The MOSEK C API manual. Version 6.0 (Revision 135).

www.mosek.com

Published by MOSEK ApS, Denmark.

Contact information

Phone	+45 3917 9907	
Fax	+45 3917 9823	
WEB	http://www.mosek.com	
Email	sales@mosek.com	Sales, pricing, and licensing.
	support@mosek.com	Technical support, questions and bug reports.
	info@mosek.com	Everything else.
Mail	MOSEK ApS C/O Symbion Science Park Fruebjergvej 3, Box 16 2100 Copenhagen Ø Denmark	

Contents

License agreement

Before using the MOSEK software, please read the license agreement available in the distribution at

`mosek\6\license.pdf`

Chapter 1

Changes and new features in MOSEK

The section presents improvements and new features added to MOSEK in version 6.0.

1.1 Compilers used to build MOSEK

MOSEK has been build with the compiler shown in Table 1.1.

Platform	C compiler
linux32x86	Intel C 11.0 (gcc 4.3, glibc 2.3.4)
linux64x86	Intel C 11.0 (gcc 4.3, glibc 2.3.4)
osx32x86	Intel C 11.1 (gcc 4.0)
osx64x86	Intel C 11.1 (gcc 4.0)
solaris32x86	Sun Studio 12
solaris64x86	Sun Studio 12
win32x86	Intel C 11.0 (VS 2005)
win64x86	Intel C 11.0 (VS 2005)

Table 1.1: Compiler version used to build MOSEK

.

1.2 General changes

- A problem analyzer is now available. It generates an simple report with of statisics and information about the optimization problem and relevant warnings about the problem formulation are included.

- A solution analyzer is now available.
- All timing measures are now wall clock times
- MOSEK employs version 1.2.3 of the zlib library.
- MOSEK employs version 11.6.1 of the FLEXnet licensing tools.
- The convexity of quadratic and quadratic constrained optimization is checked explicitly.
- On Windows all DLLs and EXEs are now signed.
- On all platforms the Jar files are signed.
- MOSEK no longer deals with ctrl-c. The user is responsible for terminating MOSEK in the callback.

1.3 Optimizers

1.3.1 Interior point optimizer

- The speed and stability of interior-point optimizer for linear problems has been improved.
- The speed and stability of the interior-point optimizer for conic problems has been improved. In particular, it is much better at dealing with primal or dual infeasible problems.

1.3.2 The simplex optimizers

- Presolve is now much more effective for simplex optimizers hot-starts.

1.3.3 Mixed-integer optimizer

- The stopping criteria for the mixed-integer optimizer have been changed to conform better with industry standards.

1.4 API changes

- The Mosek/Java API is now built for SUN Java 1.5 and later.
- The Mosek/.NET API is now built for MS .NET 2.0 and later.
- The Mosek/Python API is now based on Python CTypes and uses NumPy instead of Numeric. Python 2.5 and later is supported on all platforms where the `ctypes` module is available.

1.5 License system

- The license conditions have been relaxed, so that a license is shared among all tasks using a single environment. This means that running several optimizations in parallel will only consume one license, as long as the associated tasks share a single MOSEK environment. Please note this is NOT useful when using the MATLAB parallel toolbox.
- By default a license remains checked out for the lifetime of the environment. This behavior can be changed using the parameter `MSK_IPAR_CACHE_LICENSE`.
- Flexlm has been upgraded to version 11.6 from version 11.4.

1.6 Other changes

- The documentation has been improved.

1.7 Interfaces

- The AMPL interface has been augmented so it is possible to pass an initial (feasible) integer solution to mixed-integer optimizer.
- The AMPL interface is now capable of reading the constraint and variable names if they are avialable.

1.8 Platform changes

- MAC OSX on the PowerPC platform is no longer supported.
- Solaris on the SPARC platform is no longer supported.
- MAC OSX is supported on Intel 64 bit X86 i.e. osx64x86.
- Add support for MATLAB R2009b.

Chapter 2

About this manual

This manual covers the general functionality of MOSEK and the usage of the MOSEK C API.

The MOSEK C Application Programming Interface allows access to the full functionality of MOSEK from C and C++.

The C API consists of a header file `mosek.h` and a dynamic link library which an application can link to. This manual covers usage of the dynamic link library.

New users of the MOSEK C API are encouraged to read:

- Chapter 4 on compiling and running the distributed examples.
- The relevant parts of Chapter 5, i.e. at least the general introduction and the linear optimization section.
- Chapter 14 for a set of guidelines about developing, testing, and debugging applications employing MOSEK.

This should introduce most of the data structures and functionality necessary to implement and solve an optimization problem.

Chapter 7 contains general material about the mathematical formulations of optimization problems compatible with MOSEK, as well as common tips and tricks for reformulating problems so that they can be solved by MOSEK.

Hence, Chapter 7 is useful when trying to find a good formulation of a specific model.

More advanced examples of modelling and model debugging are located in

- Chapter 11 which deals with analysis of infeasible problems,
- Chapter 12 about the sensitivity analysis interface, and
- Chapter 13 which contains a few larger case studies.

Finally, the C API reference material is located in

- Chapter 15 which lists all types and functions,
- Chapter 16 which lists all available parameters,
- Chapter 17 which lists all response codes, and
- Chapter 18 which lists all symbolic constants.

Chapter 3

Getting support and help

3.1 MOSEK documentation

For an overview of the available MOSEK documentation please see

`mosek\6\help\index.html`

in the distribution.

3.2 Additional reading

In this manual it is assumed that the reader is familiar with mathematics and in particular mathematical optimization. Some introduction to linear programming is found in books such as "Linear programming" by Chvátal [15] or "Computer Solution of Linear Programs" by Nazareth [20]. For more theoretical aspects see e.g. "Nonlinear programming: Theory and algorithms" by Bazaraa, Shetty, and Sherali [11]. Finally, the book "Model building in mathematical programming" by Williams [26] provides an excellent introduction to modeling issues in optimization.

Another useful resource is "Mathematical Programming Glossary" available at

http://glossary.computing.society.informs.org

Chapter 4

Testing installation and compiling examples

This chapter describes how to verify that MOSEK has been installed and set up correctly, and how to compile, link and execute a C example distributed with MOSEK.

4.1 Setting up MOSEK

Usage of the MOSEK C API requires a working installation of MOSEK and the installation of a valid license file — see the MOSEK Installation Manual for instructions.

If MOSEK is installed correctly, you should be able to execute the MOSEK command line tool.

4.1.1 Windows: Checking the MOSEK installation

If MOSEK was installed using the automatic installer, the default location is

```
C:\Program Files\mosek\6\
```

unless a different path was specified.

To check that MOSEK is installed correctly, please do the following.

1. Open a DOS command prompt (DOS box).
2. Enter

   ```
   mosek.exe -f
   ```

   This will execute the MOSEK command line tool and print some relevant information. For example:

```
MOSEK Version 6.0.0.29(beta) (Build date: 2009-7-28 17:28:26)
Copyright (c) 1998-2009 MOSEK ApS, Denmark. WWW: http://www.mosek.com
Global optimizer version: 6.0.1.282. Global optimizer build date: Jul 20 2009 15:24:58
Barrier Solver Version 6.0.0.029,
Platform Windows x64.
64 bit architecture.
Using FLEXlm version: 11.6.
Hostname: 'morud' Hostid: '"0015c5f32f32 0015c5f32f30"'

Operating system variables
MOSEKLM_LICENSE_FILE     :
PATH                     : c:\local\python24;c:\local\bin;C:\Program Files\mosek\6\tools\platfor

*** Warning: No input file specified.
             Common usage of the MOSEK command line tool is:

             mosek file_name

Return code - 0  [MSK_RES_OK]
```

3. Verify that

   - The program is executed. If the system was unable to recognize `mosek.exe` as a valid command, then the `PATH` environment variable has not been set correctly.
   - The MOSEK version printed matches the expected version.
   - The `MOSEKLM_LICENSE_FILE` points to the correct license file or to the directory containing it. Note that if it points to a directory containing several license files, there is a risk that it will use the wrong one.
   - The `PATH` contains the path to the correct MOSEK installation.

4.1.2 Linux: Checking the MOSEK installation

There is no automatic installer for MOSEK on Linux, thus installation is performed manually: See MOSEK Installation Manual for details.

To check that MOSEK is installed correctly, please do the following:

1. Open a command prompt.
2. Enter

   ```
   mosek -f
   ```

   This will execute the MOSEK command line tool and print some relevant information. For example:

```
MOSEK Version 5.0.0.3(alpha) (Build date: Nov 23 2006 10:56:35)
Copyright (c) 1998-2006 MOSEK ApS, Denmark. WWW: http://www.mosek.com
Global optimizer version: 4.50.343. Global optimizer build date: Nov 10 2006 08:37:51.
Using FLEXlm version: 11.3.
Hostname: 'kolding' Hostid: '00001a1a5a6a'

Operating system variables
MOSEKLM_LICENSE_FILE     : /home/ulfw/mosek/5/licenses
LD_LIBRARY_PATH          : /home/ulfw/mosek/5/tools/platform/win/bin:/home/ulfw/lib

*** Warning: No input file specified.
             Common usage of the MOSEK command line tool is:

             mosek file_name

Return code - 0  [MSK_RES_OK]
```

3. Verify that
   - The program is executed. If the system was unable to locate `mosek`, then the `PATH` environment variable bas not been set correctly.
   - The MOSEK version printed matches the expected version.
   - The `MOSEKLM_LICENSE_FILE` points to the correct license file or to the directory containing it. If it points to a directory containing several license files, there is a risk that it will use the wrong one.
   - The `LD_LIBRARY_PATH` contains the path to the correct MOSEK installation.

4.1.3 MacOSX: Checking the MOSEK installation

There is no automatic installer for MOSEK on Linux. Installation is performed manually: See MOSEK Installation Manual for details.

To check that MOSEK is correctly installed, go though the following steps.

1. Open a command prompt.
2. Enter

   ```
   mosek -f
   ```

   This will execute the MOSEK command line tool and print some relevant information.
3. Verify that
   - The program was executed. If the system was unable to locate `mosek`, then the `PATH` environment variable was not correctly set.
   - The MOSEK version printed matches the expected version.

- The `MOSEKLM_LICENSE_FILE` points to the correct license file or the directory containing it. If it points to a directory containing several license files, there is a risk that it will use to wrong one.
- The `DYLD_LIBRARY_PATH` should contain the path to the correct MOSEK installation.

4.2 Compiling and linking

This section demonstrates how to compile, link and run the example `lo1.c` included with MOSEK. The general requirements for a program linking to the MOSEK library are the same as for `lo1.c`.

It is assumed that MOSEKis installed, and that there is a working C compiler on the system.

4.2.1 Compiling under Microsoft Windows

We assume that MOSEK is installed under the default path

```
c:\Program Files\mosek\6
```

and that the platform-specific files are located in

```
c:\Program Files\mosek\6\tools\platform\<platform>\
```

where `<platform>` is `win` (32-bit Windows), `win64x86` (64-bit Windows AMD64 or Intel64) or `winia64` (Windows Itanium).

4.2.1.1 Compiling examples using NMake

The example directory contains makefiles for use with Microsoft NMake. This requires that paths and environment are set up for the Visual Studio tool chain (usually, the submenu containing Visual Studio also contains a *Visual Studio Command Prompt* which does the necessary setup).

To build the examples, open a DOS box and change directory to the examples directory. For Windows with default installation directories, the example directory is

```
c:\Program Files\mosek\6\tools\examples\c
```

The directory contains a makefile named "`Makefile`". To compile all examples, run the command

```
nmake /f Makefile all
```

To only build a single example instead of all examples, replace "`all`" by the corresponding executable name. For example, to build `lo1.exe` type

```
nmake /f Makefile lo1.exe
```

4.2.1.2 Compiling from command line

To compile and run a C example using the MOSEK `dll`, the following files are required:

- `mosek.h`. The header file defining all functions and constants in MOSEK

  `c:\Program Files\mosek\6\tools\platform\<platform>\h\mosek.h`

- The MOSEK `lib` file located in

  `c:\Program Files\mosek\6\tools\platform\<platform>\dll`

  The relevant `lib` file is

  - on 64-bit Microsoft Windows (AMD x64 or Intel EMT64)

    `mosek64_6_0.lib`

  - on 32-bit Microsoft Windows

    `mosek6_0.lib`

- The MOSEK solver `dll` located in

  `c:\Program Files\mosek\6\tools\platform\<platform>\bin`

  The relevant `dll` file is

  - on 64-bit Microsoft Windows (AMD x64 or Intel EMT64)

    `mosek64_6_0.dll`

  - on 32-bit Microsoft Windows

    `mosek6_0.dll`

Finally, the distributed C examples are located in the directory

`c:\Program Files\mosek\6\tools\examples\c`

To compile and execute the distributed example `lo1.c`, do the following:

1. Change directory:

   ```
   c:
   cd "\Program Files\mosek\6\tools"
   ```

2. Compile the example into an executable `lo1.exe` (we assume that the Visual Studio C compiler `cl.exe` is available). For Windows 32

   ```
   cl examples\c\lo1.c /out:lo1.exe /I platform\win32x86\h\mosek.h platform\win32x86\bin\mosek6_0.l
   ```

For Windows 64:

```
cl examples\c\lo1.c /out:lo1.exe /I platform\win64x86\h\mosek.h platform\win64x86\bin\mosek64_6_
```

3. To run the compiled examples, enter

```
./lo1.exe
```

4.2.1.3 Adding MOSEK to a Visual Studio Project

The following walk-through is specific for Microsoft Visual Studio 7 (.NET), but may work for other versions too.

To compile a project linking to MOSEK in Visual Studio, the following steps are necessary:

- Create a project or open an existing project in Visual Studio.
- In the **Solution Explorer** right-click on the relevant project and select **Properties**. This will open the **Property pages** dialog.
- In the selection box **Configuration:** select **All Configurations**.
- In the tree-view open **Configuration Properties** → **C/C++** → **General**.
- In the properties view select **Additional Include Directories** and click on the ellipsis "**...**".
- Click on the **New Folder** button and write the *full path* to the `mosek.h` header file or browse for the file by clicking the ellipsis "**...**". For example, for 32-bit Windows enter

  ```
  C:\Program Files\mosek\6\tools\platform\win32x86\h
  ```

- Click **OK**.
- Back in the **Property Pages** dialog select from the tree-view **Configuration Properties** → **Linker** → **Input**.
- In the properties view select **Additional Dependencies** and click on the ellipsis "**...**". This will open the **Additional Dependencies** dialog.
- In the text field enter the full path of the MOSEK `lib` on a new line. For example, for 32-bit Windows

  ```
  C:\Program Files\mosek\6\tools\platform\win32x86\bin\mosek6_0.lib
  ```

- Click **OK**.
- Back in the **Property Pages** dialog click **OK**.

Additionally, if you want to add the `mosek.h` header file to your project, do the following:

- In the **Solution Explorer** right-click on the relevant project and select **Add** → **Add Existing Item**.
- Locate and select the `mosek.h` header file and click **OK**.

4.2.2 UNIX versions

The `mosek.h` header file which must be included in all files that uses MOSEK functions is located in the directory

`mosek/6/tools/platform/<platform>/h/mosek.h`

and the MOSEK shared (or dynamic) library is located in

`mosek/6/tools/platform/<platform>/bin/libmosek64.so.6.0`

for 64-bit architectures, and in

`mosek/6/tools/platform/<platform>/bin/libmosek.so.6.0`

for 32-bit architectures, where `<platform>` represents a particular UNIX platform, e.g.

- `linux32x86`,
- `linux64x86`,
- `osx32ppc`,
- `osx32x86`,
- `solarissparc`, or
- `solarissparc64`.

Programs linking with MOSEK must be linked to several libraries. A script for linking the MOSEK examples can is located in

`mosek/<version>/test/testunix.sh`

This script contains the definitions:

```
case $MSKPLATFORM in

  linux32x86)
    MSKCC="gcc"
    MSKLINKFLAGS="-lc -ldl -lm"
```

```
    ;;

  linux64x86)
    MSKCC="gcc -m64"
    MSKLINKFLAGS="-lc -ldl -lm"
    ;;

  solarissparc )
    MSKDIR=solaris/sparc
    MSKCC=cc
    MSKLINKFLAGS="-lsocket -lnsl -lintl -lthread -lpthread -lc -ldl -lm"
    ;;

  solarissparc64 )
    MSKDIR=solaris/sparc64
    MSKPLATFORM=solaris/sparc64
    MSKCC="cc -xtarget=generic64"
    MSKLINKFLAGS="-lsocket -lnsl -lintl -lthread -lpthread -lc -ldl -lm"
    ;;
esac
```

In the `testunix.sh` script the `MSKLINKFLAGS` variable is defined for each platform. `MSKLINKFLAGS` contains the link flags that must be added to the command line when linking against the MOSEK dynamic library.

4.2.2.1 Compiling examples using GMake

The example directory contains makefiles for use with GNU Make.

To build the examples, open a prompt and change directory to the examples directory. For Linux with default installation path, the examples directory is

```
mosek/6/tools/examples/c
```

The directory contains a makefile for GNU Make and gcc. To build all examples, go to the examples and enter

```
gmake all
```

To build one example instead of all examples, replace "`all`" by the corresponding executable name. For example, to build the `lo1` executable type

```
gmake lo1
```

4.2.2.2 Example: Linking with GNU C under Linux

The following example shows how to link to the MOSEK shared library on a 64bit platform:

```
# The -L. tells gcc to look for shared libraries in the directory ./
# The -lmosek  tells gcc to link to the mosek library

# Set environment variable so the MOSEK shared library
# can be located. Must be done at both link time and run time.

export LD_LIBRARY_PATH=./platform/linux64x86/bin/:$LD_LIBRARY_PATH

# Replace -lmosek with -lmosek if you are linking on a 32-bit platform.

gcc examp/lo1.c -o lo1 -I h/ -L platform/linux32x86/bin/ \
                -lmosek -pthread -lc -ldl -lm

# Run lo1 executable
./lo1
```

Please note that linking with the "`-pthread`" flag is requires by Intel's OpenMP library.

The distribution contains a MOSEK library built without OpenMP; to use this instead of the normal MOSEK library, replace "`-lmosek`" by "`-lmoseknoomp`, and "`-lmosek64`" by "`-lmoseknoomp64`".

4.2.2.3 Example: Linking with Sun C on Solaris

The following example shows how to link to the MOSEK shared library.

```
# The -L. tells cc to look for shared libraries in the directory ./
# The -lmosek  tells cc to link to the mosek library.

# Replace -lmosek with -lmosek64 if you are linking on a 64-bit platform.

cc examp/lo1.c -o lo1 -I h/ -L platform/solaris/sparc/dll/ -lmosek \
                -lsocket -lnsl -lintl -lthread -lpthread -lc -ldl -lm

# Set environment variable so the MOSEK shared library
# can be located.
LD_LIBRARY_PATH=./platform/linux/intel/dll/:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH

# Run lo1 executable
./lo1
```

Chapter 5

Basic API tutorial

In this chapter the reader will learn how to build a simple application that uses MOSEK.

A number of examples is provided to demonstrate the functionality required for solving linear, quadratic, and conic problems as well as mixed integer problems.

Please note that the section on linear optimization also describes most of the basic functionality that is not specific to linear problems. Hence, it is recommended to read Section 5.2 before reading the rest of this chapter.

5.1 The basics

A typical program using the MOSEK C interface can be described shortly:

1. Create an environment (`MSKenv_t`) object.
2. Set up some environment specific data and initialize the environment object.
3. Create a task (`MSKtask_t`) object.
4. Load a problem into the task object.
5. Optimize the problem.
6. Fetch the result.
7. Delete the environment and task objects.

5.1.1 The environment and the task

The first MOSEK related step in any program that employs MOSEK is to create an environment (`MSKenv_t`) object. The environment contains environment specific data such as information about the

license file, streams for environment messages etc. Before creating any task objects, the environment must be initialized using MSK_initenv. When this is done one or more task (MSKtask_t) objects can be created. Each task is associated with a single environment and defines a complete optimization problem as well as task message streams and optimization parameters.

In C, the creation of an environment and a task could like this:

```
{
  MSKenv_t    env = NULL;
  MSKtask_t   task = NULL;
  MSKrescodee res;

  /* Create an environment */
  res = MSK_makeenv(&env, NULL,NULL,NULL,NULL);

  /* You may connect streams and other callbacks to env here */

  /* Initialize the environment */
  if (res == MSK_RES_OK)
    res = MSK_initenv(env)

  /* Create a task */
  if (res == MSK_RES_OK)
    res = MSK_maketask(env, 0,0, &task);
  ...
  /* input some task data, optimize etc. */
  ...
  MSK_deletetask(&task);
  MSK_deleteenv(&env);
}
```

Please note that an environment should, if possible, be shared between multiple tasks.

5.1.2 A simple working example

The following simple example shows a working C program which

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

```
1 /*
2   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
```

```
3
4     File:    simple.c
5
6     Purpose: To demonstrate a very simple example using MOSEK by
7              reading a problem file, solving the problem and
8              writing the solution to a file.
9  */
10
11 #include "mosek.h"
12
13 int main (int argc, char * argv[])
14 {
15   MSKtask_t   task = NULL;
16   MSKenv_t    env  = NULL;
17   MSKrescodee res  = MSK_RES_OK;
18
19   if (argc <= 1)
20   {
21     printf ("Missing argument. The syntax is:\n");
22     printf (" simple inputfile [ solutionfile ]\n");
23   }
24   else
25   {
26     /* Create the mosek environment.
27        The 'NULL' arguments here, are used to specify customized
28        memory allocators and a memory debug file. These can
29        safely be ignored for now. */
30
31     res = MSK_makeenv(&env, NULL, NULL, NULL, NULL);
32
33     /* Initialize the environment */
34     if ( res==MSK_RES_OK )
35       MSK_initenv (env);
36
37     /* Create a task object linked to the environment env.
38        Initially we create it with 0 variables and 0 columns,
39        since we do not know the size of the problem. */
40     if ( res==MSK_RES_OK )
41       res = MSK_maketask (env, 0,0, &task);
42
43     /* We assume that a problem file was given as the first command
44        line argument (received in 'argv'). */
45     if ( res==MSK_RES_OK )
46       res = MSK_readdata (task, argv[1]);
47
48     /* Solve the problem */
49     if ( res==MSK_RES_OK )
50       MSK_optimize(task);
51
52     /* Print a summary of the solution. */
53     MSK_solutionsummary(task, MSK_STREAM_MSG);
54
55     /* If an output file was specified, write a solution */
56     if ( res==MSK_RES_OK && argc>2 )
57     {
58       /* We define the output format to be OPF, and tell MOSEK to
59          leave out parameters and problem data from the output file. */
60       MSK_putintparam (task,MSK_IPAR_WRITE_DATA_FORMAT,    MSK_DATA_FORMAT_OP);
```

```
61        MSK_putintparam (task,MSK_IPAR_OPF_WRITE_SOLUTIONS,  MSK_ON);
62        MSK_putintparam (task,MSK_IPAR_OPF_WRITE_HINTS,      MSK_OFF);
63        MSK_putintparam (task,MSK_IPAR_OPF_WRITE_PARAMETERS, MSK_OFF);
64        MSK_putintparam (task,MSK_IPAR_OPF_WRITE_PROBLEM,    MSK_OFF);
65        MSK_writedata(task,argv[2]);
66      }
67
68      MSK_deletetask(&task);
69      MSK_deleteenv(&env);
70    }
71    return res;
72  }
```

5.1.2.1 Writing a problem to a file

It is frequently beneficial to write a problem to a file that can be stored for later use or inspected visually. The MSK_writedata function is used write a problem to a file as follows

```
65  MSK_writedata(task,argv[2]);
```

By default the extension of the filename is the format written. I.e. the filename `somename.opf` implies the file is written in the OPF format.

Similarly, the function MSK_readdata reads a problem from a file:

```
46  res = MSK_readdata (task, argv[1]);
```

5.1.2.2 Inputting and outputting problem data

An optimization problem consists of several components; objective, objective sense, constraints, variable bounds etc. Therefore, the task (`MSKtask_t`) provides a number of methods to operate on the task specific data, all of which are listed in Section 15.4.

5.1.2.3 Setting parameters

Apart from the problem data, the task contains a number of parameters defining the behavior of MOSEK. For example the MSK_IPAR_OPTIMIZER parameter defines which optimizer to use. A complete list of all parameters are listed in Chapter 16.

5.1.3 Compiling and running examples

All examples presented in this chapter are distributed with MOSEK and are available in the directory

```
mosek/6/tools/examples/
```

in the MOSEK installation. Chapter 4 describes how to compile and run the examples.

It is recommended to copy examples to a different directory before modifying and compiling them.

5.2 Linear optimization

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

Minimize or maximize the objective function

$$\sum_{j=0}^{n-1} c_j x_j + c^f \tag{5.1}$$

subject to the linear constraints

$$l_k^c \leq \sum_{j=0}^{n-1} a_{kj} x_j \leq u_k^c,\ k = 0, \ldots, m-1, \tag{5.2}$$

and the bounds

$$l_j^x \leq x_j \leq u_j^x,\ j = 0, \ldots, n-1, \tag{5.3}$$

where we have used the problem elements

m and n, which are the number of constraints and variables respectively,

x, which is the variable vector of length n,

c, which is a coefficient vector of size n

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

c^f, which is a constant,

A, which is a $m \times n$ matrix of coefficients is given by

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

l^c and u^c, which specify the lower and upper bounds on constraints respectively, and

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

Please note the unconventional notation using 0 as the first index rather than 1. Hence, x_0 is the first element in variable vector x. This convention has been adapted from C arrays which are indexed from 0.

5.2.1 Linear optimization example: lo1

The following is an example of a linear optimization problem:

$$\begin{array}{lrcrcrcrcl} \text{maximize} & 3x_0 & + & 1x_1 & + & 5x_2 & + & 1x_3 & & \\ \text{subject to} & 3x_0 & + & 1x_1 & + & 2x_2 & & & = & 30, \\ & 2x_0 & + & 1x_1 & + & 3x_2 & + & 1x_3 & \geq & 15, \\ & & & 2x_1 & & & + & 3x_3 & \leq & 25, \end{array} \tag{5.4}$$

having the bounds

$$\begin{array}{rcrcl} 0 & \leq & x_0 & \leq & \infty, \\ 0 & \leq & x_1 & \leq & 10, \\ 0 & \leq & x_2 & \leq & \infty, \\ 0 & \leq & x_3 & \leq & \infty. \end{array} \tag{5.5}$$

5.2.1.1 Solving the problem

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

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

Below we explain each of these steps. For the complete source code see section 5.2.1.2. The code can also be found in:

```
mosek\6\tools\examples\c\lo1.c
```

Create an environment. Before setting up the optimization problem, a MOSEK environment must be created and initialized. This is done in the lines:

```
56 r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
57
58 /* Directs the env log stream to the 'printstr' function. */
59 if ( r==MSK_RES_OK )
60   MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
61
62 /* Initialize the environment. */
63 if ( r==MSK_RES_OK )
64   r = MSK_initenv(env);
```

We connect a call-back function to the environment log stream. In this case the call-back function simply prints messages to the standard output stream.

Create an optimization task. Next, an empty task object is created:

```
68 /* Create the optimization task. */
69 r = MSK_maketask(env,NUMCON,NUMVAR,&task);
70
71 /* Directs the log task stream to the 'printstr' function. */
72 if ( r==MSK_RES_OK )
73   MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
```

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

Load a problem into the task object. First an estimate of the size of the input data is set. This is done to increase the speed of inputting data and is optional.

```
78 if (r == MSK_RES_OK)
79   r = MSK_putmaxnumvar(task,NUMVAR);
80
81 if (r == MSK_RES_OK)
82   r = MSK_putmaxnumcon(task,NUMCON);
83
84 if (r == MSK_RES_OK)
85   r = MSK_putmaxnumanz(task,NUMANZ);
```

Before any problem data can be set, variables and constraints must be added to the problem via calls to the function MSK_append.

```
87 /* Append 'NUMCON' empty constraints.
88  The constraints will initially have no bounds. */
89 if ( r == MSK_RES_OK )
90   r = MSK_append(task,MSK_ACC_CON,NUMCON);
91
92 /* Append 'NUMVAR' variables.
93  The variables will initially be fixed at zero (x=0). */
94 if ( r == MSK_RES_OK )
95   r = MSK_append(task,MSK_ACC_VAR,NUMVAR);
```

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

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

```
102 /* Set the linear term c_j in the objective.*/
103 if(r == MSK_RES_OK)
104   r = MSK_putcj(task,j,c[j]);
```

The bounds on variables are stored in the arrays

```
48 MSKboundkeye bkx[]  = {MSK_BK_LO,      MSK_BK_RA, MSK_BK_LO,      MSK_BK_LO      };
49 double        blx[]  = {0.0,            0.0,       0.0,            0.0            };
50 double        bux[]  = {+MSK_INFINITY, 10.0,      +MSK_INFINITY, +MSK_INFINITY };
```

Bound key	Type of bound	Lower bound	Upper bound
MSK_BK_FX	$\cdots = l_j$	Finite	Identical to the lower bound
MSK_BK_FR	Free	Minus infinity	Plus infinity
MSK_BK_LO	$l_j \leq \cdots$	Finite	Plus infinity
MSK_BK_RA	$l_j \leq \cdots \leq u_j$	Finite	Finite
MSK_BK_UP	$\cdots \leq u_j$	Minus infinity	Finite

Table 5.1: Interpretation of the bound keys.

and are set with calls to MSK_putbound.

```
106 /* Set the bounds on variable j.
107  blx[j] <= x_j <= bux[j] */
108 if(r == MSK_RES_OK)
109   r = MSK_putbound(task,
110                    MSK_ACC_VAR, /* Put bounds on variables.*/
111                    j,           /* Index of variable.*/
112                    bkx[j],      /* Bound key.*/
113                    blx[j],      /* Numerical value of lower bound.*/
114                    bux[j]);     /* Numerical value of upper bound.*/
```

The *Bound key* stored in bkx specify the type of the bound according to Table 5.1. For instance bkx[0]= MSK_BK_LO means that $x_0 \geq l_0^x$. Finally, the numerical values of the bounds on variables are given by

$$l_j^x = \texttt{blx[j]} \tag{5.6}$$

and

$$u_j^x = \texttt{bux[j]}. \tag{5.7}$$

Recall that in our example the A matrix is given by

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

This matrix is stored in sparse format in the arrays:

```
32 MSKlidxt      aptrb[] = {0, 2, 5, 7};
33 MSKlidxt      aptre[] = {2, 5, 7, 9};
34 MSKidxt       asub[] = { 0, 1,
35                          0, 1, 2,
36                          0, 1,
37                          1, 2};
38 double        aval[] = { 3.0, 2.0,
39                          1.0, 1.0, 2.0,
40                          2.0, 3.0,
41                          1.0, 3.0};
```

The ptrb, ptre, asub, and aval arguments define the constraint matrix A in the column ordered sparse format (for details, see Section 5.8.3.2).

Using the function MSK_putavec we set column j of A

```
/* Input column j of A */
if(r == MSK_RES_OK)
  r = MSK_putavec(task,
                  MSK_ACC_VAR,       /* Input columns of A.*/
                  j,                 /* Variable (column) index.*/
                  aptre[j]-aptrb[j], /* Number of non-zeros in column j.*/
                  asub+aptrb[j],     /* Pointer to row indexes of column j.*/
                  aval+aptrb[j]);    /* Pointer to Values of column j.*/
```

Alternatively, the same A matrix can be set one row at a time; please see section 5.2.2 for an example.

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

```
/* Set the bounds on constraints.
   for i=1, ...,NUMCON : blc[i] <= constraint i <= buc[i] */
for(i=0; i<NUMCON && r==MSK_RES_OK; ++i)
  r = MSK_putbound(task,
                   MSK_ACC_CON, /* Put bounds on constraints.*/
                   i,           /* Index of constraint.*/
                   bkc[i],      /* Bound key.*/
                   blc[i],      /* Numerical value of lower bound.*/
                   buc[i]);     /* Numerical value of upper bound.*/
```

Optimization: After the problem is set-up the task can be optimized by calling the function MSK_optimizetrm.

```
r = MSK_optimizetrm(task,&trmcode);
```

Extracting the solution. After optimizing the status of the solution is examined with a call to MSK_getsolutionstatus. If the solution status is reported as MSK_SOL_STA_OPTIMAL or MSK_SOL_STA_NEAR_OPTIMAL the solution is extracted in the lines below:

```
MSK_getsolutionslice(task,
                     MSK_SOL_BAS,    /* Request the basic solution. */
                     MSK_SOL_ITEM_XX,/* Which part of solution.     */
                     0,              /* Index of first variable.    */
                     NUMVAR,         /* Index of last variable+1.   */
                     xx);
```

The MSK_getsolutionslice function obtains a "slice" of the solution. MOSEK may compute several solutions depending on the optimizer employed. In this example the *basic solution* is requested by setting the second argument to MSK_SOL_BAS. The third argument MSK_SOL_ITEM_XX specifies that we want the variable values of the solution. The two following arguments `0` and `NUMVAR` specifies the range of variable values we want.

The range specified is the first index (here "0") up to but not including the second index (`here ''NUMVAR''`).

5.2.1.2 Source code for lo1

```
/*
  Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

  File:      lo1.c

  Purpose:   To demonstrate how to solve a small linear
             optimization problem using the MOSEK C API.
*/

#include <stdio.h>
#include "mosek.h"

/* This function prints log output from MOSEK to the terminal. */
static void MSKAPI printstr(void *handle,
                            char str[])
{
  printf("%s",str);
} /* printstr */

#define NUMVAR 4
#define NUMCON 3
#define NUMANZ 9

int main(int argc,char *argv[])
{
  MSKrescodee  r;
  MSKidxt i,j;
  double       c[]    = {3.0, 1.0, 5.0, 1.0};

  /* Below is the sparse representation of the A
     matrix stored by column. */
  MSKlidxt     aptrb[] = {0, 2, 5, 7};
  MSKlidxt     aptre[] = {2, 5, 7, 9};
  MSKidxt      asub[] = { 0, 1,
                          0, 1, 2,
                          0, 1,
                          1, 2};
  double       aval[] = { 3.0, 2.0,
                          1.0, 1.0, 2.0,
                          2.0, 3.0,
                          1.0, 3.0};

  /* Bounds on constraints. */
  MSKboundkeye bkc[]  = {MSK_BK_FX, MSK_BK_LO,     MSK_BK_UP    };
  double       blc[]  = {30.0,      15.0,          -MSK_INFINITY};
  double       buc[]  = {30.0,      +MSK_INFINITY, 25.0         };
  /* Bounds on variables. */
  MSKboundkeye bkx[]  = {MSK_BK_LO,     MSK_BK_RA, MSK_BK_LO,     MSK_BK_LO     };
  double       blx[]  = {0.0,           0.0,       0.0,           0.0           };
  double       bux[]  = {+MSK_INFINITY, 10.0,      +MSK_INFINITY, +MSK_INFINITY };
  double xx[NUMVAR];
  MSKenv_t     env  = NULL;
  MSKtask_t    task = NULL;

  /* Create the mosek environment. */
  r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);

```

```
58    /* Directs the env log stream to the 'printstr' function. */
59    if ( r==MSK_RES_OK )
60      MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
61
62    /* Initialize the environment. */
63    if ( r==MSK_RES_OK )
64      r = MSK_initenv(env);
65
66    if ( r==MSK_RES_OK )
67    {
68      /* Create the optimization task. */
69      r = MSK_maketask(env,NUMCON,NUMVAR,&task);
70
71      /* Directs the log task stream to the 'printstr' function. */
72      if ( r==MSK_RES_OK )
73        MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
74
75      /* Give MOSEK an estimate of the size of the input data.
76       This is done to increase the speed of inputting data.
77       However, it is optional. */
78      if (r == MSK_RES_OK)
79        r = MSK_putmaxnumvar(task,NUMVAR);
80
81      if (r == MSK_RES_OK)
82        r = MSK_putmaxnumcon(task,NUMCON);
83
84      if (r == MSK_RES_OK)
85        r = MSK_putmaxnumanz(task,NUMANZ);
86
87      /* Append 'NUMCON' empty constraints.
88       The constraints will initially have no bounds. */
89      if ( r == MSK_RES_OK )
90        r = MSK_append(task,MSK_ACC_CON,NUMCON);
91
92      /* Append 'NUMVAR' variables.
93       The variables will initially be fixed at zero (x=0). */
94      if ( r == MSK_RES_OK )
95        r = MSK_append(task,MSK_ACC_VAR,NUMVAR);
96
97      /* Optionally add a constant term to the objective. */
98      if ( r ==MSK_RES_OK )
99        r = MSK_putcfix(task,0.0);
100     for(j=0; j<NUMVAR && r == MSK_RES_OK; ++j)
101     {
102       /* Set the linear term c_j in the objective.*/
103       if(r == MSK_RES_OK)
104         r = MSK_putcj(task,j,c[j]);
105
106       /* Set the bounds on variable j.
107        blx[j] <= x_j <= bux[j] */
108       if(r == MSK_RES_OK)
109         r = MSK_putbound(task,
110                          MSK_ACC_VAR, /* Put bounds on variables.*/
111                          j,           /* Index of variable.*/
112                          bkx[j],      /* Bound key.*/
113                          blx[j],      /* Numerical value of lower bound.*/
114                          bux[j]);     /* Numerical value of upper bound.*/
115
```

```
116       /* Input column j of A */
117       if(r == MSK_RES_OK)
118         r = MSK_putavec(task,
119                         MSK_ACC_VAR,       /* Input columns of A.*/
120                         j,                 /* Variable (column) index.*/
121                         aptre[j]-aptrb[j], /* Number of non-zeros in column j.*/
122                         asub+aptrb[j],     /* Pointer to row indexes of column j.*/
123                         aval+aptrb[j]);    /* Pointer to Values of column j.*/
124
125     }
126
127     /* Set the bounds on constraints.
128        for i=1, ...,NUMCON : blc[i] <= constraint i <= buc[i] */
129     for(i=0; i<NUMCON && r==MSK_RES_OK; ++i)
130       r = MSK_putbound(task,
131                        MSK_ACC_CON, /* Put bounds on constraints.*/
132                        i,           /* Index of constraint.*/
133                        bkc[i],      /* Bound key.*/
134                        blc[i],      /* Numerical value of lower bound.*/
135                        buc[i]);     /* Numerical value of upper bound.*/
136
137     /* Maximize objective function. */
138     if (r == MSK_RES_OK)
139       r = MSK_putobjsense(task,
140                           MSK_OBJECTIVE_SENSE_MAXIMIZE);
141
142     if ( r==MSK_RES_OK )
143     {
144       MSKrescodee trmcode;
145
146       /* Run optimizer */
147       r = MSK_optimizetrm(task,&trmcode);
148
149       /* Print a summary containing information
150        about the solution for debugging purposes. */
151       MSK_solutionsummary (task,MSK_STREAM_LOG);
152
153       if ( r==MSK_RES_OK )
154       {
155         MSKsolstae solsta;
156         int j;
157         MSK_getsolutionstatus (task,
158                                MSK_SOL_BAS,
159                                NULL,
160                                &solsta);
161         switch(solsta)
162         {
163           case MSK_SOL_STA_OPTIMAL:
164           case MSK_SOL_STA_NEAR_OPTIMAL:
165             MSK_getsolutionslice(task,
166                                  MSK_SOL_BAS,    /* Request the basic solution. */
167                                  MSK_SOL_ITEM_XX,/* Which part of solution.     */
168                                  0,              /* Index of first variable.    */
169                                  NUMVAR,         /* Index of last variable+1.   */
170                                  xx);
171
172             printf("Optimal primal solution\n");
173             for(j=0; j<NUMVAR; ++j)
```

```
174               printf("x[%d]: %e\n",j,xx[j]);
175
176             break;
177           case MSK_SOL_STA_DUAL_INFEAS_CER:
178           case MSK_SOL_STA_PRIM_INFEAS_CER:
179           case MSK_SOL_STA_NEAR_DUAL_INFEAS_CER:
180           case MSK_SOL_STA_NEAR_PRIM_INFEAS_CER:
181             printf("Primal or dual infeasibility certificate found.\n");
182             break;
183
184           case MSK_SOL_STA_UNKNOWN:
185             printf("The status of the solution could not be determined.\n");
186             break;
187           default:
188             printf("Other solution status.");
189             break;
190         }
191       }
192       else
193       {
194         printf("Error while optimizing.\n");
195       }
196     }
197
198     if (r != MSK_RES_OK)
199     {
200       /* In case of an error print error code and description. */
201       char symname[MSK_MAX_STR_LEN];
202       char desc[MSK_MAX_STR_LEN];
203
204       printf("An error occurred while optimizing.\n");
205       MSK_getcodedesc (r,
206                        symname,
207                        desc);
208       printf("Error %s - '%s'\n",symname,desc);
209     }
210
211     MSK_deletetask(&task);
212
213     MSK_deleteenv(&env);
214   }
215
216   return r;
217 }
```

5.2.2 Row-wise input

In the previous example the A matrix is set one column at a time. Alternatively the same matrix can be set one row at a time or the two methods can be mixed as in the example in section 5.6. The following example show how to set the A matrix by rows.

The source code for this example can be found in:

```
mosek\6\tools\examples\c\lo2.c
```

```
/*
  Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

  File:      lo2.c

  Purpose:   To demonstrate how to solve a small linear
             optimization problem using the MOSEK C API.
*/

#include <stdio.h>
#include "mosek.h"

/* This function prints log output from MOSEK to the terminal. */
static void MSKAPI printstr(void *handle,
                            char str[])
{
  printf("%s",str);
} /* printstr */

#define NUMVAR 4
#define NUMCON 3
#define NUMANZ 9

int main(int argc,char *argv[])
{
  MSKrescodee  r;
  MSKidxt i,j;
  double       c[]    = {3.0, 1.0, 5.0, 1.0};

  /* Below is the sparse representation of the A
     matrix stored by row. */
  MSKlidxt     aptrb[] = {0, 3, 7};
  MSKlidxt     aptre[] = {3, 7, 9};
  MSKidxt      asub[] = { 0,1,2,
                          0,1,2,3,
                          1,3};
  double       aval[] = { 3.0, 1.0, 2.0,
                          2.0, 1.0, 3.0, 1.0,
                          2.0, 3.0};
  /* Bounds on constraints. */
  MSKboundkeye bkc[]  = {MSK_BK_FX, MSK_BK_LO,     MSK_BK_UP    };
  double       blc[]  = {30.0,      15.0,          -MSK_INFINITY};
  double       buc[]  = {30.0,      +MSK_INFINITY, 25.0         };
  /* Bounds on variables. */
  MSKboundkeye bkx[]  = {MSK_BK_LO,     MSK_BK_RA, MSK_BK_LO,     MSK_BK_LO     };
  double       blx[]  = {0.0,           0.0,       0.0,           0.0           };
  double       bux[]  = {+MSK_INFINITY, 10.0,      +MSK_INFINITY, +MSK_INFINITY };

  double xx[NUMVAR];
  MSKenv_t     env  = NULL;
  MSKtask_t    task = NULL;

  /* Create the mosek environment. */
  r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);

  /* Directs the env log stream to the 'printstr' function. */
  if ( r==MSK_RES_OK )
```

```
58      MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
59
60    /* Initialize the environment. */
61    if ( r==MSK_RES_OK )
62      r = MSK_initenv(env);
63
64    if ( r==MSK_RES_OK )
65    {
66      /* Create the optimization task. */
67      r = MSK_maketask(env,NUMCON,NUMVAR,&task);
68
69      /* Directs the log task stream to the 'printstr' function. */
70      if ( r==MSK_RES_OK )
71        MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
72
73      /* Give MOSEK an estimate of the size of the input data.
74       This is done to increase the speed of inputting data.
75       However, it is optional. */
76      if (r == MSK_RES_OK)
77        r = MSK_putmaxnumvar(task,NUMVAR);
78
79      if (r == MSK_RES_OK)
80        r = MSK_putmaxnumcon(task,NUMCON);
81
82      if (r == MSK_RES_OK)
83        r = MSK_putmaxnumanz(task,NUMANZ);
84
85      /* Append 'NUMCON' empty constraints.
86       The constraints will initially have no bounds. */
87      if ( r == MSK_RES_OK )
88        r = MSK_append(task,MSK_ACC_CON,NUMCON);
89
90      /* Append 'NUMVAR' variables.
91       The variables will initially be fixed at zero (x=0). */
92      if ( r == MSK_RES_OK )
93        r = MSK_append(task,MSK_ACC_VAR,NUMVAR);
94
95      /* Optionally add a constant term to the objective. */
96      if ( r ==MSK_RES_OK )
97        r = MSK_putcfix(task,0.0);
98      for(j=0; j<NUMVAR && r == MSK_RES_OK; ++j)
99      {
100       /* Set the linear term c_j in the objective.*/
101       if(r == MSK_RES_OK)
102         r = MSK_putcj(task,j,c[j]);
103
104       /* Set the bounds on variable j.
105        blx[j] <= x_j <= bux[j] */
106       if(r == MSK_RES_OK)
107         r = MSK_putbound(task,
108                          MSK_ACC_VAR, /* Put bounds on variables.*/
109                          j,           /* Index of variable.*/
110                          bkx[j],      /* Bound key.*/
111                          blx[j],      /* Numerical value of lower bound.*/
112                          bux[j]);     /* Numerical value of upper bound.*/
113     }
114
115     /* Set the bounds on constraints.
```

```
116       for i=1, ...,NUMCON : blc[i] <= constraint i <= buc[i] */
117     for(i=0; i<NUMCON && r==MSK_RES_OK; ++i)
118     {
119       r = MSK_putbound(task,
120                        MSK_ACC_CON, /* Put bounds on constraints.*/
121                        i,           /* Index of constraint.*/
122                        bkc[i],      /* Bound key.*/
123                        blc[i],      /* Numerical value of lower bound.*/
124                        buc[i]);     /* Numerical value of upper bound.*/
125       /* Input column j of A */
126       if(r == MSK_RES_OK)
127         r = MSK_putavec(task,
128                         MSK_ACC_CON,       /* Input row of A.*/
129                         i,                 /* Row index.*/
130                         aptre[i]-aptrb[i], /* Number of non-zeros in row i.*/
131                         asub+aptrb[i],     /* Pointer to column indexes of row i.*/
132                         aval+aptrb[i]);    /* Pointer to Values of row i.*/
133     }
134
135     /* Maximize objective function. */
136     if (r == MSK_RES_OK)
137       r = MSK_putobjsense(task,
138                           MSK_OBJECTIVE_SENSE_MAXIMIZE);
139
140     if ( r==MSK_RES_OK )
141     {
142       MSKrescodee trmcode;
143
144       /* Run optimizer */
145       r = MSK_optimizetrm(task,&trmcode);
146
147       /* Print a summary containing information
148        about the solution for debugging purposes. */
149       MSK_solutionsummary (task,MSK_STREAM_LOG);
150
151       if ( r==MSK_RES_OK )
152       {
153         MSKsolstae solsta;
154         int j;
155         MSK_getsolutionstatus (task,
156                                MSK_SOL_BAS,
157                                NULL,
158                                &solsta);
159         switch(solsta)
160         {
161           case MSK_SOL_STA_OPTIMAL:
162           case MSK_SOL_STA_NEAR_OPTIMAL:
163             MSK_getsolutionslice(task,
164                                  MSK_SOL_BAS,    /* Request the basic solution. */
165                                  MSK_SOL_ITEM_XX,/* Which part of solution.     */
166                                  0,              /* Index of first variable.    */
167                                  NUMVAR,         /* Index of last variable+1.   */
168                                  xx);
169
170             printf("Optimal primal solution\n");
171             for(j=0; j<NUMVAR; ++j)
172               printf("x[%d]: %e\n",j,xx[j]);
173
```

```
174             break;
175           case MSK_SOL_STA_DUAL_INFEAS_CER:
176           case MSK_SOL_STA_PRIM_INFEAS_CER:
177           case MSK_SOL_STA_NEAR_DUAL_INFEAS_CER:
178           case MSK_SOL_STA_NEAR_PRIM_INFEAS_CER:
179             printf("Primal or dual infeasibility certificate found.\n");
180             break;
181
182           case MSK_SOL_STA_UNKNOWN:
183             printf("The status of the solution could not be determined.\n");
184             break;
185           default:
186             printf("Other solution status.");
187             break;
188         }
189       }
190       else
191       {
192         printf("Error while optimizing.\n");
193       }
194     }
195
196     if (r != MSK_RES_OK)
197     {
198       /* In case of an error print error code and description. */
199       char symname[MSK_MAX_STR_LEN];
200       char desc[MSK_MAX_STR_LEN];
201
202       printf("An error occurred while optimizing.\n");
203       MSK_getcodedesc (r,
204                        symname,
205                        desc);
206       printf("Error %s - '%s'\n",symname,desc);
207     }
208
209     MSK_deletetask(&task);
210
211     MSK_deleteenv(&env);
212   }
213
214   return r;
215 }
```

5.3 Quadratic optimization

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

$$\begin{array}{lccccll} \text{minimize} & & & \frac{1}{2}x^T Q^o x + c^T x + c^f & & & \\ \text{subject to} & l_k^c & \leq & \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j}x_j & \leq & u_k^c, & k = 0, \ldots, m-1, \\ & l^x & \leq & x & \leq & u^x, & j = 0, \ldots, n-1. \end{array} \tag{5.8}$$

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

$$x^T Q x = 0.5 x^T (Q + Q^T) x.$$

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

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

$$l_k^c \leq \frac{1}{2} x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j \tag{5.9}$$

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

$$\frac{1}{2} x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j \leq u_k^c. \tag{5.10}$$

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

5.3.1 Example: Quadratic objective

The following is an example if a quadratic, linearly constrained problem:

$$\begin{array}{lrcc} \text{minimize} & & x_1^2 + 0.1 x_2^2 + x_3^2 - x_1 x_3 - x_2 & \\ \text{subject to} & 1 & \leq & x_1 + x_2 + x_3 \\ & & & x \geq 0 \end{array} \tag{5.11}$$

This can be written equivalently as

$$\begin{array}{lccc} \text{minimize} & 1/2 x^T Q^o x + c^T x & & \\ \text{subject to} & Ax & \geq & b \\ & x & \geq & 0, \end{array} \tag{5.12}$$

where

$$Q^o = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 0.2 & 0 \\ -1 & 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \text{ and } b = 1. \tag{5.13}$$

Please note that MOSEK always assumes that there is a 1/2 in front of the $x^T Q x$ term in the objective. Therefore, the 1 in front of x_0^2 becomes 2 in Q, i.e. $Q_{0,0}^o = 2$.

5.3.1.1 Source code

```
1 /*
2    Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3
4    File:      qo1.c
```

```
5
6     Purpose: To demonstrate how to solve a quadratic optimization
7              problem using the MOSEK API.
8  */
9
10 #include <stdio.h>
11
12 #include "mosek.h" /* Include the MOSEK definition file. */
13
14 #define NUMCON 1   /* Number of constraints.             */
15 #define NUMVAR 3   /* Number of variables.               */
16 #define NUMANZ 3   /* Number of non-zeros in A.           */
17 #define NUMQNZ 4   /* Number of non-zeros in Q.           */
18
19 static void MSKAPI printstr(void *handle,
20                             char str[])
21 {
22   printf("%s",str);
23 } /* printstr */
24
25 int main(int argc,char *argv[])
26 {
27
28   double        c[]   = {0.0,-1.0,0.0};
29
30   MSKboundkeye  bkc[] = {MSK_BK_LO};
31   double        blc[] = {1.0};
32   double        buc[] = {+MSK_INFINITY};
33
34   MSKboundkeye  bkx[] = {MSK_BK_LO,
35                          MSK_BK_LO,
36                          MSK_BK_LO};
37   double        blx[] = {0.0,
38                          0.0,
39                          0.0};
40   double        bux[] = {+MSK_INFINITY,
41                          +MSK_INFINITY,
42                          +MSK_INFINITY};
43
44   MSKintt       aptrb[] = {0, 1, 2 };
45   MSKintt       aptre[] = {1, 2, 3};
46   MSKidxt       asub[] = {0,   0,   0};
47   double        aval[] = {1.0, 1.0, 1.0};
48
49   MSKidxt       qsubi[NUMQNZ];
50   MSKidxt       qsubj[NUMQNZ];
51   double        qval[NUMQNZ];
52
53   MSKidxt       i,j;
54   double        xx[NUMVAR];
55
56   MSKenv_t      env;
57   MSKtask_t     task;
58   MSKrescodee   r;
59
60   /* Create the mosek environment. */
61   r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
62
```

```
63  /* Check whether the return code is ok. */
64  if ( r==MSK_RES_OK )
65  {
66    /* Directs the log stream to the 'printstr' function. */
67    MSK_linkfunctoenvstream(env,
68                            MSK_STREAM_LOG,
69                            NULL,
70                            printstr);
71  }
72
73  /* Initialize the environment. */
74  r = MSK_initenv(env);
75  if ( r==MSK_RES_OK )
76  {
77
78  /* Create the optimization task. */
79    r = MSK_maketask(env,NUMCON,NUMVAR,&task);
80
81    if ( r==MSK_RES_OK )
82    {
83      r = MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
84
85      /* Give MOSEK an estimate of the size of the input data.
86       This is done to increase the speed of inputting data.
87       However, it is optional. */
88      if (r == MSK_RES_OK)
89        r = MSK_putmaxnumvar(task,NUMVAR);
90
91      if (r == MSK_RES_OK)
92        r = MSK_putmaxnumcon(task,NUMCON);
93
94      if (r == MSK_RES_OK)
95        r = MSK_putmaxnumanz(task,NUMANZ);
96
97      /* Append 'NUMCON' empty constraints.
98       The constraints will initially have no bounds. */
99      if ( r == MSK_RES_OK )
100       r = MSK_append(task,MSK_ACC_CON,NUMCON);
101
102     /* Append 'NUMVAR' variables.
103      The variables will initially be fixed at zero (x=0). */
104     if ( r == MSK_RES_OK )
105       r = MSK_append(task,MSK_ACC_VAR,NUMVAR);
106
107     /* Optionally add a constant term to the objective. */
108     if ( r ==MSK_RES_OK )
109       r = MSK_putcfix(task,0.0);
110     for(j=0; j<NUMVAR && r == MSK_RES_OK; ++j)
111     {
112       /* Set the linear term c_j in the objective.*/
113       if(r == MSK_RES_OK)
114         r = MSK_putcj(task,j,c[j]);
115
116       /* Set the bounds on variable j.
117        blx[j] <= x_j <= bux[j] */
118       if(r == MSK_RES_OK)
119         r = MSK_putbound(task,
120                          MSK_ACC_VAR, /* Put bounds on variables.*/
```

```
                          j,           /* Index of variable.*/
                          bkx[j],      /* Bound key.*/
                          blx[j],      /* Numerical value of lower bound.*/
                          bux[j]);     /* Numerical value of upper bound.*/

        /* Input column j of A */
        if(r == MSK_RES_OK)
          r = MSK_putavec(task,
                          MSK_ACC_VAR,       /* Input columns of A.*/
                          j,                 /* Variable (column) index.*/
                          aptre[j]-aptrb[j], /* Number of non-zeros in column j.*/
                          asub+aptrb[j],     /* Pointer to row indexes of column j.*/
                          aval+aptrb[j]);    /* Pointer to Values of column j.*/

      }

      /* Set the bounds on constraints.
         for i=1, ...,NUMCON : blc[i] <= constraint i <= buc[i] */
      for(i=0; i<NUMCON && r==MSK_RES_OK; ++i)
        r = MSK_putbound(task,
                         MSK_ACC_CON, /* Put bounds on constraints.*/
                         i,           /* Index of constraint.*/
                         bkc[i],      /* Bound key.*/
                         blc[i],      /* Numerical value of lower bound.*/
                         buc[i]);     /* Numerical value of upper bound.*/

      if ( r==MSK_RES_OK )
      {
        /*
         * The lower triangular part of the Q
         * matrix in the objective is specified.
         */

        qsubi[0] = 0;   qsubj[0] = 0;  qval[0] = 2.0;
        qsubi[1] = 1;   qsubj[1] = 1;  qval[1] = 0.2;
        qsubi[2] = 2;   qsubj[2] = 0;  qval[2] = -1.0;
        qsubi[3] = 2;   qsubj[3] = 2;  qval[3] = 2.0;

        /* Input the Q for the objective. */

        r = MSK_putqobj(task,NUMQNZ,qsubi,qsubj,qval);
      }

      if ( r==MSK_RES_OK )
      {
        MSKrescodee trmcode;

        /* Run optimizer */
        r = MSK_optimizetrm(task,&trmcode);

        /* Print a summary containing information
           about the solution for debugging purposes*/
        MSK_solutionsummary (task,MSK_STREAM_LOG);

        if ( r==MSK_RES_OK )
        {
          MSKsolstae solsta;
          int j;
```

```
179
180           MSK_getsolutionstatus (task,
181                                  MSK_SOL_ITR,
182                                  NULL,
183                                  &solsta);
184
185           switch(solsta)
186           {
187             case MSK_SOL_STA_OPTIMAL:
188             case MSK_SOL_STA_NEAR_OPTIMAL:
189               MSK_getsolutionslice(task,
190                                    MSK_SOL_ITR,    /* Request the interior solution. */
191                                    MSK_SOL_ITEM_XX,/* Which part of solution.     */
192                                    0,              /* Index of first variable.    */
193                                    NUMVAR,         /* Index of last variable+1.   */
194                                    xx);
195
196               printf("Optimal primal solution\n");
197               for(j=0; j<NUMVAR; ++j)
198                 printf("x[%d]: %e\n",j,xx[j]);
199
200               break;
201             case MSK_SOL_STA_DUAL_INFEAS_CER:
202             case MSK_SOL_STA_PRIM_INFEAS_CER:
203             case MSK_SOL_STA_NEAR_DUAL_INFEAS_CER:
204             case MSK_SOL_STA_NEAR_PRIM_INFEAS_CER:
205               printf("Primal or dual infeasibility certificate found.\n");
206               break;
207
208             case MSK_SOL_STA_UNKNOWN:
209               printf("The status of the solution could not be determined.\n");
210               break;
211             default:
212               printf("Other solution status.");
213               break;
214           }
215         }
216         else
217         {
218           printf("Error while optimizing.\n");
219         }
220       }
221
222       if (r != MSK_RES_OK)
223       {
224         /* In case of an error print error code and description. */
225         char symname[MSK_MAX_STR_LEN];
226         char desc[MSK_MAX_STR_LEN];
227
228         printf("An error occurred while optimizing.\n");
229         MSK_getcodedesc (r,
230                          symname,
231                          desc);
232         printf("Error %s - '%s'\n",symname,desc);
233       }
234     }
235   }
236   MSK_deletetask(&task);
```

```
237   MSK_deleteenv(&env);
238
239   return (r);
240 } /* main */
```

5.3.1.2 Example code comments

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

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

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

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

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

The lower triangular part of the matrix Q^o is specified using an unordered sparse triplet format (for details, see Section 5.8.3):

```
154 qsubi[0] = 0;   qsubj[0] = 0;   qval[0] = 2.0;
155 qsubi[1] = 1;   qsubj[1] = 1;   qval[1] = 0.2;
156 qsubi[2] = 2;   qsubj[2] = 0;   qval[2] = -1.0;
157 qsubi[3] = 2;   qsubj[3] = 2;   qval[3] = 2.0;
```

Please note that

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

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

```
161 r = MSK_putqobj(task,NUMQNZ,qsubi,qsubj,qval);
```

5.3.2 Example: Quadratic constraints

In this section describes how to solve a problem with quadratic constraints. Please note that quadratic constraints are subject to the convexity requirement (5.9).

Consider the problem:

$$\begin{array}{lcl} \text{minimize} & & x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ \text{subject to} \quad 1 & \leq & x_1 + x_2 + x_3 - x_1^2 - x_2^2 - 0.1x_3^2 + 0.2x_1x_3, \\ & & x \geq 0. \end{array} \tag{5.14}$$

This is equivalent to

$$\begin{array}{lll} \text{minimize} & 1/2x^T Q^o x + c^T x & \\ \text{subject to} & 1/2x^T Q^0 x + Ax & \geq \quad b, \end{array} \tag{5.15}$$

where

$$Q^o = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 0.2 & 0 \\ -1 & 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \quad b = 1. \tag{5.16}$$

$$Q^0 = \begin{bmatrix} -2 & 0 & 0.2 \\ 0 & -2 & 0 \\ 0.2 & 0 & -0.2 \end{bmatrix}. \tag{5.17}$$

5.3.2.1 Source code

```
/*
   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

   File:      qcqo1.c

   Purpose:   To demonstrate how to solve a quadratic
              optimization problem using the MOSEK API.

              minimize  x_1^2 + 0.1 x_2^2 +  x_3^2 - x_1 x_3 - x_2
              s.t 1 <=  x_1 + x_2 + x_3 - x_1^2 - x_2^2 - 0.1 x_3^2 + 0.2 x_1 x_3
              x >= 0

 */

#include <stdio.h>

#include "mosek.h" /* Include the MOSEK definition file. */

#define NUMCON 1   /* Number of constraints.             */
#define NUMVAR 3   /* Number of variables.               */
#define NUMANZ 3   /* Number of non-zeros in A.          */
#define NUMQNZ 4   /* Number of non-zeros in Q.          */

static void MSKAPI printstr(void *handle,
                            char str[])
{
  printf("%s",str);
} /* printstr */

int main(int argc,char *argv[])
{

  MSKrescodee  r;

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

  MSKboundkeye bkc[]  = {MSK_BK_LO};
  double       blc[]  = {1.0};
  double       buc[]  = {+MSK_INFINITY};
```

```
40
41    MSKboundkeye bkx[]  = {MSK_BK_LO,
42                           MSK_BK_LO,
43                           MSK_BK_LO};
44    double       blx[]  = {0.0,
45                           0.0,
46                           0.0};
47    double       bux[]  = {+MSK_INFINITY,
48                           +MSK_INFINITY,
49                           +MSK_INFINITY};
50
51    MSKlidxt     aptrb[] = {0, 1, 2 };
52    MSKlidxt     aptre[] = {1, 2, 3};
53    MSKidxt      asub[] = { 0,   0,   0};
54    double       aval[] = { 1.0, 1.0, 1.0};
55
56    MSKidxt      qsubi[NUMQNZ];
57    MSKidxt      qsubj[NUMQNZ];
58    double       qval[NUMQNZ];
59
60    MSKidxt      j,i;
61    double       xx[NUMVAR];
62    MSKenv_t     env;
63    MSKtask_t    task;
64
65    /* Create the mosek environment. */
66    r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
67
68    /* Check whether the return code is ok. */
69    if ( r==MSK_RES_OK )
70    {
71      /* Directs the log stream to the 'printstr' function. */
72      MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
73    }
74
75    /* Initialize the environment. */
76    r = MSK_initenv(env);
77    if ( r==MSK_RES_OK )
78    {
79    /* Create the optimization task. */
80      r = MSK_maketask(env,NUMCON,NUMVAR,&task);
81
82      if ( r==MSK_RES_OK )
83      {
84        r = MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
85
86
87      /* Give MOSEK an estimate of the size of the input data.
88       This is done to increase the speed of inputting data.
89       However, it is optional. */
90      if (r == MSK_RES_OK)
91        r = MSK_putmaxnumvar(task,NUMVAR);
92
93      if (r == MSK_RES_OK)
94        r = MSK_putmaxnumcon(task,NUMCON);
95
96      if (r == MSK_RES_OK)
97        r = MSK_putmaxnumanz(task,NUMANZ);
```

```
98
99      /* Append 'NUMCON' empty constraints.
100      The constraints will initially have no bounds. */
101     if ( r == MSK_RES_OK )
102       r = MSK_append(task,MSK_ACC_CON,NUMCON);
103
104     /* Append 'NUMVAR' variables.
105      The variables will initially be fixed at zero (x=0). */
106     if ( r == MSK_RES_OK )
107       r = MSK_append(task,MSK_ACC_VAR,NUMVAR);
108
109     /* Optionally add a constant term to the objective. */
110     if ( r ==MSK_RES_OK )
111       r = MSK_putcfix(task,0.0);
112     for(j=0; j<NUMVAR && r == MSK_RES_OK; ++j)
113     {
114       /* Set the linear term c_j in the objective.*/
115       if(r == MSK_RES_OK)
116         r = MSK_putcj(task,j,c[j]);
117
118       /* Set the bounds on variable j.
119        blx[j] <= x_j <= bux[j] */
120       if(r == MSK_RES_OK)
121         r = MSK_putbound(task,
122                          MSK_ACC_VAR, /* Put bounds on variables.*/
123                          j,           /* Index of variable.*/
124                          bkx[j],      /* Bound key.*/
125                          blx[j],      /* Numerical value of lower bound.*/
126                          bux[j]);     /* Numerical value of upper bound.*/
127
128       /* Input column j of A */
129       if(r == MSK_RES_OK)
130         r = MSK_putavec(task,
131                         MSK_ACC_VAR,       /* Input columns of A.*/
132                         j,                 /* Variable (column) index.*/
133                         aptre[j]-aptrb[j], /* Number of non-zeros in column j.*/
134                         asub+aptrb[j],     /* Pointer to row indexes of column j.*/
135                         aval+aptrb[j]);    /* Pointer to Values of column j.*/
136
137     }
138
139     /* Set the bounds on constraints.
140        for i=1, ...,NUMCON : blc[i] <= constraint i <= buc[i] */
141     for(i=0; i<NUMCON && r==MSK_RES_OK; ++i)
142       r = MSK_putbound(task,
143                        MSK_ACC_CON, /* Put bounds on constraints.*/
144                        i,           /* Index of constraint.*/
145                        bkc[i],      /* Bound key.*/
146                        blc[i],      /* Numerical value of lower bound.*/
147                        buc[i]);     /* Numerical value of upper bound.*/
148
149       if ( r==MSK_RES_OK )
150       {
151         /*
152          * The lower triangular part of the Q^o
153          * matrix in the objective is specified.
154          */
155
```

```
156           qsubi[0] = 0;   qsubj[0] = 0;  qval[0] = 2.0;
157           qsubi[1] = 1;   qsubj[1] = 1;  qval[1] = 0.2;
158           qsubi[2] = 2;   qsubj[2] = 0;  qval[2] = -1.0;
159           qsubi[3] = 2;   qsubj[3] = 2;  qval[3] = 2.0;
160
161           /* Input the Q^o for the objective. */
162
163           r = MSK_putqobj(task,NUMQNZ,qsubi,qsubj,qval);
164         }
165
166         if ( r==MSK_RES_OK )
167         {
168            /*
169            * The lower triangular part of the Q^0
170            * matrix in the first constraint is specified.
171            This corresponds to adding the term
172            - x_1^2 - x_2^2 - 0.1 x_3^2 + 0.2 x_1 x_3
173            */
174
175           qsubi[0] = 0;   qsubj[0] = 0;  qval[0] = -2.0;
176           qsubi[1] = 1;   qsubj[1] = 1;  qval[1] = -2.0;
177           qsubi[2] = 2;   qsubj[2] = 2;  qval[2] = -0.2;
178           qsubi[3] = 2;   qsubj[3] = 0;  qval[3] = 0.2;
179
180           /* Put Q^0 in constraint with index 0. */
181
182            r = MSK_putqconk(task,
183                             0,
184                             4,
185                             qsubi,
186                             qsubj,
187                             qval);
188         }
189
190         if ( r==MSK_RES_OK )
191           r = MSK_putobjsense(task, MSK_OBJECTIVE_SENSE_MINIMIZE);
192
193         if ( r==MSK_RES_OK )
194         {
195           MSKrescodee trmcode;
196
197           /* Run optimizer */
198           r = MSK_optimizetrm(task,&trmcode);
199
200           /* Print a summary containing information
201              about the solution for debugging purposes*/
202           MSK_solutionsummary (task,MSK_STREAM_LOG);
203
204           if ( r==MSK_RES_OK )
205           {
206             MSKsolstae solsta;
207             int j;
208
209             MSK_getsolutionstatus (task,
210                                    MSK_SOL_ITR,
211                                    NULL,
212                                    &solsta);
213
```

```
214           switch(solsta)
215           {
216             case MSK_SOL_STA_OPTIMAL:
217             case MSK_SOL_STA_NEAR_OPTIMAL:
218               MSK_getsolutionslice(task,
219                                    MSK_SOL_ITR,    /* Request the interior solution. */
220                                    MSK_SOL_ITEM_XX,/* Which part of solution.     */
221                                    0,              /* Index of first variable.    */
222                                    NUMVAR,         /* Index of last variable+1.   */
223                                    xx);
224
225               printf("Optimal primal solution\n");
226               for(j=0; j<NUMVAR; ++j)
227                 printf("x[%d]: %e\n",j,xx[j]);
228
229               break;
230             case MSK_SOL_STA_DUAL_INFEAS_CER:
231             case MSK_SOL_STA_PRIM_INFEAS_CER:
232             case MSK_SOL_STA_NEAR_DUAL_INFEAS_CER:
233             case MSK_SOL_STA_NEAR_PRIM_INFEAS_CER:
234               printf("Primal or dual infeasibility certificate found.\n");
235               break;
236
237             case MSK_SOL_STA_UNKNOWN:
238               printf("The status of the solution could not be determined.\n");
239               break;
240             default:
241               printf("Other solution status.");
242               break;
243           }
244         }
245         else
246         {
247           printf("Error while optimizing.\n");
248         }
249       }
250
251       if (r != MSK_RES_OK)
252       {
253         /* In case of an error print error code and description. */
254         char symname[MSK_MAX_STR_LEN];
255         char desc[MSK_MAX_STR_LEN];
256
257         printf("An error occurred while optimizing.\n");
258         MSK_getcodedesc (r,
259                          symname,
260                          desc);
261         printf("Error %s - '%s'\n",symname,desc);
262       }
263     }
264
265     MSK_deletetask(&task);
266   }
267   MSK_deleteenv(&env);
268
269   return ( r );
270 } /* main */
```

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

5.4 Conic optimization

Conic problems are a generalization of linear problems, allowing constraints of the type

$$x \in \mathcal{C}$$

where $\mathcal{C}$ is a convex cone.

MOSEK can solve conic optimization problems of the following form

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x \in \mathcal{C} & & \end{array} \tag{5.18}$$

where $\mathcal{C}$ is a cone. $\mathcal{C}$ can be a product of cones, i.e.

$$\mathcal{C} = \mathcal{C}_0 \times \cdots \times \mathcal{C}_{p-1}$$

in which case $x \in \mathcal{C}$ means $x^t \in \mathcal{C}_t \subseteq \mathbb{R}^{n_t}$. Please note that the set of real numbers $\mathbb{R}$ is itself a cone, so linear variables are still allowed.

MOSEK supports two specific cones apart from the real numbers:

- The quadratic cone:
$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n_t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}.$$
- The rotated quadratic cone:
$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n_t} : 2x_1 x_2 \geq \sum_{j=3}^{n^t} x_j^2,\ x_1, x_2 \geq 0 \right\}.$$

When creating a conic problem in MOSEK, each cone is defined by a *cone type* (quadratic or rotated quadratic cone) and a list of variable indexes. To summarize:

- In MOSEK all variables belong to the set $\mathbb{R}$ of reals, unless they are explicitly declared as belonging to a cone.
- Each variable may belong to one cone *at most*.

5.4.1 Example: cqo1

The problem

$$\begin{array}{lrcl} \text{minimize} & x_4 + x_5 & & \\ \text{subject to} & x_0 + x_1 + x_2 + x_3 & = & 1, \\ & x_0, x_1, x_2, x_3 & \geq & 0, \\ & x_4 \geq \sqrt{x_0^2 + x_2^2}, & & \\ & x_5 \geq \sqrt{x_1^2 + x_3^2} & & \end{array} \tag{5.19}$$

is an example of a conic quadratic optimization problem. The problem includes a set of linear constraints and two quadratic cones.

5.4.1.1 Source code

```
/*
   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

   File:      cqo1.c

   Purpose:   To demonstrate how to solve a small conic quadratic
              optimization problem using the MOSEK API.
 */

#include <stdio.h>

#include "mosek.h" /* Include the MOSEK definition file. */

#define NUMCON 1   /* Number of constraints.             */
#define NUMVAR 6   /* Number of variables.               */
#define NUMANZ 4   /* Number of non-zeros in A.          */

static void MSKAPI printstr(void *handle,
                            char str[])
{
  printf("%s",str);
} /* printstr */

int main(int argc,char *argv[])
{
  MSKrescodee  r;

  MSKboundkeye bkc[] = { MSK_BK_FX };
  double       blc[] = { 1.0 };
  double       buc[] = { 1.0 };

  MSKboundkeye bkx[] = {MSK_BK_LO,
                        MSK_BK_LO,
                        MSK_BK_LO,
                        MSK_BK_LO,
                        MSK_BK_FR,
                        MSK_BK_FR};
  double       blx[] = {0.0,
                        0.0,
                        0.0,
```

```
                          0.0,
                          -MSK_INFINITY,
                          -MSK_INFINITY};
  double       bux[] = {+MSK_INFINITY,
                          +MSK_INFINITY,
                          +MSK_INFINITY,
                          +MSK_INFINITY,
                          +MSK_INFINITY,
                          +MSK_INFINITY};

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

  MSKintt     aptrb[] = {0, 1, 2, 3, 5, 5};
  MSKintt     aptre[] = {1, 2, 3, 4, 5, 5};
  double      aval[] = {1.0, 1.0, 1.0, 1.0};
  MSKidxt     asub[] = {0, 0, 0, 0};

  MSKidxt     i,j,csub[3];
  double      xx[NUMVAR];
  MSKenv_t    env;
  MSKtask_t   task;

  /* Create the mosek environment. */
  r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
  /* Check if return code is ok. */
  if ( r==MSK_RES_OK )
  {
    /* Directs the log stream to the
       'printstr' function. */
    MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
  }

  /* Initialize the environment. */
  if ( r==MSK_RES_OK )
    r = MSK_initenv(env);

  if ( r==MSK_RES_OK )
  {
    /* Create the optimization task. */
    r = MSK_maketask(env,NUMCON,NUMVAR,&task);

    if ( r==MSK_RES_OK )
    {
      MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);

      /* Give MOSEK an estimate of the size of the input data.
     This is done to increase the speed of inputting data.
     However, it is optional. */
      if (r == MSK_RES_OK)
        r = MSK_putmaxnumvar(task,NUMVAR);

      if (r == MSK_RES_OK)
        r = MSK_putmaxnumcon(task,NUMCON);
```

```
99
100       if (r == MSK_RES_OK)
101         r = MSK_putmaxnumanz(task,NUMANZ);
102
103       /* Append 'NUMCON' empty constraints.
104      The constraints will initially have no bounds. */
105       if ( r == MSK_RES_OK )
106         r = MSK_append(task,MSK_ACC_CON,NUMCON);
107
108       /* Append 'NUMVAR' variables.
109      The variables will initially be fixed at zero (x=0). */
110       if ( r == MSK_RES_OK )
111         r = MSK_append(task,MSK_ACC_VAR,NUMVAR);
112
113       /* Optionally add a constant term to the objective. */
114       if ( r ==MSK_RES_OK )
115         r = MSK_putcfix(task,0.0);
116       for(j=0; j<NUMVAR && r == MSK_RES_OK; ++j)
117       {
118         /* Set the linear term c_j in the objective.*/
119         if(r == MSK_RES_OK)
120           r = MSK_putcj(task,j,c[j]);
121
122         /* Set the bounds on variable j.
123        blx[j] <= x_j <= bux[j] */
124         if(r == MSK_RES_OK)
125           r = MSK_putbound(task,
126                            MSK_ACC_VAR, /* Put bounds on variables.*/
127                            j,           /* Index of variable.*/
128                            bkx[j],      /* Bound key.*/
129                            blx[j],      /* Numerical value of lower bound.*/
130                            bux[j]);     /* Numerical value of upper bound.*/
131
132         /* Input column j of A */
133         if(r == MSK_RES_OK)
134           r = MSK_putavec(task,
135                           MSK_ACC_VAR,       /* Input columns of A.*/
136                           j,                 /* Variable (column) index.*/
137                           aptre[j]-aptrb[j], /* Number of non-zeros in column j.*/
138                           asub+aptrb[j],     /* Pointer to row indexes of column j.*/
139                           aval+aptrb[j]);    /* Pointer to Values of column j.*/
140
141       }
142
143       /* Set the bounds on constraints.
144        for i=1, ...,NUMCON : blc[i] <= constraint i <= buc[i] */
145       for(i=0; i<NUMCON && r==MSK_RES_OK; ++i)
146         r = MSK_putbound(task,
147                          MSK_ACC_CON, /* Put bounds on constraints.*/
148                          i,           /* Index of constraint.*/
149                          bkc[i],      /* Bound key.*/
150                          blc[i],      /* Numerical value of lower bound.*/
151                          buc[i]);     /* Numerical value of upper bound.*/
152
153       if ( r==MSK_RES_OK )
154       {
155         /* Append the first cone. */
156         csub[0] = 4;
```

```
157         csub[1] = 0;
158         csub[2] = 2;
159         r = MSK_appendcone(task,
160                            MSK_CT_QUAD,
161                            0.0, /* For future use only, can be set to 0.0 */
162                            3,
163                            csub);
164       }
165
166       if ( r==MSK_RES_OK )
167       {
168         /* Append the second cone. */
169         csub[0] = 5;
170         csub[1] = 1;
171         csub[2] = 3;
172
173         r       =  MSK_appendcone(task,
174                                   MSK_CT_QUAD,
175                                   0.0,
176                                   3,
177                                   csub);
178       }
179
180
181       if ( r==MSK_RES_OK )
182       {
183         MSKrescodee trmcode;
184
185         /* Run optimizer */
186         r = MSK_optimizetrm(task,&trmcode);
187
188         /* Print a summary containing information
189            about the solution for debugging purposes*/
190         MSK_solutionsummary (task,MSK_STREAM_LOG);
191
192         if ( r==MSK_RES_OK )
193         {
194           MSKsolstae solsta;
195           MSKidxt    j;
196
197           MSK_getsolutionstatus (task,
198                                  MSK_SOL_ITR,
199                                  NULL,
200                                  &solsta);
201
202           switch(solsta)
203           {
204             case MSK_SOL_STA_OPTIMAL:
205             case MSK_SOL_STA_NEAR_OPTIMAL:
206               MSK_getsolutionslice(task,
207                                    MSK_SOL_ITR,    /* Request the interior solution. */
208                                    MSK_SOL_ITEM_XX,/* Which part of solution.     */
209                                    0,              /* Index of first variable.    */
210                                    NUMVAR,         /* Index of last variable+1.   */
211                                    xx);
212
213               printf("Optimal primal solution\n");
214               for(j=0; j<NUMVAR; ++j)
```

```
215                 printf("x[%d]: %e\n",j,xx[j]);
216
217               break;
218             case MSK_SOL_STA_DUAL_INFEAS_CER:
219             case MSK_SOL_STA_PRIM_INFEAS_CER:
220             case MSK_SOL_STA_NEAR_DUAL_INFEAS_CER:
221             case MSK_SOL_STA_NEAR_PRIM_INFEAS_CER:
222               printf("Primal or dual infeasibility certificate found.\n");
223               break;
224
225             case MSK_SOL_STA_UNKNOWN:
226               printf("The status of the solution could not be determined.\n");
227               break;
228             default:
229               printf("Other solution status.");
230               break;
231           }
232         }
233         else
234         {
235           printf("Error while optimizing.\n");
236         }
237       }
238
239       if (r != MSK_RES_OK)
240       {
241         /* In case of an error print error code and description. */
242         char symname[MSK_MAX_STR_LEN];
243         char desc[MSK_MAX_STR_LEN];
244
245         printf("An error occurred while optimizing.\n");
246         MSK_getcodedesc (r,
247                          symname,
248                          desc);
249         printf("Error %s - '%s'\n",symname,desc);
250       }
251     }
252     /* Delete the task and the associated data. */
253     MSK_deletetask(&task);
254   }
255
256   /* Delete the environment and the associated data. */
257   MSK_deleteenv(&env);
258
259   return ( r );
260 } /* main */
```

5.4.1.2 Source code comments

The only new function introduced in the example is MSK_appendcone, which is called here:

```
159 r = MSK_appendcone(task,
160                     MSK_CT_QUAD,
161                     0.0, /* For future use only, can be set to 0.0 */
162                     3,
163                     csub);
```

Here MSK_CT_QUAD defines the cone type, in this case it is a *quadratic cone*. The cone parameter `0.0` is currently not used by MOSEK — simply passing 0.0 will work.

The next argument denotes the number of variables in the cone, in this case 3, and the last argument is a list of indexes of the variables in the cone.

5.5 Integer optimization

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

5.5.1 Example: milo1

In this section the example

$$\begin{array}{lllll} \text{maximize} & x_0 + 0.64x_1 & & & \\ \text{subject to} & 50x_0 + 31x_1 & \leq & 250, & \\ & 3x_0 - 2x_1 & \geq & -4, & \\ & x_0, x_1 \geq 0 & & \text{and integer} & \end{array} \tag{5.20}$$

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

5.5.1.1 Source code

The example (5.20) is almost identical to a linear optimization problem except for some variables being integer constrained. Therefore, only the specification of the integer constraints requires something new compared to the linear optimization problem discussed previously. In MOSEK these constraints are specified using the function MSK_putvartype as shown in the code:

```
129 for(j=0; j<NUMVAR && r == MSK_RES_OK; ++j)
130   r = MSK_putvartype(task,j,MSK_VAR_TYPE_INT);
```

The complete source for the example is listed below.

```
1  /*
2     Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3
4     File:      milo1.c
5
6     Purpose:   To demonstrate how to solve a small mixed
7                integer linear optimization problem using
8                the MOSEK API.
9  */
10
11 #include <stdio.h>
12
13 #include "mosek.h" /* Include the MOSEK definition file. */
14
15 #define NUMCON 2   /* Number of constraints.             */
```

```
16 #define NUMVAR 2   /* Number of variables.                */
17 #define NUMANZ 4   /* Number of non-zeros in A.           */
18
19 static void MSKAPI printstr(void *handle,
20                              char str[])
21 {
22   printf("%s",str);
23 } /* printstr */
24
25 int main(int argc,char *argv[])
26 {
27   MSKrescodee  r;
28   double       c[]   = {  1.0, 0.64 };
29   MSKboundkeye bkc[] = { MSK_BK_UP,    MSK_BK_LO };
30   double       blc[] = { -MSK_INFINITY,-4.0 };
31   double       buc[] = { 250.0,        MSK_INFINITY };
32
33   MSKboundkeye bkx[] = { MSK_BK_LO,    MSK_BK_LO };
34   double       blx[] = { 0.0,          0.0 };
35   double       bux[] = { MSK_INFINITY, MSK_INFINITY };
36
37
38   MSKintt      aptrb[] = { 0, 2 };
39   MSKintt      aptre[] = { 2, 4 };
40   MSKidxt      asub[] = { 0,    1,   0,    1 };
41   double       aval[] = { 50.0, 3.0, 31.0, -2.0 };
42   MSKidxt      i,j;
43
44   double       xx[NUMVAR];
45   MSKenv_t     env;
46   MSKtask_t    task;
47
48   /* Create the mosek environment. */
49   r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
50
51   /* Initialize the environment. */
52   if ( r==MSK_RES_OK )
53     r = MSK_initenv(env);
54
55   /* Check if return code is ok. */
56   if ( r==MSK_RES_OK )
57   {
58     /* Directs the log stream to the 'printstr' function. */
59     MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
60
61     /* Create the optimization task. */
62     r = MSK_maketask(env,NUMCON,NUMVAR,&task);
63
64     if ( r==MSK_RES_OK )
65       r = MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
66         /* Give MOSEK an estimate of the size of the input data.
67      This is done to increase the speed of inputting data.
68      However, it is optional. */
69     if (r == MSK_RES_OK)
70       r = MSK_putmaxnumvar(task,NUMVAR);
71
72     if (r == MSK_RES_OK)
73       r = MSK_putmaxnumcon(task,NUMCON);
```

```
74
75      if (r == MSK_RES_OK)
76        r = MSK_putmaxnumanz(task,NUMANZ);
77
78      /* Append 'NUMCON' empty constraints.
79       The constraints will initially have no bounds. */
80      if ( r == MSK_RES_OK )
81        r = MSK_append(task,MSK_ACC_CON,NUMCON);
82
83      /* Append 'NUMVAR' variables.
84       The variables will initially be fixed at zero (x=0). */
85      if ( r == MSK_RES_OK )
86        r = MSK_append(task,MSK_ACC_VAR,NUMVAR);
87
88      /* Optionally add a constant term to the objective. */
89      if ( r ==MSK_RES_OK )
90        r = MSK_putcfix(task,0.0);
91      for(j=0; j<NUMVAR && r == MSK_RES_OK; ++j)
92      {
93        /* Set the linear term c_j in the objective.*/
94        if(r == MSK_RES_OK)
95          r = MSK_putcj(task,j,c[j]);
96
97        /* Set the bounds on variable j.
98         blx[j] <= x_j <= bux[j] */
99        if(r == MSK_RES_OK)
100         r = MSK_putbound(task,
101                          MSK_ACC_VAR, /* Put bounds on variables.*/
102                          j,           /* Index of variable.*/
103                          bkx[j],      /* Bound key.*/
104                          blx[j],      /* Numerical value of lower bound.*/
105                          bux[j]);     /* Numerical value of upper bound.*/
106
107       /* Input column j of A */
108       if(r == MSK_RES_OK)
109         r = MSK_putavec(task,
110                         MSK_ACC_VAR,       /* Input columns of A.*/
111                         j,                 /* Variable (column) index.*/
112                         aptre[j]-aptrb[j], /* Number of non-zeros in column j.*/
113                         asub+aptrb[j],     /* Pointer to row indexes of column j.*/
114                         aval+aptrb[j]);    /* Pointer to Values of column j.*/
115
116     }
117
118     /* Set the bounds on constraints.
119        for i=1, ...,NUMCON : blc[i] <= constraint i <= buc[i] */
120     for(i=0; i<NUMCON && r==MSK_RES_OK; ++i)
121       r = MSK_putbound(task,
122                        MSK_ACC_CON, /* Put bounds on constraints.*/
123                        i,           /* Index of constraint.*/
124                        bkc[i],      /* Bound key.*/
125                        blc[i],      /* Numerical value of lower bound.*/
126                        buc[i]);     /* Numerical value of upper bound.*/
127
128     /* Specify integer variables. */
129     for(j=0; j<NUMVAR && r == MSK_RES_OK; ++j)
130       r = MSK_putvartype(task,j,MSK_VAR_TYPE_INT);
131
```

```
132     if ( r==MSK_RES_OK )
133       r =  MSK_putobjsense(task,
134                            MSK_OBJECTIVE_SENSE_MAXIMIZE);
135
136     if ( r==MSK_RES_OK )
137       {
138         MSKrescodee trmcode;
139
140         /* Run optimizer */
141         r = MSK_optimizetrm(task,&trmcode);
142
143         /* Print a summary containing information
144            about the solution for debugging purposes*/
145         MSK_solutionsummary (task,MSK_STREAM_LOG);
146
147         if ( r==MSK_RES_OK )
148         {
149           MSKsolstae solsta;
150           int j;
151
152           MSK_getsolutionstatus (task,
153                                  MSK_SOL_ITG,
154                                  NULL,
155                                  &solsta);
156
157           switch(solsta)
158           {
159             case MSK_SOL_STA_INTEGER_OPTIMAL:
160             case MSK_SOL_STA_NEAR_INTEGER_OPTIMAL :
161               MSK_getsolutionslice(task,
162                                    MSK_SOL_ITG,    /* Request the integer */
163                                    MSK_SOL_ITEM_XX,/* Which part of solution.     */
164                                    0,              /* Index of first variable.    */
165                                    NUMVAR,         /* Index of last variable+1.   */
166                                    xx);
167
168               printf("Optimal solution.\n");
169               for(j=0; j<NUMVAR; ++j)
170                 printf("x[%d]: %e\n",j,xx[j]);
171
172               break;
173             case MSK_SOL_STA_PRIM_FEAS:
174               /* A feasible but not necessarily optimal solution was located. */
175               MSK_getsolutionslice(task,
176                                    MSK_SOL_ITG,    /* Request the integer.*/
177                                    MSK_SOL_ITEM_XX,/* Which part of solution.     */
178                                    0,              /* Index of first variable.    */
179                                    NUMVAR,         /* Index of last variable+1.   */
180                                    xx);
181               printf("Feasible solution.\n");
182               for(j=0; j<NUMVAR; ++j)
183                 printf("x[%d]: %e\n",j,xx[j]);
184               break;
185             case MSK_SOL_STA_UNKNOWN:
186               printf("The status of the solution could not be determined.\n");
187               break;
188             default:
189               printf("Other solution status.");
```

```
190                 break;
191             }
192           }
193           else
194           {
195             printf("Error while optimizing.\n");
196           }
197         }
198
199         if (r != MSK_RES_OK)
200         {
201           /* In case of an error print error code and description. */
202           char symname[MSK_MAX_STR_LEN];
203           char desc[MSK_MAX_STR_LEN];
204
205           printf("An error occurred while optimizing.\n");
206           MSK_getcodedesc (r,
207                            symname,
208                            desc);
209           printf("Error %s - '%s'\n",symname,desc);
210         }
211     }
212
213     MSK_deletetask(&task);
214     MSK_deleteenv(&env);
215
216     printf("Return code: %d.\n",r);
217
218     return ( r );
219   } /* main */
```

5.5.1.2 Code comments

Please note that when MSK_getsolutionslice is called, the integer solution is requested by using MSK_SOL_ITG. No dual solution is defined for integer optimization problems.

5.5.2 Specifying an initial solution

Integer optimization problems are generally hard to solve, but the solution time can often be reduced by providing an initial solution for the solver. Solution values can be set using MSK_putsolution (for inputting a whole solution) or MSK_putsolutioni (for inputting solution values related to a single variable or constraint).

It is not necessary to specify the whole solution. By setting the MSK_IPAR_MIO_CONSTRUCT_SOL parameter to MSK_ON and inputting values for the integer variables only, will force MOSEK to compute the remaining continuous variable values.

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

5.5.3 Example: Specifying an integer solution

Consider the problem

$$\begin{array}{ll} \text{maximize} & 7x_0 + 10x_1 + x_2 + 5x_3 \\ \text{subject to} & x_0 + x_1 + x_2 + x_3 \leq 2.5 \\ & x_0, x_1, x_2 \text{ integer}, \quad x_0, x_1, x_2, x_3 \geq 0 \end{array} \tag{5.21}$$

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

```
/*
   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

   File:      mioinitsol.c

   Purpose:   To demonstrate how to solve a MIP with a start guess.

 */

#include "mosek.h"
#include <stdio.h>

static void MSKAPI printstr(void *handle,
                            char str[])
{
  printf("%s",str);
} /* printstr */

#define NUMVAR     4
#define NUMCON     1
#define NUMINTVAR  3


int main(int argc,char *argv[])
{
  char         buffer[512];

  MSKrescodee  r;

  MSKenv_t     env;
  MSKtask_t    task;

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

  MSKboundkeye bkc[] = {MSK_BK_UP};
  double       blc[] = {-MSK_INFINITY};
  double       buc[] = {2.5};

  MSKboundkeye bkx[] = {MSK_BK_LO, MSK_BK_LO, MSK_BK_LO,MSK_BK_LO};
  double       blx[] = {0.0,       0.0,       0.0,      0.0      };
  double       bux[] = {MSK_INFINITY,MSK_INFINITY,MSK_INFINITY,MSK_INFINITY};

  MSKlidxt     ptrb[] = {0,1,2,3};
  MSKlidxt     ptre[] = {1,2,3,4};
  double       aval[] = {1.0, 1.0, 1.0, 1.0};
  MSKidxt      asub[] = {0,   0,   0,   0  };
```

```
47  MSKidxt      intsub[] = {0,1,2};
48  MSKidxt      j;
49
50  r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
51
52  if ( r==MSK_RES_OK )
53  {
54    MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
55  }
56
57  r = MSK_initenv(env);
58
59  if ( r==MSK_RES_OK )
60    r = MSK_maketask(env,0,0,&task);
61
62  if ( r==MSK_RES_OK )
63    MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
64
65  if (r == MSK_RES_OK)
66    r = MSK_inputdata(task,
67                      NUMCON,NUMVAR,
68                      NUMCON,NUMVAR,
69                      c,
70                      0.0,
71                      ptrb,
72                      ptre,
73                      asub,
74                      aval,
75                      bkc,
76                      blc,
77                      buc,
78                      bkx,
79                      blx,
80                      bux);
81
82  if (r == MSK_RES_OK)
83    MSK_putobjsense(task,MSK_OBJECTIVE_SENSE_MAXIMIZE);
84
85  for(j=0; j<NUMINTVAR && r == MSK_RES_OK; ++j)
86    r = MSK_putvartype(task,intsub[j],MSK_VAR_TYPE_INT);
87
88
89  /* Construct an initial feasible solution from the
90     values of the integer variables specified */
91
92  if (r == MSK_RES_OK)
93    r = MSK_putintparam(task,MSK_IPAR_MIO_CONSTRUCT_SOL,MSK_ON);
94
95  /* Set status of all variables to unknown */
96  if (r == MSK_RES_OK)
97    r = MSK_makesolutionstatusunknown(task, MSK_SOL_ITG);
98
99  /* Assign values 1,1,0 to integer variables */
100
101 if (r == MSK_RES_OK)
102   r = MSK_putsolutioni (
103                         task,
104                         MSK_ACC_VAR,
```

```
105                           0,
106                           MSK_SOL_ITG,
107                           MSK_SK_SUPBAS,
108                           0.0,
109                           0.0,
110                           0.0,
111                           0.0);
112
113     if (r == MSK_RES_OK)
114       r = MSK_putsolutioni (
115                           task,
116                           MSK_ACC_VAR,
117                           1,
118                           MSK_SOL_ITG,
119                           MSK_SK_SUPBAS,
120                           2.0,
121                           0.0,
122                           0.0,
123                           0.0);
124
125     if (r == MSK_RES_OK)
126       r = MSK_putsolutioni (
127                           task,
128                           MSK_ACC_VAR,
129                           2,
130                           MSK_SOL_ITG,
131                           MSK_SK_SUPBAS,
132                           0.0,
133                           0.0,
134                           0.0,
135                           0.0);
136
137     /* solve */
138
139     if (r == MSK_RES_OK)
140       r = MSK_optimize(task);
141
142     /* Did mosek construct a feasible initial solution ? */
143
144     {
145       int isok;
146
147       if (r == MSK_RES_OK)
148         r = MSK_getintinf(task,MSK_IINF_MIO_CONSTRUCT_SOLUTION,&isok);
149
150       if ( isok>0 && r == MSK_RES_OK)
151         printf("MOSEK constructed a feasible initial soulution.\n");
152     }
153     /* Delete the task. */
154
155     MSK_deletetask(&task);
156
157     MSK_deleteenv(&env);
158
159     printf("Return code: %d\n",r);
160     if ( r!=MSK_RES_OK )
161     {
162       MSK_getcodedisc(r,buffer,NULL);
```

```
163     printf("Description: %s\n",buffer);
164   }
165
166   return (r);
167 }
```

5.6 Problem modification and reoptimization

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

5.6.1 A production planning problem

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

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

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

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

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

$$\begin{array}{lrcrcrcl} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & & \\ \text{subject to} & 2x_0 & + & 4x_1 & + & 3x_2 & \leq & 100000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & \leq & 50000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & \leq & 60000, \end{array} \tag{5.22}$$

and

$$x_0, x_1, x_2 \geq 0. \tag{5.23}$$

The following code loads this problem into the optimization task.

```
27 MSKrescodee  r;
28 MSKidxt      i,j;
29 double       c[]    = {1.5, 2.5, 3.0};
30 MSKlidxt     ptrb[] = {0, 3, 6};
31 MSKlidxt     ptre[] = {3, 6, 9};
32
```

```
33 MSKidxt      asub[] = { 0, 1, 2,
34                         0, 1, 2,
35                         0, 1, 2};
36
37 double       aval[] = { 2.0, 3.0, 2.0,
38                         4.0, 2.0, 3.0,
39                         3.0, 3.0, 2.0};
40
41 MSKboundkeye bkc[]  = {MSK_BK_UP, MSK_BK_UP, MSK_BK_UP    };
42 double       blc[]  = {-MSK_INFINITY, -MSK_INFINITY, -MSK_INFINITY};
43 double       buc[]  = {100000, 50000, 60000};
44
45 MSKboundkeye bkx[]  = {MSK_BK_LO,     MSK_BK_LO,    MSK_BK_LO};
46 double       blx[]  = {0.0,           0.0,          0.0,};
47 double       bux[]  = {+MSK_INFINITY, +MSK_INFINITY,+MSK_INFINITY};
48
49 double       xx[NUMVAR];
50
51 MSKenv_t     env;
52 MSKtask_t    task;
53 MSKintt      numvar,numcon;
54
55 /* Create the mosek environment. */
56 r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
57
58 /* Check if return code is ok. */
59 if ( r==MSK_RES_OK )
60 {
61   /* Directs the env log stream to the
62      'printstr' function. */
63   MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
64 }
65
66 /* Initialize the environment. */
67 r = MSK_initenv(env);
68
69 if ( r==MSK_RES_OK )
70 {
71   /* Create the optimization task. */
72   r = MSK_maketask(env,NUMCON,NUMVAR,&task);
73
74   /* Directs the log task stream to the
75      'printstr' function. */
76
77   MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
78
79   /* Give MOSEK an estimate of the size of the input data. This is
80      done to increase the efficiency of inputting data, however it is
81      optional.*/
82
83   if (r == MSK_RES_OK)
84     r = MSK_putmaxnumvar(task,NUMVAR);
85
86   if (r == MSK_RES_OK)
87     r = MSK_putmaxnumcon(task,NUMCON);
88
89   if (r == MSK_RES_OK)
90     r = MSK_putmaxnumanz(task,NUMANZ);
```

```
91
92    /* Append the constraints. */
93    if (r == MSK_RES_OK)
94      r = MSK_append(task,MSK_ACC_CON,NUMCON);
95
96    /* Append the variables. */
97    if (r == MSK_RES_OK)
98      r = MSK_append(task,MSK_ACC_VAR,NUMVAR);
99
100   /* Put C. */
101   if (r == MSK_RES_OK)
102     r = MSK_putcfix(task, 0.0);
103
104   if (r == MSK_RES_OK)
105     for(j=0; j<NUMVAR; ++j)
106       r = MSK_putcj(task,j,c[j]);
107
108   /* Put constraint bounds. */
109   if (r == MSK_RES_OK)
110     for(i=0; i<NUMCON; ++i)
111       r = MSK_putbound(task,MSK_ACC_CON,i,bkc[i],blc[i],buc[i]);
112
113   /* Put variable bounds. */
114   if (r == MSK_RES_OK)
115     for(j=0; j<NUMVAR; ++j)
116       r = MSK_putbound(task,MSK_ACC_VAR,j,bkx[j],blx[j],bux[j]);
117
118   /* Put A. */
119   if (r == MSK_RES_OK)
120     if ( NUMCON>0 )
121       for(j=0; j<NUMVAR; ++j)
122         r = MSK_putavec(task,
123                         MSK_ACC_VAR,
124                         j,
125                         ptre[j]-ptrb[j],
126                         asub+ptrb[j],
127                         aval+ptrb[j]);
128
129   if (r == MSK_RES_OK)
130     r = MSK_putobjsense(task,
131                         MSK_OBJECTIVE_SENSE_MAXIMIZE);
132
133   if (r == MSK_RES_OK)
134     r = MSK_optimizetrm(task,NULL);
135
136   if (r == MSK_RES_OK)
137     MSK_getsolutionslice(task,
138                          MSK_SOL_BAS,       /* Basic solution.       */
139                          MSK_SOL_ITEM_XX,   /* Which part of solution.  */
140                          0,                 /* Index of first variable. */
141                          NUMVAR,            /* Index of last variable+1 */
142                          xx);
```

5.6.2 Changing the A matrix

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

```
145 if (r == MSK_RES_OK)
146   r = MSK_putaij(task, 0, 0, 3.0);
```

The problem now has the form:

$$\begin{array}{lrcrcrcl} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & & \\ \text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & \leq & 100000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & \leq & 50000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & \leq & 60000, \end{array} \tag{5.24}$$

and

$$x_0, x_1, x_2 \geq 0. \tag{5.25}$$

After changing the A matrix we can find the new optimal solution by calling

MSK_optimize

again

5.6.3 Appending variables

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

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

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

```
149 /* Append a new variable x_3 to the problem */
150 if (r == MSK_RES_OK)
151   r = MSK_append(task,MSK_ACC_VAR,1);
152
153 /* Get index of new variable, this should be 3 */
154 if (r == MSK_RES_OK)
155   r = MSK_getnumvar(task,&numvar);
156
157 /* Set bounds on new variable */
158 if (r == MSK_RES_OK)
159   r = MSK_putbound(task,
160                    MSK_ACC_VAR,
161                    numvar-1,
162                    MSK_BK_LO,
163                    0,
164                    +MSK_INFINITY);
165
166 /* Change objective */
```

```
if (r == MSK_RES_OK)
  r = MSK_putcj(task,numvar-1,1.0);

/* Put new values in the A matrix */
if (r == MSK_RES_OK)
{
  MSKidxt acolsub[] = {0,   2};
  double  acolval[] =  {4.0, 1.0};

   r = MSK_putavec(task,
                   MSK_ACC_VAR,
                   numvar-1, /* column index */
                   2, /* num nz in column*/
                   acolsub,
                   acolval);
}
```

After this operation the problem looks this way:

$$\begin{array}{lrcrcrcrcr} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & + & 1.0x_3 & & \\ \text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & + & 4x_3 & \leq & 100000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & & & \leq & 50000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & + & 1x_3 & \leq & 60000, \end{array} \tag{5.26}$$

and

$$x_0, x_1, x_2, x_3 \geq 0. \tag{5.27}$$

5.6.4 Reoptimization

When

MSK_optimize

is called MOSEK will store the optimal solution internally. After a task has been modified and

MSK_optimize

is called again the solution will automatically be used to reduce solution time of the new problem, if possible.

In this case an optimal solution to problem (5.24) was found and then added a column was added to get (5.26). The simplex optimizer is well suited for exploiting an existing primal or dual feasible solution. Hence, the subsequent code instructs MOSEK to choose the simplex optimizer freely when optimizing.

```
/* Change optimizer to free simplex and reoptimize */
if (r == MSK_RES_OK)
  r = MSK_putintparam(task,MSK_IPAR_OPTIMIZER,MSK_OPTIMIZER_FREE_SIMPLEX);

if (r == MSK_RES_OK)
  r = MSK_optimizetrm(task,NULL);
```

5.6.5 Appending constraints

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

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

This corresponds to adding the constraint

$$x_0 + 2x_1 + x_2 + x_3 \leq 30000 \tag{5.28}$$

to the problem which is done in the following code:

```
190 /* Append a new constraint */
191 if (r == MSK_RES_OK)
192   r = MSK_append(task,MSK_ACC_CON,1);
193
194 /* Get index of new constraint, this should be 4 */
195 if (r == MSK_RES_OK)
196   r = MSK_getnumcon(task,&numcon);
197
198 /* Set bounds on new constraint */
199 if (r == MSK_RES_OK)
200   r = MSK_putbound(task,
201                    MSK_ACC_CON,
202                    numcon-1,
203                    MSK_BK_UP,
204                    -MSK_INFINITY,
205                    30000);
206
207 /* Put new values in the A matrix */
208 if (r == MSK_RES_OK)
209 {
210   MSKidxt arowsub[] = {0,   1,   2,   3  };
211   double arowval[] =  {1.0, 2.0, 1.0, 1.0};
212
213   r = MSK_putavec(task,
214                   MSK_ACC_CON,
215                   numcon-1, /* row index */
216                   4, /* num nz in row*/
217                   arowsub,
218                   arowval);
219 }
```

5.7 Efficiency considerations

Although MOSEK is implemented to handle memory efficiently, the user may have valuable knowledge about a problem, which could be used to improve the performance of MOSEK. This section discusses

some tricks and general advice that hopefully make MOSEK process your problem faster.

Avoid memory fragmentation: MOSEK stores the optimization problem in internal data structures in the memory. Initially MOSEK will allocate structures of a certain size, and as more items are added to the problem the structures are reallocated. For large problems the same structures may be reallocated many times causing memory fragmentation. One way to avoid this is to give MOSEK an estimated size of your problem using the functions:

- MSK_putmaxnumvar. Estimate for the number of variables.
- MSK_putmaxnumcon. Estimate for the number of constraints.
- MSK_putmaxnumcone. Estimate for the number of cones.
- MSK_putmaxnumanz. Estimate for the number of non-zeros in A.
- MSK_putmaxnumqnz. Estimate for the number of non-zeros in the quadratic terms.

None of these functions change the problem, they only give hints to the eventual dimension of the problem. If the problem ends up growing larger than this, the estimates are automatically increased.

Tune the reallocation process: It is possible to obtain information about how often MOSEK reallocates storage for the A matrix by inspecting MSK_IINF_STO_NUM_A_REALLOC. A large value indicates that `maxnumanz` has been reestimated many times and that the initial estimate should be increased.

Do not mix `put-` and `get-` functions: For instance, the functions MSK_putavec and MSK_getavec. MOSEK will queue `put-` commands internally until a `get-` function is called. If every `put-` function call is followed by a `get-` function call, the queue will have to be flushed often, decreasing efficiency.

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

Use the LIFO principle when removing constraints and variables: MOSEK can more efficiently remove constraints and variables with a high index than a small index.

An alternative to removing a constraint or a variable is to fix it at 0, and set all relevant coefficients to 0. Generally this will not have any impact on the optimization speed.

Add more constraints and variables than you need (now): The cost of adding one constraint or one variable is about the same as adding many of them. Therefore, it may be worthwhile to add many variables instead of one. Initially fix the unused variable at zero, and then later unfix them as needed. Similarly, you can add multiple free constraints and then use them as needed.

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

Do not remove basic variables: When doing reoptimizations, instead of removing a basic variable it may be more efficient to fix the variable at zero and then remove it when the problem is reoptimized and it has left the basis. This makes it easier for MOSEK to restart the simplex optimizer.

C name	C type	Dimension	Related problem parameter
numcon	int		m
numvar	int		n
numcone	int		t
numqonz	int		q^o_{ij}
qosubi	int[]	numqonz	q^o_{ij}
qosubj	int[]	numqonz	q^o_{ij}
qoval	double*	numqonz	q^o_{ij}
c	double[]	numvar	c_j
cfix	double		c^f
numqcnz	int		q^k_{ij}
qcsubk	int[]	qcnz	q^k_{ij}
qcsubi	int[]	qcnz	q^k_{ij}
qcsubj	int[]	qcnz	q^k_{ij}
qcval	double*	qcnz	q^k_{ij}
aptrb	int[]	numvar	a_{ij}
aptre	int[]	numvar	a_{ij}
asub	int[]	aptre[numvar-1]	a_{ij}
aval	double[]	aptre[numvar-1]	a_{ij}
bkc	MSKboundkeye*	numcon	l^c_k and u^c_k
blc	double[]	numcon	l^c_k
buc	double[]	numcon	u^c_k
bkx	MSKboundkeye *	numvar	l^x_k and u^x_k
blx	double[]	numvar	l^x_k
bux	double[]	numvar	u^x_k

Table 5.2: Naming convention used in MOSEK

5.8 Conventions employed in the API

5.8.1 Naming conventions for arguments

In the definition of the MOSEK C API a consistent naming convention has been used. This implies that whenever for example `numcon` is an argument in a function definition it indicates the number of constraints.

In Table 5.2 the variable names used to specify the problem parameters are listed.

The relation between the variable names and the problem parameters is as follows:

- The quadratic terms in the objective:

$$q^o_{\texttt{qosubi[t]},\texttt{qosubj[t]}} = \texttt{qoval[t]},\ t = 0, \ldots, \texttt{numqonz} - 1. \tag{5.29}$$

Symbolic constant	Lower bound	Upper bound
MSK_BK_FX	finite	identical to the lower bound
MSK_BK_FR	minus infinity	plus infinity
MSK_BK_LO	finite	plus infinity
MSK_BK_RA	finite	finite
MSK_BK_UP	minus infinity	finite

Table 5.3: Interpretation of the bound keys.

- The linear terms in the objective:
$$c_j = \texttt{c[j]},\ j = 0, \ldots, \texttt{numvar} - 1 \tag{5.30}$$
- The fixed term in the objective:
$$c^f = \texttt{cfix}. \tag{5.31}$$
- The quadratic terms in the constraints:
$$q^{\texttt{qcsubk[t]}}_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} = \texttt{qcval[t]},\ t = 0, \ldots, \texttt{numqcnz} - 1. \tag{5.32}$$
- The linear terms in the constraints:
$$a_{\texttt{asub[t]},\texttt{j}} = \texttt{aval[t]}, \quad \begin{array}{l} t = \texttt{ptrb[j]}, \ldots, \texttt{ptre[j]} - 1, \\ j = 0, \ldots, \texttt{numvar} - 1. \end{array} \tag{5.33}$$
- The bounds on the constraints are specified using the variables `bkc`, `blc`, and `buc`. The components of the integer array `bkc` specify the bound type according to Table 5.3. For instance `bkc[2]=`MSK_BK_LO means that $-\infty < l^c_2$ and $u^c_2 = \infty$. Finally, the numerical values of the bounds are given by
$$l^c_k = \texttt{blc[k]},\ k = 0, \ldots, \texttt{numcon} - 1 \tag{5.34}$$
and
$$u^c_k = \texttt{buc[k]},\ k = 0, \ldots, \texttt{numcon} - 1. \tag{5.35}$$
- The bounds on the variables are specified using the variables `bkx`, `blx`, and `bux`. The components in the integer array `bkx` specify the bound type according to Table 5.3. The numerical values for the lower bounds on the variables are given by
$$l^x_j = \texttt{blx[j]},\ j = 0, \ldots, \texttt{numvar} - 1. \tag{5.36}$$
The numerical values for the upper bounds on the variables are given by
$$u^x_j = \texttt{bux[j]},\ j = 0, \ldots, \texttt{numvar} - 1. \tag{5.37}$$

5.8.1.1 Bounds

A bound on a variable or on a constraint in MOSEK consists of a *bound key*, as defined in Table 5.3, a lower bound value and an upper bound value. Even if a variable or constraint is bounded only from below, e.g. $x \geq 0$, both bounds are inputted or extracted; the value inputted as upper bound for $(x \geq 0)$ is ignored.

5.8.2 Vector formats

Three different vector formats are used in the MOSEK API:

Full vector: This is simply an array where the first element corresponds to the first item, the second element to the second item etc. For example to get the linear coefficients of the objective in `task`, one would write

```
30 MSKrealt * c = MSK_calloctask(task, numvar, sizeof(MSKrealt));
31 MSK_getc(task,c);
```

where `numvar` is the number of variables in the problem.

Vector slice: A vector slice is a range of values. For example, to get the bounds associated constraint 3 through 10 (both inclusive) one would write

```
36 MSKrealt * upper_bound   = MSK_calloctask(task,8,sizeof(MSKrealt));
37 MSKrealt * lower_bound   = MSK_calloctask(task,8,sizeof(MSKrealt));
38 MSKboundkeye * bound_key = MSK_calloctask(task,8,sizeof(MSKboundkeye));
39 MSK_getboundslice(task,MSK_ACC_CON, 2,10,
40                   bound_key,lower_bound,upper_bound);
```

Please note that items in MOSEK are numbered from 0, so that the index of the first item is 0, and the index of the n'th item is $n-1$.

Sparse vector: A sparse vector is given as an array of indexes and an array of values. For example, to input a set of bounds associated with constraints number 1, 6, 3, and 9, one might write

```
44 MSKidxt bound_index[]    = {         1,         6,         3,         9 };
45 MSKboundkeye bound_key[] = { MSK_BK_FR, MSK_BK_LO, MSK_BK_UP, MSK_BK_FX };
46 MSKrealt lower_bound[]   = {       0.0,     -10.0,       0.0,       5.0 };
47 MSKrealt upper_bound[]   = {       0.0,       0.0,       6.0,       5.0 };
48 MSK_putboundlist(task,MSK_ACC_CON, 4, bound_index,
49                   bound_key,lower_bound,upper_bound);
```

Note that the list of indexes need not be ordered.

5.8.3 Matrix formats

The coefficient matrices in a problem are inputted and extracted in a sparse format, either as complete or a partial matrices. Basically there are two different formats for this.

5.8.3.1 Unordered triplets

In unordered triplet format each entry is defined as a row index, a column index and a coefficient. For example, to input the A matrix coefficients for $a_{1,2} = 1.1$, $a_{3,3} = 4.3$, and $a_{5,4} = 0.2$, one would write as follows:

```
53 MSKidxt subi[] = {   1,   3,   5 };
54 MSKidxt subj[] = {   2,   3,   4 };
55 MSKrealt cof[] = { 1.1, 4.3, 0.2 };
56 MSK_putaijlist(task,3, subi,subj,cof);
```

Please note that in some cases (like MSK_putaijlist) *only* the specified indexes remain modified — all other are unchanged. In other cases (such as MSK_putqconk) the triplet format is used to modify *all* entries — entries that are not specified are set to 0.

5.8.3.2 Row or column ordered sparse matrix

In a sparse matrix format only the non-zero entries of the matrix are stored. MOSEK uses a sparse matrix format ordered either by rows or columns. In the column-wise format the position of the non-zeros are given as a list of row indexes. In the row-wise format the position of the non-zeros are given as a list of column indexes. Values of the non-zero entries are given in column or row order.

A sparse matrix in column ordered format consists of:

`asub`: List of row indexes.

`aval`: List of non-zero entries of A ordered by columns.

`ptrb`: Where `ptrb[j]` is the position of the first value/index in `aval` / `asub` for column j.

`ptre`: Where `ptre[j]` is the position of the last value/index plus one in `aval` / `asub` for column j.

The values of a matrix A with `numcol` columns are assigned so that for

$$j = 0, \ldots, \texttt{numcol} - 1.$$

We define

$$a_{\texttt{asub}[k],j} = \texttt{aval}[k], \quad k = \texttt{ptrb}[j], \ldots, \texttt{ptre}[j] - 1. \tag{5.38}$$

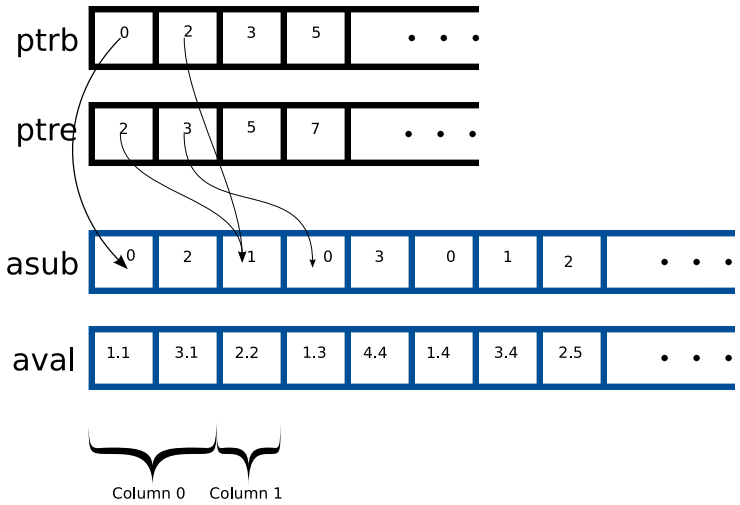

Figure 5.1: The matrix A (5.39) represented in column ordered sparse matrix format.

As an example consider the matrix

$$A = \begin{bmatrix} 1.1 & & 1.3 & 1.4 & \\ & 2.2 & & & 2.5 \\ 3.1 & & & 3.4 & \\ & & 4.4 & & \end{bmatrix}. \tag{5.39}$$

which can be represented in the column ordered sparse matrix format as

$$\begin{array}{lcl} \texttt{ptrb} & = & [0, 2, 3, 5, 7], \\ \texttt{ptre} & = & [2, 3, 5, 7, 8], \\ \texttt{asub} & = & [0, 2, 1, 0, 3, 0, 2, 1], \\ \texttt{aval} & = & [1.1, 3.1, 2.2, 1.3, 4.4, 1.4, 3.4, 2.5]. \end{array}$$

Fig. 5.1 illustrates how the matrix A (5.39) is represented in column ordered sparse matrix format.

5.8.3.3 Row ordered sparse matrix

The matrix A (5.39) can also be represented in the row ordered sparse matrix format as:

$$\begin{array}{lcl} \texttt{ptrb} & = & [0, 3, 5, 7], \\ \texttt{ptre} & = & [3, 5, 7, 8], \\ \texttt{asub} & = & [0, 2, 3, 1, 4, 0, 3, 2], \\ \texttt{aval} & = & [1.1, 1.3, 1.4, 2.2, 2.5, 3.1, 3.4, 4.4]. \end{array}$$

5.9 The license system

By default a license token is checked out when MSK_optimizetrm is first called and is returned when the MOSEK environment is deleted. Calling MSK_optimizetrm from different threads using the same MOSEK environment only consumes one license token.

To change the license systems behavior to returning the license token after each call to MSK_optimizetrm set the parameter MSK_IPAR_CACHE_LICENSE to MSK_OFF. Please note that there is a small overhead associated with setting this parameter, since checking out a license token from the license server can take a small amount of time.

Additionally license checkout and checkin can be controlled manually with the functions MSK_checkinlicense and MSK_checkoutlicense.

5.9.1 Waiting for a free license

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

Chapter 6

Advanced API tutorial

This chapter provides information about additional problem classes and functionality provided in the C API.

6.1 Separable convex optimization

In this section we will discuss solution of nonlinear **separable** convex optimization problems using MOSEK. We allow both nonlinear constraints and objective, but restrict ourselves to separable functions.

6.1.1 The problem

A general separable nonlinear optimization problem can be specified as follows:

$$\begin{array}{lccccc} \text{minimize} & & & f(x) + c^T x & & \\ \text{subject to} & l^c & \leq & g(x) + Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \tag{6.1}$$

where

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

- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $f : \mathbb{R}^n \to \mathbb{R}$ is a nonlinear function.
- $g : \mathbb{R}^n \to \mathbb{R}^m$ is a nonlinear vector function.

This implies that the ith constraint essentially has the form

$$l_i^c \leq g_i(x) + \sum_{j=1}^{n} a_{ij} x_j \leq u_i^c.$$

The problem (6.1) must satisfy the three important requirements:

1. Separability: This requirement implies that all nonlinear functions can be written on the form

   $$f(x) = \sum_{j=1}^{n} f^j(x_j)$$

   and

   $$g_i(x) = \sum_{j=1}^{n} g_i^j(x_j)$$

   where

   $$f^j : \mathbb{R} \to \mathbb{R} \text{ and } g_i^j : \mathbb{R} \to \mathbb{R}.$$

   Hence, the nonlinear functions can be written as a sum of functions which depends only one variable.

2. Differentiability: All functions should be twice differentiable for all x_j satisfying

   $$l_j^x < x < u_j^x$$

   if x_j occurs in at least one nonlinear function.

3. Convexity: The problem should be a convex optimization problem. See Section 7.4 for a discussion of this requirement.

6.1.2 A numerical example

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

$$\begin{array}{lll} \text{minimize} & x_1 - \ln(x_1 + 2x_2) & \\ \text{subject to} & x_1^2 + x_2^2 & \leq \quad 1 \end{array} \tag{6.2}$$

First note that the problem (6.2) is not a separable optimization problem because the logarithmic term in the objective is not a function of a single variable. However, by introducing a constraint and a variable the problem can be made separable as follows

$$\begin{array}{lccc} \text{minimize} & x_1 - \ln(x_3) & & \\ \text{subject to} & x_1^2 + x_2^2 & \leq & 1, \\ & x_1 + 2x_2 - x_3 & = & 0, \\ & x_3 \geq 0. & & \end{array} \tag{6.3}$$

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

$$x_3 > 0.$$

This assures sure that function evaluation errors will not occur during the optimization process because MOSEK will only evaluate $\ln(x_3)$ for $x_3 > 0$.

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

6.1.3 `scopt` an optimizer for separable convex optimization

`scopt` is an easy-to-use interface to MOSEKs nonlinear capabilities for solving separable convex problems. As currently implemented `scopt` is not capable of handling arbitrary nonlinear expressions. In fact `scopt` can handle only the nonlinear expressions $x\log(x)$, e^x, $\log(x)$, and x^g. However, in a subsequent section we will demonstrate that it is easy to expand the number of nonlinear expressions that `scopt` can handle.

6.1.3.1 Design principles of `scopt`

All the linear data of the problem, such as c and A, is inputted to MOSEK as usual, i.e. using the relevant functions in the MOSEK API.

The nonlinear part of the problem is specified using some arrays which indicate the type of the nonlinear expressions and where these should be added.

For example given the three `int` arrays — `oprc`, `opric`, and `oprjc` — and the two `double` arrays — `oprfc` and `oprgc` — the nonlinear expressions in the constraints can be coded in those arrays using the following table:

oprc[k]	opric[k]	oprjc[k]	oprfc[k]	oprgc[k]	oprhc[k]	Expression added to constraint i
0	i	j	f	g	h	$fx_j \ln(x_j)$
1	i	j	f	g	h	fe^{gx_j+h}
2	i	j	f	g	h	$f \ln(gx_j + h)$
3	i	j	f	g	h	$f(x_j + h)^g$

Hence, `oprc[k]` specifies the nonlinear expression type, `opric[k]` indicates to which constraint the nonlinear expression should be added. `oprfc[k]` and `oprgc[k]` are parameters used when the nonlinear expression is evaluated. This implies that nonlinear expressions can be added to an arbitrary constraint and hence you can create multiple nonlinear constraints.

Using the same method all the nonlinear terms in the objective can be specified using `opro[k]`, `oprjo[k]`, `oprfo[k]` and `oprho[k]` as shown below:

opro[k]	oprjo[k]	oprfo[k]	oprgo[k]	oprho[k]	Expression added in objective
0	j	f	g	h	$fx_j \ln(x_j)$
1	j	f	g	h	fe^{gx_j+h}
2	j	f	g	h	$f \ln(gx_j + h)$
3	j	f	g	h	$f(x_j + h)^g$

6.1.3.2 Example

Suppose we want to add the nonlinear expression $-ln(x_3)$ to the objective. This is an expression on the form $f \ln(gx_j + h)$ where $f = -1$, $g = 1$, $h = 0$ and $j = 3$. This can be represented by:

```
opro[0]  = 2
oprjo[0] = 3
oprfo[0] = -1.0
oprgo[0] = 1.0
oprho[0] = 0.0
```

6.1.3.3 Source code

The source code for `scopt` consists of the files:

- `scopt.h`: An include file that defines the two functions `MSK_scbegin` and `MSK_scend`, which are used to initialize and remove the nonlinear function data.
- `scopt.c` : This file implements the nonlinear initialization and evaluation functions.
- `tstscopt.c` : This file solves the example problem (6.2) using `scopt.c`.

These three files are all available in the directory

```
mosek\6\tools\examples\c
```

We will not discuss the implementation of `scopt` in details but rather refer the reader to the source code found in `scopt.c` which is included in the distribution. However, the driver program `tstscopt.c` which solves the example (6.2).

```
/*
   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

   File     : tstscopt.c

   Purpose  : To solve the problem

              minimize    x_1 - log(x_3)
              subject to  x_1^2 + x_2^2 <= 1
                          x_1 + 2*x_2 - x_3 = 0
                          x_3 >=0
 */

#include "scopt.h"

#define NUMOPRO  1 /* Number of nonlinear expressions in the obj. */
#define NUMOPRC  2 /* Number of nonlinear expressions in the con. */
#define NUMVAR   3 /* Number of variables.     */
#define NUMCON   2 /* Number of constraints.   */
#define NUMANZ   3 /* Number of non-zeros in A. */

static void MSKAPI printstr(void *handle,
                            char str[])
{
  printf("%s",str);
} /* printstr */

int main()
{
  char          buffer[MSK_MAX_STR_LEN];
  double        oprfo[NUMOPRO],oprgo[NUMOPRO],oprho[NUMOPRO],
                oprfc[NUMOPRC],oprgc[NUMOPRC],oprhc[NUMOPRC],
                c[NUMVAR],aval[NUMANZ],
                blc[NUMCON],buc[NUMCON],blx[NUMVAR],bux[NUMVAR];
  int           numopro,numoprc,
                numcon=NUMCON,numvar=NUMVAR,
                opro[NUMOPRO],oprjo[NUMOPRO],
                oprc[NUMOPRC],opric[NUMOPRC],oprjc[NUMOPRC],
                aptrb[NUMVAR],aptre[NUMVAR],asub[NUMANZ];
  MSKboundkeye bkc[NUMCON],bkx[NUMVAR];
  MSKenv_t      env;
  MSKrescodee   r;
  MSKtask_t     task;
  schand_t      sch;

  /* Specify nonlinear terms in the objective. */
  numopro   = NUMOPRO;
  opro[0]   = MSK_OPR_LOG; /* Defined in scopt.h */
  oprjo[0] = 2;
  oprfo[0] = -1.0;
  oprgo[0] = 1.0;  /* This value is never used. */
  oprho[0] = 0.0;

  /* Specify nonlinear terms in the constraints. */
```

```
55  numoprc  = NUMOPRC;
56
57  oprc[0]  = MSK_OPR_POW;
58  opric[0] = 0;
59  oprjc[0] = 0;
60  oprfc[0] = 1.0;
61  oprgc[0] = 2.0;
62  oprhc[0] = 0.0;
63
64  oprc[1]  = MSK_OPR_POW;
65  opric[1] = 0;
66  oprjc[1] = 1;
67  oprfc[1] = 1.0;
68  oprgc[1] = 2.0;
69  oprhc[1] = 0.0;
70
71  /* Specify c */
72  c[0] = 1.0; c[1] = 0.0; c[2] = 0.0;
73
74  /* Specify a. */
75  aptrb[0] = 0;   aptrb[1] = 1;   aptrb[2] = 2;
76  aptre[0] = 1;   aptre[1] = 2;   aptre[2] = 3;
77  asub[0]  = 1;   asub[1]  = 1;   asub[2]  = 1;
78  aval[0]  = 1.0; aval[1]  = 2.0; aval[2]  = -1.0;
79
80  /* Specify bounds for constraints. */
81  bkc[0] = MSK_BK_UP;     bkc[1] = MSK_BK_FX;
82  blc[0] = -MSK_INFINITY; blc[1] = 0.0;
83  buc[0] = 1.0;           buc[1] = 0.0;
84
85  /* Specify bounds for variables. */
86  bkx[0] = MSK_BK_FR;      bkx[1] = MSK_BK_FR;     bkx[2] = MSK_BK_LO;
87  blx[0] = -MSK_INFINITY;  blx[1] = -MSK_INFINITY; blx[2] = 0.0;
88  bux[0] = MSK_INFINITY;   bux[1] = MSK_INFINITY;  bux[2] = MSK_INFINITY;
89
90  /* Create  the mosek environment. */
91  r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
92
93  /* Check whether the return code is ok. */
94  if ( r==MSK_RES_OK )
95  {
96    /* Directs the log stream to the user
97       specified procedure 'printstr'. */
98    MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
99  }
100
101 if ( r==MSK_RES_OK )
102 {
103   /* Initialize the environment. */
104   r = MSK_initenv(env);
105 }
106
107 if ( r==MSK_RES_OK )
108 {
109   /* Make the optimization task. */
110   r = MSK_makeemptytask(env,&task);
111   if ( r==MSK_RES_OK )
112     MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
```

```
113
114     if ( r==MSK_RES_OK )
115     {
116       r = MSK_inputdata(task,
117                         numcon,numvar,
118                         numcon,numvar,
119                         c,0.0,
120                         aptrb,aptre,
121                         asub,aval,
122                         bkc,blc,buc,
123                         bkx,blx,bux);
124     }
125
126     if ( r== MSK_RES_OK )
127     {
128       /* Set-up of nonlinear expressions. */
129       r = MSK_scbegin(task,
130                       numopro,opro,oprjo,oprfo,oprgo,oprho,
131                       numoprc,oprc,opric,oprjc,oprfc,oprgc,oprhc,
132                       &sch);
133
134
135       if ( r==MSK_RES_OK )
136       {
137         printf("Start optimizing\n");
138
139         r = MSK_optimize(task);
140
141         printf("Done optimizing\n");
142
143         MSK_solutionsummary(task,MSK_STREAM_MSG);
144       }
145
146       /* The nonlinear expressions are no longer needed. */
147       MSK_scend(task,&sch);
148     }
149     MSK_deletetask(&task);
150   }
151   MSK_deleteenv(&env);
152
153   printf("Return code: %d\n",r);
154   if ( r!=MSK_RES_OK )
155   {
156     MSK_getcodedisc(r,buffer,NULL);
157     printf("Description: %s\n",buffer);
158   }
159 } /* main */
```

6.1.3.4 Adding more nonlinear expression types

`scopt` handles only a limited number of nonlinear expression types. However, it is easy to add a new operator such as the square root operator. First step is to define the new operator in the file `scopt.h` that after modification contains the lines

```
15 #define MSK_OPR_ENT 0
```

```
16 #define MSK_OPR_EXP 1
17 #define MSK_OPR_LOG 2
18 #define MSK_OPR_POW 3
19 #define MSK_OPR_SQRT 4 /* constant for square root operator */
```

Next the function `evalopr` in the file `scopt.c` should be modified. The purpose of `evalopr` is to compute the function value, the gradient (first derivative), and the Hessian (second derivative) for a nonlinear expression. The modified function has the form:

```
310 static int evalopr(int    opr,
311                     double f,
312                     double g,
313                     double h,
314                     double xj,
315                     double *fxj,
316                     double *grdfxj,
317                     double *hesfxj)
318 /* Purpose: To evaluate an operator and its derivatives.
319      fxj:    Is the function value
320      grdfxj: Is the first derivative.
321      hexfxj: Is the second derivative.
322  */
323 {
324   double rtemp;
325
326   switch ( opr )
327   {
328     case MSK_OPR_ENT:
329       if ( fxj )
330       fxj[0] = f*xj*log(xj);
331
332       if ( grdfxj )
333       grdfxj[0] = f*(1.0+log(xj));
334
335       if ( hesfxj )
336       hesfxj[0] = f/xj;
337       break;
338     case MSK_OPR_EXP:
339       if ( fxj || grdfxj || hesfxj )
340       {
341         rtemp = exp(g*xj+h);
342
343         if ( fxj )
344         fxj[0] = f*rtemp;
345
346         if ( grdfxj )
347         grdfxj[0] = f*g*rtemp;
348
349         if ( hesfxj )
350         hesfxj[0] = f*g*g*rtemp;
351       }
352       break;
353     case MSK_OPR_LOG:
354       rtemp = g*xj+h;
355       if ( rtemp<=0.0 )
356         return ( 1 );
357
```

```
358       if ( fxj )
359       fxj[0] = f*log(rtemp);
360
361       if ( grdfxj )
362       grdfxj[0] = (g*f)/(rtemp);
363
364       if ( hesfxj )
365       hesfxj[0] = -(f*g*g)/(rtemp*rtemp);
366       break;
367     case MSK_OPR_POW:
368       if ( fxj )
369       fxj[0] = f*pow(xj+h,g);
370
371       if ( grdfxj )
372       grdfxj[0] = f*g*pow(xj+h,g-1.0);
373
374       if ( hesfxj )
375         hesfxj[0] = f*g*(g-1.0)*pow(xj+h,g-2.0);
376       break;
377     case MSK_OPR_SQRT: /* handle operator f * sqrt(x+g) */
378       if ( fxj )
379         fxj[0] = f*sqrt(g*xj+h); /* The function value. */
380
381       if ( grdfxj )
382         grdfxj[0] = 0.5*f*g/sqrt(g*xj+h); /* The gradient. */
383
384
385       if ( hesfxj )
386         hesfxj[0] = -0.25*f*g*g*pow(g*xj+h,-1.5);
387       break;
388     default:
389       printf("scopt.c: Unknown operator %d\n",opr);
390       exit(0);
391   }
392
393   return ( 0 );
394 } /* evalopr */
```

6.2 Exponential optimization

6.2.1 The problem

An exponential optimization problem has the form

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{k\in J_0} c_k e^{\left(\sum_{j=0}^{n-1} a_{k,j}x_j\right)} & \\ \text{subject to} & \displaystyle\sum_{k\in J_i} c_k e^{\left(\sum_{j=0}^{n-1} a_{k,j}x_j\right)} \quad \leq \quad 1, \quad i=1,\ldots,m, \\ & x \in \mathbb{R}^n & \end{array} \tag{6.4}$$

where it is assumed that

$$\cup_{i=0}^{m} J_k = \{1, \ldots, T\}$$

and

$$J_i \cap J_j = \emptyset$$

if $i \neq j$.

Given

$$c_i > 0, \ , i = 1, \ldots, T$$

the problem (6.4) is a convex optimization problem which can be solved using MOSEK. We will call

$$c_t e^{\left(\sum_{j=0}^{n-1} a_{t,j} x_j\right)} = e^{\left(\log(c_t) + \sum_{j=0}^{n-1} a_{t,j} x_j\right)}$$

for a term and hence the number of terms is T.

As stated the problem (6.4) is a nonseparable problem. However, using

$$v_t = e^{\left(\log(c_t) + \sum_{j=0}^{n-1} a_{tj} x_j\right)}$$

we obtain the separable problem

$$\begin{array}{lllll} \text{minimize} & \sum_{t \in J_0} e^{v_t} & & & \\ \text{subject to} & \sum_{t \in J_i} e^{v_t} & \leq & 1, & i = 1, \ldots, m, \\ & \sum_{j=0}^{n-1} a_{t,j} x_j - v_t & = & -\log(c_t), & t = 0, \ldots, T, \end{array} \tag{6.5}$$

which could be solved using the scopt interface discussed in Section 6.1. A warning about this approach is that computing the function

$$e^x$$

using double-precision floating point numbers is only possible for small values of x in absolute value. Indeed e^x grows very rapidly for larger x values, and numerical problems may arise when solving the problem on this form.

It is also possible to reformulate the exponential optimization problem (6.4) as a dual geometric geometric optimization problem (6.12). This is often the preferred solution approach because it is computationally more efficient and the numerical problems associated with evaluating e^x for moderately large x values are avoided.

6.2.2 Source code

The MOSEK distribution includes the source code for a program that enables you to:

1. Read (and write) a data file stating an exponential optimization problem.
2. Verify that the input data is reasonable.

3. Solve the problem via the exponential optimization problem (6.5) or the dual geometric optimization problem (6.12).

4. Write a solution file.

6.2.3 Solving from the command line.

In the following we will discuss the program `mskexpopt`, which is included in the MOSEK distribution, in both source code and compiled form. Hence, you can solve exponential optimization problems using the operating system command line or directly from your own C program.

6.2.3.1 The input format

First we will define a text input format for specifying an exponential optimization problem. This is as follows:

$$
\begin{array}{rlll}
\text{* This is a comment} & & & \\
numcon & & & \\
numvar & & & \\
numter & & & \\
c_1 & & & \\
c_2 & & & \\
\vdots & & & \\
c_{numter} & & & \\
i_1 & & & \\
i_2 & & & \\
\vdots & & & \\
i_{numter} & & & \\
t_1 & j_1 & a_{t_1,j_1} & \\
t_2 & j_2 & a_{t_2,j_2} & \\
\vdots & \vdots & \vdots &
\end{array}
$$

The first line is an optional comment line. In general everything occurring after a * is considered a comment. Lines 2 to 4 inclusive define the number of constraints (m), the number of variables (n), and the number of terms T in the problem. Then follows three sections containing the problem data.

The first section

$$
\begin{array}{l}
c_1 \\
c_2 \\
\vdots \\
c_{numter}
\end{array}
$$

lists the coefficients c_t of each term t in their natural order.

The second section

$$\begin{array}{l} i_1 \\ i_2 \\ \vdots \\ i_{numter} \end{array}$$

specifies to which constraint each term belongs. Hence, for instance $i_2 = 5$ means that the term number 2 belongs to constraint 5. $i_t = 0$ means that term number t belongs to the objective.

The third section

$$\begin{array}{lll} t_1 & j_1 & a_{t_1,j_1} \\ t_2 & j_2 & a_{t_2,j_2} \\ \vdots & \vdots & \vdots \end{array}$$

defines the non-zero $a_{t,j}$ values. For instance the entry

$$1 \quad 3 \quad 3.3$$

implies that $a_{t,j} = 3.3$ for $t = 1$ and $j = 3$.

Please note that each $a_{t,j}$ should be specified only once.

6.2.4 Choosing primal or dual form

One can choose to solve the exponential optimization problem directly in the primal form (6.5) or on the dual form. By default `mskexpopt` solves a problem on the dual form since usually this is more efficient. The command line option

```
-primal
```

chooses the primal form.

6.2.5 An example

Consider the problem:

$$\begin{array}{lll} \text{minimize} & 40e^{-x_1-1/2x_2-x_3} + 20e^{x_1+x_3} + 40e^{x_1+x_2+x_3} & \\ \text{subject to} & \frac{1}{3}e^{-2x_1-2x_2} + \frac{4}{3}e^{1/2x_2-x_3} & \leq \quad 1. \end{array} \tag{6.6}$$

This small problem can be specified as follows using the input format:

```
* File : expopt1.eo

1   * numcon
3   * numvar
5   * numter

* Coefficients of terms
```

```
40
20
40
0.3333333
1.3333333

* For each term, the index of the
* constraints to the term belongs

0
0
0
1
1

* Section defining a_kj

0 0 -1
0 1 -0.5
0 2 -1
1 0 1.0
1 2 1.0
2 0 1.0
2 1 1.0
2 2 1.0
3 0 -2
3 1 -2
4 1 0.5
4 2 -1.0
```

Using the program `mskexpopt` included in the MOSEK distribution the example can be solved. Indeed the command line

```
mskexpopt expopt1.eo
```

will produce the solution file `expopt1.sol` shown below.

```
PROBLEM STATUS      : PRIMAL_AND_DUAL_FEASIBLE
SOLUTION STATUS     : OPTIMAL
PRIMAL OBJECTIVE    : 1.331371e+02

VARIABLES
INDEX   ACTIVITY
1       6.931471e-01
2       -6.931472e-01
3       3.465736e-01
```

6.2.6 Solving from your C code

The C source code for solving an exponential optimization problem is included in the MOSEK distribution. The relevant source code consists of the files:

`expopt.h`: Defines prototypes for the functions:

`MSK_expoptread`: Reads a problem from a file.

`MSK_expoptsetup`: Sets up a problem. The function takes the arguments:

- `expopttask`: A MOSEK task structure.
- `solveform`: If 0, then the optimizer will choose whether the problem is solved on primal or dual form. If -1 the primal form is used and if 1 the dual form.
- `numcon`: Number of constraints.
- `numvar`: Number of variables.
- `numter`: Number of terms T.
- `*subi`: Array of length `numter` defining which constraint a term belongs to or zero for the objective.
- `*c`: Array of length `numter` containing coefficients for the terms.
- `numanz`: Length of `subk`, `subj`, and `akj`.
- `*subk`: Term indexes.
- `*subj`: Variable indexes.
- `*akj`: `akj[i]` is coefficient of variable `subj[i]` in term `subk[i]`, i.e.

$$a_{\texttt{subk}[i],\texttt{subj}[i]} = \texttt{akj}[i].$$

- `*expopthnd`: Data structure containing nonlinear information.

`MSK_expoptimize`: Solves the problem and returns the problem status and the optimal primal solution.

`MSK_expoptfree`: Frees data structures allocated by `MSK_expoptsetup`.

`expopt.c`: Implements the functions specified in `expopt.h`.

`mskexpopt.c`: A command line interface.

As a demonstration of the interface a C program that solves (6.6) is included below.

```
1  /*
2     Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3
4     File     : tstexpopt.c
5
6     Purpose  : To demonstrate a simple interface for exponential optimization.
7  */
8
9  #include <string.h>
10
11 #include "expopt.h"
12
```

```
13 void MSKAPI printcb(void* handle,char str[])
14 {
15   printf("%s",str);
16 }
17
18
19 int main (int argc, char **argv)
20 {
21   int          r = MSK_RES_OK, numcon = 1, numvar = 3 , numter = 5;
22
23   int          subi[]   = {0,0,0,1,1};
24   int          subk[]   = {0,0,0,1,1,2,2,2,3,3,4,4};
25   double       c[]      = {40.0,20.0,40.0,0.333333,1.333333};
26   int          subj[]   = {0,1,2,0,2,0,1,2,0,1,1,2};
27   double       akj[]    = {-1,-0.5,-1.0,1.0,1.0,1.0,1.0,1.0,-2.0,-2.0,0.5,-1.0};
28   int          numanz   = 12;
29   double       objval;
30   double       xx[3];
31   double       y[5];
32   MSKenv_t     env;
33   MSKprostae   prosta;
34   MSKsolstae   solsta;
35   MSKtask_t    expopttask;
36   expopthand_t expopthnd = NULL;
37   /* Pointer to data structure that holds nonlinear information */
38
39   if (r == MSK_RES_OK)
40     r = MSK_makeenv (
41                      &env,
42                      NULL,
43                      NULL,
44                      NULL,
45                      NULL);
46
47   if (r == MSK_RES_OK)
48     r = MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printcb);
49
50   if (r == MSK_RES_OK)
51     r = MSK_initenv(env);
52
53   if (r == MSK_RES_OK)
54     MSK_makeemptytask(env,&expopttask);
55
56   if (r == MSK_RES_OK)
57     r = MSK_linkfunctotaskstream(expopttask,MSK_STREAM_LOG,NULL,printcb);
58
59   if (r == MSK_RES_OK)
60   {
61     /* Initialize expopttask with problem data */
62     r =  MSK_expoptsetup(expopttask,
63                          1, /* Solve the dual formulation */
64                          numcon,
65                          numvar,
66                          numter,
67                          subi,
68                          c,
69                          subk,
70                          subj,
```

```
                          akj,
                          numanz,
                          &expopthnd
                          /* Pointer to data structure holding nonlinear data */
                          );
  }

  /* Any parameter can now be changed with standard mosek function calls */
  if (r == MSK_RES_OK)
    r = MSK_putintparam(expopttask,MSK_IPAR_INTPNT_MAX_ITERATIONS,200);

  /* Optimize, xx holds the primal optimal solution,
   y holds solution to the dual problem if the dual formulation is used
  */

  if (r == MSK_RES_OK)
    r = MSK_expoptimize(expopttask,
                        &prosta,
                        &solsta,
                        &objval,
                        xx,
                        y,
                        &expopthnd);

  /* Free data allocated by expoptsetup */
  if (expopthnd)
    MSK_expoptfree(expopttask,
                   &expopthnd);

  MSK_deletetask(&expopttask);
  MSK_deleteenv(&env);

}
```

6.2.7 A warning about exponential optimization problems

Exponential optimization problem may in some cases have a final optimal objective value for a solution containing infinite values. Consider the simple example

$$\begin{array}{ll} \min & e^x \\ \text{s.t.} & x \in \mathbb{R}, \end{array}$$

which has the optimal objective value 0 at $x = -\infty$. Similar problems can occur in constraints.

Such a solution can not in general be obtained by numerical methods, which means that MOSEK will act unpredictably in these situations — possibly failing to find a meaningful solution or simply stalling.

6.3 General convex optimization

MOSEK provides an interface for general convex optimization which is discussed in this section.

6.3.1 A warning

Using the general convex optimization interface in MOSEK is complicated. It is recommended to use the conic solver, the quadratic solver or the `scopt` interface whenever possible. Alternatively GAMS or AMPL with MOSEK as solver are well-suited for general convex optimization problems.

6.3.2 The problem

A general nonlinear convex optimization problem is to minimize or maximize an objective function of the form

$$f(x) + \frac{1}{2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} q_{i,j}^{o} x_i x_j + \sum_{j=0}^{n-1} c_j x_j + c^f \tag{6.7}$$

subject to the functional constraints

$$l_k^c \leq g_k(x) + \frac{1}{2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} q_{i,j}^{k} x_i x_j + \sum_{j=0}^{n-1} a_{k,j} x_j \leq u_k^c,\ k = 0, \ldots, m-1, \tag{6.8}$$

and the bounds

$$l_j^x \leq x_j \leq u_j^x,\ j = 0, \ldots, n-1. \tag{6.9}$$

Please note that this problem is a generalization of linear and quadratic optimization. This implies that the parameters c, A, Q^o, Q, and so forth denote the same as in the case of linear and quadratic optimization. All linear and quadratic terms should be inputted to MOSEK as described for these problem classes. The general convex part of the problems is defined by the functions $f(x)$ and $g_k(x)$, which must be general nonlinear, twice differentiable functions.

6.3.3 Assumptions about a nonlinear optimization problem

MOSEK makes two assumptions about the optimization problem.

The first assumption is that all functions are at least twice differentiable on their domain. More precisely, $f(x)$ and $g(x)$ must be at least twice differentiable for all x so that

$$l^x < x < u^x.$$

The second assumption is that

$$f(x) + \frac{1}{2}x^T Q^o x \tag{6.10}$$

must be a convex function if the objective is minimized. Otherwise if the objective is maximized it must be a concave function. Moreover,

$$g_k(x) + \frac{1}{2}x^T Q^k x \tag{6.11}$$

must be a convex function if

$$u_k^c < \infty$$

and a concave function if

$$l_k^c > -\infty.$$

Note in particular that nonlinear equalities are not allowed.

If these two assumptions are not satisfied, then it cannot be guaranteed that MOSEK produces correct results or works at all.

6.3.4 Specifying general convex terms

MOSEK receives information about the general convex terms via two call-back functions implemented by the user:

- `MSK_nlgetspfunc`: For parsing information on structural information about f and g.
- `MSK_nlgetvafunc`: For parsing information on numerical information about f and g.

The call-back functions are passed to MOSEK with the function `MSK_putnlfunc`.

For an example of using the general convex framework see Section 6.4.

6.4 Dual geometric optimization

Dual geometric is a special class of nonlinear optimization problems involving a nonlinear and non-separable objective function. In this section we will show how to solve dual geometric optimization problems using MOSEK.

6.4.1 The problem

Consider the dual geometric optimization problem

$$\begin{array}{lll} \text{maximize} & f(x) & \\ \text{subject to} & Ax & = b, \\ & x \geq 0, & \end{array} \tag{6.12}$$

where $A \in \mathbb{R}^{m \times n}$ and all other quantities have conforming dimensions. Let t be an integer and p be a vector of $t+1$ integers satisfying the conditions

$$\begin{array}{rcll} p_0 & = & 0, & \\ p_i & < & p_{i+1}, & i = 1, \ldots, t, \\ p_t & = & n. & \end{array}$$

Then f can be stated as follows

$$f(x) = \sum_{j=0}^{n-1} x_j \ln\left(\frac{v_j}{x_j}\right) + \sum_{i=1}^{t} \left(\sum_{j=p_i}^{p_{i+1}-1} x_j \right) \ln\left(\sum_{j=p_i}^{p_{i+1}-1} x_j \right)$$

where $v \in \mathbb{R}^n$ is a vector positive positive values.

Given these assumptions, it can be proven that f is a concave function and therefore the dual geometric optimization problem can be solved using MOSEK.

For a thorough discussion of geometric optimization see [11, pp. 531-538].

We will introduce the following definitions:

$$x^i := \begin{bmatrix} x_{p_i} \\ x_{p_i+1} \\ \vdots \\ x_{p_{i+1}-1} \end{bmatrix}, \quad X^i := \operatorname{diag}(x^i), \quad \text{and} \quad e^i := \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^{p_{i+1}-p_i}.$$

which make it possible to state f on the form

$$f(x) = \sum_{j=0}^{n-1} x_j \ln\left(\frac{v_j}{x_j}\right) + \sum_{i=1}^{t}((e^i)^T x^i)\ln((e^i)^T x^i).$$

Furthermore, we have that

$$\nabla f(x) = \begin{bmatrix} \ln(v_0) - 1 - \ln(x_0) \\ \vdots \\ \ln(v_j) - 1 - \ln(x_j) \\ \vdots \\ \ln(v_{n-1}) - 1 - \ln(x_{n-1}) \end{bmatrix} + \begin{bmatrix} 0e^0 \\ (1 + \ln((e^1)^T x^1))e^1 \\ \vdots \\ (1 + \ln((e^i)^T x^i))e^i \\ \vdots \\ (1 + \ln((e^t)^T x^t))e^t \end{bmatrix}$$

and

$$\nabla^2 f(x) = \begin{bmatrix} -(X^0)^{-1} & 0 & 0 & \cdots & 0 \\ 0 & \frac{e^1(e^1)^T}{(e^1)^T x^1} - (X^1)^{-1} & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \frac{e^t(e^t)^T}{(e^t)^T x^t} - (X^t)^{-1} \end{bmatrix}.$$

Please note that the Hessian is a block diagonal matrix and, especially if t is large, it is very sparse — MOSEK will automatically exploit these features to speed up computations. Moreover, the Hessian can be computed cheaply, specificly in

$$O\left(\sum_{i=0}^{t}(p_{i+1} - p_i)^2\right)$$

operations.

6.4.2 A numerical example

In the following we will use the data

$$A = \begin{bmatrix} -1 & 1 & 1 & -2 & 0 \\ -0.5 & 0 & 1 & -2 & 0.5 \\ -1 & 1 & 1 & 0 & -1 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \quad v = \begin{bmatrix} 40 \\ 20 \\ 40 \\ \frac{1}{3} \\ \frac{4}{3} \end{bmatrix}$$

and the function f given by

$$\begin{aligned} f(x) &= \sum_{j=0}^{4} x_j \ln\left(\frac{v_j}{x_j}\right) \\ &\quad +(x_3 + x_4)\ln(x_3 + x_4) \end{aligned}$$

for demonstration purposes.

6.4.3 `dgopt`: A program for dual geometric optimization

The generic dual geometric optimization problem and a numerical example have been presented and we will now develop a program which can solve the dual geometric optimization problem using the MOSEK API.

6.4.3.1 Data input

The first problem is how to feed the problem data into MOSEK. Since the constraints of the optimization problem are linear, they can be specified fully using an MPS file as in the purely linear case. The MPS file for the numerical data above will look as follows:

```
NAME
ROWS
 N  obj
 E  c1
 E  c2
 E  c3
 E  c4
COLUMNS
    x1        obj       0
    x1        c1        -1
    x1        c2        -0.5
    x1        c3        -1
    x1        c4        1
    x2        obj       0
    x2        c1        1
    x2        c3        1
    x2        c4        1
```

```
    x3        obj       0
    x3        c1        1
    x3        c2        1
    x3        c3        1
    x3        c4        1
    x4        obj       0
    x4        c1        -2
    x4        c2        -2
    x5        obj       0
    x5        c2        0.5
    x5        c3        -1
RHS
    rhs       c4        1
RANGES
BOUNDS
ENDATA
```

Moreover, a file specifying f is required so for that purpose we define a file:

$$
\begin{array}{c}
t \\
v_0 \\
v_1 \\
\vdots \\
v_{n-1} \\
p_1 - p_0 \\
p_2 - p_1 \\
\vdots \\
p_t - p_{t-1}
\end{array}
$$

Hence, for the numerical example this file has the format:

```
2
40.0
20.0
40.0
0.33333333333333
1.33333333333333
3
2
```

6.4.3.2 Solving the numerical example

The example is solved by executing the command line

```
mskdgopt examp/data/dgo.mps examples/data/dgo.f
```

6.4.4 The source code: `dgopt`

The source code for the `dgopt` consists of the files:

- `dgopt.h` and `dgopt.c`: Functions for reading and solving the dual geometric optimization problem.
- `mskdgopt.c` : The command line interface.

These files are available in the MOSEK distribution in the directory:

```
mosek/6/tools/examples/c
```

The basic functionality of `dgopt` can be gathered by studying the function `main` in `mskdgopt.c`. This function first loads the linear part of the problem from an MPS file into the task. Next, the nonlinear part of the problem is read from a file with the function `MSK_dgoptread`. Finally, the nonlinear function is created and inputted with `MSK_dgoptsetup` and the problem is solved. The solution is written to the file `dgopt.sol`.

The following functions in `dgopt.c` are used to set up the information about the evaluation of the nonlinear objective function:

`MSK_dgoread` The purpose of this function is to read data from a file which specifies the nonlinear function f in the objective.

`MSK_dgosetup` This function creates the problem in the task. The information parsed to the function is stored in a data structure called `nlhandt`, defined in the program. This structure is later passed to the functions `gostruc` and `goeval` which are used to compute the gradient and the Hessian of f.

`gostruc` This function is a call-back function used by MOSEK. The function reports structural information about f such as the number of non-zeros in the Hessian and the sparsity pattern of the Hessian.

`goeval` This function is a call-back function used by MOSEK. It reports numerical information about f such as the objective value and gradient for a particular x value.

6.5 Linear network flow problems

Network flow problems are a special class of linear optimization problems which has many applications. A network consists of a set of points connected by a set of lines. Usually the points and lines are called *nodes* and *arcs*. Arcs may have an direction on them. The network is directed if all arcs are directed. The class of network flow problems is defined as follows.
Let $G = (\mathcal{N}, \mathcal{A})$ be a directed network of nodes $\mathcal{N}$ and arcs $\mathcal{A}$. Associated with every arc $(i, j) \in \mathcal{A}$ is a cost c_{ij} and a capacity $[l_{ij}^x, u_{ij}^x]$. Moreover, associated with each node $i \in \mathcal{N}$ in the network is a lower

limit l_i^c and an upper limit u_i^c on the demand (supply) of the node. The minimum cost of a network flow problem can be stated as follows:

$$\begin{array}{lrcccl} \text{minimize} & & & \displaystyle\sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} & & \\ \text{subject to} & l_i^c & \leq & \displaystyle\sum_{\{j:(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j:(j,i)\in\mathcal{A}\}} x_{ji} & \leq & u_i^c \quad \forall i \in \mathcal{N}, \\ & l_{ij}^x & \leq & x_{ij} & \leq & u_{ij}^x \quad \forall (i,j) \in \mathcal{A}. \end{array} \tag{6.13}$$

A classical example of a network flow problem is the transportation problem where the objective is to distribute goods from warehouses to customers at lowest possible total cost, see [2] for a detailed application reference.

The above graph formulation of the network flow problem implies the structural properties. Each variable appears in exactly two constraints with a numerical value of either -1.0 or $+1.0$.

It is well-known that problems with network flow structure can be solved efficiently with a specialized version of the simplex method. MOSEK includes such a network simplex implementation which can be called either directly using `MSK_netoptimize` or indirectly by letting the standard simplex optimizer extract the embedded network. This section shows how to solve a network problem by a direct call to `MSK_netoptimize`. For further details on how to exploit embedded network in the standard simplex optimizer, see Section 8.3.1.

6.5.1 A linear network flow problem example

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

$$\begin{array}{lrcrcrcrcrcl} \text{maximize} & x_0 & & & + & x_2 & + & & - & x_4 & + & x_5 & & \\ \text{subject to} & -x_0 & & & & & + & x_3 & & & & & = & 1, \\ & & & & & x_2 & - & x_3 & + & x_4 & + & x_5 & = & -2, \\ & x_0 & - & x_1 & & & & & - & x_4 & - & x_5 & = & 0, \\ & & & x_1 & - & x_2 & + & & & & & & = & 0, \end{array} \tag{6.14}$$

having the bounds $0 \leq x_j \leq \infty$ for $j = 0 \ldots 5$.

The corresponding graph $G = (\mathcal{N}, \mathcal{A})$ is displayed in fig.6.1.

6.5.1.1 Source code

In this section we will show how to solve (6.14) with the network optimizer.

The C program included below, which solves this problem, is distributed with MOSEK and can be found in the directory

```
mosek\6\tools\examples\c
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
```

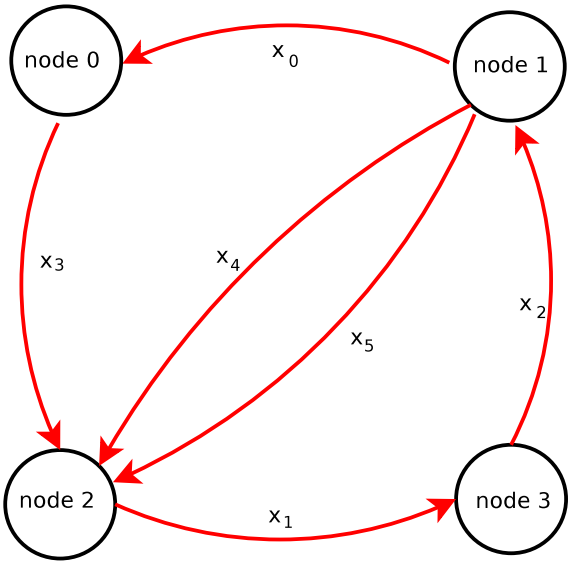

Figure 6.1: Simple network.

```
4
5  #include "mosek.h"
6
7  /*
8    Demonstrates a simple use of the network optimizer.
9
10    Purpose: 1. Specify data for a network.
11             2. Solve the network problem with the network optimizer.
12   */
13
14 /* Network sizes */
15 #define NUMCON 4 /* Nodes in network */
16 #define NUMVAR 6 /* Arcs in network */
17
18 void MSKAPI printlog(void *ptr,
19                      char s[])
20 {
21   printf("%s",s);
22 } /* printlog */
23
24 /* Main function  */
25 int main(int argc, char *argv[])
26 {
27   MSKrescodee         r;
28   MSKenv_t            env;
29   MSKtask_t           dummytask=NULL;
30   MSKidxt             from[] = {0,2,3,1,1,1},
31                       to[]   = {2,3,1,0,2,2};
32   MSKrealt            cc[]   = {0.0,0.0},
33                       cx[]   = {1.0,0.0,1.0,0.0,-1.0,1.0},
```

```
                        blc[]  = {1.0,1.0,-2.0,0.0},
                        buc[]  = {1.0,1.0,-2.0,0.0},
                        blx[]  = {0.0,0.0,0.0,0.0,0.0,0.0},
                        bux[]  = {MSK_INFINITY,MSK_INFINITY,MSK_INFINITY,
                                  MSK_INFINITY,MSK_INFINITY,MSK_INFINITY},
                        xc[NUMCON],xx[NUMVAR],y[NUMCON],slc[NUMCON],
                        suc[NUMCON],slx[NUMVAR],sux[NUMVAR];
  MSKboundkeye          bkc[] = {MSK_BK_FX,MSK_BK_FX,MSK_BK_FX,MSK_BK_FX},
                        bkx[] = {MSK_BK_LO,MSK_BK_LO,MSK_BK_LO,MSK_BK_LO,
                                  MSK_BK_LO,MSK_BK_LO};
  MSKstakeye            skc[NUMCON],skx[NUMVAR];
  MSKprostae            prosta;
  MSKsolstae            solsta;
  int                   i,j;

  r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);

  if ( r==MSK_RES_OK )
    MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printlog);

  if ( r==MSK_RES_OK )
    r = MSK_initenv(env);

  if ( r==MSK_RES_OK )
  {
    /* Create an optimization task.
       Will be used as a dummy task in MSK_netoptimize, parameters can be set here */
    r = MSK_maketask(env,0,0,&dummytask);
  }

  if ( r==MSK_RES_OK )
  {
    MSK_linkfunctotaskstream(dummytask, MSK_STREAM_LOG, NULL,     printlog);
  }

  if ( r==MSK_RES_OK )
  {
    r = MSK_putobjsense(dummytask, MSK_OBJECTIVE_SENSE_MAXIMIZE);
  }

  if ( r==MSK_RES_OK )
  {
    /* Solve network problem with a direct call into the network optimizer */
    r =  MSK_netoptimize(dummytask,
                         NUMCON,
                         NUMVAR,
                         cc,
                         cx,
                         bkc,
                         blc,
                         buc,
                         bkx,
                         blx,
                         bux,
                         from,
                         to,
                         &prosta,
                         &solsta,
```

```
                          0,
                          skc,
                          skx,
                          xc,
                          xx,
                          y,
                          slc,
                          suc,
                          slx,
                          sux);

    if ( solsta == MSK_SOL_STA_OPTIMAL )
    {
      printf("Network problem is optimal\n");

      printf("Primal solution is :\n");
      for( i = 0; i < NUMCON; ++i )
        printf("xc[%d] = %-16.10e\n",i,xc[i]);

      for( j = 0; j < NUMVAR; ++j )
        printf("Arc(%d,%d) -> xx[%d] = %-16.10e\n",from[j],to[j],j,xx[j]);
    }
    else if ( solsta == MSK_SOL_STA_PRIM_INFEAS_CER )
    {
      printf("Network problem is primal infeasible\n");
    }
    else if ( solsta == MSK_SOL_STA_DUAL_INFEAS_CER )
    {
      printf("Network problem is dual infeasible\n");
    }
    else
    {
      printf("Network problem solsta : %d\n",solsta);
    }
  }

  MSK_deletetask(&dummytask);
  MSK_deleteenv(&env);
}
```

6.5.1.2 Example code comments

There are a few important differences between the linear network optimization example in section 6.5.1.1 and the general linear optimization problem in section 5.2.

- MOSEK allows that network problems can be inputted and optimized using one function call to the function `MSK_netoptimize`. This is more efficient and uses less memory than a call to the standard optimizer.
- Since we know that each column of matrix A has two non-zeroes, it can be stored in two arrays, `from` and `to`, specifying the origin and destination of the arcs (variables), see graph in fig.fig-network.
- The solution is written directly to `skc`, `skx`, `xc`, `xx`, `y`, `slc`, `suc`, `slx` and `sux` by `MSK_netoptimize`.

6.6 Embedded network flow problems

Often problems contains both large parts with network structure and some non-network constraints or variables — such problems are said to have *embedded network structure*.

A linear optimization with embedded network structure problem can be written as :

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l_N^c & \leq & Nx & \leq & u_N^c, \\ & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \tag{6.15}$$

Where the constraints

$$l_N^c \leq Nx \leq u_N^c \tag{6.16}$$

defines a network as explained in section 6.5, and the constraints

$$l^c \leq Ax \leq u^c \tag{6.17}$$

defines the general non-network linear constraints. As an example consider the small linear optimization problem

$$\begin{array}{lrcrcrcrcrcrcl} \text{maximize} & -x_0 & & & + & x_2 & & & - & x_4 & + & x_5 & & \\ \text{subject to} & 0.50x_0 & & & & & + & 0.50x_3 & & & & & = & 0.5, \\ & & & & & 0.50x_2 & - & 0.50x_3 & + & 0.50x_4 & + & 0.50x_5 & = & -1, \\ & -0.25x_0 & + & -2.50x_1 & + & & & & - & 0.25x_4 & - & 0.25x_5 & = & 0, \\ & & & 2.50x_1 & - & 0.25x_2 & & & & & & & = & 0, \\ & & - & x_1 & + & x_2 & + & x_3 & & & + & x_5 & \geq & 6, \end{array} \tag{6.18}$$

with the bounds

$$-\infty \leq x_0 \leq 0, 0 \leq x_j \leq \infty \text{ for } j = 1 \ldots 5.$$

Recalling the network flow problem structural properties from section 6.13, each variable should appear in exactly two constraints with coefficients of either -1.0 or $+1.0$.

At first glance it does not seem to contain any network structure, but if we scale constraints 1-4 by respectively 2.0, 2.0, 4.0, 4.0 and columns 1-2 by -1.0, 0.1 we get the following problem :

$$\begin{array}{lrcrcrcrcrcrcl} \text{maximize} & x_0 & & & + & x_2 & + & & - & x_4 & + & x_5 & & \\ \text{subject to} & -x_0 & & & & & + & x_3 & & & & & = & 1, \\ & & & & & x_2 & - & x_3 & + & x_4 & + & x_5 & = & -2, \\ & x_0 & - & x_1 & & & & & - & x_4 & - & x_5 & = & 0, \\ & & & x_1 & - & x_2 & + & & & & & & = & 0, \\ & & & x_1 & + & x_2 & + & x_3 & & & + & x_5 & \geq & 6, \end{array} \tag{6.19}$$

with the bounds

$$0 \leq x_j \leq \infty \text{ for } j = 0 \ldots 5.$$

This corresponds to the network flow problem in section 6.5.1 plus one extra non-network constraint. We cannot use the network optimizer directly on the above problem since the last constraint destroys the network property. Finding the largest possible network structure in a linear optimization problem is computationally difficult, so MOSEK offers a heuristic `MSK_netextraction` that attempts to find suitable scaling factors maximizing numbers of network constraints and variables. Assuming that the embedded network structure is dominant and the problem has few non-network constraints, we can exploit this structure and potentially speed up the optimization. Since the network constraints can be handled efficiently by the specialized network optimizer, the following idea is used:

- Disregard the non-network constraints and optimize the network problem.
- Use the network solution to hot-start the standard dual simplex optimizer.

An embedded network can be exploited by this scheme in two ways:

- Use the extraction heuristics directly by the `MSK_netextraction` function and optimize with the `MSK_netoptimize` function.
- Let the simplex optimizer exploit embedded network structure automatically.

The first method is more difficult than the second, but also offers much more flexibility. In 6.6.1 the first method is demonstrated by a code example below. For further details on exploiting embedded network structure in the standard simplex optimizer, see section 8.3.1.

6.6.1 Example: Exploit embedded network flow structure in the simplex optimizer

MOSEK is distributed with some network examples which can be found in the directory

```
mosek\6\tools\examples
```

The example given in this section demonstrates how to extract and optimize embedded network structure in a arbitrary linear optimization problem. The following idea is used

- Read an arbitrary linear optimization problem into a task.
- Use the `MSK_netextraction` function to extract embedded network structure.
- Optimize the network problem using the `MSK_netoptimize` function.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "mosek.h"

/*
  Demonstrates a simple use of network structure in a model.

   Purpose: 1. Read an optimization problem from an
                          user specified MPS file.
                    2. Extract the embedded network.
                    3. Solve the embedded network with the network optimizer.

   Note that the general simplex optimizer called though MSK_optimize can also extract
   embedded network and solve it with the network optimizer. The direct call to the
   network optimizer, which is demonstrated here, is offered as an option to save
   memory and overhead when solving either many or large network problems.
 */

/* Helper functions  */
void addext(char filename[],
            char extension[])
{
  char *cptr;

  cptr = strrchr(filename,'.');
  if ( cptr && cptr[1]!='\\' )
    strcpy(cptr+1,extension);
  else
  {
    strcat(filename,".");
    strcat(filename,extension);
  }
} /* addext */

void MSKAPI printlog(void *ptr,
                     char s[])
{
  printf("%s",s);
} /* printlog */

/* Main function  */
int main(int argc, char *argv[])
{
  MSKrescodee         r;
  MSKenv_t            env;
  MSKtask_t           task=NULL,dummytask=NULL;
  MSKintt             numcon=0,numvar=0,numnetcon,numnetvar;
  MSKidxt             *netcon,*netvar,*from,*to;
  MSKrealt            *scalcon,*scalvar,*cc,*cx,*blc,*buc,*blx,*bux,
                      *xc,*xx,*y,*slc,*suc,*slx,*sux;
  MSKboundkeye        *bkc,*bkx;
  MSKstakeye          *skc,*skx;
  MSKprostae          prosta;
  MSKsolstae          solsta;
  int                 i,j,k,*rmap,*cmap,hotstart=0;
```

```
58  char                filename[1024];
59
60  if ( argc<2 )
61  {
62    printf("No input file specified\n");
63    exit(0);
64  }
65  else
66    printf("Inputfile:  %s\n",argv[1]);
67
68  r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
69
70  if ( r==MSK_RES_OK )
71    MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printlog);
72
73  if ( r==MSK_RES_OK )
74    r = MSK_initenv(env);
75
76  if ( r==MSK_RES_OK )
77  {
78    strcpy(filename,argv[1]);
79    addext(filename,"log");
80
81    /* Create an (empty) optimization task. */
82    r = MSK_maketask(env,0,0,&task);
83
84    if ( r==MSK_RES_OK )
85    {
86      MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL,     printlog);
87      MSK_linkfiletotaskstream(task, MSK_STREAM_LOG, filename, 0);
88    }
89
90    if ( r==MSK_RES_OK )
91    {
92     r = MSK_readdata(task,argv[1]);
93    }
94  }
95
96  if ( r==MSK_RES_OK )
97  {
98    r = MSK_getnumcon(task,&numcon);
99  }
100
101 if ( r==MSK_RES_OK )
102 {
103   r = MSK_getnumvar(task,&numvar);
104 }
105
106 if ( r==MSK_RES_OK )
107 {
108   /* Create an (empty) optimization task. Will be used as a dummy task
109      in MSK_netoptimize, parameters can be set here */
110   r = MSK_maketask(env,0,0,&dummytask);
111 }
112
113 if ( r==MSK_RES_OK )
114 {
115   MSK_linkfunctotaskstream(dummytask, MSK_STREAM_LOG, NULL,     printlog);
```

```
116     MSK_linkfiletotaskstream(dummytask, MSK_STREAM_LOG, filename, 0);
117   }
118
119   /* Allocate memory for embedded network (maximum sizez => (numcon and numvar) */
120   if ( r==MSK_RES_OK )
121   {
122     /* Sizes of the embedded network structure is unknown use maximum size
123        required  by MSK_networkextraction */
124     rmap     = MSK_calloctask(dummytask,numcon,sizeof(int));
125     cmap     = MSK_calloctask(dummytask,numvar,sizeof(int));
126     netcon   = MSK_calloctask(dummytask,numcon,sizeof(MSKidxt));
127     netvar   = MSK_calloctask(dummytask,numvar,sizeof(MSKidxt));
128     from     = MSK_calloctask(dummytask,numvar,sizeof(MSKidxt));
129     to       = MSK_calloctask(dummytask,numvar,sizeof(MSKidxt));
130
131     scalcon  = MSK_calloctask(dummytask,numcon,sizeof(MSKrealt));
132     scalvar  = MSK_calloctask(dummytask,numvar,sizeof(MSKrealt));
133     cc       = MSK_calloctask(dummytask,numcon,sizeof(MSKrealt));
134     cx       = MSK_calloctask(dummytask,numvar,sizeof(MSKrealt));
135     blc      = MSK_calloctask(dummytask,numcon,sizeof(MSKrealt));
136     buc      = MSK_calloctask(dummytask,numcon,sizeof(MSKrealt));
137     blx      = MSK_calloctask(dummytask,numvar,sizeof(MSKrealt));
138     bux      = MSK_calloctask(dummytask,numvar,sizeof(MSKrealt));
139     xx       = MSK_calloctask(dummytask,numvar,sizeof(MSKrealt));
140     xc       = MSK_calloctask(dummytask,numcon,sizeof(MSKrealt));
141     y        = MSK_calloctask(dummytask,numcon,sizeof(MSKrealt));
142     slc      = MSK_calloctask(dummytask,numcon,sizeof(MSKrealt));
143     suc      = MSK_calloctask(dummytask,numcon,sizeof(MSKrealt));
144     slx      = MSK_calloctask(dummytask,numvar,sizeof(MSKrealt));
145     sux      = MSK_calloctask(dummytask,numvar,sizeof(MSKrealt));
146
147     bkc      = MSK_calloctask(dummytask,numcon,sizeof(MSKboundkeye));
148     bkx      = MSK_calloctask(dummytask,numvar,sizeof(MSKboundkeye));
149
150     skc      = MSK_calloctask(dummytask,numvar,sizeof(MSKstakeye));
151     skx      = MSK_calloctask(dummytask,numvar,sizeof(MSKstakeye));
152
153     if( !( rmap    && cmap    && netcon && netvar && from && to  &&
154            scalcon && scalvar && cc     && cx     && blc  && buc &&
155            blx     && bux     && xx     && xc     && y    && slc &&
156            suc     && slx     && sux    && bkc    && bkx  && skc &&
157            skx ) )
158     {
159       r = MSK_RES_ERR_SPACE;
160     }
161     else
162     {
163       /* We just use zero cost on slacks */
164       for( i = 0; i < numcon; ++i )
165         cc[i] = 0.0;
166     }
167   }
168
169   if ( r==MSK_RES_OK )
170   {
171     /* Extract embedded network */
172     r = MSK_netextraction(task,
173                           &numnetcon,
```

```
                          &numnetvar,
                          netcon,
                          netvar,
                          scalcon,
                          scalvar,
                          cx,
                          bkc,
                          blc,
                          buc,
                          bkx,
                          blx,
                          bux,
                          from,
                          to);

    MSK_deletetask(&task);
  }

  if ( r==MSK_RES_OK )
  {
    /* Solve embedded network with a direct call into the network optimizer */
    r =  MSK_netoptimize(dummytask,
                         numnetcon,
                         numnetvar,
                         cc,
                         cx,
                         bkc,
                         blc,
                         buc,
                         bkx,
                         blx,
                         bux,
                         from,
                         to,
                         &prosta,
                         &solsta,
                         hotstart,
                         skc,
                         skx,
                         xc,
                         xx,
                         y,
                         slc,
                         suc,
                         slx,
                         sux);

    if ( solsta == MSK_SOL_STA_OPTIMAL )
    {
      printf("Embedded network problem is optimal\n");
    }
    else if ( solsta == MSK_SOL_STA_PRIM_INFEAS_CER )
    {
      printf("Embedded network problem is primal infeasible\n");
    }
    else if ( solsta == MSK_SOL_STA_DUAL_INFEAS_CER )
    {
      printf("Embedded network problem is dual infeasible\n");
```

```
232       }
233       else
234       {
235         printf("Embedded network problem solsta : %d\n",solsta);
236       }
237     }
238
239     /* Free allocated memory */
240     MSK_freetask(dummytask,rmap);
241     MSK_freetask(dummytask,cmap);
242     MSK_freetask(dummytask,netcon);
243     MSK_freetask(dummytask,netvar);
244     MSK_freetask(dummytask,from);
245     MSK_freetask(dummytask,to);
246     MSK_freetask(dummytask,scalcon);
247     MSK_freetask(dummytask,scalvar);
248     MSK_freetask(dummytask,cc);
249     MSK_freetask(dummytask,cx);
250     MSK_freetask(dummytask,blc);
251     MSK_freetask(dummytask,buc);
252     MSK_freetask(dummytask,blx);
253     MSK_freetask(dummytask,bux);
254     MSK_freetask(dummytask,xx);
255     MSK_freetask(dummytask,xc);
256     MSK_freetask(dummytask,y);
257     MSK_freetask(dummytask,slc);
258     MSK_freetask(dummytask,suc);
259     MSK_freetask(dummytask,slx);
260     MSK_freetask(dummytask,sux);
261     MSK_freetask(dummytask,bkc);
262     MSK_freetask(dummytask,bkx);
263     MSK_freetask(dummytask,skc);
264     MSK_freetask(dummytask,skx);
265
266     MSK_deletetask(&dummytask);
267     MSK_deleteenv(&env);
268   }
```

In the above example we only optimize the embedded network problem. We still need to use the found network solution as a hot-start for the simplex optimizer and solve the original problem. This involves unscaling the network solution back to same unit measure as the original problem. In the example

```
mosek\6\tools\examples\c\network3.c
```

we show how to convert the network solution into a valid hot-start for the simplex optimizer.

6.7 Solving linear systems involving the basis matrix

A linear optimization problem always has an optimal solution which is also a basic solution. In an optimal basic solution there are exactly m basic variables where m is the number of rows in the constraint matrix A. Define

$$B \in \mathbb{R}^{m \times m}$$

as a matrix consisting of the columns of A corresponding to the basic variables.

The basis matrix B is always non-singular, i.e.

$$\det(B) \neq 0$$

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

$$B\bar{x} = w \tag{6.20}$$

and

$$B^T\bar{x} = w \tag{6.21}$$

each has a unique solution for all w.

MOSEK provides functions for solving the linear systems (6.20) and (6.21) for an arbitrary w.

6.7.1 Identifying the basis

To use the solutions to (6.20) and (6.21) it is important to know how the basis matrix B is constructed.

Internally MOSEK employs the linear optimization problem

$$\begin{array}{lccccc} \text{maximize} & & & c^T x & & \\ \text{subject to} & & & Ax - x^c & = & 0 \\ & l^x & \leq & x & \leq & u^x, \\ & l^c & \leq & x^c & \leq & u^c. \end{array} \tag{6.22}$$

where

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

The basis matrix is constructed of m columns taken from

$$[\ A \quad -I\ ].$$

If variable x_j is a basis variable, then the j'th column of A denoted $a_{:,j}$ will appear in B. Similarly, if x^c_i is a basis variable, then the i'th column of $-I$ will appear in the basis. The ordering of the basis variables and therefore the ordering of the columns of B is arbitrary. The ordering of the basis variables may be retrieved by calling the function:

```
MSK_initbasissolve (MSKtask_t task
                    MSKidxt   *basis);
```

This function initializes data structures for later use and returns the indexes of the basic variables in the array `basis`. The interpretation of the `basis` is as follows. If

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

then the i'th basis variable is x^c_i. Moreover, the i'th column in B will be the i'th column of $-I$. On the other hand if

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

then the i'th basis variable is variable

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

and the i'th column of B is the column

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

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

6.7.2 An example

Consider the linear optimization problem:

$$\begin{array}{lrcl} \text{minimize} & x_0 + x_1 & & \\ \text{subject to} & x_0 + 2x_1 & \leq & 2, \\ & x_0 + x_1 & \leq & 6, \\ & x_0, x_1 \geq 0. & & \end{array} \tag{6.23}$$

Suppose a call to MSK_initbasissolve returns an array `basis` so that

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

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

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix}. \tag{6.24}$$

Please note the ordering of the columns in B.

The following program demonstrates the use of MSK_solvewithbasis.

```
/*
  Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

  File:      solvebasis.c

  Purpose:   To demonstrate the usage of
             MSK_solvewithbasis on the problem:

             maximize  x0 + x1
             st.
                     x0 + 2.0 x1 <= 2
                     x0  +    x1 <= 6
                     x0 >= 0, x1>= 0

               The problem has the slack variables
               xc0, xc1 on the constraints
               and the variables x0 and x1.

```

```
19                maximize  x0 + x1
20                st.
21                   x0 + 2.0 x1 -xc1       = 2
22                   x0  +    x1       -xc2 = 6
23                   x0 >= 0, x1>= 0,
24                   xc1 <=  0 , xc2 <= 0
25
26
27             problem data is read from basissolve.lp.
28
29   Syntax:    solvebasis basissolve.lp
30
31  */
32 #include "mosek.h"
33
34 static void MSKAPI printstr(void *handle,
35                             char str[])
36 {
37   printf("%s",str);
38 } /* printstr */
39
40 int main(int argc,char **argv)
41 {
42   MSKenv_t  env;
43   MSKtask_t task;
44   MSKintt NUMCON = 2;
45   MSKintt NUMVAR = 2;
46
47   double       c[]    = {1.0, 1.0};
48   MSKintt      ptrb[] = {0, 2};
49   MSKintt      ptre[] = {2, 3};
50   MSKidxt      asub[] = {0, 1,
51                          0, 1};
52   double aval[] = {1.0, 1.0,
53                    2.0, 1.0};
54   MSKboundkeye bkc[]  = {MSK_BK_UP,
55                          MSK_BK_UP};
56
57   double blc[]  = {-MSK_INFINITY,
58                    -MSK_INFINITY};
59   double buc[]  = {2.0,
60                    6.0};
61
62   MSKboundkeye  bkx[]  = {MSK_BK_LO,
63                           MSK_BK_LO};
64   double  blx[]  = {0.0,
65                     0.0};
66
67   double  bux[]  = {+MSK_INFINITY,
68                     +MSK_INFINITY};
69
70
71   MSKrescodee       r = MSK_RES_OK;
72   MSKidxt       i,nz;
73   double    w1[] = {2.0,6.0};
74   double    w2[] = {1.0,0.0};
75   MSKidxt   sub[] = {0,1};
76   MSKidxt   *basis;
```

```
77
78    if (r == MSK_RES_OK)
79      r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
80
81    if ( r==MSK_RES_OK )
82      MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
83
84    if ( r==MSK_RES_OK )
85      r = MSK_initenv(env);
86
87    if ( r==MSK_RES_OK )
88      r = MSK_makeemptytask(env,&task);
89
90    if ( r==MSK_RES_OK )
91        MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
92
93    if ( r == MSK_RES_OK)
94      r = MSK_inputdata(task, NUMCON,NUMVAR, NUMCON,NUMVAR, c, 0.0,
95                        ptrb, ptre, asub, aval, bkc, blc, buc, bkx, blx, bux);
96
97    if (r == MSK_RES_OK)
98      r = MSK_putobjsense(task,MSK_OBJECTIVE_SENSE_MAXIMIZE);
99
100
101
102   if (r == MSK_RES_OK)
103     r = MSK_optimize(task);
104
105   if (r == MSK_RES_OK)
106     basis = MSK_calloctask(task,NUMCON,sizeof(MSKidxt));
107
108   if (r == MSK_RES_OK)
109     r = MSK_initbasissolve(task,basis);
110
111   /* List basis variables corresponding to columns of B */
112   for (i=0;i<NUMCON && r == MSK_RES_OK;++i)
113   {
114     printf("basis[%d] = %d\n",i,basis[i]);
115     if (basis[sub[i]] < NUMCON)
116       printf ("Basis variable no %d is xc%d.\n",i, basis[i]);
117     else
118       printf ("Basis variable no %d is x%d.\n",i,basis[i] - NUMCON);
119   }
120
121   nz = 2;
122   /* solve Bx = w1 */
123   /* sub contains index of non-zeros in w1.
124      On return w1 contains the solution x and sub
125      the index of the non-zeros in x.
126    */
127   if (r == MSK_RES_OK)
128     r = MSK_solvewithbasis(task,0,&nz,sub,w1);
129
130   if (r == MSK_RES_OK)
131   {
132     printf("\nSolution to Bx = w1:\n\n");
133
134     /* Print solution and b. */
```

```
135
136     for (i=0;i<nz;++i)
137     {
138       if (basis[sub[i]] < NUMCON)
139         printf ("xc%d = %e\n",basis[sub[i]] , w1[sub[i]] );
140       else
141         printf ("x%d = %e\n",basis[sub[i]] - NUMCON , w1[sub[i]] );
142     }
143   }
144     /* Solve B^Tx = c */
145   nz = 2;
146   sub[0] = 0;
147   sub[1] = 1;
148
149   if (r == MSK_RES_OK)
150     r = MSK_solvewithbasis(task,1,&nz,sub,w2);
151
152   if (r == MSK_RES_OK)
153   {
154     printf("\nSolution to B^Tx = w2:\n\n");
155     /* Print solution and y. */
156     for (i=0;i<nz;++i)
157     {
158       if (basis[sub[i]] < NUMCON)
159         printf ("xc%d = %e\n",basis[sub[i]] , w2[sub[i]] );
160       else
161         printf ("x%d = %e\n",basis[sub[i]] - NUMCON , w2[sub[i]] );
162     }
163   }
164
165    printf("Return code: %d (0 means no error occurred.)\n",r);
166
167    return ( r );
168
169 }/* main */
```

In the example above the linear system is solved using the optimal basis for (6.23) and the original right-hand side of the problem. Thus the solution to the linear system is the optimal solution to the problem. When running the example program the following output is produced.

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

Solution to Bx = b:

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

Solution to B^Tx = c:

x1 = -1.000000e+00
```

```
x0 = 1.000000e+00
```

Please note that the ordering of the basis variables is

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

and thus the basis is given by:

$$B = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \tag{6.25}$$

It can be verified that

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

is a solution to

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

6.7.3 Solving arbitrary linear systems

MOSEK can be used to solve an arbitrary (rectangular) linear system

$$Ax = b$$

using the MSK_solvewithbasis function without optimizing the problem as in the previous example. This is done by setting up an A matrix in the task, setting all variables to basic and calling the MSK_solvewithbasis function with the b vector as input. The solution is returned by the function.

Below we demonstrate how to solve the linear system

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{6.26}$$

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

```
1  /*
2     Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3
4     File      :  solvelinear.c
5
6     Purpose   :  To demonstrate the usage of MSK_solvewithbasis
7                  to solve the linear system:
8
9                   1.0  x1             = b1
10                 -1.0  x0  +  1.0  x1 = b2
11
12                 with two different right hand sides
13
14                 b = (1.0, -2.0)
15
16                 and
17
```

```
18                b = (7.0, 0.0)
19  */
20
21 #include "mosek.h"
22
23 static void MSKAPI printstr(void *handle,
24                             char str[])
25 {
26   printf("%s",str);
27 } /* printstr */
28
29
30 MSKrescodee put_a(MSKtask_t task,
31                   double *aval,
32                   MSKidxt *asub,
33                   MSKidxt *ptrb,
34                   MSKidxt *ptre,
35                   int numvar,
36                  MSKidxt *basis
37                  )
38
39 {
40   MSKrescodee r = MSK_RES_OK;
41   int i;
42   MSKstakeye *skx = NULL , *skc = NULL;
43
44
45   skx = (MSKstakeye *) calloc(numvar,sizeof(MSKstakeye));
46   if (skx == NULL && numvar)
47     r = MSK_RES_ERR_SPACE;
48
49   skc = (MSKstakeye *) calloc(numvar,sizeof(MSKstakeye));
50   if (skc == NULL && numvar)
51     r = MSK_RES_ERR_SPACE;
52
53   for (i=0;i<numvar && r == MSK_RES_OK;++i)
54   {
55     skx[i] = MSK_SK_BAS;
56     skc[i] = MSK_SK_FIX;
57   }
58
59
60   /* Create a coefficient matrix and right hand
61      side with the data from the linear system */
62   if (r == MSK_RES_OK)
63     r = MSK_append(task,MSK_ACC_VAR,numvar);
64
65   if (r == MSK_RES_OK)
66     r = MSK_append(task,MSK_ACC_CON,numvar);
67
68   for (i=0;i<numvar && r == MSK_RES_OK;++i)
69     r = MSK_putavec(task,MSK_ACC_VAR,i,ptre[i]-ptrb[i],asub+ptrb[i],aval+ptrb[i]);
70
71   for (i=0;i<numvar && r == MSK_RES_OK;++i)
72     r = MSK_putbound(task,MSK_ACC_CON,i,MSK_BK_FX,0,0);
73
74   for (i=0;i<numvar && r == MSK_RES_OK;++i)
75     r = MSK_putbound(task,MSK_ACC_VAR,i,MSK_BK_FR,-MSK_INFINITY,MSK_INFINITY);
```

```
76
77    /* Allocate space for the solution and set status to unknown */
78
79    if (r == MSK_RES_OK)
80    {
81      r = MSK_makesolutionstatusunknown(task, MSK_SOL_BAS);
82    }
83
84    /* Define a basic solution by specifying
85       status keys for variables & constraints. */
86    for (i=0; i<numvar && r==MSK_RES_OK;++i)
87        r = MSK_putsolutioni (
88                              task,
89                              MSK_ACC_VAR,
90                              i,
91                              MSK_SOL_BAS,
92                              skx[i],
93                              0.0,
94                              0.0,
95                              0.0,
96                              0.0);
97
98    for (i=0;i<numvar && r == MSK_RES_OK;++i)
99        r = MSK_putsolutioni (
100                             task,
101                             MSK_ACC_CON,
102                             i,
103                             MSK_SOL_BAS,
104                             skc[i],
105                             0.0,
106                             0.0,
107                             0.0,
108                             0.0);
109
110
111   if (r == MSK_RES_OK)
112     r = MSK_initbasissolve(task,basis);
113
114   free (skx);
115   free (skc);
116
117   return ( r );
118
119 }
120
121 #define NUMCON 2
122 #define NUMVAR 2
123
124
125 int main(int argc,char **argv)
126 {
127   MSKenv_t  env;
128   MSKtask_t task;
129   MSKrescodee r = MSK_RES_OK;
130   MSKintt   numvar = NUMCON;
131   MSKintt   numcon = NUMVAR;   /* we must have numvar == numcon */
132   int       i,nz;
133   double    aval[] = {-1.0,1.0,1.0};
```

```
134   MSKidxt   asub[] = {1,0,1};
135   MSKidxt   ptrb[] = {0,1};
136   MSKidxt   ptre[] = {1,3};
137
138   MSKidxt   bsub[NUMCON];
139   double    b[NUMCON];
140
141   MSKidxt   *basis = NULL;
142
143   if (r == MSK_RES_OK)
144     r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
145
146   if ( r==MSK_RES_OK )
147     MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
148
149   if ( r==MSK_RES_OK )
150     r = MSK_initenv(env);
151
152   if ( r==MSK_RES_OK )
153     r = MSK_makeemptytask(env,&task);
154
155   if ( r==MSK_RES_OK )
156       MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
157
158   basis = (MSKidxt *) calloc(numcon,sizeof(MSKidxt));
159   if ( basis == NULL && numvar)
160     r = MSK_RES_ERR_SPACE;
161
162
163   /* Put A matrix and factor A.
164      Call this function only once for a given task. */
165   if (r == MSK_RES_OK)
166     r = put_a( task,
167                aval,
168                asub,
169                ptrb,
170                ptre,
171                numvar,
172                basis
173                );
174
175   /* now solve rhs */
176   b[0] = 1;
177   b[1] = -2;
178   bsub[0] = 0;
179   bsub[1] = 1;
180   nz = 2;
181
182   if (r == MSK_RES_OK)
183     r = MSK_solvewithbasis(task,0,&nz,bsub,b);
184
185   if (r == MSK_RES_OK)
186   {
187     printf("\nSolution to Bx = b:\n\n");
188     /* Print solution and show correspondents
189        to original variables in the problem */
190     for (i=0;i<nz;++i)
191     {
```

```
192       if (basis[bsub[i]] < numcon)
193         printf("This should never happen\n");
194       else
195         printf ("x%d = %e\n",basis[bsub[i]] - numcon , b[bsub[i]] );
196     }
197   }
198
199   b[0] = 7;
200   bsub[0] = 0;
201   nz = 1;
202
203   if (r == MSK_RES_OK)
204     r = MSK_solvewithbasis(task,0,&nz,bsub,b);
205
206   if (r == MSK_RES_OK)
207   {
208     printf("\nSolution to Bx = b:\n\n");
209     /* Print solution and show correspondents
210        to original variables in the problem */
211     for (i=0;i<nz;++i)
212     {
213       if (basis[bsub[i]] < numcon)
214         printf("This should never happen\n");
215       else
216         printf ("x%d = %e\n",basis[bsub[i]] - numcon , b[bsub[i]] );
217     }
218   }
219
220   free (basis);
221   return r;
222 }
```

The most important step in the above example is the definition of the basic solution using the MSK_putsolutioni function, where we define the status key for each variable. The actual values of the variables are not important and can be selected arbitrarily, so we set them to zero. All variables corresponding to columns in the linear system we want to solve are set to basic and the slack variables for the constraints, which are all non-basic, are set to their bound.

The program produces the output:

```
Solution to Bx = b:

x1 = 1
x0 = 3

Solution to Bx = b:

x1 = 7
x0 = 7
```

and we can verify that $x_0 = 2, x_1 = -4$ is indeed a solution to (6.26).

6.8 The progress call-back

Some of the API function calls, notably MSK_optimize, may take a long time to complete. Therefore, during the optimization a call-back function is called frequently. From the call-back function it is possible

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

6.8.1 Source code example

The following source code example documents how the progress call-back function can be used.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

   File:      callback.c

   Purpose:   To demonstrate how to use the progress
              callback.

              Compile and link the file with  MOSE, then
              it is used as follows:

              callback psim 25fv47.mps
              callback dsim 25fv47.mps
              callback intpnt 25fv47.mps

              The first argument tells which optimizer to use
              i.e. psim is primal simplex, dsim is dual simplex
              and intpnt is interior-point.
 */


#include "mosek.h"


/* Note: This function is declared using MSKAPI,
         so the correct calling convention is
         employed. */
static int MSKAPI usercallback(MSKtask_t        task,
                               MSKuserhandle_t  handle,
                               MSKcallbackcodee caller)
{
  int    iter;
  double pobj,dobj,opttime=0.0,stime=0.0,
         *maxtime=(double *) handle;
```

```
39
40    switch ( caller )
41    {
42      case MSK_CALLBACK_BEGIN_INTPNT:
43        printf("Starting interior-point optimizer\n");
44        break;
45      case MSK_CALLBACK_INTPNT:
46        MSK_getintinf(task,
47                      MSK_IINF_INTPNT_ITER,
48                      &iter);
49        MSK_getdouinf(task,
50                      MSK_DINF_INTPNT_PRIMAL_OBJ,
51                      &pobj);
52        MSK_getdouinf(task,
53                      MSK_DINF_INTPNT_DUAL_OBJ,
54                      &dobj);
55        MSK_getdouinf(task,
56                      MSK_DINF_INTPNT_TIME,
57                      &stime);
58        MSK_getdouinf(task,
59                      MSK_DINF_OPTIMIZER_TIME,
60                      &opttime);
61
62        printf("Iterations: %-3d  Time: %6.2f(%.2f)  ",
63               iter,opttime,stime);
64        printf("Primal obj.: %-18.6e  Dual obj.: %-18.6e\n",
65                pobj,dobj);
66        break;
67      case MSK_CALLBACK_END_INTPNT:
68        printf("Interior-point optimizer finished.\n");
69        break;
70      case MSK_CALLBACK_BEGIN_PRIMAL_SIMPLEX:
71        printf("Primal simplex optimizer started.\n");
72        break;
73      case MSK_CALLBACK_UPDATE_PRIMAL_SIMPLEX:
74        MSK_getintinf(task,
75                      MSK_IINF_SIM_PRIMAL_ITER,
76                      &iter);
77        MSK_getdouinf(task,
78                      MSK_DINF_SIM_OBJ,
79                      &pobj);
80        MSK_getdouinf(task,
81                      MSK_DINF_SIM_TIME,
82                      &stime);
83        MSK_getdouinf(task,
84                      MSK_DINF_OPTIMIZER_TIME,
85                      &opttime);
86
87        printf("Iterations: %-3d  ",iter);
88        printf("  Elapsed time: %6.2f(%.2f)\n",
89               opttime,stime);
90        printf("Obj.: %-18.6e\n",pobj);
91        break;
92      case MSK_CALLBACK_END_PRIMAL_SIMPLEX:
93        printf("Primal simplex optimizer finished.\n");
94        break;
95      case MSK_CALLBACK_BEGIN_DUAL_SIMPLEX:
96        printf("Dual simplex optimizer started.\n");
```

```
      break;
    case MSK_CALLBACK_UPDATE_DUAL_SIMPLEX:
      MSK_getintinf(task,
                    MSK_IINF_SIM_DUAL_ITER,
                    &iter);
      MSK_getdouinf(task,
                    MSK_DINF_SIM_OBJ,
                    &pobj);
      MSK_getdouinf(task,
                    MSK_DINF_SIM_TIME,
                    &stime);
      MSK_getdouinf(task,
                    MSK_DINF_OPTIMIZER_TIME,
                    &opttime);

      printf("Iterations: %-3d  ",iter);
      printf("  Elapsed time: %6.2f(%.2f)\n",
             opttime,stime);
      printf("Obj.: %-18.6e\n",pobj);
      break;
    case MSK_CALLBACK_END_DUAL_SIMPLEX:
      printf("Dual simplex optimizer finished.\n");
      break;
    case MSK_CALLBACK_BEGIN_BI:
      printf("Basis identification started.\n");
      break;
    case MSK_CALLBACK_END_BI:
      printf("Basis identification finished.\n");
      break;
  }

  if ( opttime>=maxtime[0] )
  {
    /* mosek is spending too much time.
       Terminate it. */
    return ( 1 );
  }

  return ( 0 );
} /* usercallback */

static void MSKAPI printtxt(void *info,
                            char *buffer)
{
  printf("%s",buffer);
} /* printtxt */

int main(int argc, char *argv[])
{
  double    maxtime,
            *xx,*y;
  int       r,j,i,numcon,numvar;
  FILE      *f;
  MSKenv_t  env;
  MSKtask_t task;

  if ( argc<3 )
  {
```

```
155     printf("Too few input arguments. mosek intpnt myfile.mps\n");
156     exit(0);
157   }
158
159   /*
160    * It is assumed that we are working in a
161    * windows environment.
162    */
163
164   /* Create mosek environment. */
165   r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
166
167   /* Check the return code. */
168   if ( r==MSK_RES_OK )
169     r = MSK_initenv(env);
170
171   /* Check the return code. */
172   if ( r==MSK_RES_OK )
173   {
174     /* Create an (empty) optimization task. */
175     r = MSK_makeemptytask(env,&task);
176
177     if ( r==MSK_RES_OK )
178     {
179       MSK_linkfunctotaskstream(task,MSK_STREAM_MSG,NULL, printtxt);
180       MSK_linkfunctotaskstream(task,MSK_STREAM_ERR,NULL, printtxt);
181     }
182
183     /* Specifies that data should be read from the
184        file argv[2].
185      */
186
187     if ( r==MSK_RES_OK )
188       r = MSK_readdata(task,argv[2]);
189
190     if ( r==MSK_RES_OK )
191     {
192       if ( 0==strcmp(argv[1],"psim") )
193         MSK_putintparam(task,MSK_IPAR_OPTIMIZER,MSK_OPTIMIZER_PRIMAL_SIMPLEX);
194       else  if ( 0==strcmp(argv[1],"dsim") )
195         MSK_putintparam(task,MSK_IPAR_OPTIMIZER,MSK_OPTIMIZER_DUAL_SIMPLEX);
196       else  if ( 0==strcmp(argv[1],"intpnt") )
197         MSK_putintparam(task,MSK_IPAR_OPTIMIZER,MSK_OPTIMIZER_INTPNT);
198
199
200       /* Tell mosek about the call-back function. */
201       maxtime = 3600;
202       MSK_putcallbackfunc(task,
203                           usercallback,
204                           (void *) &maxtime);
205
206       /* Turn all MOSEK logging off. */
207       MSK_putintparam(task,
208                       MSK_IPAR_LOG,
209                       0);
210
211       r = MSK_optimize(task);
212
```

```
      MSK_solutionsummary(task,MSK_STREAM_MSG);
    }


    MSK_deletetask(&task);
  }
  MSK_deleteenv(&env);

  printf("Return code - %d\n",r);

  return ( r );
} /* main */
```

6.9 Customizing the warning and error reporting

You can customize the warning and error reporting in the C API. The MSK_putresponsefunc function can be used to register a user-defined function to be called every time a warning or an error is encountered by MOSEK. This user-defined function will then handle the error/warning as desired.

The following code shows how to define and register an error handling function:

```
/*
   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

   File: errorreporting.c

   Purpose:   To demonstrate how the error reporting can be customized.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#include "mosek.h"

MSKrescodee MSKAPI handleresponse(MSKuserhandle_t handle,
                                  MSKrescodee     r,
                                  MSKCONST char   msg[])
/* A custom response handler. */
{
  if ( r==MSK_RES_OK )
  {
    /* Do nothing */
  }
  else if ( r<MSK_FIRST_ERR_CODE )
  {
    printf("MOSEK reports warning number %d: %s\n",r,msg);
  }
  else
  {
    printf("MOSEK reports error number %d: %s\n",r,msg);
  }

```

```
34    return ( MSK_RES_OK );
35
36  } /* handlerespone */
37
38
39  int main(int argc, char *argv[])
40  {
41    MSKenv_t    env;
42    MSKrescodee r;
43    MSKtask_t   task;
44
45    r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
46
47    if ( r==MSK_RES_OK )
48      r = MSK_initenv(env);
49
50    if ( r==MSK_RES_OK )
51    {
52
53      r = MSK_makeemptytask(env,&task);
54
55      if ( r==MSK_RES_OK )
56      {
57        /*
58         * Input a custom warning and error handler function.
59         */
60
61        MSK_putresponsefunc(task,handleresponse,NULL);
62
63        /* User defined code goes here */
64        /* This will provoke an error */
65
66        if (r == MSK_RES_OK)
67          r = MSK_putaij(task,10,10,1.0);
68
69      }
70      MSK_deletetask(&task);
71    }
72    MSK_deleteenv(&env);
73
74    printf("Return code - %d\n",r);
75
76    if (r == MSK_RES_ERR_INDEX_IS_TOO_LARGE)
77      return ( MSK_RES_OK);
78    else
79      return (-1);
80  } /* main */
```

The output from the code above is:

```
  MOSEK reports error number 1204: The index value 10 occurring in argument 'i' is too large.
Return code - 1204
```

6.10 Unicode strings

All strings i.e. `char *` in the `C` API are assumed to be UTF8 strings. Please note that

- an ASCII string is always a valid UTF8 string, and
- an UTF8 string is stored in an array of chars.

For more information about UTF8 encoded strings, please see http://en.wikipedia.org/wiki/UTF-8.

It is possible to convert a `wchar_t` string to a UTF8 string using the function `MSK_wchartoutf8`. The inverse function `MSK_utf8towchar` converts a UTF8 string to a `wchar_t` string.

6.10.1 A source code example

The example below documents how to convert a `wchar_t` string to a UTF8 string.

```
/*
   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

   File: unicode.c

   Purpose:   To demonstrate how to use a unicoded strings.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#include "mosek.h"

int main(int argc, char *argv[])
{
  char         output[512];
  wchar_t      *input=L"myfile.mps";
  MSKenv_t     env;
  MSKrescodee r;
  MSKtask_t    task;
  size_t       len,conv;


  r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);

  if ( r==MSK_RES_OK )
    r = MSK_initenv(env);

  if ( r==MSK_RES_OK )
  {
    r = MSK_makeemptytask(env,&task);

    if ( r==MSK_RES_OK )
    {
      /*
```

```
38          The wchar_t string "input" specifying a file name
39          is converted to a UTF8 string that can be inputted
40          to MOSEK.
41        */
42
43       r = MSK_wchartoutf8(sizeof(output),&len,&conv,output,input);
44
45       if ( r==MSK_RES_OK )
46       {
47         /* output is now an UTF8 encoded string. */
48         r = MSK_readdata(task,output);
49       }
50
51       if ( r==MSK_RES_OK )
52       {
53         r = MSK_optimize(task);
54         MSK_solutionsummary(task,MSK_STREAM_MSG);
55       }
56     }
57     MSK_deletetask(&task);
58   }
59   MSK_deleteenv(&env);
60
61   printf("Return code - %d\n",r);
62
63   return ( r );
64 } /* main */
```

6.10.2 Limitations

Please note that the MPS and LP format are based ASCII formats whereas the OPF, MBT, and XML are UTF8 based formats. This implies that problems which contains non-ASCII variable or constraint names cannot be written correctly to an MPS or LP formatted file.

Chapter 7

Modelling

In this chapter we will discuss the following issues:

- The formal definitions of the problem types that MOSEK can solve.
- The solution information produced by MOSEK.
- The information produced by MOSEK if the problem is infeasible.
- A set of examples showing different ways of formulating commonly occurring problems so that they can be solved by MOSEK.
- Recommendations for formulating optimization problems.

7.1 Linear optimization

A linear optimization problem can be written as

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \tag{7.1}$$

where

- m is the number of constraints.
- n is the number of decision variables.
- $x \in \mathbb{R}^n$ is a vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear part of the objective function.
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.

- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.

A primal solution (x) is *(primal) feasible* if it satisfies all constraints in (7.1). If (7.1) has at least one primal feasible solution, then (7.1) is said to be (primal) feasible.

In case (7.1) does not have a feasible solution, the problem is said to be *(primal) infeasible*.

7.1.1 Duality for linear optimization

Corresponding to the primal problem (7.1), there is a dual problem

$$\begin{array}{lcll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & \end{array} \tag{7.2}$$

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

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

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

A solution

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

to the dual problem is feasible if it satisfies all the constraints in (7.2). If (7.2) has at least one feasible solution, then (7.2) is *(dual) feasible*, otherwise the problem is *(dual) infeasible*.

We will denote a solution

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

so that x is a solution to the primal problem (7.1), and

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

is a solution to the corresponding dual problem (7.2). A solution which is both primal and dual feasible is denoted a *primal-dual feasible solution*.

7.1.1.1 A primal-dual feasible solution

Let

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

be a primal-dual feasible solution, and let

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

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

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

where the first relation can be obtained by multiplying the dual constraints (7.2) by x and x^c respectively, and the second relation comes from the fact that each term in each sum is nonnegative. It follows that the primal objective will always be greater than or equal to the dual objective.

We then define the *duality gap* as the difference between the primal objective value and the dual objective value, i.e.

$$c^T x^* + c^f - ((l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f)$$

Please note that the duality gap will always be nonnegative.

7.1.1.2 An optimal solution

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

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

are satisfied.

If (7.1) 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.

7.1.1.3 Primal infeasible problems

If the problem (7.1) is infeasible (has no feasible solution), MOSEK will report a certificate of primal infeasibility: The dual solution reported is a certificate of infeasibility, and the primal solution is undefined.

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

$$\begin{array}{lcll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & 0, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & \end{array} \tag{7.3}$$

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

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

to (7.3) so that

$$(l^c)^T(s_l^c)^* - (u^c)^T(s_u^c)^* + (l^x)^T(s_l^x)^* - (u^x)^T(s_u^x)^* > 0.$$

Such a solution implies that (7.3) is unbounded, and that its dual is infeasible.

We note that the dual of (7.3) is a problem which constraints are identical to the constraints of the original primal problem (7.1): If the dual of (7.3) is infeasible, so is the original primal problem.

7.1.1.4 Dual infeasible problems

If the problem (7.2) is infeasible (has no feasible solution), MOSEK will report a certificate of dual infeasibility: The primal solution reported is a certificate of infeasibility, and the dual solution is undefined.

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

$$\begin{array}{lccccc} \text{minimize} & & & c^T x & & \\ \text{subject to} & & & Ax - x^c & = & 0, \\ & \bar{l}^c & \leq & x^c & \leq & \bar{u}^c, \\ & \bar{l}^x & \leq & x & \leq & \bar{u}^x \end{array} \tag{7.4}$$

where

$$\bar{l}_i^c = \begin{cases} 0, & \text{if } l_i^c > -\infty, \\ -\infty & \text{otherwise} \end{cases} \quad \text{and} \quad \bar{u}_i^c := \begin{cases} 0, & \text{if } u_i^c < \infty, \\ \infty & \text{otherwise} \end{cases}$$

and

$$\bar{l}_j^x = \begin{cases} 0, & \text{if } l_j^x > -\infty, \\ -\infty & \text{otherwise} \end{cases} \quad \text{and} \quad \bar{u}_j^x := \begin{cases} 0, & \text{if } u_j^x < \infty, \\ \infty & \text{otherwise} \end{cases}$$

so that the objective value $c^T x$ is negative. Such a solution implies that (7.4) is unbounded, and that the dual of (7.4) is infeasible.

We note that the dual of (7.4) is a problem which constraints are identical to the constraints of the original dual problem (7.2): If the dual of (7.4) is infeasible, so is the original dual problem.

7.1.2 Primal and dual infeasible case

In case that both the primal problem (7.1) and the dual problem (7.2) are infeasible, MOSEK will report only one of the two possible certificates — which one is not defined (MOSEK returns the first certificate found).

7.2 Quadratic and quadratically constrained optimization

A convex quadratic optimization problem is an optimization problem of the form

$$\begin{array}{lccccl} \text{minimize} & & \frac{1}{2}x^T Q^o x + c^T x + c^f & & & \\ \text{subject to} & l_k^c & \leq & \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,i}x_j & \leq & u_k^c, \quad k = 0, \ldots, m-1, \\ & l^x & \leq & x & \leq & u^x, \quad j = 0, \ldots, n-1, \end{array} \tag{7.5}$$

where the convexity requirement implies that

- Q^o is a symmetric positive semi-definite matrix.
- If $l_k^c = -\infty$, then Q^k is a symmetric positive semi-definite matrix.
- If $u_k^c = \infty$, then Q^k is a symmetric negative semi-definite matrix.
- If $l_k > -\infty$ and $u_k^k < \infty$, then Q^k is a zero matrix.

The convexity requirement is very important and it is strongly recommended that MOSEK is applied to convex problems only.

7.2.1 A general recommendation

Any convex quadratic optimization problem can be reformulated as a conic optimization problem. It is our experience that for the majority of practical applications it is better to cast them as conic problems because

- the resulting problem is convex by construction, and
- the conic optimizer is more efficient than the optimizer for general quadratic problems.

See Section 7.3.3.1 for further details.

7.2.2 Reformulating as a separable quadratic problem

The simplest quadratic optimization problem is

$$\begin{array}{lccc} \text{minimize} & 1/2x^T Qx + c^T x & & \\ \text{subject to} & Ax & = & b, \\ & x \geq 0. & & \end{array} \tag{7.6}$$

The problem (7.6) is said to be a separable problem if Q is a diagonal matrix or, in other words, if the quadratic terms in the objective all have this form

$$x_j^2$$

instead of this form

$$x_j x_i.$$

The separable form has the following advantages:

- It is very easy to check the convexity assumption, and
- the simpler structure in a separable problem usually makes it easier to solve.

It is well-known that a positive semi-definite matrix Q can always be factorized, i.e. a matrix F exists so that

$$Q = F^T F. \tag{7.7}$$

In many practical applications of quadratic optimization F is known explicitly; e.g. if Q is a covariance matrix, F is the set of observations producing it.

Using (7.7), the problem (7.6) can be reformulated as

$$\begin{array}{lrcl} \text{minimize} & 1/2y^T Iy + c^T x & & \\ \text{subject to} & Ax & = & b, \\ & Fx - y & = & 0, \\ & x \geq 0. & & \end{array} \tag{7.8}$$

The problem (7.8) is also a quadratic optimization problem and has more constraints and variables than (7.6). However, the problem is separable. Normally, if F has fewer rows than columns, it is worthwhile to reformulate as a separable problem. Indeed consider the extreme case where F has one dense row and hence Q will be a dense matrix.

The idea presented above is applicable to quadratic constraints too. Now, consider the constraint

$$1/2x^T(F^T F)x \leq b \tag{7.9}$$

where F is a matrix and b is a scalar. (7.9) can be reformulated as

$$\begin{array}{rcl} 1/2y^T Iy & \leq & b, \\ Fx - y & = & 0. \end{array}$$

It should be obvious how to generalize this idea to make any convex quadratic problem separable.

Next, consider the constraint

$$1/2x^T(D + F^T F)x \leq b$$

where D is a positive semi-definite matrix, F is a matrix, and b is a scalar. We assume that D has a simple structure, e.g. that D is a diagonal or a block diagonal matrix. If this is the case, it may be worthwhile performing the reformulation

$$\begin{array}{rcl} 1/2((x^T Dx) + y^T Iy) & \leq & b, \\ Fx - y & = & 0. \end{array}$$

Now, the question may arise: When should a quadratic problem be reformulated to make it separable or near separable? The simplest rule of thumb is that it should be reformulated if the number of non-zeros used to represent the problem decreases when reformulating the problem.

7.3 Conic optimization

Conic optimization can be seen as a generalization of linear optimization. Indeed a conic optimization problem is a linear optimization problem plus a constraint of the form

$$x \in \mathcal{C}$$

where $\mathcal{C}$ is a convex cone. A complete conic problem has the form

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x \in \mathcal{C}. & & \end{array} \tag{7.10}$$

The cone $\mathcal{C}$ can be a Cartesian product of p convex cones, i.e.

$$\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_p$$

in which case $x \in \mathcal{C}$ can be written as

$$x = (x_1, \ldots, x_p),\ x_1 \in \mathcal{C}_1, \ldots, x_p \in \mathcal{C}_p$$

where each $x_t \in \mathbb{R}^{n_t}$. Please note that the n-dimensional Euclidean space $\mathbb{R}^n$ is a cone itself, so simple linear variables are still allowed.

MOSEK supports only a limited number of cones, specifically

$$\mathcal{C} = \mathcal{C}_1 \times \cdot \times \mathcal{C}_p$$

where each $\mathcal{C}_t$ has one of the following forms

- $\mathbb{R}$ set:
$$\mathcal{C}_t = \{x \in \mathbb{R}^{n^t}\}.$$
- Quadratic cone:
$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}.$$
- Rotated quadratic cone:
$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1 x_2 \geq \sum_{j=3}^{n^t} x_j^2,\ x_1, x_2 \geq 0 \right\}.$$

Although these cones may seem to provide only limited expressive power they can be used to model a large range of problems as demonstrated in Section 7.3.3.

7.3.1 Duality for conic optimization

The dual problem corresponding to the conic optimization problem (7.10) is given by

$$\begin{array}{lccl} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x + s_n^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, \\ & s_n^x \in \mathcal{C}^* & & \end{array} \tag{7.11}$$

where the dual cone $\mathcal{C}^*$ is a product of the cones

$$\mathcal{C}^* = \mathcal{C}_1^* \times \cdots \mathcal{C}_p^*$$

where each $\mathcal{C}_t^*$ is the dual cone of $\mathcal{C}_t$. For the cone types MOSEK can handle, the relation between the primal and dual cone is given as follows:

- $\mathbb{R}$ set:
$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n^t} \right\} \quad \Leftrightarrow \quad \mathcal{C}_t^* := \left\{ s \in \mathbb{R}^{n^t} : \ s = 0 \right\}.$$
- Quadratic cone:
$$\mathcal{C}_t := \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\} \quad \Leftrightarrow \quad \mathcal{C}_t^* = \mathcal{C}_t.$$
- Rotated quadratic cone:
$$\mathcal{C}_t := \left\{ x \in \mathbb{R}^{n^t} : 2x_1 x_2 \geq \sum_{j=3}^{n^t} x_j^2, \ x_1, x_2 \geq 0 \right\}. \quad \Leftrightarrow \quad \mathcal{C}_t^* = \mathcal{C}_t.$$

Please note that the dual problem of the dual problem is identical to the original primal problem.

7.3.2 Infeasibility

In case MOSEK finds a problem to be infeasible it reports a certificate of the infeasibility. This works exactly as for linear problems (see Sections 7.1.1.3 and 7.1.1.4).

7.3.3 Examples

This section contains several examples of inequalities and problems that can be cast as conic optimization problems.

7.3.3.1 Quadratic objective and constraints

From Section 7.2.2 we know that any convex quadratic problem can be stated on the form

$$\begin{array}{lll} \text{minimize} & 0.5\left\|Fx\right\|^2 + c^T x, & \\ \text{subject to} & 0.5\left\|Gx\right\|^2 + a^T x & \leq \quad b, \end{array} \tag{7.12}$$

where F and G are matrices and c and a are vectors. For simplicity we assume that there is only one constraint, but it should be obvious how to generalize the methods to an arbitrary number of constraints.

Problem (7.12) can be reformulated as

$$\begin{array}{lrcl} \text{minimize} & 0.5\left\|t\right\|^2 + c^T x, & & \\ \text{subject to} & 0.5\left\|z\right\|^2 + a^T x & \leq & b, \\ & Fx - t & = & 0, \\ & Gx - z & = & 0 \end{array} \tag{7.13}$$

after the introduction of the new variables t and z. It is easy to convert this problem to a conic quadratic optimization problem, i.e.

$$\begin{array}{lrcll} \text{minimize} & v + c^T x, & & & \\ \text{subject to} & p + a^T x & = & b, & \\ & Fx - t & = & 0, & \\ & Gx - z & = & 0, & \\ & w & = & 1, & \\ & q & = & 1, & \\ & \left\|t\right\|^2 & \leq & 2vw, & v, w \geq 0, \\ & \left\|z\right\|^2 & \leq & 2pq, & p, q \geq 0. \end{array} \tag{7.14}$$

In this case we can model the last two inequalities using rotated quadratic cones.

If we assume that F is a non-singular matrix — e.g. a diagonal matrix — then

$$x = F^{-1}t$$

and hence we can eliminate x from the problem to obtain:

$$\begin{array}{lrcll} \text{minimize} & v + c^T F^{-1} t, & & & \\ \text{subject to} & p + a^T F^{-1} t & = & b, & \\ & GF^{-1}t - z & = & 0, & \\ & w & = & 1, & \\ & q & = & 1, & \\ & \left\|t\right\|^2 & \leq & 2vw, & v, w \geq 0, \\ & \left\|z\right\|^2 & \leq & 2pq, & p, q \geq 0. \end{array} \tag{7.15}$$

In most cases MOSEK performs this reduction automatically during the presolve phase before the optimization is performed.

7.3.3.2 Minimizing a sum of norms

The next example is the problem of minimizing a sum of norms, i.e. the problem

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{k} \left\|x^i\right\| & \\ \text{subject to} & Ax = b, & \end{array} \tag{7.16}$$

where

$$x := \begin{bmatrix} x^1 \\ \vdots \\ x^k \end{bmatrix}.$$

This problem is equivalent to

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{k} z_i & \\ \text{subject to} & Ax = b, & \\ & \left\|x^i\right\| \leq z_i, & i = 1, \ldots, k, \end{array} \tag{7.17}$$

which in turn is equivalent to

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{k} z_i & \\ \text{subject to} & Ax = b, & \\ & (z_i, x^i) \in \mathcal{C}_i, & i = 1, \ldots, k \end{array} \tag{7.18}$$

where all $\mathcal{C}_i$ are of the quadratic type, i.e.

$$\mathcal{C}_i := \left\{(z_i, x^i) : \; z_i \geq \left\|x^i\right\|\right\}.$$

The dual problem corresponding to (7.18) is

$$\begin{array}{lll} \text{maximize} & b^T y & \\ \text{subject to} & A^T y + s = c, & \\ & t_i = 1, & i = 1, \ldots, k, \\ & (t_i, s^i) \in \mathcal{C}_i, & i = 1, \ldots, k \end{array} \tag{7.19}$$

where

$$s := \begin{bmatrix} s^1 \\ \vdots \\ s^k \end{bmatrix}.$$

This problem is equivalent to

$$\begin{array}{lll} \text{maximize} & b^T y & \\ \text{subject to} & A^T y + s = c, & \\ & \left\|s^i\right\|_2^2 \leq 1, & i = 1, \ldots, k. \end{array} \tag{7.20}$$

Please note that in this case the dual problem can be reduced to an "ordinary" convex quadratically constrained optimization problem due to the special structure of the primal problem. In some cases it turns out that it is much better to solve the dual problem (7.19) rather than the primal problem (7.18).

7.3.3.3 Modelling polynomial terms using conic optimization

Generally an arbitrary polynomial term of the form

$$fx^g$$

cannot be represented with conic quadratic constraints, however in the following we will demonstrate some special cases where it is possible.

A particular simple polynomial term is the reciprocal, i.e.

$$\frac{1}{x}.$$

Now, a constraint of the form

$$\frac{1}{x} \leq y$$

where it is required that $x > 0$ is equivalent to

$$1 \leq xy \text{ and } x > 0$$

which in turn is equivalent to

$$\begin{array}{rcl} z & = & \sqrt{2}, \\ z^2 & \leq & 2xy. \end{array}$$

The last formulation is a conic constraint plus a simple linear equality.

E.g., consider the problem

$$\begin{array}{lll} \text{minimize} & c^T x & \\ \text{subject to} & \sum_{j=1}^{n} \frac{f_j}{x_j} & \leq \quad b, \\ & x \geq 0, & \end{array}$$

where it is assumed that $f_j > 0$ and $b > 0$. This problem is equivalent to

$$\begin{array}{lrcll} \text{minimize} & c^T x & & & \\ \text{subject to} & \sum_{j=1}^{n} f_j z_j & = & b, & \\ & v_j & = & \sqrt{2}, & j = 1, \ldots, n, \\ & v_j^2 & \leq & 2 z_j x_j, & j = 1, \ldots, n, \\ & x, z \geq 0, & & & \end{array} \tag{7.21}$$

because

$$v_j^2 = 2 \leq 2 z_j x_j$$

implies that

$$\frac{1}{x_j} \leq z_j \text{ and } \sum_{j=1}^{n} \frac{f_j}{x_j} \leq \sum_{j=1}^{n} f_j z_j = b.$$

The problem (7.21) is a conic quadratic optimization problem having n 3-dimensional rotated quadratic cones.

The next example is the constraint

$$\begin{array}{rcl} \sqrt{x} & \geq & |t|, \\ x & \geq & 0, \end{array}$$

where both t and x are variables. This set is identical to the set

$$\begin{array}{rcl} t^2 & \leq & 2xz, \\ z & = & 0.5, \\ x, z, & \geq & 0. \end{array} \tag{7.22}$$

Occasionally, when modeling the *market impact* term in portfolio optimization, the polynomial term $x^{\frac{3}{2}}$ occurs. Therefore, consider the set defined by the inequalities

$$\begin{array}{rcl} x^{1.5} & \leq & t, \\ 0 & \leq & x. \end{array} \tag{7.23}$$

We will exploit that $x^{1.5} = x^2/\sqrt{x}$. First define the set

$$\begin{array}{rcl} x^2 & \leq & 2st, \\ s, t & \geq & 0. \end{array} \tag{7.24}$$

Now, if we can make sure that

$$2s \leq \sqrt{x},$$

then we have the desired result since this implies that

$$x^{1.5} = \frac{x^2}{\sqrt{x}} \leq \frac{x^2}{2s} \leq t.$$

Please note that s can be chosen freely and that $\sqrt{x} = 2s$ is a valid choice.

Let

$$\begin{array}{rcl} x^2 & \leq & 2st, \\ w^2 & \leq & 2vr, \\ x & = & v, \\ s & = & w, \\ r & = & \frac{1}{8}, \\ s, t, v, r & \geq & 0, \end{array} \tag{7.25}$$

then

$$\begin{array}{rcl} s^2 & = & w^2 \\ & \leq & 2vr \\ & = & \frac{v}{4} \\ & = & \frac{x}{4}. \end{array}$$

Moreover,

$$\begin{array}{rcl} x^2 & \leq & 2st, \\ & \leq & 2\sqrt{\frac{x}{4}}t \end{array}$$

leading to the conclusion that

$$x^{1.5} \leq t.$$

(7.25) is a conic reformulation which is equivalent to (7.23). Please note that the $x \geq 0$ constraint does not appear explicitly in (7.24) and (7.25), but implicitly since $x = v \geq 0$.

As we shall see next, any polynomial term of the form x^g where g is a positive rational number can be represented using conic quadratic constraints [3, pp. 12-13], [13].

7.3.3.4 Optimization with rational polynomials

We next demonstrate how to model convex polynomial constraints of the form $x^{p/q} \leq t$ (where p and q are both positive integers) as a set of rotated quadratic cone constraints.

Following Ben-Tal et al. [13, p. 105] we use an intermediate result, namely that the set

$$\{s \in \mathbb{R},\, y \in \mathbb{R}^{2^l}_+ \mid s \leq (2^{l2^{l-1}} y_1 y_2 \cdots y_{2^l})^{1/2^l}\}$$

is convex and can be represented as a set of rotated quadratic cone constraints. To see this, we rewrite the condition (exemplified for $l = 3$),

$$s \leq \left(2^{12} \cdot y_1 \cdot y_2 \cdot y_3 \cdot y_4 \cdot y_5 \cdot y_6 \cdot y_7 \cdot y_8\right)^{1/8} \tag{7.26}$$

as

$$s^8 \leq \left(2^{12} \cdot y_1 \cdot y_2 \cdot y_3 \cdot y_4 \cdot y_5 \cdot y_6 \cdot y_7 \cdot y_8\right) \tag{7.27}$$

since all $y_i \geq 0$. We next introduce l levels of auxiliary variables and (rotated cone) constraints

$$y_{11}^2 \leq 2y_1 y_2, \quad y_{12}^2 \leq 2y_3 y_4, \quad y_{13}^2 \leq 2y_5 y_6, \quad y_{14}^2 \leq 2y_7 y_8, \tag{7.28}$$

$$y_{21}^2 \leq 2y_{11} y_{12}, \quad y_{22}^2 \leq 2y_{13} y_{14}, \tag{7.29}$$

and finally

$$s^2 \leq 2y_{21} y_{22}. \tag{7.30}$$

By simple substitution we see that (7.30) and (7.27) are equivalent, and since (7.30) involves only a set of simple rotated conic constraints then the original constraint (7.26) can be represented using only rotated conic constraints.

7.3.3.5 Convex increasing power functions

Using the intermediate result in section 7.3.3.4 we can include convex power functions with positive rational powers, i.e., constraints of the form

$$x^{p/q} \leq t, \quad x \geq 0$$

where p and q are positive integers and $p/q \geq 1$. For example, consider the constraints

$$x^{5/3} \leq t, \quad x \geq 0.$$

We rewrite it as

$$x^8 \leq x^3 t^3, \quad x \geq 0$$

which in turn is equivalent to

$$x^8 \leq 2^{12} y_1 y_2 \cdots y_8, \quad x = y_1 = y_2 = y_3, \quad y_4 = y_5 = y_6 = t, \quad y_6 = 1, \quad y_7 = 2^{-12}, \quad x, y_i \geq 0,$$

i.e., it can be represented as a set of rotated conic and linear constraints using the reformulation above.

For general p and q we choose l as the smallest integer such that $p \leq 2^l$ and we construct the problem as

$$x^{2^l} \leq 2^{l2^{l-1}} y_1 y_2 \cdots y_{2^l}, \quad x, y_i \geq 0,$$

with the first $2^l - p$ elements of y set to x, the next q elements set to t, and the product of the remaining elements as $1/2^{l2^{l-1}}$, i.e.,

$$x^{2^l} \leq x^{2^l - p} t^q, \quad x \geq 0 \qquad \Longleftrightarrow \qquad x^{p/q} \leq t, \quad x \geq 0.$$

7.3.3.6 Decreasing power functions

We can also include decreasing power functions with positive rational powers

$$x^{-p/q} \leq t, \quad x \geq 0$$

where p and q are positive integers. For example, consider

$$x^{-5/2} \leq t, \quad x \geq 0,$$

or equivalently

$$1 \leq x^5 t^2, \quad x \geq 0,$$

which, in turn, can be rewritten as

$$s^8 \leq 2^{12} y_1 y_2 \cdots y_8, \quad s = 2^{3/2}, \quad y_1 = \cdots = y_5 = x, \quad y_6 = y_7 = y_8 = t, \quad x, y_i \geq 0.$$

For general p and q we choose l as the smallest integer such that $p + q \leq 2^l$ and we construct the problem as

$$s^{2^l} \leq y_1 y_2 \cdots y_{2^l}, \quad y_i \geq 0,$$

with $s = 2^{l/2}$ and the first p elements of y set to x, the next q elements set to t, and the remaining elements set to 1, i.e.,

$$1 \leq x^p t^q, \quad x \geq 0 \qquad \Longleftrightarrow \qquad x^{-p/q} \leq t, \quad x \geq 0.$$

7.3.3.7 Minimizing general polynomials

Using the formulations in section 7.3.3.5 and section 7.3.3.6 it is straightforward to minimize general polynomials. For example, we can minimize

$$f(x) = x^2 + x^{-2}$$

which is used in statistical matching. We first formulate the problem

$$\begin{array}{ll} \text{minimize} & u + v \\ \text{subject to} & x^2 \leq u \\ & x^{-2} \leq v, \end{array}$$

which is equivalent to the quadratic conic optimization problem

$$\begin{array}{ll} \text{minimize} & u+v \\ \text{subject to} & x^2 \leq 2uw \\ & s^2 \leq 2y_{21}y_{22} \\ & y_{21}^2 \leq 2y_1y_2 \\ & y_{22}^2 \leq 2y_3y_4 \\ & w = 1 \\ & s = 2^{3/4} \\ & y_1 = y_2 = x \\ & y_3 = v \\ & y_4 = 1 \end{array}$$

in the variables $(x, u, v, w, s, y_1, y_2, y_3, y_4, y_{21}, y_{22})$.

7.3.3.8 Further reading

If you want to learn more about what can be modeled as a conic optimization problem we recommend the references [3, 13, 18].

7.3.4 Potential pitfalls in conic optimization

While a linear optimization problem either has a bounded optimal solution or is infeasible, the conic case is not as simple as that.

7.3.4.1 Non-attainment in the primal problem

Consider the example

$$\begin{array}{lrcl} \text{minimize} & z & & \\ \text{subject to} & 2yz & \geq & x^2, \\ & x & = & \sqrt{2}, \\ & y, z & \geq & 0, \end{array} \tag{7.31}$$

which corresponds to the problem

$$\begin{array}{lrcl} \text{minimize} & \frac{1}{y} & & \\ \text{subject to} & y & \geq & 0. \end{array} \tag{7.32}$$

Clearly, the optimal objective value is zero but it is never attained because implicitly we assume that the optimal y is finite.

7.3.4.2 Non-attainment in the dual problem

Next, consider the example

$$\begin{array}{lrcl} \text{minimize} & x_4 & & \\ \text{subject to} & x_3 + x_4 & = & 1, \\ & x_1 & = & 0, \\ & x_2 & = & 1, \\ & 2x_1x_2 & \geq & x_3^2, \\ & x_1, x_2 & \geq & 0, \end{array} \tag{7.33}$$

which has the optimal solution

$$x_1^* = 0,\ x_2^* = 1,\ x_3^* = 0 \text{ and } x_4^* = 1$$

implying that the optimal primal objective value is 1.

Now, the dual problem corresponding to (7.33) is

$$\begin{array}{lrcl} \text{maximize} & y_1 + y_3 & & \\ \text{subject to} & y_2 + s_1 & = & 0, \\ & y_3 + s_2 & = & 0, \\ & y_1 + s_3 & = & 0, \\ & y_1 & = & 1, \\ & 2s_1s_2 & \geq & s_3^2, \\ & s_1, s_2 & \geq & 0. \end{array} \tag{7.34}$$

Therefore,

$$y_1^* = 1$$

and

$$s_3^* = -1.$$

This implies that

$$2s_1^*s_2^* \geq (s_3^*)^2 = 1$$

and hence $s_2^* > 0$. Given this fact we can conclude that

$$\begin{array}{rcl} y_1^* + y_3^* & = & 1 - s_2^* \\ & < & 1 \end{array}$$

implying that the optimal dual objective value is 1, however, this is never attained. Hence, no primal-dual bounded optimal solution with zero duality gap exists. Of course it is possible to find a primal-dual feasible solution such that the duality gap is close to zero, but then s_1^* will be similarly large. This is likely to make the problem (7.33) hard to solve.

An inspection of the problem (7.33) reveals the constraint $x_1 = 0$, which implies that $x_3 = 0$. If we either add the redundant constraint

$$x_3 = 0$$

to the problem (7.33) or eliminate x_1 and x_3 from the problem it becomes easy to solve.

7.4 Nonlinear convex optimization

MOSEK is capable of solving smooth (twice differentiable) convex nonlinear optimization problems of the form

$$\begin{array}{lrcccl} \text{minimize} & & & f(x) + c^T x & & \\ \text{subject to} & & & g(x) + Ax - x^c & = & 0, \\ & l^c & \leq & x^c & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \tag{7.35}$$

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

This means that the ith constraint has the form

$$l_i^c \leq g_i(x) + \sum_{j=1}^{n} a_{i,j} x_j \leq u_i^c$$

when the x_i^c variable has been eliminated.

The linear term Ax is not included in $g(x)$ since it can be handled much more efficiently as a separate entity when optimizing.

The nonlinear functions f and g must be smooth in all $x \in [l^x; u^x]$. Moreover, $f(x)$ must be a convex function and $g_i(x)$ must satisfy

$$\begin{array}{rcll} l_i^c = -\infty & \Rightarrow & g_i(x) & \text{is convex,} \\ u_i^c = \infty & \Rightarrow & g_i(x) & \text{is concave,} \\ -\infty < l_i^c \leq u_i^c < \infty & \Rightarrow & g_i(x) = 0. & \end{array}$$

7.4.1 Duality

So far, we have not discussed what happens when MOSEK is used to solve a primal or dual infeasible problem. In the following section these issues are addressed.

Similar to the linear case, MOSEK reports dual information in the general nonlinear case. Indeed in this case the Lagrange function is defined by

$$\begin{array}{rcl} L(x^c,x,y,s_l^c,s_u^c,s_l^x,s_u^x) & := & f(x)+c^Tx+c^f \\ & & -y^T(Ax+g(x)-x^c) \\ & & -(s_l^c)^T(x^c-l^c)-(s_u^c)^T(u^c-x^c) \\ & & -(s_l^x)^T(x-l^x)-(s_u^x)^T(u^x-x). \end{array}$$

and the dual problem is given by

$$\begin{array}{lcl} \text{maximize} & L(x^c,x,y,s_l^c,s_u^c,s_l^x,s_u^x) & \\ \text{subject to} & \nabla_{(x^c,x)}L(x^c,x,y,s_l^c,s_u^c,s_l^x,s_u^x) = 0, & \\ & s_l^c,s_u^c,s_l^x,s_u^x \geq 0. & \end{array}$$

which is equivalent to

$$\begin{array}{lcll} \text{maximize} & f(x)-y^Tg(x)-x^T(\nabla f(x)^T-\nabla g(x)^Ty) & & \\ & +((l^c)^Ts_l^c-(u^c)^Ts_u^c+(l^x)^Ts_l^x-(u^x)^Ts_u^x+c^f & & \\ \text{subject to} & -\nabla f(x)^T+A^Ty+\nabla g(x)^Ty+s_l^x-s_u^x & = & c, \\ & -y+s_l^c-s_u^c & = & 0, \\ & s_l^c,s_u^c,s_l^x,s_u^x \geq 0. & & \end{array} \tag{7.36}$$

7.5 Recommendations

Often an optimization problem can be formulated in several different ways, and the exact formulation used may have a significant impact on the solution time and the quality of the solution. In some cases the difference between a "good" and a "bad" formulation means the ability to solve the problem or not.

Below is a list of several issues that you should be aware of when developing a good formulation.

1. Sparsity is very important. The constraint matrix A is assumed to be a sparse matrix, where sparse means that it contains many zeros (typically less than 10% non-zeros). Normally, when A is sparser, less memory is required to store the problem and it can be solved faster.

2. Avoid large bounds as these can introduce all sorts of numerical problems. Assume that a variable x_j has the bounds

   $$0.0 \leq x_j \leq 1.0e16.$$

   The number 1.0e16 is large and it is very likely that the constraint $x_j \leq 1.0e16$ is non-binding at optimum, and therefore that the bound $1.0e16$ will not cause problems. Unfortunately, this is a naïve assumption because the bound $1.0e16$ may actually affect the presolve, the scaling, the computation of the dual objective value, etc. In this case the constraint $x_j \geq 0$ is likely to be sufficient, i.e. $1.0e16$ is just a way of representing infinity.

3. Avoid large penalty terms in the objective, i.e. do not have large terms in the linear part of the objective function. They will most likely cause numerical problems.

4. On a computer all computations are performed in finite precision, which implies that

$$1 = 1 + \varepsilon$$

where ε is about 10^{-16}. This means that the results of all computations are truncated and therefore causing rounding errors. The upshot is that very small numbers and very large numbers should be avoided, e.g. it is recommended that all elements in A either are zero or belong to the interval $[10^{-6}, 10^{6}]$. The same holds for the bounds and the linear objective.

5. Decreasing the number of variables or constraints does not *necessarily* make it easier to solve a problem. In certain cases, i.e. in nonlinear optimization, it may be a good idea to introduce more constraints and variables if it makes the model separable. Furthermore, a big but sparse problem may be advantageous compared to a smaller but denser problem.

6. Try to avoid linearly dependent rows among the linear constraints. Network flow problems and multi-commodity network flow problems, for example, often contain one or more linearly dependent rows.

7. Finally, it is recommended to consult some of the papers about preprocessing to get some ideas about efficient formulations. See e.g. [4, 5, 16, 17].

7.5.1 Avoid near infeasible models

Consider the linear optimization problem

$$\begin{array}{lrcl} \text{minimize} & & & \\ \text{subject to} & x + y & \leq & 10^{-10} + \alpha, \\ & 1.0e4x + 2.0e4y & \geq & 10^{-6}, \\ & x, y \geq 0. & & \end{array} \tag{7.37}$$

Clearly, the problem is feasible for $\alpha = 0$. However, for $\alpha = -1.0e-10$ the problem is infeasible. This implies that an insignificant change in the right side of the constraints makes the problem status switch from feasible to infeasible. Such a model should be avoided.

7.6 Examples continued

7.6.1 The absolute value

Assume that we have a constraint for the form

$$|f^T x + g| \leq b \tag{7.38}$$

where $x \in \mathbb{R}^n$ is a vector of variables, and $f \in \mathbb{R}^n$ and $g, b \in \mathbb{R}$ are constants.

It is easy to verify that the constraint (7.38) is equivalent to

$$-b \leq f^T x + g \leq b \tag{7.39}$$

which is a set of ordinary linear inequality constraints.

Please note that equalities involving an absolute value such as

$$|x| = 1$$

cannot be formulated as a linear or even a as convex nonlinear optimization problem. It requires integer constraints.

7.6.2 The Markowitz portfolio model

In this section we will show how to model several versions of the Markowitz portfolio model using conic optimization.

The Markowitz portfolio model deals with the problem of selecting a portfolio of assets, i.e. stocks, bonds, etc. The goal is to find a portfolio such that for a given return the risk is minimized. The assumptions are:

- A portfolio can consist of n traded assets numbered $1, 2, \ldots$ held over a period of time.
- w_j^0 is the initial holding of asset j where $\sum_j w_j^0 > 0$.
- r_j is the return on asset j and is assumed to be a random variable. r has a known mean $\bar{r}$ and covariance Σ.

The variable x_j denotes the amount of asset j traded in the given period of time and has the following meaning:

- If $x_j > 0$, then the amount of asset j is increased (by purchasing).
- If $x_j < 0$, then the amount of asset j is decreased (by selling).

The model deals with two central quantities:

- Expected return:
$$E[r^T(w^0 + x)] = \bar{r}^T(w^0 + x).$$
- Variance (Risk):
$$V[r^T(w^0 + x)] = (w^0 + x)^T \Sigma (w^0 + x).$$

By definition Σ is positive semi-definite and

$$\begin{array}{rcl} \text{Std. dev.} & = & \left\| \Sigma^{\frac{1}{2}}(w^0 + x) \right\| \\ & = & \left\| L^T(w^0 + x) \right\| \end{array}$$

where L is **any** matrix such that

$$\Sigma = LL^T$$

A low rank of Σ is advantageous from a computational point of view. A valid L can always be computed as the Cholesky factorization of Σ.

7.6.2.1 Minimizing variance for a given return

In our first model we want to minimize the variance while selecting a portfolio with a specified expected target return t. Additionally, the portfolio must satisfy the budget (self-financing) constraint asserting that the total amount of assets sold must equal the total amount of assets purchased. This is expressed in the model

$$\begin{array}{lrcl} \text{minimize} & V[r^T(w^0+x)] & & \\ \text{subject to} & E[r^T(w^0+x)] & = & t, \\ & e^T x & = & 0, \end{array} \tag{7.40}$$

where $e := (1, \ldots, 1)^T$. Using the definitions above this may be formulated as a quadratic optimization problem:

$$\begin{array}{lrcl} \text{minimize} & (w^0+x)^T\Sigma(w^0+x) & & \\ \text{subject to} & \bar{r}^T(w^0+x) & = & t, \\ & e^T x & = & 0. \end{array} \tag{7.41}$$

7.6.2.2 Conic quadratic reformulation

An equivalent conic quadratic reformulation is given by:

$$\begin{array}{lrcl} \text{minimize} & f & & \\ \text{subject to} & \Sigma^{\frac{1}{2}}(w^0+x) - g & = & 0, \\ & \bar{r}^T(w^0+x) & = & t, \\ & e^T x & = & 0, \\ & f \geq \|g\|. & & \end{array} \tag{7.42}$$

Here we minimize the standard deviation instead of the variance. Please note that $\Sigma^{\frac{1}{2}}$ can be replaced by any matrix L where $\Sigma = LL^T$. A low rank L is computationally advantageous.

7.6.2.3 Transaction costs with market impact term

We will now expand our model to include transaction costs as a fraction of the traded volume. [1, pp. 445-475] argues that transaction costs can be modeled as follows

$$\text{commission} + \frac{\text{bid}}{\text{ask}} - \text{spread} + \theta\sqrt{\frac{\text{trade volume}}{\text{daily volume}}}, \tag{7.43}$$

and that it is important to incorporate these into the model.

In the following we deal with the last of these terms denoted the *market impact term*. If you sell (buy) a lot of assets the price is likely to go down (up). This can be captured in the market impact term

$$\theta\sqrt{\frac{\text{trade volume}}{\text{daily volume}}} \approx m_j\sqrt{|x_j|}.$$

The θ and "daily volume" have to be estimated in some way, i.e.

$$m_j = \frac{\theta}{\sqrt{\text{daily volume}}}$$

has to be estimated. The market impact term gives the cost as a fraction of daily traded volume ($|x_j|$). Therefore, the total cost when trading an amount x_j of asset j is given by

$$|x_j|(m_j|x_j|^{\frac{1}{2}}).$$

This leads us to the model:

$$\begin{array}{lrcl} \text{minimize} & f & & \\ \text{subject to} & \Sigma^{\frac{1}{2}}(w^0+x)-g & = & 0, \\ & \bar{r}^T(w^0+x) & = & t, \\ & e^Tx+e^Ty & = & 0, \\ & |x_j|(m_j|x_j|^{\frac{1}{2}}) & \leq & y_j, \\ & f \geq \|g\|\,. & & \end{array} \tag{7.44}$$

Now, defining the variable transformation

$$y_j = m_j\bar{y}_j$$

we obtain

$$\begin{array}{lrcl} \text{minimize} & f & & \\ \text{subject to} & \Sigma^{\frac{1}{2}}(w^0+x)-g & = & 0, \\ & \bar{r}^T(w^0+x) & = & t, \\ & e^Tx+m^T\bar{y} & = & 0, \\ & |x_j|^{3/2} & \leq & \bar{y}_j, \\ & f \geq \|g\|\,. & & \end{array} \tag{7.45}$$

As shown in Section 7.3.3.3 the set

$$|x_j|^{3/2} \leq \bar{y}_j$$

can be modeled by

$$\begin{array}{rcl} x_j & \leq & z_j, \\ -x_j & \leq & z_j, \\ z_j^2 & \leq & 2s_j\bar{y}_j, \\ u_j^2 & \leq & 2v_jq_j, \\ z_j & = & v_j, \\ s_j & = & u_j, \\ q_j & = & \frac{1}{8}, \\ q_j, s_j, \bar{y}_j, v_j, q_j & \geq & 0. \end{array} \tag{7.46}$$

7.6.2.4 Further reading

For further reading please see [19] in particular, and [23] and [1], which also contain relevant material.

Chapter 8

The optimizers for continuous problems

The most essential part of MOSEK is the optimizers. Each optimizer is designed to solve a particular class of problems i.e. linear, conic, or general nonlinear problems. The purpose of the present chapter is to discuss which optimizers are available for the continuous problem classes and how the performance of an optimizer can be tuned, if needed.

This chapter deals with the optimizers for *continuous problems* with no integer variables.

8.1 How an optimizer works

When the optimizer is called, it roughly performs the following steps:

Presolve: Preprocessing to reduce the size of the problem.

Dualizer: Choosing whether to solve the primal or the dual form of the problem.

Scaling: Scaling the problem for better numerical stability.

Optimize: Solve the problem using selected method.

The first three preprocessing steps are transparent to the user, but useful to know about for tuning purposes. In general, the purpose of the preprocessing steps is to make the actual optimization more efficient and robust.

8.1.1 Presolve

Before an optimizer actually performs the optimization the problem is preprocessed using the so-called presolve. The purpose of the presolve is to

- remove redundant constraints,
- eliminate fixed variables,
- remove linear dependencies,
- substitute out free variables, and
- 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 [4, 5].

It is possible to fine-tune the behavior of the presolve or to turn it off entirely. If presolve consumes too much time or memory compared to the reduction in problem size gained it may be disabled. This is done by setting the parameter MSK_IPAR_PRESOLVE_USE to MSK_PRESOLVE_MODE_OFF.

The two most time-consuming steps of the presolve are

- the eliminator, and
- the linear dependency check.

Therefore, in some cases it is worthwhile to disable one or both of these.

8.1.1.1 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 with the parameter MSK_IPAR_PRESOLVE_ELIMINATOR_USE to MSK_OFF.

8.1.1.2 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 practise to build models without linear dependencies. If the linear dependencies are removed at the modeling stage, the linear dependency check can safely be disabled by setting the parameter MSK_IPAR_PRESOLVE_LINDEP_USE to MSK_OFF.

8.1.2 Dualizer

All linear, conic, and convex optimization problems have an equivalent dual problem associated with them. MOSEK has built-in heuristics to determine if it is most efficient to solve the primal or dual problem. The form (primal or dual) solved is displayed in the MOSEK log. Should the internal heuristics not choose the most efficient form of the problem it may be worthwhile to set the dualizer manually by setting the parameters:

- 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 problems may be dualized.

8.1.3 Scaling

Problems containing data with large and/or small coefficients, say $1.0e+9$ or $1.0e-7$, are often hard to solve. Significant digits may be truncated in calculations with finite precision, which can result in the optimizer relying on inaccurate calculations. Since computers work in finite precision, extreme coefficients should be avoided. In general, data around the same "order of magnitude" is preferred, and we will refer to a problem, satisfying this loose property, as being *well-scaled*. If the problem is not well scaled, MOSEK will try to scale (multiply) constraints and variables by suitable constants. MOSEK solves the scaled problem to improve the numerical properties.

The scaling process is transparent, i.e. the solution to the original problem is reported. It is important to be aware that the optimizer terminates when the termination criterion is met on the scaled problem, therefore significant primal or dual infeasibilities may occur after unscaling for badly scaled problems. The best solution to this problem is to reformulate it, 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.

8.1.4 Using multiple CPU's

The interior-point optimizers in MOSEK have been parallelized. This means that if you solve linear, quadratic, conic, or general convex optimization problem using the interior-point optimizer, you can take advantage of multiple CPU's.

By default MOSEK uses one thread to solve the problem, but the number of threads (and thereby CPUs) employed can be changed by setting the parameter MSK_IPAR_INTPNT_NUM_THREADS This should never exceed the number of CPU's on the machine.

The speed-up obtained when using multiple CPUs is highly problem and hardware dependent, and consequently, it is advisable to compare single threaded and multi threaded performance for the given problem type to determine the optimal settings.

For small problems, using multiple threads will probably not be worthwhile.

8.2 Linear optimization

8.2.1 Optimizer selection

Two different types of optimizers are available for linear problems: The default is an interior-point method, and the alternatives are simplex methods. The optimizer can be selected using the parameter MSK_IPAR_OPTIMIZER.

8.2.2 The interior-point optimizer

The purpose of this section is to provide information about the algorithm employed in MOSEK interior-point optimizer.

In order to keep the discussion simple it is assumed that MOSEK solves linear optimization problems on standard form

$$\begin{array}{lrcl} \text{minimize} & c^T x & & \\ \text{subject to} & Ax & = & b, \\ & x \geq 0. & & \end{array} \tag{8.1}$$

This is in fact what happens inside MOSEK; for efficiency reasons MOSEK converts the problem to standard form before solving, then convert it back to the input form when reporting the solution.

Since it is not known beforehand whether problem (8.1) has an optimal solution, is primal infeasible or is dual infeasible, the optimization algorithm must deal with all three situations. This is the reason that MOSEK solves the so-called homogeneous model

$$\begin{array}{rcl} Ax - b\tau & = & 0, \\ A^T y + s - c\tau & = & 0, \\ -c^T x + b^T y - \kappa & = & 0, \\ x, s, \tau, \kappa & \geq & 0, \end{array} \tag{8.2}$$

where y and s correspond to the dual variables in (8.1), and τ and κ are two additional scalar variables.

Note that the homogeneous model (8.2) 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 (8.2) 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

$$\begin{array}{rcl} A\frac{x^*}{\tau^*} & = & b, \\ A^T\frac{y^*}{\tau^*} + \frac{s^*}{\tau *} & = & c, \\ -c^T\frac{x^*}{\tau^*} + b^T\frac{y^*}{\tau^*} & = & 0, \\ x^*, s^*, \tau^*, \kappa^* & \geq & 0. \end{array} \tag{8.3}$$

This shows that $\frac{x^*}{\tau^*}$ is a primal optimal solution and $(\frac{y^*}{\tau^*}, \frac{s^*}{\tau *})$ is a dual optimal solution; this is reported as the optimal interior-point solution since

$$(x, y, s) = \left(\frac{x^*}{\tau^*}, \frac{y^*}{\tau^*}, \frac{s^*}{\tau *}\right)$$

is a primal-dual optimal solution.

On other hand, if $\kappa^* > 0$ then

$$\begin{array}{rcl} Ax^* & = & 0, \\ A^T y^* + s^* & = & 0, \\ -c^T x^* + b^T y^* & = & \kappa^*, \\ x^*, s^*, \tau^*, \kappa^* & \geq & 0. \end{array} \tag{8.4}$$

This implies that at least one of

$$-c^T x^* > 0 \tag{8.5}$$

or

$$b^T y^* > 0 \tag{8.6}$$

is satisfied. If (8.5) is satisfied then x^* is a certificate of dual infeasibility, whereas if (8.6) is satisfied then y^* is a certificate of dual infeasibility.

In summary, by computing an appropriate solution to the homogeneous model, all information required for a solution to the original problem is obtained. A solution to the homogeneous model can be computed using a primal-dual interior-point algorithm [10].

8.2.2.1 Interior-point termination criterion

For efficiency reasons it is not practical to solve the homogeneous model exactly. Hence, an exact optimal solution or an exact infeasibility certificate cannot be computed and a reasonable termination criterion has to be employed.

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

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

to homogeneous model is generated where

$$x^k, s^k, \tau^k, \kappa^k > 0.$$

Whenever the trial solution satisfies the criterion

$$\begin{array}{rcl} \left\| A\frac{x^k}{\tau^k} - b \right\| & \leq & \varepsilon_p(1 + \|b\|), \\ \left\| A^T\frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c \right\| & \leq & \varepsilon_d(1 + \|c\|), \text{ and} \\ \min\left(\frac{(x^k)^T s^k + \tau^k \kappa^k}{(\tau^k)^2}, \left|\frac{c^T x^k}{\tau^k} - \frac{b^T y^k}{\tau^k}\right|\right) & \leq & \varepsilon_g \max\left(1, \left|\frac{c^T x^k}{\tau^k}\right|\right), \end{array} \tag{8.7}$$

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 (8.7) is that the optimizer is terminated if

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

On the other hand, if the trial solution satisfies

$$-\varepsilon_i c^T x^k > \frac{\|c\|}{\max(\|b\|, 1)} \left\| Ax^k \right\| \tag{8.8}$$

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

$$\left\| Ax^k \right\| > 0,$$

and define

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

Tolerance	Parameter 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

Table 8.1: Parameters employed in termination criterion.

It is easy to verify that

$$\|A\bar{x}\| = \varepsilon_i \text{ and } -c^T\bar{x} > 1,$$

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

Finally, if

$$\varepsilon_i b^T y^k \geq \frac{\|b\|}{\max(1,\|c\|)} \left\|A^T y^k + s^k\right\| \tag{8.9}$$

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

It is possible to adjust the tolerances ε_p, ε_d, ε_g and ε_i using parameters; see table 8.1 for details.

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 (8.7) reveals that quality of the solution is dependent on $\|b\|$ and $\|c\|$; 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 [10]. This means that if the optimizer is stopped prematurely then it is very unlikely that either the primal or dual solution is feasible. Another consequence is that in most cases all the tolerances, ε_p, ε_d and ε_g, has to be relaxed together to achieve an effect.

The basis identification discussed in section 8.2.2.2 requires an optimal solution to work well; hence basis identification should turned off if the termination criterion is relaxed.

To conclude the discussion in this section, relaxing the termination criterion is usually is not worthwhile.

8.2.2.2 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 [7].

Please note that a basic solution is often more accurate than an interior-point solution.

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

controls when basis identification is performed.

8.2.2.3 The interior-point log

Below is a typical log output from the interior-point optimizer presented:

```
Optimizer  - threads                : 1
Optimizer  - solved problem         : the dual
Optimizer  - constraints            : 2                variables             : 6
Factor     - setup time             : 0.04             order time            : 0.00
Factor     - GP order used          : no               GP order time         : 0.00
Factor     - nonzeros before factor : 3                after factor          : 3
Factor     - offending columns      : 0                flops                 : 1.70e+001
ITE PFEAS    DFEAS    KAP/TAU  POBJ              DOBJ              MU       TIME
0   2.0e+002 2.9e+001 2.0e+002 -0.000000000e+000 -1.204741644e+003 2.0e+002 0.44
1   2.2e+001 3.1e+000 7.3e+002 -5.885951891e+003 -5.856764353e+003 2.2e+001 0.57
2   3.8e+000 5.4e-001 9.7e+001 -7.405187479e+003 -7.413054916e+003 3.8e+000 0.58
3   4.0e-002 5.7e-003 2.6e-001 -7.664507945e+003 -7.665313396e+003 4.0e-002 0.58
4   4.2e-006 6.0e-007 2.7e-005 -7.667999629e+003 -7.667999714e+003 4.2e-006 0.59
5   4.2e-010 6.0e-011 2.7e-009 -7.667999994e+003 -7.667999994e+003 4.2e-010 0.59
```

The first line displays the number of threads used by the optimizer and second line tells that the optimizer choose to solve the dual problem rather the primal problem. The next line displays the problem dimensions as seen by the optimizer, and the "`Factor...`" lines show various statistics. This is followed by the iteration log.

Using the same notation as in section 8.2.2 the columns of the iteration log has the following meaning:

- `ITE`: Iteration index.
- `PFEAS`: $\left\|Ax^k - b\tau^k\right\|$. The numbers in this column should converge monotonically towards to zero.
- `DFEAS`: $\left\|A^T y^k + s^k - c\tau^k\right\|$. The numbers in this column should converge monotonically toward to zero.
- `KAP/TAU`: κ^k/τ^k. If the numbers in this column converge toward zero then the problem has an optimal solution. Otherwise if the numbers converge towards infinity, the problem is primal or/and dual infeasible.
- `POBJ`: $c^T x^k/\tau^k$. An estimate for the primal objective value.
- `DOBJ`: $b^T y^k/\tau^k$. An estimate for the dual objective value.
- `MU`: $\frac{(x^k)^T s^k + \tau^k \kappa^k}{n+1}$. The numbers in this column should always converge monotonically to zero.
- `TIME`: Time spend since the optimization started.

8.2.3 The simplex based optimizer

An alternative to the interior-point optimizer is the simplex optimizer.

The simplex optimizer uses a different method that allows exploiting an initial guess for the optimal solution to reduce the solution time. Depending on the problem it may be faster or slower to use an initial guess; see section 8.2.4 for a discussion.

MOSEK provides both a primal and a dual variant of the simplex optimizer — we will return to this later.

8.2.3.1 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 (7.1) and (7.2) for a definition of the primal and dual problem. Due the fact that to computations are performed in finite precision MOSEK allows violation of primal and dual feasibility within certain tolerances. The user can control the allowed primal and dual infeasibility with the parameters MSK_DPAR_BASIS_TOL_X and MSK_DPAR_BASIS_TOL_S.

8.2.3.2 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 *hot-start*. If the user is solving a sequence of optimization problems by solving the problem, making modifications, and solving again, MOSEK will hot-start automatically.

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.

By default MOSEK uses presolve when performing a hot-start. If the optimizer only needs very few iterations to find the optimal solution it may be better to turn off the presolve.

8.2.3.3 Numerical difficulties in the simplex optimizers

Though MOSEK is designed to minimize numerical instability, completely avoiding it is impossible when working in finite precision. MOSEK counts a "numerical unexpected behavior" event inside the optimizer as a *set-back*. The user can define how many set-backs the optimizer accepts; if that number is exceeded, the optimization will be aborted. Set-backs are implemented to avoid long sequences where the optimizer tries to recover from an unstable situation.

Set-backs are, for example, repeated singularities when factorizing the basis matrix, repeated loss of feasibility, degeneracy problems (no progress in objective) and other events indicating numerical difficulties. If the simplex optimizer encounters a lot of set-backs the problem is usually badly scaled; in such a situation try to reformulate into a better scaled problem. Then, if a lot of set-backs still occur, trying one or more of the following suggestions may be worthwhile:

- Raise tolerances for allowed primal or dual feasibility: Hence, increase the value of
  - 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 hot-starts, in rare cases switching off this feature may improve stability. This is controlled by the MSK_IPAR_SIM_HOTSTART parameter.
- Increase maximum 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.

8.2.4 The interior-point or the simplex optimizer?

Given a linear optimization problem, which optimizer is the best: The primal simplex, the dual simplex or the interior-point optimizer?

It is impossible to provide a general answer to this question, however, the interior-point optimizer behaves more predictably — it tends to use between 20 and 100 iterations, almost independently of problem size — but cannot perform hot-start, while simplex can take advantage of an initial solution, but is less predictable for cold-start. The interior-point optimizer is used by default.

8.2.5 The primal or the dual simplex variant?

MOSEK provides both a primal and a dual simplex optimizer. Predicting which simplex optimizer is faster is impossible, however, in recent years the dual optimizer has seen several algorithmic and computational improvements, which, in our experience, makes it faster on average than the primal simplex optimizer. Still, it depends much on the problem structure and size.

Setting the MSK_IPAR_OPTIMIZER parameter to MSK_OPTIMIZER_FREE_SIMPLEX instructs MOSEK to choose which simplex optimizer to use automatically.

To summarize, if you want to know which optimizer is faster for a given problem type, you should try all the optimizers.

Alternatively, use the concurrent optimizer presented in Section 8.6.3.

8.3 Linear network optimization

8.3.1 Network flow problems

Linear optimization problems with the network flow structure as specified in section 6.5 can often be solved significantly faster with a specialized version of the simplex method [2] than with the general solvers.

MOSEK includes a network simplex solver which, on avarage, solves network problems 10 to 100 times faster than the standard simplex optimizers.

To use the network simplex optimizer, do the following:

- Input the network flow problem as an ordinary linear optimization problem.
- Set the parameters
  - MSK_IPAR_SIM_NETWORK_DETECT to 0, and
  - MSK_IPAR_OPTIMIZER to MSK_OPTIMIZER_FREE_SIMPLEX.
- Optimize the problem using MSK_optimize.

MOSEK will automatically detect the network structure and apply the specialized simplex optimizer.

8.3.2 Embedded network problems

Often problems contains both large parts with network structure and some non-network constraints or variables — such problems are said to have *embedded network structure*.

If the procedure described in section 8.3.1 is applied, MOSEK will attemt to exploit this structure to speed up the optimization.

This is done heuristically by detecting the largest network embedded in the problem, solving this subproblem using the network simplex optimizer, and using the solution to hot-start a normal simplex optimizer.

The MSK_IPAR_SIM_NETWORK_DETECT parameter defines how large a percentage of the problem should be a network before the specialized solver is applied. In general, it is recommended to use the network optimizer only on problems containing a substantial embedded network.

If MOSEK only finds limited network structure in a problem, consider trying to switch off presolve MSK_IPAR_PRESOLVE_USE and scaling MSK_IPAR_SIM_SCALING, since in rare cases it might disturb the network heuristic.

The network detection heuristic can also be called directly through MSK_netextraction.

Parameter name	Purpose
MSK_DPAR_INTPNT_CO_TOL_PFEAS	Controls primal feasibility
MSK_DPAR_INTPNT_CO_TOL_DFEAS	Controls dual feasibility
MSK_DPAR_INTPNT_CO_TOL_REL_GAP	Controls relative gap
MSK_DPAR_INTPNT_TOL_INFEAS	Controls when the problem is declared infeasible
MSK_DPAR_INTPNT_CO_TOL_MU_RED	Controls when the complementarity is reduced enough

Table 8.2: Parameters employed in termination criterion.

8.4 Conic optimization

8.4.1 The interior-point optimizer

For conic optimization problems only an interior-point type optimizer is available. The interior-point optimizer is an implementation of the so-called homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see [6].

8.4.1.1 Interior-point termination criteria

The parameters controlling when the conic interior-point optimizer terminates are shown in Table 8.2.

8.5 Nonlinear convex optimization

8.5.1 The interior-point optimizer

For quadratic, quadratically constrained, and general convex optimization problems an interior-point type optimizer is available. The interior-point optimizer is an implementation of the homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see [8, 9].

8.5.1.1 The convexity requirement

Continuous nonlinear problems are required to be convex. For quadratic problems MOSEK test this requirement before optimizing. Specifying a non-convex problem results in an error message.

The following parameters are available to control the convexity check:

- MSK_IPAR_CHECK_CONVEXITY: Turn convexity check on/off.
- MSK_DPAR_CHECK_CONVEXITY_REL_TOL: Tolerance for convexity check.
- MSK_IPAR_LOG_CHECK_CONVEXITY: Turn on more log information for debugging.

8.5.1.2 The differentiabilty requirement

The nonlinear optimizer in MOSEK requires both first order and second order derivatives. This of course implies care should be taken when solving problems involving non-differentiable functions.

For instance, the function

$$f(x) = x^2$$

is differentiable everywhere whereas the function

$$f(x) = \sqrt{x}$$

is only diffrentiable for $x > 0$. In order to make sure that MOSEK evaulates the functions at points where they are differentiable, the function domains must be defined by setting appropriate variable bounds.

In general, if a variable is not ranged MOSEK will only evaluate that variable at points strictly within the bounds. Hence, imposing the bound

$$x \geq 0$$

in the case of $\sqrt{x}$ is sufficient to guarantee that the function will only be evaluated in points where it is differentiable.

However, if a function is differentiable on closed a range, specifying the variable bounds is not sufficient. Consider the function

$$f(x) = \frac{1}{x} + \frac{1}{1-x}. \tag{8.10}$$

In this case the bounds

$$0 \leq x \leq 1$$

will not guarantee that MOSEK only evalues the function for x between 0 and 1. To force MOSEK to strictly satisfy both bounds on ranged variables set the parameter MSK_IPAR_INTPNT_STARTING_POINT to MSK_STARTING_POINT_SATISFY_BOUNDS.

For efficiency reasons it may be better to reformulate the problem than to force MOSEK to observe ranged bounds strictly. For instance, (8.10) can be reformulated as follows

$$\begin{array}{rcl} f(x) & = & \frac{1}{x} + \frac{1}{y} \\ 0 & = & 1 - x - y \\ 0 & \leq & x \\ 0 & \leq & y. \end{array}$$

8.5.1.3 Interior-point termination criteria

The parameters controlling when the general convex interior-point optimizer terminates are shown in Table 8.3.

8.6 Solving problems in parallel

If a computer has multiple CPUs, or has a CPU with multiple cores, it is possible for MOSEK to take advantage of this to speed up solution times.

Parameter name	Purpose
MSK_DPAR_INTPNT_NL_TOL_PFEAS	Controls primal feasibility
MSK_DPAR_INTPNT_NL_TOL_DFEAS	Controls dual feasibility
MSK_DPAR_INTPNT_NL_TOL_REL_GAP	Controls relative gap
MSK_DPAR_INTPNT_TOL_INFEAS	Controls when the problem is declared infeasible
MSK_DPAR_INTPNT_NL_TOL_MU_RED	Controls when the complementarity is reduced enough

Table 8.3: Parameters employed in termination criteria.

8.6.1 Thread safety

The MOSEK API is thread-safe provided that a task is only modified or accessed from one thread at any given time — accessing two separate tasks from two separate threads at the same time is safe. Sharing an environment between threads is safe.

8.6.2 The parallelized interior-point optimizer

The interior-point optimizer is capable of using multiple CPUs or cores. This implies that whenever the MOSEK interior-point optimizer solves an optimization problem, it will try to divide the work so that each CPU gets a share of the work. The user decides how many CPUs MOSEK should exploit.

It is not always possible to divide the work equally, and often parts of the computations and the coordination of the work is processed sequentially, even if several CPUs are present. Therefore, the speed-up obtained when using multiple CPUs is highly problem dependent. However, as a rule of thumb, if the problem solves very quickly, i.e. in less than 60 seconds, it is not advantageous to use the parallel option.

The MSK_IPAR_INTPNT_NUM_THREADS parameter sets the number of threads (and therefore the number of CPUs) that the interior point optimizer will use.

8.6.3 The concurrent optimizer

An alternative to the parallel interior-point optimizer is the *concurrent optimizer*. The idea of the concurrent optimizer is to run multiple optimizers on the same problem concurrently, for instance, it allows you to apply the interior-point and the dual simplex optimizers to a linear optimization problem concurrently. The concurrent optimizer terminates when the first of the applied optimizers has terminated successfully, and it reports the solution of the fastest optimizer. In that way a new optimizer has been created which essentially performs as the fastest of the interior-point and the dual simplex optimizers.Hence, the concurrent optimizer is the best one to use if there are multiple optimizers available in MOSEK for the problem and you cannot say beforehand which one will be faster.

Note in particular that any solution present in the task will also be used for hot-starting the simplex algorithms. One possible scenario would therefore be running a hot-start dual simplex in parallel with interior point, taking advantage of both the stability of the interior-point method and the ability of the simplex method to use an initial solution.

Optimizer	Associated parameter	Default priority
MSK_OPTIMIZER_INTPNT	MSK_IPAR_CONCURRENT_PRIORITY_INTPNT	4
MSK_OPTIMIZER_FREE_SIMPLEX	MSK_IPAR_CONCURRENT_PRIORITY_FREE_SIMPLEX	3
MSK_OPTIMIZER_PRIMAL_SIMPLEX	MSK_IPAR_CONCURRENT_PRIORITY_PRIMAL_SIMPLEX	2
MSK_OPTIMIZER_DUAL_SIMPLEX	MSK_IPAR_CONCURRENT_PRIORITY_DUAL_SIMPLEX	1

Table 8.4: Default priorities for optimizer selection in concurrent optimization.

By setting the

`MSK_IPAR_OPTIMIZER`

parameter to

`MSK_OPTIMIZER_CONCURRENT`

the concurrent optimizer chosen.

The number of optimizers used in parallel is determined by the

`MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS.`

parameter. Moreover, the optimizers are selected according to a preassigned priority with optimizers having the highest priority being selected first. The default priority for each optimizer is shown in Table 8.6.3. For example, setting the MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS parameter to 2 tells the concurrent optimizer to the apply the two optimizers with highest priorities: In the default case that means the interior-point optimizer and one of the simplex optimizers.

8.6.3.1 Concurrent optimization through the API

The following example shows how to call the concurrent optimizer through the API.

```
1  /*
2     Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
3
4     File:      concurrent1.c
5
6     Purpose:   To demonstrate how to solve a problem
7                with the concurrent optimizer.
8   */
9
10 #include <stdio.h>
11
12 #include "mosek.h"
13
14 static void MSKAPI printstr(void *handle,
15                             char str[])
```

```
{
  printf("%s",str);
} /* printstr */

int main(int argc,char *argv[])
{
  MSKenv_t  env;
  MSKtask_t task;
  MSKintt r = MSK_RES_OK;

  /* Create mosek environment. */
  r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);

  if ( r==MSK_RES_OK )
    MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);

  /* Initialize the environment. */
  r = MSK_initenv(env);

  if ( r==MSK_RES_OK )
    r = MSK_maketask(env,0,0,&task);

  if ( r==MSK_RES_OK )
    MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);

  if (r == MSK_RES_OK)
    r = MSK_readdata(task,argv[1]);

  MSK_putintparam(task,MSK_IPAR_OPTIMIZER,MSK_OPTIMIZER_CONCURRENT);
  MSK_putintparam(task,MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS,2);

  if (r == MSK_RES_OK)
    r = MSK_optimize(task);

  MSK_solutionsummary(task,MSK_STREAM_LOG);


  MSK_deletetask(&task);
  MSK_deleteenv(&env);

  printf("Return code: %d (0 means no error occured.)\n",r);

  return ( r );
} /* main */
```

8.6.4 A more flexible concurrent optimizer

MOSEK also provides a more flexible method of concurrent optimization by using the function MSK_optimizeconcurrent. The main advantages of this function are that it allows the calling application to assign arbitrary values to the parameters of each tasks, and that call-back functions can be attached to each task. This may be useful in the following situation: Assume that you know the primal simplex optimizer to be the best optimizer for your problem, but that you do not know which of the available selection strategies (as defined by the MSK_IPAR_SIM_PRIMAL_SELECTION parameter) is the best. In this case you can solve the problem with the primal simplex optimizer using several

different selection strategies concurrently.

An example demonstrating the usage of the MSK_optimizeconcurrent function is included below. The example solves a single problem using the interior-point and primal simplex optimizers in parallel.

```
/*
  Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

  File:      concurrent2.c

  Purpose:   To demonstrate a more flexible interface for concurrent optimization.
*/


#include "mosek.h"

static void MSKAPI printstr(void *handle,
                            char str[])
{
  printf("simplex: %s",str);
} /* printstr */

static void MSKAPI printstr2(void *handle,
                             char str[])
{
  printf("intrpnt: %s",str);
} /* printstr */

#define NUMTASKS 1

int main(int argc,char **argv)
{
  MSKintt   r=MSK_RES_OK,i;
  MSKenv_t  env;
  MSKtask_t task;
  MSKtask_t task_list[NUMTASKS];

  /* Create mosek environment. */
  r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);

  if ( r==MSK_RES_OK )
    MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);

  /* Initialize the environment. */
  if ( r==MSK_RES_OK )
    r = MSK_initenv(env);

  /* Create a task for each concurrent optimization.
     The 'task' is the master task that will hold the problem data.
  */

  if ( r==MSK_RES_OK )
    r = MSK_maketask(env,0,0,&task);

  if (r == MSK_RES_OK)
    r = MSK_maketask(env,0,0,&task_list[0]);

  if (r == MSK_RES_OK)
```

```
    r = MSK_readdata(task,argv[1]);

  /* Assign different parameter values to each task.
     In this case different optimizers. */

  if (r == MSK_RES_OK)
    r = MSK_putintparam(task,
                        MSK_IPAR_OPTIMIZER,
                        MSK_OPTIMIZER_PRIMAL_SIMPLEX);

  if (r == MSK_RES_OK)
    r = MSK_putintparam(task_list[0],
                        MSK_IPAR_OPTIMIZER,
                        MSK_OPTIMIZER_INTPNT);


  /* Assign call-back functions to each task */

  if (r == MSK_RES_OK)
    MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);

  if (r == MSK_RES_OK)
    MSK_linkfunctotaskstream(task_list[0],
                             MSK_STREAM_LOG,
                             NULL,
                             printstr2);

  if (r == MSK_RES_OK)
    r = MSK_linkfiletotaskstream(task,
                                 MSK_STREAM_LOG,
                                 "simplex.log",
                                 0);

  if (r == MSK_RES_OK)
    r = MSK_linkfiletotaskstream(task_list[0],
                                 MSK_STREAM_LOG,
                                 "intpnt.log",
                                 0);


  /* Optimize task and task_list[0] in parallel.
     The problem data i.e. C, A, etc.
     is copied from task to task_list[0].
   */

  if (r == MSK_RES_OK)
    r = MSK_optimizeconcurrent (
                                task,
                                task_list,
                                NUMTASKS);

  printf ("Return Code = %d\n",r);

  MSK_solutionsummary(task,
                      MSK_STREAM_LOG);
  return r;
}
```

8.7 Understanding solution quality

MOSEK will, in general, not produce an *exact* optimal solution; for efficiency reasons computations are performed in finite precision. This means that it is important to evaluate the quality of the reported solution. To evaluate the solution quality inspect the following properties:

- The solution status reported by MOSEK.
- Primal feasibility: How much the solution violates the original constraints of the problem.
- Dual feasibility: How much the dual solution violates the constraints of the dual problem.
- Duality gap: The difference between the primal and dual objective values.

Ideally, the primal and dual solutions should only violate the constraints of their respective problem *slightly* and the primal and dual objective values should be *close*. This should be evaluated in the context of the problem: How good is the data precision in the problem, and how exact a solution is required.

8.7.1 The solution summary

The solution summary is a small display generated by MOSEK that makes it easy to check the quality of the solution.

The function `MSK_solutionsummary` should be used to generate solution summary.

8.7.1.1 The optimal case

The solution summary has the format

```
Problem status  : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL
Primal - objective: 5.5018458883e+03   eq. infeas.: 1.20e-12 max bound infeas.: 2.31e-14
Dual   - objective: 5.5018458883e+03   eq. infeas.: 1.15e-14 max bound infeas.: 7.11e-15
```

i.e. it shows status information, objective values and quality measures for the primal and dual solutions.

Assumeing that we are solving a linear optimization problem and referring to the problems (7.1) and (7.2), the interpretation of the solution summary is as follows:

- Problem status: The status of the problem.
- Solution status: The status of the solution.
- Primal objective: The primal objective value.
- Primal eq. infeas: $\|Ax^x - x^c\|_\infty$.
- Primal max bound infeas.: $\max(l^c - x^c; x^c - u^c; l^x - x^x; x^x - u^x; 0)$.

- Dual objective: The dual objective value.
- Dual eq. infeas: $\left\|-y + s_l^c - s_u^c; A^T y + s_l^x - s_u^x - c\right\|_\infty$.
- Dual max bound infeas.: $\max(-s_l^c; -s_u^c; -s_l^x; -s_u^x; 0)$.

In the solution summary above the solution is classified as `OPTIMAL`, meaning that the solution should be a good approximation to the true optimal solution. This seems very reasonable since the primal and dual solutions only violate their respective constraints slightly. Moreover, the duality gap is small, i.e. the primal and dual objective values are almost identical.

8.7.1.2 The primal infeasible case

For an infeasible problem the solution summary might look like this:

```
Problem status  : PRIMAL_INFEASIBLE
Solution status : PRIMAL_INFEASIBLE_CER
Primal - objective: 0.0000000000e+00   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00
Dual   - objective: 1.0000000000e+02   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00
```

It is known that if the problem is primal infeasible then an infeasibility certificate exists, which is a solution to the problem (7.3) having a positive objective value. Note that the primal solution plays no role and only the dual solution is used to specify the certificate.

Therefore, in the primal infeasible case the solution summery should report how good the dual solution is to the problem (7.3). The interpretation of the solution summary is as follows:

- Problem status: The status of the problem.
- Solution status: The status of the solution.
- Primal objective: Should be ignored.
- Primal eq. infeas: Should be ignored.
- Primal max bound infeas.: Should be ignored.
- Dual objective: $(l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x$.
- Dual eq. infeas: $\left\|-y + s_l^c - s_u^c; A^T y + s_l^x - s_u^x - 0\right\|_\infty$.
- Dual max bound infeas.: $\max(-s_l^c; -s_u^c; -s_l^x; -s_u^x)$.

Please note that

- any information about the primal solution should be ignored.
- the dual objective value should be strictly positive if primal problem is minimization problem. Otherwise it should be strictly negative.

- the bigger the ratio

$$\frac{(l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x}{\max(\|-y + s_l^c - s_u^c; A^T y + s_l^x - s_u^x - 0\|_\infty, \max(-s_l^c; -s_u^c; -s_l^x; -s_u^x))}$$

  is, the better the certificate is. The reason is that a certificate is a ray, and hence only the direction is important. Therefore, in principle, the certificate should be normalized before using it.

Please see Section 10.2 for more information about certificates of infeasibility.

8.7.2 Retrieving solution quality information with the API

Information about the solution quality may be retrieved in the API with the help of the following functions:

- MSK_getsolutioninf: Obtains maximum infeasibility and primal/dual objective value.
- MSK_analyzesolution: Print additional information about the solution, e.g basis condition number and optionally a list of violated constraints.
- MSK_getpeqi, MSK_getdeqi, MSK_getpbi, MSK_getdbi, MSK_getdcni, MSK_getpcni: Obtains violation of the individual constraints.

The example below shows how to use these function to determine the quality of the solution.

```
/*
  Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

  File:    solutionquality.c

  Purpose: To demonstrate how to examine the quality of a solution.
*/
#include "mosek.h"
#include "math.h"

static void MSKAPI printstr(void *handle,
                            char str[])
{
  printf("%s",str);
} /* printstr */


double double_max(double arg1,double arg2)
{
  return arg1<arg2 ? arg2 : arg1;
}

int main (int argc, char * argv[])
{
  MSKtask_t   task = NULL;
  MSKenv_t    env  = NULL;
```

```
27  MSKrescodee r  = MSK_RES_OK;
28
29  if (argc <= 1)
30  {
31    printf ("Missing argument. The syntax is:\n");
32    printf (" simple inputfile [ solutionfile ]\n");
33  }
34  else
35  {
36    /* Create the mosek environment.
37       The 'NULL' arguments here, are used to specify customized
38       memory allocators and a memory debug file. These can
39       safely be ignored for now. */
40
41    r = MSK_makeenv(&env, NULL, NULL, NULL, NULL);
42
43    /* Initialize the environment */
44    if ( r==MSK_RES_OK )
45      MSK_initenv (env);
46
47    /* Create a task object linked to the environment env.
48       Initially we create it with 0 variables and 0 columns,
49       since we do not know the size of the problem. */
50    if ( r==MSK_RES_OK )
51      r = MSK_maketask (env, 0,0, &task);
52
53    if (r == MSK_RES_OK)
54      MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
55
56    /* We assume that a problem file was given as the first command
57       line argument (received in 'argv'). */
58    if ( r==MSK_RES_OK )
59      r = MSK_readdata (task, argv[1]);
60
61
62    /* Solve the problem */
63    if ( r==MSK_RES_OK )
64    {
65      MSKrescodee trmcode;
66
67      MSK_optimizetrm(task,&trmcode);
68    }
69
70    /* Print a summary of the solution. */
71    MSK_solutionsummary(task, MSK_STREAM_MSG);
72
73    if (r == MSK_RES_OK)
74    {
75      MSKprostae prosta;
76      MSKsolstae  solsta;
77      MSKrealt primalobj,maxpbi,maxpcni,maxpeqi,maxinti,
78        dualobj, maxdbi, maxdcni, maxdeqi;
79      MSKintt isdef;
80      MSKsoltypee whichsol = MSK_SOL_BAS;
81      int accepted = 1;
82
83
84      MSK_getsolutioninf (
```

```
                          task,
                          whichsol,
                          &prosta,
                          &solsta,
                          &primalobj,
                          &maxpbi,
                          &maxpcni,
                          &maxpeqi,
                          &maxinti,
                          &dualobj,
                          &maxdbi,
                          &maxdcni,
                          &maxdeqi);

        switch(solsta)
        {
          case MSK_SOL_STA_OPTIMAL:
          case MSK_SOL_STA_NEAR_OPTIMAL:
            {
              double max_primal_infeas = 0.0; /* maximal primal infeasibility */
              double max_dual_infeas   = 0.0; /* maximal dual infeasibility */
              double obj_gap = fabs(dualobj-primalobj);


              max_primal_infeas = double_max(max_primal_infeas,maxpbi);
              max_primal_infeas = double_max(max_primal_infeas,maxpcni);
              max_primal_infeas = double_max(max_primal_infeas,maxpeqi);

              max_dual_infeas = double_max(max_dual_infeas,maxdbi);
              max_dual_infeas = double_max(max_dual_infeas,maxdcni);
              max_dual_infeas = double_max(max_dual_infeas,maxdeqi);

              /* Assume the application needs the solution to be within
                 1e-6 ofoptimality in an absolute sense. Another approach
                 would be looking at the relative objective gap */
              printf("Objective gap: %e\n",obj_gap);
              if (obj_gap > 1e-6)
              {
                printf("Warning: The objective gap is too large.");
                accepted = 0;
              }

              printf("Max primal infeasibility: %e\n", max_primal_infeas);
              printf("Max dual infeasibility: %e\n"  , max_dual_infeas);

              /* We will accept a primal infeasibility of 1e-8 and
                 dual infeasibility of 1e-6 */

              if (max_primal_infeas > 1e-8)
              {
                printf("Warning: Primal infeasibility is too large");
                accepted = 0;
              }

              if (max_dual_infeas > 1e-6)
              {
                printf("Warning: Dual infeasibility is too large");
                accepted = 0;
```

```
143               }
144             }
145
146             if (accepted && r == MSK_RES_OK)
147             {
148               MSKintt numvar,j;
149               MSKrealt *xx = NULL;
150
151               MSK_getnumvar(task,&numvar);
152
153               xx = (double *) malloc(numvar*sizeof(MSKrealt));
154
155               MSK_getsolutionslice(task,
156                                    MSK_SOL_BAS,    /* Request the basic solution. */
157                                    MSK_SOL_ITEM_XX,/* Which part of solution.     */
158                                    0,              /* Index of first variable.    */
159                                    numvar,         /* Index of last variable+1.   */
160                                    xx);
161
162
163               printf("Optimal primal solution\n");
164               for(j=0; j<numvar; ++j)
165                 printf("x[%d]: %e\n",j,xx[j]);
166
167               free(xx);
168             }
169             else
170             {
171               /* Print detailed information about the solution */
172               if (r == MSK_RES_OK)
173                 r = MSK_analyzesolution(task,MSK_STREAM_LOG,whichsol);
174             }
175             break;
176           case MSK_SOL_STA_DUAL_INFEAS_CER:
177           case MSK_SOL_STA_PRIM_INFEAS_CER:
178           case MSK_SOL_STA_NEAR_DUAL_INFEAS_CER:
179           case MSK_SOL_STA_NEAR_PRIM_INFEAS_CER:
180             printf("Primal or dual infeasibility certificate found.\n");
181             break;
182           case MSK_SOL_STA_UNKNOWN:
183             printf("The status of the solution could not be determined.\n");
184             break;
185           default:
186             printf("Other solution status");
187             break;
188         }
189     }
190     else
191     {
192       printf("Error while optimizing.\n");
193     }
194
195     MSK_deletetask(&task);
196     MSK_deleteenv(&env);
197   }
198   return r;
199 }
```

Chapter 9

The optimizer for mixed integer problems

A problem is a mixed-integer optimization problem when one or more of the variables are constrained to be integers. The integer optimizer available in MOSEK can solve integer optimization problems involving

- linear,
- quadratic and
- conic

constraints. However, a problem is not allowed to have both conic constraints and quadratic objective or constraints.

Readers unfamiliar with integer optimization are strongly recommended to consult some relevant literature, e.g. the book [27] by Wolsey is a good introduction to integer optimization.

9.1 Some notation

In general, an integer optimization problem has the form

$$\begin{array}{rcllccll} z^* & = & \text{minimize} & & & c^T x & & \\ & & \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & & & l^x & \leq & x & \leq & u^x, \\ & & & & & x_j \in \mathcal{Z}, & & \forall j \in \mathcal{J}, \end{array} \tag{9.1}$$

where $\mathcal{J}$ is an index set specifying which variables are integer-constrained. Frequently we talk about the continuous relaxation of an integer optimization problem defined as

$$\begin{array}{rcllcll} \underline{z} & = & \text{minimize} & & c^T x & & \\ & & \text{subject to} & l^c & \leq \quad Ax & \leq & u^c, \\ & & & l^x & \leq \quad x & \leq & u^x \end{array} \tag{9.2}$$

i.e. we ignore the constraint

$$x_j \in \mathcal{Z},\ \forall j \in \mathcal{J}.$$

Moreover, let $\hat{x}$ be any feasible solution to (9.1) and define

$$\overline{z} := c^T \hat{x}.$$

It should be obvious that

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

holds. This is an important observation since if we assume that it is not possible to solve the mixed-integer optimization problem within a reasonable time frame, but that a feasible solution can be found, then the natural question is: How far is the *obtained* solution from the *optimal* solution? The answer is that no feasible solution can have an objective value smaller than $\underline{z}$, which implies that the obtained solution is no further away from the optimum than $\overline{z} - \underline{z}$.

9.2 An important fact about integer optimization problems

It is important to understand that in a worst-case scenario, the time required to solve integer optimization problems grows exponentially with the size of the problem. For instance, assume that a problem contains n binary variables, then the time required to solve the problem in the worst case may be proportional to 2^n. It is a simple exercise to verify that 2^n is huge even for moderate values of n.

In practice this implies that the focus should be on computing a near optimal solution quickly rather than at locating an optimal solution.

9.3 How the integer optimizer works

The process of solving an integer optimization problem can be split in three phases:

Presolve: In this phase the optimizer tries to reduce the size of the problem using preprocessing techniques. Moreover, it strengthens the continuous relaxation, if possible.

Heuristic: Using heuristics the optimizer tries to guess a good feasible solution.

Optimization: The optimal solution is located using a variant of the branch-and-cut method.

In some cases the integer optimizer may locate an optimal solution in the preprocessing stage or conclude that the problem is infeasible. Therefore, the heuristic and optimization stages may never be performed.

9.3.1 Presolve

In the preprocessing stage redundant variables and constraints are removed. The presolve stage can be turned off using the MSK_IPAR_MIO_PRESOLVE_USE parameter.

9.3.2 Heuristic

Initially, the integer optimizer tries to guess a good feasible solution using different heuristics:

- First a very simple rounding heuristic is employed.
- Next, if deemed worthwhile, the *feasibility pump* heuristic is used.
- Finally, if the two previous stages did not produce a good initial solution, more sophisticated heuristics are used.

The following parameters can be used to control the effort made by the integer optimizer to find an initial feasible solution.

- MSK_IPAR_MIO_HEURISTIC_LEVEL: Controls how sophisticated and computationally expensive a heuristic to employ.
- MSK_DPAR_MIO_HEURISTIC_TIME: The minimum amount of time to spend in the heuristic search.
- MSK_IPAR_MIO_FEASPUMP_LEVEL: Controls how aggressively the feasibility pump heuristic is used.

9.3.3 The optimization phase

This phase solves the problem using the branch and cut algorithm.

9.4 Termination criterion

In general, it is impossible to find an exact feasible and optimal solution to an integer optimization problem in a reasonable amount of time, though in many practical cases it may be possible. Therefore, the integer optimizer employs a relaxed feasibility and optimality criterion to determine when a satisfactory solution is located.

A candidate solution, i.e. a solution to (9.2), is said to be an integer feasible solution if the criterion

$$\min(|x_j| - \lfloor x_j \rfloor, \lceil x_j \rceil - |x_j|) \leq \max(\delta_1, \delta_2 |x_j|) \ \forall j \in \mathcal{J}$$

is satisfied. Hence, such a solution is defined as a feasible solution to (9.1).

Whenever the integer optimizer locates an integer feasible solution it will check if the criterion

$$\overline{z} - \underline{z} \leq \max(\delta_3, \delta_4 \max(1, |\overline{z}|))$$

Tolerance	Parameter name
δ_1	MSK_DPAR_MIO_TOL_ABS_RELAX_INT
δ_2	MSK_DPAR_MIO_TOL_REL_RELAX_INT
δ_3	MSK_DPAR_MIO_TOL_ABS_GAP
δ_4	MSK_DPAR_MIO_TOL_REL_GAP
δ_5	MSK_DPAR_MIO_NEAR_TOL_ABS_GAP
δ_6	MSK_DPAR_MIO_NEAR_TOL_REL_GAP

Table 9.1: Integer optimizer tolerances.

Parameter name	Delayed	Explanation
MSK_IPAR_MIO_MAX_NUM_BRANCHES	Yes	Maximum number of branches allowed.
MSK_IPAR_MIO_MAX_NUM_RELAXS	Yes	Maximum number of relaxations allowed.
MSK_IPAR_MIO_MAX_NUM_SOLUTIONS	Yes	Maximum number of feasible integer solutions allowed.

Table 9.2: Parameters affecting the termination of the integer optimizer.

is satisfied. If this is the case, the integer optimizer terminates and reports the integer feasible solution as an optimal solution. Please note that $\underline{z}$ is a valid lower bound determined by the integer optimizer during the solution process, i.e.

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

The lower bound $\underline{z}$ normally increases during the solution process.

The δ tolerances can are specified using parameters — see Table 9.1. If an optimal solution cannot be located within a reasonable time, it may be advantageous to employ a relaxed termination criterion after some time. Whenever the integer optimizer locates an integer feasible solution and has spent at least the number of seconds defined by the MSK_DPAR_MIO_DISABLE_TERM_TIME parameter on solving the problem, it will check whether the criterion

$$\overline{z} - \underline{z} \leq \max(\delta_5, \delta_6 \max(1, |\overline{z}|))$$

is satisfied. If it is satisfied, the optimizer will report that the candidate solution is **near optimal** and then terminate. All δ tolerances can be adjusted using suitable parameters — see Table 9.1. In Table 9.2 some other parameters affecting the integer optimizer termination criterion are shown. Please note that if the effect of a parameter is delayed, the associated termination criterion is applied only after some time, specified by the MSK_DPAR_MIO_DISABLE_TERM_TIME parameter.

9.5 How to speed up the solution process

As mentioned previously, in many cases it is not possible to find an optimal solution to an integer optimization problem in a reasonable amount of time. Some suggestions to reduce the solution time are:

- Relax the termination criterion: In case the run time is not acceptable, the first thing to do is to relax the termination criterion — see Section 9.4 for details.

- Specify a good initial solution: In many cases a good feasible solution is either known or easily computed using problem specific knowledge. If a good feasible solution is known, it is usually worthwhile to use this as a starting point for the integer optimizer.
- Improve the formulation: A mixed-integer optimization problem may be impossible to solve in one form and quite easy in another form. However, it is beyond the scope of this manual to discuss good formulations for mixed-integer problems. For discussions on this topic see for example [27].

9.6 Understanding solution quality

To determine the quality of the solution one should check the following:

- The solution status key returned by MOSEK.
- The *optimality gap*: A messure for how much the located solution can deviate from the optimal solution to the problem.
- Feasibility. How much the solution violates the constraints of the problem.

The *optimality gap* is a measure for how close the solution is to the optimal solution. The optimality gap is given by

$$\epsilon = |(\text{objective value of feasible solution}) - (\text{objective bound})|.$$

The objective value of the solution is guarentted to be within ϵ of the optimal solution.

The optimality gap can be retrived through the solution item MSK_DINF_MIO_OBJ_ABS_GAP. Often it is more meaningful to look at the optimality gap normalized with the magnitude of the solution. The relative optimality gap is available in MSK_DINF_MIO_OBJ_REL_GAP.

9.6.1 Solutionsummary

The function MSK_solutionsummary prints the most important solution information to the screen.

After a call to the optimizer the solution summary might look like this:

```
Problem status  : PRIMAL_FEASIBLE
Solution status : INTEGER_OPTIMAL
Primal - objective: 1.2015000000e+06   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00
cone infeas.: 0.00e+00 integer infeas.: 0.00e+00
```

The second line contains the solution status key. This shows how MOSEK classified the solution. In this case it is INTEGER_OPTIMAL, meaning that the solution is considered to be optimal within the selected tolerances.

The third line contains information relating to the solution. The first number is the primal objective function. The second and third number is the maximum infeasibility in the equality constraints and

bounds respectfully. The fourth and fifth number is the maximum infeasibility in the conic and integral contraints. All the numbers relating to the feasibility of the solution should be small for the solution to be valid.

9.6.2 Retrieving solution quality information with the API

Information about the solution quality may be retrieved in the API with the help of the following functions:

- MSK_getsolutioninf: Obtains maximum infeasibility.
- MSK_analyzesolution Print additional information about the solution. E.g basis condition number and optionally a list of violated constraints.
- MSK_getpeqi, MSK_getdeqi, MSK_getpbi, MSK_getdbi, MSK_getdcni, MSK_getpcni,MSK_getinti: Obtains violation of the individual constraints.

Chapter 10

The analyzers

10.1 The problem analyzer

The problem analyzer prints a detailed survey of the model's

- linear constraints and objective
- quadratic constraints
- conic constraints
- variables

In the initial stages of model formulation the problem analyzer may be used as a quick way of verifying that the model has been built or imported correctly. In later stages it can help revealing special structures within the model that may be used to tune the optimizer's performance or to identify the causes of numerical difficulties.

The problem analyzer is run from the command line using the `-anapro` argument and produces something similar to the following (this is the problemanalyzer's survey of the `aflow30a` problem from the MIPLIB 2003 collection, see Appendix B for more examples):

```
Analyzing the problem

Constraints               Bounds                    Variables
 upper bd:       421       ranged  : all             cont:       421
 fixed   :        58                                 bin :       421

--------------------------------------------------------------------------------

Objective, min cx
   range: min |c|: 0.00000   min |c|>0: 11.0000     max |c|: 500.000
 distrib:        |c|        vars
                   0         421
```

```
          [11, 100)         150
         [100, 500]         271

-----------------------------------------------------------------------------

Constraint matrix A has
       479 rows (constraints)
       842 columns (variables)
      2091 (0.518449%) nonzero entries (coefficients)

Row nonzeros, A_i
   range: min A_i: 2 (0.23753%)    max A_i: 34 (4.038%)
 distrib:        A_i        rows       rows%        acc%
                   2         421       87.89       87.89
             [8, 15]          20        4.18       92.07
            [16, 31]          30        6.26       98.33
            [32, 34]           8        1.67      100.00

Column nonzeros, A|j
   range: min A|j: 2 (0.417537%)    max A|j: 3 (0.626305%)
 distrib:        A|j        cols       cols%        acc%
                   2         435       51.66       51.66
                   3         407       48.34      100.00

A nonzeros, A(ij)
   range: min |A(ij)|: 1.00000     max |A(ij)|: 100.000
 distrib:      A(ij)      coeffs
             [1, 10)        1670
           [10, 100]         421

-----------------------------------------------------------------------------

Constraint bounds, lb <= Ax <= ub
 distrib:        |b|              lbs             ubs
                   0                               421
             [1, 10]               58              58

Variable bounds, lb <= x <= ub
 distrib:        |b|              lbs             ubs
                   0              842
             [1, 10)                               421
           [10, 100]                               421

-----------------------------------------------------------------------------
```

The survey is divided into six different sections, each described below. To keep the presentation short with focus on key elements the analyzer generally attempts to display information on issues relevant for the current model only: E.g., if the model does not have any conic constraints (this is the case in the example above) or any integer variables, those parts of the analysis will not appear.

10.1.1 General characteristics

The first part of the survey consists of a brief summary of the model's linear and quadratic constraints (indexed by i) and variables (indexed by j). The summary is divided into three subsections:

`Constraints`

`upper bd`: The number of upper bounded constraints, $\sum_{j=0}^{n-1} a_{ij}x_j \leq u_i^c$

`lower bd`: The number of lower bounded constraints, $l_i^c \leq \sum_{j=0}^{n-1} a_{ij}x_j$

`ranged` : The number of ranged constraints, $l_i^c \leq \sum_{j=0}^{n-1} a_{ij}x_j \leq u_i^c$

`fixed` : The number of fixed constraints, $l_i^c = \sum_{j=0}^{n-1} a_{ij}x_j = u_i^c$

`free` : The number of free constraints

`Bounds`

`upper bd`: The number of upper bounded variables, $x_j \leq u_j^x$

`lower bd`: The number of lower bounded variables, $l_k^x \leq x_j$

`ranged` : The number of ranged variables, $l_k^x \leq x_j \leq u_j^x$

`fixed` : The number of fixed variables, $l_k^x = x_j = u_j^x$

`free` : The number of free variables

`Variables`

`cont`: The number of continuous variables, $x_j \in \mathbb{R}$

`bin` : The number of binary variables, $x_j \in \{0, 1\}$

`int` : The number of general integer variables, $x_j \in \mathbb{Z}$

Only constraints, bounds and domains actually in the model will be reported on, cf. appendix B; if all entities in a section turn out to be of the same kind, the number will be replaced by `all` for brevity.

10.1.2 Objective

The second part of the survey focuses on (the linear part of) the objective, summarizing the optimization sense and the coefficients' absolute value range and distribution. The number of 0 (zero) coefficients is singled out (if any such variables are in the problem).

The range is displayed using three terms:

`min |c|`: The minimum absolute value among all coeffecients

`min |c|>0`: The minimum absolute value among the nonzero coefficients

`max |c|`: The maximum absolute value among the coefficients

If some of these extrema turn out to be equal, the display is shortened accordingly:

- If `min |c|` is greater than zero, the `min |c|>0` term is obsolete and will not be displayed
- If only one or two different coefficients occur this will be displayed using `all` and an explicit listing of the coefficients

The absolute value distribution is displayed as a table summarizing the numbers by orders of magnitude (with a ratio of 10). Again, the number of variables with a coefficient of 0 (if any) is singled out. Each line of the table is headed by an interval (half-open intervals including their lower bounds), and is followed by the number of variables with their objective coefficient in this interval. Intervals with no elements are skipped.

10.1.3 Linear constraints

The third part of the survey displays information on the nonzero coefficients of the linear constraint matrix.

Following a brief summary of the matrix dimensions and the number of nonzero coefficients in total, three sections provide further details on how the nonzero coefficients are distributed by row-wise count (`A_i`), by column-wise count (`A|j`), and by absolute value (`|A(ij)|`). Each section is headed by a brief display of the distribution's range (`min` and `max`), and for the row/column-wise counts the corresponding densities are displayed too (in parentheses).

The distribution tables single out three particularly interesting counts: zero, one, and two nonzeros per row/column; the remaining row/column nonzeros are displayed by orders of magnitude (ratio 2). For each interval the relative and accumulated relative counts are also displayed.

Note that constraints may have both linear and quadratic terms, but the empty rows and columns reported in this part of the survey relate to the linear terms only. If empty rows and/or columns are found in the linear constraint matrix, the problem is analyzed further in order to determine if the corresponding constraints have any quadratic terms or the corresponding variables are used in conic or quadratic constraints; cf. the last two examples of appendix B.

The distribution of the absolute values, `|A(ij)|`, is displayed just as for the objective coefficients described above.

10.1.4 Constraint and variable bounds

The fourth part of the survey displays distributions for the absolute values of the finite lower and upper bounds for both constraints and variables. The number of bounds at 0 is singled out and, otherwise, displayed by orders of magnitude (with a ratio of 10).

10.1.5 Quadratic constraints

The fifth part of the survey displays distributions for the nonzero elements in the gradient of the quadratic constraints, i.e. the nonzero row counts for the column vectors Qx. The table is similar to

the tables for the linear constraints' nonzero row and column counts described in the survey's third part.

Note: Quadratic constraints may also have a linear part, but that will be included in the linear constraints survey; this means that if a problem has one or more pure quadratic constraints, part three of the survey will report an equal number of linear constraint rows with 0 (zero) nonzeros, cf. the last example in appendix B. Likewise, variables that appear in quadratic terms only will be reported as empty columns (0 nonzeros) in the linear constraint report.

10.1.6 Conic constraints

The last part of the survey summarizes the model's conic constraints. For each of the two types of cones, quadratic and rotated quadratic, the total number of cones are reported, and the distribution of the cones' dimensions are displayed using intervals. Cone dimensions of 2, 3, and 4 are singled out.

10.2 Analyzing infeasible problems

When developing and implementing a new optimization model, the first attempts will often be either infeasible, due to specification of inconsistent constraints, or unbounded, if important constraints have been left out.

In this chapter we will

- go over an example demonstrating how to locate infeasible constraints using the MOSEK infeasibility report tool,
- discuss in more general terms which properties that may cause infeasibilities, and
- present the more formal theory of infeasible and unbounded problems.

Furthermore, chapter 11 contains a discussion on a specific method for repairing infeasibility problems where infeasibilities are caused by model parameters rather than errors in the model or the implementation.

10.2.1 Example: Primal infeasibility

A problem is said to be *primal infeasible* if no solution exists that satisfy all the constraints of the problem.

As an example of a primal infeasible problem consider the problem of minimizing the cost of transportation between a number of production plants and stores: Each plant produces a fixed number of goods, and each store has a fixed demand that must be met. Supply, demand and cost of transportation per unit are given in figure 10.1.

The problem represented in figure 10.1 is infeasible, since the total demand

$$2300 = 1100 + 200 + 500 + 500 \tag{10.1}$$

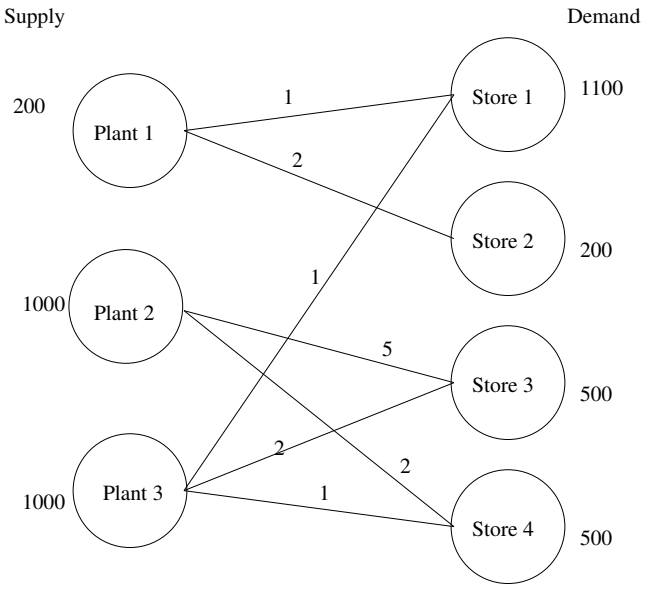

Figure 10.1: Supply, demand and cost of transportation.

exceeds the total supply

$$2200 = 200 + 1000 + 1000 \tag{10.2}$$

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

$$\begin{array}{lrcrcrcrcrcrcrl}
\text{minimize} & x_{11} & + & 2x_{12} & + & 5x_{23} & + & 2x_{24} & + & x_{31} & + & 2x_{33} & + & x_{34} & \\
\text{subject to} & x_{11} & + & x_{12} & & & & & & & & & & & \leq \ 200, \\
 & & & & & x_{23} & + & x_{24} & & & & & & & \leq \ 1000, \\
 & & & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq \ 1000, \\
 & x_{11} & & & & & & & + & x_{31} & & & & & = \ 1100, \\
 & & & x_{12} & & & & & & & & & & & = \ 200, \\
 & & & & & x_{23} & + & & & & & x_{33} & & & = \ 500, \\
 & & & & & & & x_{24} & + & & & & & x_{34} & = \ 500, \\
x_{ij} \geq 0. & & & & & & & & & & & & & &
\end{array} \tag{10.3}$$

Solving the problem (10.3) using MOSEK will result in a solution, a solution status and a problem status. Among the log output from the execution of MOSEK on the above problem are the lines:

```
Basic solution
Problem status  : PRIMAL_INFEASIBLE
Solution status : PRIMAL_INFEASIBLE_CER
```

The first line indicates that the problem status is primal infeasible. The second line says that a *certificate of the infeasibility* was found. The certificate is returned in place of the solution to the

problem.

10.2.2 Locating the cause of primal infeasibility

Usually a primal infeasible problem status is caused by a mistake in formulating the problem and therefore the question arises: "What is the cause of the infeasible status?" When trying to answer this question, it is often advantageous to follow these steps:

- Remove the objective function. This does not change the infeasible status but simplifies the problem, eliminating any possibility of problems related to the objective function.
- Consider whether your problem has some necessary conditions for feasibility and examine if these are satisfied, e.g. total supply should be greater than or equal to total demand.
- Verify that coefficients and bounds are reasonably sized in your problem.

If the problem is still primal infeasible, some of the constraints must be relaxed or removed completely. The MOSEK infeasibility report (Section 10.2.4) may assist you in finding the constraints causing the infeasibility.

Possible ways of relaxing your problem include:

- Increasing (decreasing) upper (lower) bounds on variables and constraints.
- Removing suspected constraints from the problem.

Returning to the transportation example, we discover that removing the fifth constraint

$$x_{12} = 200 \tag{10.4}$$

makes the problem feasible.

10.2.3 Locating the cause of dual infeasibility

A problem may also be *dual infeasible*. In this case the primal problem is often unbounded, mening that feasbile solutions exists such that the objective tends towards infinity. An example of a dual infeasible and primal unbounded problem is:

$$\begin{array}{ll} \text{minimize} & x_1 \\ \text{subject to} & x_1 \leq 5. \end{array} \tag{10.5}$$

To resolve a dual infeasibility the primal problem must be made more restricted by

- Adding upper or lower bounds on variables or constraints.
- Removing variables.
- Changing the objective.

10.2.3.1 A cautious note

The problem

$$\begin{array}{ll} \text{minimize} & 0 \\ \text{subject to} & 0 \leq x_1, \\ & x_j \leq x_{j+1}, \quad j = 1, \ldots, n-1, \\ & x_n \leq -1 \end{array} \tag{10.6}$$

is clearly infeasible. Moreover, if any one of the constraints are dropped, then the problem becomes feasible.

This illustrates the worst case scenario that all, or at least a significant portion, of the constraints are involved in the infeasibility. Hence, it may not always be easy or possible to pinpoint a few constraints which are causing the infeasibility.

10.2.4 The infeasibility report

MOSEK includes functionality for diagnosing the cause of a primal or a dual infeasibility. It can be turned on by setting the MSK_IPAR_INFEAS_REPORT_AUTO to MSK_ON. This causes MOSEK to print a report on variables and constraints involved in the infeasibility.

The MSK_IPAR_INFEAS_REPORT_LEVEL parameter controls the amount of information presented in the infeasibility report. The default value is 1.

10.2.4.1 Example: Primal infeasibility

We will reuse the example (10.3) located in `infeas.lp`:

```
\
\ An example of an infeasible linear problem.
\
minimize
 obj: + 1 x11 + 2 x12 + 1 x13
      + 4 x21 + 2 x22 + 5 x23
      + 4 x31 + 1 x32 + 2 x33
st
  s0: + x11 + x12      <= 200
  s1: + x23 + x24      <= 1000
  s2: + x31 +x33 + x34 <= 1000
  d1: + x11 + x31       = 1100
  d2: + x12             = 200
  d3: + x23 + x33       = 500
  d4: + x24 + x34       = 500
bounds
end
```

Using the command line

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

MOSEK produces the following infeasibility report

```
MOSEK PRIMAL INFEASIBILITY REPORT.

Problem status: The problem is primal infeasible

The following constraints are involved in the primal infeasibility.

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

The following bound constraints are involved in the infeasibility.

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

The infeasibility report is divided into two sections where the first section shows which constraints that are important for the infeasibility. In this case the important constraints are the ones named `s0`, `s2`, `d1`, and `d2`. The values in the columns "`Dual lower`" and "`Dual upper`" are also useful, since a non-zero *dual lower* value for a constraint implies that the lower bound on the constraint is important for the infeasibility. Similarly, a non-zero *dual upper* value implies that the upper bound on the constraint is important for the infeasibility.

It is also possible to obtain the infeasible subproblem. The command line

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

produces the files `rinfeas.bas.inf.lp`. In this case the content of the file `rinfeas.bas.inf.lp` is

```
minimize
 Obj: + CFIXVAR
st
 s0: + x11 + x12 <= 200
 s2: + x31 + x33 + x34 <= 1e+003
 d1: + x11 + x31 = 1.1e+003
 d2: + x12 = 200
bounds
 x11 free
 x12 free
```

```
 x13 free
 x21 free
 x22 free
 x23 free
 x31 free
 x32 free
 x24 free
 CFIXVAR = 0e+000
end
```

which is an optimization problem. This problem is identical to (10.3), except that the objective and some of the constraints and bounds have been removed. Executing the command

```
mosek -d MSK_IPAR_INFEAS_REPORT_AUTO MSK_ON rinfeas.bas.inf.lp
```

demonstrates that the reduced problem is **primal infeasible**. Since the reduced problem is usually smaller than original problem, it should be easier to locate the cause of the infeasibility in this rather than in the original (10.3).

10.2.4.2 Example: Dual infeasibility

The example problem

```
maximize -  200 y1 - 1000 y2 - 1000 y3
         - 1100 y4 -  200 y5 -  500 y6
         -  500 y7
subject to
   x11: y1+y4 < 1
   x12: y1+y5 < 2
   x23: y2+y6 < 5
   x24: y2+y7 < 2
   x31: y3+y4 < 1
   x33: y3+y6 < 2
   x44: y3+y7 < 1
bounds
   y1 < 0
   y2 < 0
   y3 < 0
   y4 free
   y5 free
   y6 free
   y7 free
end
```

is dual infeasible. This can be verified by proving that

```
y1=-1, y2=-1, y3=0, y4=1, y5=1
```

is a certificate of dual infeasibility. In this example the following infeasibility report is produced (slightly edited):

```
The following constraints are involved in the infeasibility.

Index    Name            Activity        Objective       Lower bound     Upper bound
0        x11             -1.000000e+00                   NONE            1.000000e+00
4        x31             -1.000000e+00                   NONE            1.000000e+00

The following variables are involved in the infeasibility.

Index    Name            Activity        Objective       Lower bound     Upper bound
3        y4              -1.000000e+00   -1.100000e+03   NONE            NONE
Interior-point solution
Problem status  : DUAL_INFEASIBLE
Solution status : DUAL_INFEASIBLE_CER
Primal - objective: 1.1000000000e+03   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00 cone infeas.: 0.00e+00
Dual   - objective: 0.0000000000e+00   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00 cone infeas.: 0.00e+00
```

Let x^* denote the reported primal solution. MOSEK states

- that the problem is *dual infeasible*,
- that the reported solution is a certificate of dual infeasibility, and
- that the infeasibility measure for x^* is approximately zero.

Since it was an maximization problem, this implies that

$$c^t x^* > 0. \tag{10.7}$$

For a minimization problem this inequality would have been reversed — see (10.19).

From the infeasibility report we see that the variable `y4`, and the constraints `x11` and `x33` are involved in the infeasibility since these appear with non-zero values in the "`Activity`" column.

One possible strategy to "fix" the infeasibility is to modify the problem so that the certificate of infeasibility becomes invalid. In this case we may do one the the following things:

- Put a lower bound in `y3`. This will directly invalidate the certificate of dual infeasibility.
- Increase the object coefficient of `y3`. Changing the coefficients sufficiently will invalidate the inequality (10.7) and thus the certificate.
- Put lower bounds on `x11` or `x31`. This will directly invalidate the certificate of infeasibility.

Please note that modifying the problem to invalidate the reported certificate does *not* imply that the problem becomes dual feasible — the infeasibility may simply "move", resulting in a new infeasibility.

More often, the reported certificate can be used to give a hint about errors or inconsistencies in the model that produced the problem.

10.2.5 Theory concerning infeasible problems

This section discusses the theory of infeasibility certificates and how MOSEK uses a certificate to produce an infeasibility report. In general, MOSEK solves the problem

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x \end{array} \tag{10.8}$$

where the corresponding dual problem is

$$\begin{array}{lcccc} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & c, & \\ & -y + s_l^c - s_u^c & = & 0, & \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & & \end{array} \tag{10.9}$$

We use the convension that for any bound that is not finite, the corresponding dual variable is fixed at zero (and thus will have no influence on the dual problem). For example

$$l_j^x = -\infty \;\Rightarrow\; (s_l^x)_j = 0 \tag{10.10}$$

10.2.6 The certificate of primal infeasibility

A certificate of primal infeasibility is *any* solution to the homogenized dual problem

$$\begin{array}{lcccc} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x & & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & 0, & \\ & -y + s_l^c - s_u^c & = & 0, & \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & & \end{array} \tag{10.11}$$

with a positive objective value. That is, $(s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*})$ is a certificate of primal infeasibility if

$$(l^c)^T s_l^{c*} - (u^c)^T s_u^{c*} + (l^x)^T s_l^{x*} - (u^x)^T s_u^{x*} > 0 \tag{10.12}$$

and

$$\begin{array}{lcl} A^T y + s_l^{x*} - s_u^{x*} & = & 0, \\ -y + s_l^{c*} - s_u^{c*} & = & 0, \\ s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*} \geq 0. & & \end{array} \tag{10.13}$$

The well-known Farkas Lemma tells us that (10.8) is infeasible if and only if a certificate of primal infeasibility exists.

Let $(s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*})$ be a certificate of primal infeasibility then

$$(s_l^{c*})_i > 0 \quad ((s_u^{c*})_i > 0) \tag{10.14}$$

implies that the lower (upper) bound on the ith constraint is important for the infeasibility. Furthermore,

$$(s_l^{x*})_j > 0 \quad ((s_u^{x*})_i > 0) \tag{10.15}$$

implies that the lower (upper) bound on the jth variable is important for the infeasibility.

10.2.7 The certificate of dual infeasibility

A certificate of dual infeasibility is *any* solution to the problem

$$\begin{array}{lccccc} \text{minimize} & & & c^T x & & \\ \text{subject to} & \bar{l}^c & \leq & Ax & \leq & \bar{u}^c, \\ & \bar{l}^x & \leq & x & \leq & \bar{u}^x \end{array} \tag{10.16}$$

with negative objective value, where we use the definitions

$$\bar{l}_i^c := \begin{cases} 0, & l_i^c > -\infty, \\ -\infty, & \text{otherwise}, \end{cases} \qquad \bar{u}_i^c := \begin{cases} 0, & u_i^c < \infty, \\ \infty, & \text{otherwise}, \end{cases} \tag{10.17}$$

and

$$\bar{l}_i^x := \begin{cases} 0, & l_i^x > -\infty, \\ -\infty, & \text{otherwise}, \end{cases} \quad \text{and } \bar{u}_i^x := \begin{cases} 0, & u_i^x < \infty, \\ \infty, & \text{otherwise}. \end{cases} \tag{10.18}$$

Stated differently, a certificate of dual infeasibility is any x^* such that

$$\begin{array}{ccccc} & & c^T x^* & < & 0, \\ \bar{l}^c & \leq & Ax^* & \leq & \bar{u}^c, \\ \bar{l}^x & \leq & x^* & \leq & \bar{u}^x \end{array} \tag{10.19}$$

The well-known Farkas Lemma tells us that (10.9) is infeasible if and only if a certificate of dual infeasibility exists.

Note that if x^* is a certificate of dual infeasibility then for any j such that

$$x_j^* \neq 0, \tag{10.20}$$

variable j is involved in the dual infeasibility.

Chapter 11

Primal feasibility repair

Section 10.2.2 discusses how MOSEK treats infeasible problems. In particular, it is discussed which information MOSEK returns when a problem is infeasible and how this information can be used to pinpoint the elements causing the infeasibility.

In this section we will discuss a method for repairing a primal infeasible problem by relaxing the constraints in a controlled way. For the sake of simplicity we discuss the method in the context of linear optimization. MOSEK can also repair infeasibilities in quadratic and conic optimization problems possibly having integer constrained variables. Please note that infeasibilities in nonlinear optimization problems can't be repaired using the method described below.

11.1 The main idea

Consider the linear optimization problem with m constraints and n variables

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \tag{11.1}$$

which we assume is infeasible. Moreover, we assume that

$$(l^c)_i \leq (u^c)_i, \ \forall i \tag{11.2}$$

and

$$(l^x)_j \leq (u^x)_j, \ \forall j \tag{11.3}$$

because otherwise the problem (11.1) is trivially infeasible.

One way of making the problem feasible is to reduce the lower bounds and increase the upper bounds. If the change is sufficiently large the problem becomes feasible.

One obvious question is: What is the smallest change to the bounds that will make the problem feasible?

We associate a weight with each bound:

- $w_l^c \in \mathbb{R}^m$ (associated with l^c),
- $w_u^c \in \mathbb{R}^m$ (associated with u^c),
- $w_l^x \in \mathbb{R}^n$ (associated with l^x),
- $w_u^x \in \mathbb{R}^n$ (associated with u^x),

Now, the problem

$$\begin{array}{lrcccl} \text{minimize} & & & p & & \\ \text{subject to} & l^c & \leq & Ax + v_l^c - v_u^c & \leq & u^c, \\ & l^x & \leq & x + v_l^x - v_u^x & \leq & u^x, \\ & & & (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq & 0, \\ & & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0 & & \end{array} \tag{11.4}$$

minimizes the weighted sum of changes to the bounds that makes the problem feasible. The variables $(v_l^c)_i$, $(v_u^c)_i$, $(v_l^x)_i$ and $(v_u^c)_i$ are *elasticity* variables because they allow a constraint to be violated and hence add some elasticity to the problem. For instance, the elasticity variable $(v_l^c)_i$ shows how much the lower bound $(l^c)_i$ should be relaxed to make the problem feasible. Since p is minimized and

$$(w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p \leq 0, \tag{11.5}$$

a large $(w_l^c)_i$ tends to imply that the elasticity variable $(v_l^c)_i$ will be small in an optimal solution.

The reader may want to verify that the problem (11.4) is always feasible given the assumptions (11.2) and (11.3).

Please note that if a weight is negative then the resulting problem (11.4) is unbounded.

The weights w_l^c, w_u^c, w_l^x, and w_u^x can be regarded as a costs (penalties) for violating the associated constraints. Thus a higher weight implies that higher priority is given to the satisfaction of the associated constraint.

The main idea can now be presented as follows. If you have an infeasible problem, then form the problem (11.4) and optimize it. Next inspect the optimal solution $(v_l^c)^*, (v_u^c)^*, (v_l^x)^*$, and $(v_u^x)^*$ to problem (11.4). This solution provides a suggested relaxation of the bounds that will make the problem feasible.

Assume that p^* is an optimal objective value to (11.4). An extension of the idea presented above is to solve the problem

$$\begin{array}{lrcccl} \text{minimize} & & & c^T x & & \\ \text{subject to} & l^c & \leq & Ax + v_l^c - v_u^c & \leq & u^c, \\ & l^x & \leq & x + v_l^x - v_u^x & \leq & u^x, \\ & & & (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq & 0, \\ & & & p & = & p^*, \\ & & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0 & & \end{array} \tag{11.6}$$

which minimizes the true objective while making sure that total weighted violations of the bounds is minimal, i.e. equals to p^*.

11.2 Feasibility repair in MOSEK

MOSEK includes functionality that help you construct the problem (11.4) simply by passing a set of weights to MOSEK. This can be used for linear, quadratic, and conic optimization problems, possibly having integer constrained variables.

11.2.1 Usage of negative weights

As the problem (11.4) is presented it does not make sense to use negative weights since that makes the problem unbounded. Therefore, if the value of a weight is negative MOSEK fixes the associated elasticity variable to zero, e.g. if

$$(w_l^c)_i < 0$$

then MOSEK imposes the bound

$$(v_l^c)_i \leq 0.$$

This implies that the lower bound on the ith constraint will not be violated. (Clearly, this could also imply that the problem is infeasible so negative weight should be used with care). Associating a negative weights with a constraint tells MOSEK that the constraint should not be relaxed.

11.2.2 Automatical naming

MOSEK can automatically create a new problem of the form (11.4) starting from an existing problem by adding the elasticity variables and the extra constraints. Specificly, the variables v_l^c, v_u^c, v_l^x, v_u^x, and p are appended to existing variable vector x in their natural order. Moreover, the constraint (11.5) is appended to the constraints.

The new variables and constraints are automatically given names as follows:

- The names of the variables $(v_l^c)_i$ and $(v_u^c)_i$ are constructed from the name of the ith constraint. For instance, if the 9th original constraint is named `c9`, then by default $(v_l^c)_9$ and $(v_u^c)_9$ are given the names `LO*c9` and `UP*c9` respectively. If necessary, the character "`*`" can be replaced by a different string by changing the
  MSK_SPAR_FEASREPAIR_NAME_SEPARATOR
  parameter.

- The additional constraints
  $$l^x \leq x + v_l^x - v_u^x \leq u^x$$
  are given names as follows. There is exactly one constraint per variable in the original problem, and thus the ith of these constraints is named after the ith variable in the original problem. For instance, if the first original variable is named "`x0`", then the first of the above constraints is named "`MSK-x1`". If necessary, the prefix "`MSK-`" can be replaced by a different string by changing the
  MSK_SPAR_FEASREPAIR_NAME_PREFIX
  parameter.

- The variable p is by default given the name `WSUMVIOLVAR`, and the constraint (11.5) is given the name `WSUMVIOLCON`.

  The substring "`WSUMVIOL`" can be replaced by a different string by changing the
  MSK_SPAR_FEASREPAIR_NAME_WSUMVIOL
  parameter.

11.2.3 Feasibility repair using the API

The MSK_relaxprimal function takes an existing problem as input and creates a new task containing the problem (11.4). Moreover, if requested this function can solve the problems (11.4) and (11.6) automatically.

The parameter MSK_IPAR_FEASREPAIR_OPTIMIZE controls which problem is solved. Its value is used as follows:

- MSK_FEASREPAIR_OPTIMIZE_NONE: The problem (11.4) is constructed, but not solved.
- MSK_FEASREPAIR_OPTIMIZE_PENALTY: The problem (11.4) is constructed and solved.
- MSK_FEASREPAIR_OPTIMIZE_COMBINED: The problem (11.6) is constructed and solved.

For further details, please see the description of the function MSK_relaxprimal in the reference.

11.2.4 An example

Consider this example of linear optimization

$$\begin{array}{lrcrcl} \text{minimize} & -10x_1 & & -9x_2, & & \\ \text{subject to} & 7/10x_1 & + & 1x_2 & \leq & 630, \\ & 1/2x_1 & + & 5/6x_2 & \leq & 600, \\ & 1x_1 & + & 2/3x_2 & \leq & 708, \\ & 1/10x_1 & + & 1/4x_2 & \leq & 135, \\ & x_1, & & x_2 & \geq & 0. \\ & & x_2 \geq 650 & & & \end{array} \tag{11.7}$$

This is an infeasible problem. Suppose that we want MOSEK to suggest a modification to the bounds such that the problem becomes feasible. The following example performs this task:

```
1
2  /*
3    Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.
4
5    File:      feasrepairex1.c
6
7    Purpose:   To demonstrate how to use the MSK_relaxprimal function to
8               locate the cause of an infeasibility.
9
10   Syntax: On command line
11           feasrepairex1 feasrepair.lp
```

```
12          feasrepair.lp is located in mosek\<version>\tools\examples.
13 */
14
15
16 #include <math.h>
17 #include <stdio.h>
18
19 #include "mosek.h"
20
21
22 static void MSKAPI printstr(void *handle,
23                             char str[])
24 {
25   fputs(str,stdout);
26 } /* printstr */
27
28 int main(int argc, char** argv)
29 {
30
31   double      wlc[4] = {1.0,1.0,1.0,1.0};
32   double      wuc[4] = {1.0,1.0,1.0,1.0};
33   double      wlx[2] = {1.0,1.0};
34   double      wux[2] = {1.0,1.0};
35   double      sum_violation;
36   MSKenv_t    env;
37   MSKintt     i;
38   MSKrescodee r = MSK_RES_OK;
39   MSKtask_t   task  = NULL, task_relaxprimal = NULL;
40   char        buf[80];
41   char        buffer[MSK_MAX_STR_LEN],symnam[MSK_MAX_STR_LEN];
42
43   r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);
44
45   if (r == MSK_RES_OK)
46     MSK_initenv(env);
47
48   if ( r == MSK_RES_OK )
49     r = MSK_makeemptytask(env,&task);
50
51   if ( r==MSK_RES_OK )
52     MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
53
54   /* Read file from current dir */
55   if ( r == MSK_RES_OK )
56     r = MSK_readdata(task,argv[1]);
57
58   /* Set type of relaxation */
59
60   if (r == MSK_RES_OK)
61     r = MSK_putintparam(task,MSK_IPAR_FEASREPAIR_OPTIMIZE,MSK_FEASREPAIR_OPTIMIZE_PENALTY);
62
63   /* Call relaxprimal, minimizing sum of violations */
64
65   if (r == MSK_RES_OK)
66     r = MSK_relaxprimal(task,
67                         &task_relaxprimal,
68                         wlc,
69                         wuc,
```

```
                          wlx,
                          wux);

  if ( r==MSK_RES_OK )
    MSK_linkfunctotaskstream(task_relaxprimal,MSK_STREAM_LOG,NULL,printstr);

  if (r == MSK_RES_OK)
    r = MSK_getprimalobj(task_relaxprimal,MSK_SOL_BAS,&sum_violation);

  if (r == MSK_RES_OK)
  {
    printf ("Minimized sum of violations = %e\n",sum_violation);

    /* modified bound returned in wlc,wuc,wlx,wux */

    for (i=0;i<4;++i)
    {
      if (wlc[i] == -MSK_INFINITY)
        printf("lbc[%d] = -inf, ",i);
      else
        printf("lbc[%d] = %e, ",i,wlc[i]);

      if (wuc[i] == MSK_INFINITY)
        printf("ubc[%d] = inf\n",i);
      else
        printf("ubc[%d] = %e\n",i,wuc[i]);
    }

    for (i=0;i<2;++i)
    {
      if (wlx[i] == -MSK_INFINITY)
        printf("lbx[%d] = -inf, ",i);
      else
        printf("lbx[%d] = %e, ",i,wlx[i]);

      if (wux[i] == MSK_INFINITY)
        printf("ubx[%d] = inf\n",i);
      else
        printf("ubx[%d] = %e\n",i,wux[i]);
    }

  }

  printf("Return code: %d\n",r);
  if ( r!=MSK_RES_OK )
  {
    MSK_getcodedisc(r,symnam,buffer);
    printf("Description: %s [%s]\n",symnam,buffer);
  }

  return (r);
}
```

The output from the program above is:

```
Minimized sum of violations = 4.250000e+01
lbc[0] = -inf, ubc[0] = 6.300000e+02
```

```
lbc[1] = -inf, ubc[1] = 6.000000e+02
lbc[2] = -inf, ubc[2] = 7.080000e+02
lbc[3] = -inf, ubc[3] = 1.575000e+02
lbx[0] = 0.000000e+00, ubx[0] = inf
lbx[1] = 6.300000e+02, ubx[1] = inf
```

To make the problem feasible it is suggested increasing the upper bound on the activity of the fourth constraint from 134 to 157.5 and decreasing the lower bound on the variable x_2 to 630.

Chapter 12

Sensitivity analysis

12.1 Introduction

Given an optimization problem it is often useful to obtain information about how the optimal objective value changes when the problem parameters are perturbed. E.g, assume that a bound represents a capacity of a machine. Now, it may be possible to expand the capacity for a certain cost and hence it is worthwhile knowing what the value of additional capacity is. This is precisely the type of questions the sensitivity analysis deals with.

Analyzing how the optimal objective value changes when the problem data is changed is called sensitivity analysis.

12.2 Restrictions

Currently, sensitivity analysis is only available for continuous linear optimization problems. Moreover, MOSEK can only deal with perturbations in bounds and objective coefficients.

12.3 References

The book [15] discusses the classical sensitivity analysis in Chapter 10 whereas the book [22, Chapter 19] presents a modern introduction to sensitivity analysis. Finally, it is recommended to read the short paper [25] to avoid some of the pitfalls associated with sensitivity analysis.

12.4 Sensitivity analysis for linear problems

12.4.1 The optimal objective value function

Assume that we are given the problem

$$\begin{array}{rcll} z(l^c, u^c, l^x, u^x, c) & = & \text{minimize} & c^T x \\ & & \text{subject to} \quad l^c \leq & Ax \leq u^c, \\ & & & l^x \leq x \leq u^x, \end{array} \tag{12.1}$$

and we want to know how the optimal objective value changes as l_i^c is perturbed. To answer this question we define the perturbed problem for l_i^c as follows

$$\begin{array}{rcll} f_{l_i^c}(\beta) & = & \text{minimize} & c^T x \\ & & \text{subject to} \quad l^c + \beta e_i \leq & Ax \leq u^c, \\ & & & l^x \leq x \leq u^x, \end{array} \tag{12.2}$$

where e_i is the ith column of the identity matrix. The function

$$f_{l_i^c}(\beta) \tag{12.3}$$

shows the optimal objective value as a function of β. Please note that a change in β corresponds to a perturbation in l_i^c and hence (12.3) shows the optimal objective value as a function of l_i^c.

It is possible to prove that the function (12.3) is a piecewise linear and convex function, i.e. the function may look like the illustration in Figure 12.1.

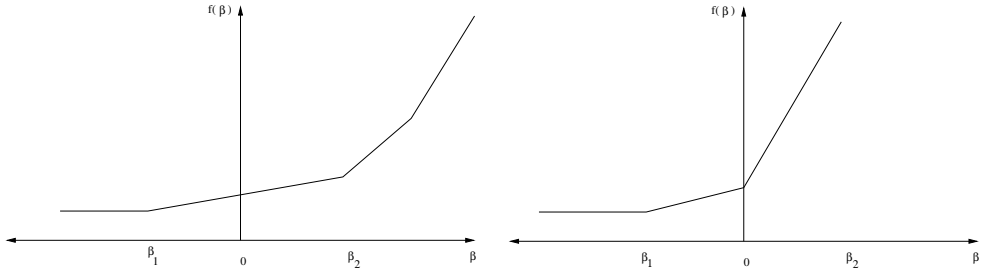

Figure 12.1: The optimal value function $f_{l_i^c}(\beta)$. Left: $\beta = 0$ is in the interior of linearity interval. Right: $\beta = 0$ is a breakpoint.

Clearly, if the function $f_{l_i^c}(\beta)$ does not change much when β is changed, then we can conclude that the optimal objective value is insensitive to changes in l_i^c. Therefore, we are interested in the rate of change in $f_{l_i^c}(\beta)$ for small changes in β — specificly the gradient

$$f'_{l_i^c}(0), \tag{12.4}$$

which is called the *shadow price* related to l_i^c. The shadow price specifies how the objective value changes for small changes in β around zero. Moreover, we are interested in the *linearity interval*

$$\beta \in [\beta_1, \beta_2] \tag{12.5}$$

for which

$$f'_{l_i^c}(\beta) = f'_{l_i^c}(0). \tag{12.6}$$

Since $f_{l_i^c}$ is not a smooth function $f'_{l_i^c}$ may not be defined at 0, as illustrated by the right example in figure 12.1. In this case we can define a left and a right shadow price and a left and a right linearity interval.

The function $f_{l_i^c}$ considered only changes in l_i^c. We can define similar functions for the remaining parameters of the z defined in (12.1) as well:

$$\begin{array}{rcll} f_{u_i^c}(\beta) & = & z(l^c, u^c + \beta e_i, l^x, u^x, c), & i = 1, \ldots, m, \\ f_{l_j^x}(\beta) & = & z(l^c, u^c, l^x + \beta e_j, u^x, c), & j = 1, \ldots, n, \\ f_{u_j^x}(\beta) & = & z(l^c, u^c, l^x, u^x + \beta e_j, c), & j = 1, \ldots, n, \\ f_{c_j}(\beta) & = & z(l^c, u^c, l^x, u^x, c + \beta e_j), & j = 1, \ldots, n. \end{array} \tag{12.7}$$

Given these definitions it should be clear how linearity intervals and shadow prices are defined for the parameters u_i^c etc.

12.4.1.1 Equality constraints

In MOSEK a constraint can be specified as either an equality constraint or a ranged constraint. If constraint i is an equality constraint, we define the optimal value function for this as

$$f_{e_i^c}(\beta) = z(l^c + \beta e_i, u^c + \beta e_i, l^x, u^x, c) \tag{12.8}$$

Thus for an equality constraint the upper and the lower bounds (which are equal) are perturbed simultaneously. Therefore, MOSEK will handle sensitivity analysis differently for a ranged constraint with $l_i^c = u_i^c$ and for an equality constraint.

12.4.2 The basis type sensitivity analysis

The classical sensitivity analysis discussed in most textbooks about linear optimization, e.g. [15, Chapter 10], is based on an optimal basic solution or, equivalently, on an optimal basis. This method may produce misleading results [22, Chapter 19] but is **computationally cheap**. Therefore, and for historical reasons this method is available in MOSEK.

We will now briefly discuss the basis type sensitivity analysis. Given an optimal basic solution which provides a partition of variables into basic and non-basic variables, the basis type sensitivity analysis computes the linearity interval $[\beta_1, \beta_2]$ so that the basis remains optimal for the perturbed problem. A shadow price associated with the linearity interval is also computed. However, it is well-known that an optimal basic solution may not be unique and therefore the result depends on the optimal basic solution employed in the sensitivity analysis. This implies that the computed interval is only a subset of the largest interval for which the shadow price is constant. Furthermore, the optimal objective value function might have a breakpoint for $\beta = 0$. In this case the basis type sensitivity method will only provide a subset of either the left or the right linearity interval.

In summary, the basis type sensitivity analysis is computationally cheap but does not provide complete information. Hence, the results of the basis type sensitivity analysis should be used with care.

12.4.3 The optimal partition type sensitivity analysis

Another method for computing the complete linearity interval is called the *optimal partition type sensitivity analysis*. The main drawback of the optimal partition type sensitivity analysis is that it is computationally expensive compared to the basis type analysts. This type of sensitivity analysis is currently provided as an experimental feature in MOSEK.

Given the optimal primal and dual solutions to (12.1), i.e. x^* and $((s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ the optimal objective value is given by

$$z^* := c^T x^*. \tag{12.9}$$

The left and right shadow prices σ_1 and σ_2 for l_i^c are given by this pair of optimization problems:

$$\begin{array}{rcll} \sigma_1 & = & \text{minimize} & e_i^T s_l^c \\ & & \text{subject to} & \begin{array}{rcl} A^T(s_l^c - s_u^c) + s_l^x - s_u^x & = & c, \\ (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) & = & z^*, \\ s_l^c, s_u^c, s_l^x, s_u^x \geq 0 & & \end{array} \end{array} \tag{12.10}$$

and

$$\begin{array}{rcll} \sigma_2 & = & \text{maximize} & e_i^T s_l^c \\ & & \text{subject to} & \begin{array}{rcl} A^T(s_l^c - s_u^c) + s_l^x - s_u^x & = & c, \\ (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) & = & z^*, \\ s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & \end{array} \end{array} \tag{12.11}$$

These two optimization problems make it easy to interpret the shadow price. Indeed, if $((s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ is an arbitrary optimal solution then

$$(s_l^c)_i^* \in [\sigma_1, \sigma_2]. \tag{12.12}$$

Next, the linearity interval $[\beta_1, \beta_2]$ for l_i^c is computed by solving the two optimization problems

$$\begin{array}{rcll} \beta_1 & = & \text{minimize} & \beta \\ & & \text{subject to} & \begin{array}{rcl} l^c + \beta e_i \leq Ax & \leq & u^c, \\ c^T x - \sigma_1 \beta & = & z^*, \\ l^x \leq x \leq u^x, & & \end{array} \end{array} \tag{12.13}$$

and

$$\begin{array}{rcll} \beta_2 & = & \text{maximize} & \beta \\ & & \text{subject to} & \begin{array}{rcl} l^c + \beta e_i \leq Ax & \leq & u^c, \\ c^T x - \sigma_2 \beta & = & z^*, \\ l^x \leq x \leq u^x. & & \end{array} \end{array} \tag{12.14}$$

The linearity intervals and shadow prices for u_i^c, l_j^x, and u_j^x are computed similarly to l_i^c.

The left and right shadow prices for c_j denoted σ_1 and σ_2 respectively are computed as follows:

$$\begin{array}{rcll} \sigma_1 & = & \text{minimize} & e_j^T x \\ & & \text{subject to} & \begin{array}{rcl} l^c + \beta e_i \leq Ax & \leq & u^c, \\ c^T x & = & z^*, \\ l^x \leq x \leq u^x & & \end{array} \end{array} \tag{12.15}$$

and

$$\begin{array}{rcll} \sigma_2 & = & \text{maximize} & e_j^T x \\ & & \text{subject to} \quad l^c + \beta e_i \leq & Ax \leq u^c, \\ & & & c^T x = z^*, \\ & & & l^x \leq x \leq u^x. \end{array} \tag{12.16}$$

Once again the above two optimization problems make it easy to interpret the shadow prices. Indeed, if x^* is an arbitrary primal optimal solution, then

$$x_j^* \in [\sigma_1, \sigma_2]. \tag{12.17}$$

The linearity interval $[\beta_1, \beta_2]$ for a c_j is computed as follows:

$$\begin{array}{rcll} \beta_1 & = & \text{minimize} & \beta \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c + \beta e_j, \\ & & & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) - \sigma_1\beta \leq z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0 \end{array} \tag{12.18}$$

and

$$\begin{array}{rcll} \beta_2 & = & \text{maximize} & \beta \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x = c + \beta e_j, \\ & & & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) - \sigma_2\beta \leq z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0. \end{array} \tag{12.19}$$

12.4.4 Example: Sensitivity analysis

As an example we will use the following transportation problem. Consider the problem of minimizing the transportation cost between a number of production plants and stores. Each plant supplies a number of goods and each store has a given demand that must be met. Supply, demand and cost of transportation per unit are shown in Figure 12.2.

If we denote the number of transported goods from location i to location j by x_{ij}, the problem can be formulated as the linear optimization problem

minimize

$$1x_{11} + 2x_{12} + 5x_{23} + 2x_{24} + 1x_{31} + 2x_{33} + 1x_{34} \tag{12.20}$$

subject to

$$\begin{array}{rcl} x_{11} + x_{12} & \leq & 400, \\ x_{23} + x_{24} & \leq & 1200, \\ x_{31} + x_{33} + x_{34} & \leq & 1000, \\ x_{11} + x_{31} & = & 800, \\ x_{12} & = & 100, \\ x_{23} + x_{33} & = & 500, \\ x_{24} + x_{34} & = & 500, \\ x_{11}, x_{12}, x_{23}, x_{24}, x_{31}, x_{33}, x_{34} & \geq & 0. \end{array} \tag{12.21}$$

The basis type and the optimal partition type sensitivity results for the transportation problem are shown in Table 12.1 and 12.2 respectively.

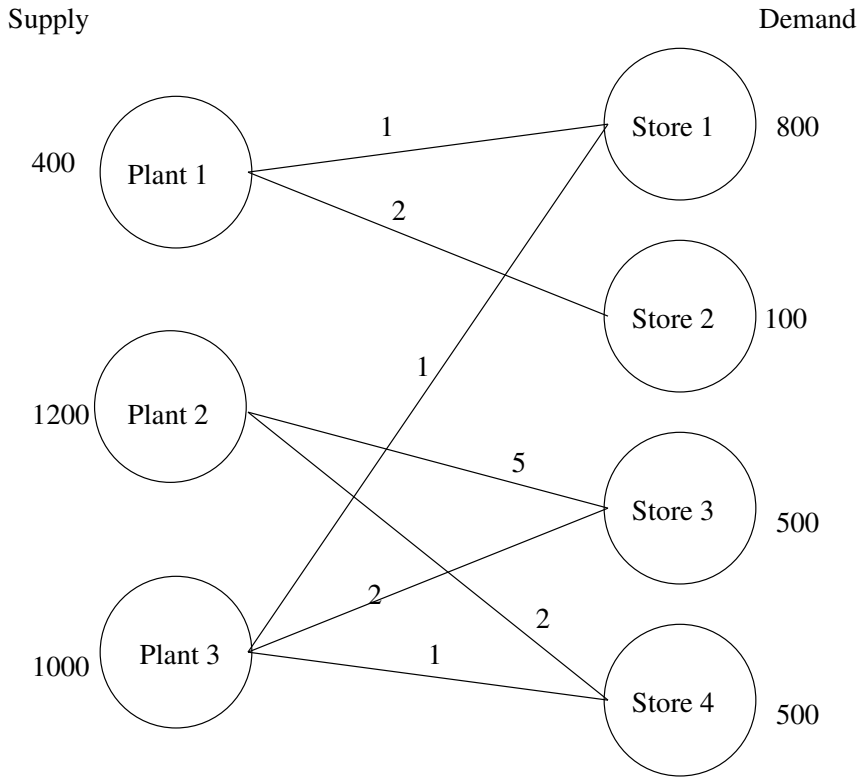

Figure 12.2: Supply, demand and cost of transportation.

Basis type

Con.	β_1	β_2	σ_1	σ_2
1	-300.00	0.00	3.00	3.00
2	-700.00	$+\infty$	0.00	0.00
3	-500.00	0.00	3.00	3.00
4	-0.00	500.00	4.00	4.00
5	-0.00	300.00	5.00	5.00
6	-0.00	700.00	5.00	5.00
7	-500.00	700.00	2.00	2.00
Var.	β_1	β_2	σ_1	σ_2
x_{11}	$-\infty$	300.00	0.00	0.00
x_{12}	$-\infty$	100.00	0.00	0.00
x_{23}	$-\infty$	0.00	0.00	0.00
x_{24}	$-\infty$	500.00	0.00	0.00
x_{31}	$-\infty$	500.00	0.00	0.00
x_{33}	$-\infty$	500.00	0.00	0.00
x_{34}	-0.000000	500.00	2.00	2.00

Optimal partition type

Con.	β_1	β_2	σ_1	σ_2
1	-300.00	500.00	3.00	1.00
2	-700.00	$+\infty$	-0.00	-0.00
3	-500.00	500.00	3.00	1.00
4	-500.00	500.00	2.00	4.00
5	-100.00	300.00	3.00	5.00
6	-500.00	700.00	3.00	5.00
7	-500.00	700.00	2.00	2.00
Var.	β_1	β_2	σ_1	σ_2
x_{11}	$-\infty$	300.00	0.00	0.00
x_{12}	$-\infty$	100.00	0.00	0.00
x_{23}	$-\infty$	500.00	0.00	2.00
x_{24}	$-\infty$	500.00	0.00	0.00
x_{31}	$-\infty$	500.00	0.00	0.00
x_{33}	$-\infty$	500.00	0.00	0.00
x_{34}	$-\infty$	500.00	0.00	2.00

Table 12.1: Ranges and shadow prices related to bounds on constraints and variables. Left: Results for the basis type sensitivity analysis. Right: Results for the optimal partition type sensitivity analysis.

Basis type

Var.	β_1	β_2	σ_1	σ_2
c_1	$-\infty$	3.00	300.00	300.00
c_2	$-\infty$	∞	100.00	100.00
c_3	-2.00	∞	0.00	0.00
c_4	$-\infty$	2.00	500.00	500.00
c_5	-3.00	∞	500.00	500.00
c_6	$-\infty$	2.00	500.00	500.00
c_7	-2.00	∞	0.00	0.00

Optimal partition type

Var.	β_1	β_2	σ_1	σ_2
c_1	$-\infty$	3.00	300.00	300.00
c_2	$-\infty$	∞	100.00	100.00
c_3	-2.00	∞	0.00	0.00
c_4	$-\infty$	2.00	500.00	500.00
c_5	-3.00	∞	500.00	500.00
c_6	$-\infty$	2.00	500.00	500.00
c_7	-2.00	∞	0.00	0.00

Table 12.2: Ranges and shadow prices related to the objective coefficients. Left: Results for the basis type sensitivity analysis. Right: Results for the optimal partition type sensitivity analysis.

Examining the results from the optimal partition type sensitivity analysis we see that for constraint number 1 we have $\sigma_1 \neq \sigma_2$ and $\beta_1 \neq \beta_2$. Therefore, we have a left linearity interval of $[-300, 0]$ and a right interval of $[0, 500]$. The corresponding left and right shadow prices are 3 and 1 respectively. This implies that if the upper bound on constraint 1 increases by

$$\beta \in [0, \beta_1] = [0, 500] \tag{12.22}$$

then the optimal objective value will decrease by the value

$$\sigma_2 \beta = 1\beta. \tag{12.23}$$

Correspondingly, if the upper bound on constraint 1 is decreased by

$$\beta \in [0, 300] \tag{12.24}$$

then the optimal objective value will increase by the value

$$\sigma_1 \beta = 3\beta. \tag{12.25}$$

12.5 Sensitivity analysis from the MOSEK API

MOSEK provides the functions MSK_primalsensitivity and MSK_dualsensitivity for performing sensitivity analysis. The code below gives an example of its use.

Example code from:

mosek/6/tools/examp/capi/sensitivity.c

```
/*
  Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

  File:      sensitivity.c

  Purpose:   To demonstrate how to perform sensitivity
             analysis from the API on a small problem:

  minimize

  obj: +1 x11 + 2 x12 + 5 x23 + 2 x24 + 1 x31 + 2 x33 + 1 x34
  st
  c1:   +  x11 +   x12                                          <= 400
  c2:                  +   x23 +   x24                          <= 1200
  c3:                                  +   x31 +   x33 +   x34  <= 1000
  c4:   +  x11                         +   x31                  = 800
  c5:         +   x12                                           = 100
  c6:                  +   x23                 +   x33          = 500
  c7:                          +   x24                 +   x34  = 500

  The example uses basis type sensitivity analysis.
*/

```

```
#include <stdio.h>

#include "mosek.h" /* Include the MOSEK definition file. */

#define NUMCON 7   /* Number of constraints.             */
#define NUMVAR 7   /* Number of variables.               */
#define NUMANZ 14  /* Number of non-zeros in A.           */

static void MSKAPI printstr(void *handle,
                            char str[])
{
  printf("%s",str);
} /* printstr */

int main(int argc,char *argv[])
{
  MSKrescodee  r;
  MSKidxt      i,j;
  MSKboundkeye bkc[] = {MSK_BK_UP, MSK_BK_UP, MSK_BK_UP, MSK_BK_FX,
                        MSK_BK_FX, MSK_BK_FX,MSK_BK_FX};
  MSKboundkeye bkx[] = {MSK_BK_LO, MSK_BK_LO, MSK_BK_LO,
                        MSK_BK_LO, MSK_BK_LO, MSK_BK_LO,MSK_BK_LO};
  MSKlidxt     ptrb[]= {0,2,4,6,8,10,12};
  MSKlidxt     ptre[]= {2,4,6,8,10,12,14};
  MSKidxt      sub[] = {0,3,0,4,1,5,1,6,2,3,2,5,2,6};
  MSKrealt     blc[] = {-MSK_INFINITY,-MSK_INFINITY,-MSK_INFINITY,800,100,500,500};
  MSKrealt     buc[] = {400,          1200,         1000,         800,100,500,500};
  MSKrealt     c[]   = {1.0,2.0,5.0,2.0,1.0,2.0,1.0};
  MSKrealt     blx[] = {0.0,0.0,0.0,0.0,0.0,0.0,0.0};
  MSKrealt     bux[] = {MSK_INFINITY,MSK_INFINITY,MSK_INFINITY,MSK_INFINITY,
                        MSK_INFINITY,MSK_INFINITY,MSK_INFINITY};
  MSKrealt     val[] = {1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0};

  MSKenv_t     env;
  MSKtask_t    task;

  /* Create mosek environment. */
  r = MSK_makeenv(&env,NULL,NULL,NULL,NULL);

  /* Check if return code is ok. */
  if ( r==MSK_RES_OK )
  {
    /* Directs the env log stream to the user
       specified procedure 'printstr'. */
    MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
  }

  /* Initialize the environment. */
  r = MSK_initenv(env);

  if ( r==MSK_RES_OK )
  {
    /* Send a message to the MOSEK Message stream. */
    MSK_echoenv(env,
                MSK_STREAM_MSG,
                "Making the MOSEK optimization task\n");

    /* Make the optimization task. */
```

```
82      r = MSK_maketask(env,NUMCON,NUMVAR,&task);
83
84      if ( r==MSK_RES_OK )
85      {
86        /* Directs the log task stream to the user
87           specified procedure 'printstr'. */
88
89        MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
90
91        MSK_echotask(task,
92                     MSK_STREAM_MSG,
93                     "Defining the problem data.\n");
94
95        /* Append the constraints. */
96        if ( r==MSK_RES_OK )
97          r = MSK_append(task,MSK_ACC_CON,NUMCON);
98
99        /* Append the variables. */
100       if ( r==MSK_RES_OK )
101         r = MSK_append(task,MSK_ACC_VAR,NUMVAR);
102
103       /* Put C. */
104       if ( r==MSK_RES_OK )
105         r = MSK_putcfix(task,0.0);
106
107       for(j=0; j<NUMVAR && r==MSK_RES_OK; ++j)
108       {
109         r = MSK_putcj(task,j,c[j]);
110         printf("%f\n",c[j]);
111       }
112
113       /* Put constraint bounds. */
114       for(i=0; i<NUMCON && r==MSK_RES_OK; ++i)
115         r = MSK_putbound(task,MSK_ACC_CON,i,bkc[i],blc[i],buc[i]);
116
117       /* Put variable bounds. */
118       for(j=0; j<NUMVAR && r==MSK_RES_OK; ++j)
119         r = MSK_putbound(task,MSK_ACC_VAR,
120                          j,bkx[j],blx[j],bux[j]);
121
122       /* Put A. */
123       if ( NUMCON>0 )
124       {
125         for(j=0; j<NUMVAR && r==MSK_RES_OK; ++j)
126           r = MSK_putavec(task,MSK_ACC_VAR,
127                           j,ptre[j]-ptrb[j],sub+ptrb[j],val+ptrb[j]);
128       }
129
130       if ( r==MSK_RES_OK )
131       {
132         MSK_putobjsense(task,MSK_OBJECTIVE_SENSE_MINIMIZE);
133
134         MSK_echotask(task,
135                      MSK_STREAM_MSG,
136                      "Start optimizing\n");
137
138         r = MSK_optimize(task);
139
```

```
140       }
141     }
142
143     if (r == MSK_RES_OK)
144     {
145       /* Analyze upper bound on c1 and the equality constraint on c4 */
146       MSKidxt subi[]  = {0,3};
147       MSKmarke marki[] = {MSK_MARK_UP,MSK_MARK_UP};
148
149       /* Analyze lower bound on the variables x12 and x31 */
150       MSKidxt subj[]  = {1,4};
151       MSKmarke markj[] = {MSK_MARK_LO,MSK_MARK_LO};
152
153       MSKrealt leftpricei[2];
154       MSKrealt rightpricei[2];
155       MSKrealt leftrangei[2];
156       MSKrealt rightrangei[2];
157       MSKrealt leftpricej[2];
158       MSKrealt rightpricej[2];
159       MSKrealt leftrangej[2];
160       MSKrealt rightrangej[2];
161
162       r = MSK_primalsensitivity( task,
163                                  2,
164                                  subi,
165                                  marki,
166                                  2,
167                                  subj,
168                                  markj,
169                                  leftpricei,
170                                  rightpricei,
171                                  leftrangei,
172                                  rightrangei,
173                                  leftpricej,
174                                  rightpricej,
175                                  leftrangej,
176                                  rightrangej);
177
178       printf("Results from sensitivity analysis on bounds:\n");
179
180       printf("For constraints:\n");
181       for (i=0;i<2;++i)
182         printf("leftprice = %e, rightprice = %e,leftrange = %e, rightrange =%e\n",
183                leftpricei[i], rightpricei[i], leftrangei[i], rightrangei[i]);
184
185       printf("For variables:\n");
186       for (i=0;i<2;++i)
187         printf("leftprice = %e, rightprice = %e,leftrange = %e, rightrange =%e\n",
188                leftpricej[i], rightpricej[i], leftrangej[i], rightrangej[i]);
189     }
190
191     if (r == MSK_RES_OK)
192     {
193
194       MSKidxt subj[]  = {2,5};
195
196       MSKrealt leftprice[2];
197       MSKrealt rightprice[2];
```

```
198       MSKrealt leftrange[2];
199       MSKrealt rightrange[2];
200
201       r = MSK_dualsensitivity(task,
202                               2,
203                               subj,
204                               leftprice,
205                               rightprice,
206                               leftrange,
207                               rightrange
208                               );
209
210       printf("Results from sensitivity analysis on objective coefficients:\n");
211
212       for (i=0;i<2;++i)
213         printf("leftprice = %e, rightprice = %e,leftrange = %e, rightrange =%e\n",
214                leftprice[i], rightprice[i], leftrange[i], rightrange[i]);
215     }
216
217     MSK_deletetask(&task);
218   }
219   MSK_deleteenv(&env);
220
221   printf("Return code: %d (0 means no error occured.)\n",r);
222
223   return ( r );
224 } /* main */
```

12.6 Sensitivity analysis with the command line tool

A sensitivity analysis can be performed with the MOSEK command line tool using the command

```
mosek myproblem.mps -sen sensitivity.ssp
```

where `sensitivity.ssp` is a file in the format described in the next section. The `ssp` file describes which parts of the problem the sensitivity analysis should be performed on.

By default results are written to a file named `myproblem.sen`. If necessary, this filename can be changed by setting the
MSK_SPAR_SENSITIVITY_RES_FILE_NAME
parameter By default a basis type sensitivity analysis is performed. However, the type of sensitivity analysis (basis or optimal partition) can be changed by setting the parameter
MSK_IPAR_SENSITIVITY_TYPE
appropriately. Following values are accepted for this parameter:

- MSK_SENSITIVITY_TYPE_BASIS
- MSK_SENSITIVITY_TYPE_OPTIMAL_PARTITION

It is also possible to use the command line

```
mosek myproblem.mps -d MSK_IPAR_SENSITIVITY_ALL MSK_ON
```

in which case a sensitivity analysis on all the parameters is performed.

12.6.1 Sensitivity analysis specification file

MOSEK employs an MPS like file format to specify on which model parameters the sensitivity analysis should be performed. As the optimal partition type sensitivity analysis can be computationally expensive it is important to limit the sensitivity analysis.

```
* A comment
BOUNDS CONSTRAINTS
 U|L|LU [cname1]
 U|L|LU [cname2]-[cname3]
BOUNDS VARIABLES
 U|L|LU [vname1]
 U|L|LU [vname2]-[vname3]
OBJECTIVE VARIABLES
 [vname1]
 [vname2]-[vname3]
```

Figure 12.3: The sensitivity analysis file format.

The format of the sensitivity specification file is shown in figure 12.3, where capitalized names are keywords, and names in brackets are names of the constraints and variables to be included in the analysis.

The sensitivity specification file has three sections, i.e.

- `BOUNDS CONSTRAINTS`: Specifies on which bounds on constraints the sensitivity analysis should be performed.
- `BOUNDS VARIABLES`: Specifies on which bounds on variables the sensitivity analysis should be performed.
- `OBJECTIVE VARIABLES`: Specifies on which objective coefficients the sensitivity analysis should be performed.

A line in the body of a section must begin with a whitespace. In the `BOUNDS` sections one of the keys `L`, `U`, and `LU` must appear next. These keys specify whether the sensitivity analysis is performed on the lower bound, on the upper bound, or on both the lower and the upper bound respectively. Next, a single constraint (variable) or range of constraints (variables) is specified.

Recall from Section 12.4.1.1 that equality constraints are handled in a special way. Sensitivity analysis of an equality constraint can be specified with either `L`, `U`, or `LU`, all indicating the same, namely that upper and lower bounds (which are equal) are perturbed simultaneously.

As an example consider

```
BOUNDS CONSTRAINTS
 L  "cons1"
 U  "cons2"
 LU "cons3"-"cons6"
```

which requests that sensitivity analysis is performed on the lower bound of the constraint named `cons1`, on the upper bound of the constraint named `cons2`, and on both lower and upper bound on the constraints named `cons3` to `cons6`.

It is allowed to use indexes instead of names, for instance

```
BOUNDS CONSTRAINTS
 L  "cons1"
 U  2
 LU 3 - 6
```

The character "*" indicates that the line contains a comment and is ignored.

12.6.2 Example: Sensitivity analysis from command line

As an example consider the `sensitivity.ssp` file shown in Figure 12.4.

```
* Comment 1

BOUNDS CONSTRAINTS
 U "c1"          * Analyze upper bound for constraint named c1
 U 2             * Analyze upper bound for the second constraint
 U 3-5           * Analyze upper bound for constraint number 3 to number 5

BOUNDS VARIABLES
 L 2-4           * This section specifies which bounds on variables should be analyzed
 L "x11"
OBJECTIVE VARIABLES
 "x11"           * This section specifies which objective coefficients should be analyzed
 2
```

Figure 12.4: Example of the sensitivity file format.

The command

```
mosek transport.lp -sen sensitivity.ssp  -d MSK_IPAR_SENSITIVITY_TYPE MSK_SENSITIVITY_TYPE_BASIS
```

produces the `transport.sen` file shown below.

```
BOUNDS CONSTRAINTS
INDEX    NAME            BOUND    LEFTRANGE       RIGHTRANGE      LEFTPRICE       RIGHTPRICE
0        c1              UP       -6.574875e-18   5.000000e+02    1.000000e+00    1.000000e+00
2        c3              UP       -6.574875e-18   5.000000e+02    1.000000e+00    1.000000e+00
3        c4              FIX      -5.000000e+02   6.574875e-18    2.000000e+00    2.000000e+00
4        c5              FIX      -1.000000e+02   6.574875e-18    3.000000e+00    3.000000e+00
5        c6              FIX      -5.000000e+02   6.574875e-18    3.000000e+00    3.000000e+00

BOUNDS VARIABLES
```

```
INDEX    NAME            BOUND    LEFTRANGE        RIGHTRANGE       LEFTPRICE        RIGHTPRICE
2        x23             LO       -6.574875e-18    5.000000e+02     2.000000e+00     2.000000e+00
3        x24             LO       -inf             5.000000e+02     0.000000e+00     0.000000e+00
4        x31             LO       -inf             5.000000e+02     0.000000e+00     0.000000e+00
0        x11             LO       -inf             3.000000e+02     0.000000e+00     0.000000e+00

OBJECTIVE VARIABLES
INDEX    NAME                     LEFTRANGE        RIGHTRANGE       LEFTPRICE        RIGHTPRICE
0        x11                      -inf             1.000000e+00     3.000000e+02     3.000000e+02
2        x23                      -2.000000e+00    +inf             0.000000e+00     0.000000e+00
```

12.6.3 Controlling log output

Setting the parameter

`MSK_IPAR_LOG_SENSITIVITY`

to 1 or 0 (default) controls whether or not the results from sensitivity calculations are printed to the message stream.

The parameter

`MSK_IPAR_LOG_SENSITIVITY_OPT`

controls the amount of debug information on internal calculations from the sensitivity analysis.

Chapter 13

Case Studies

13.1 The traveling salesman problem

The Travelling Salesman Problem (TSP) is the problem of finding the shortest cyclic tour between a set of cities, visiting each city exactly once. This can be formulated using mixed integer programming.

When solving mixed integer optimization problems it is important to use a strong formulation of the problem, otherwise MOSEK may spend a very long time solving the optimization problem. This is not only true for MOSEK but for the branch-and-bound based solution method too.

The approach explored in this section is an implementation of the approach discussed in the article "Teaching integer programming formulations using the Traveling Salesman Problem" by Gábor Pataki [21].

13.1.1 The TSP formulations

Given a set of nodes we want to find the shortest tour (a directed cycle containing all nodes) in a complete directed graph. We use the variables x_{ij} to indicate whether the arc (i, j) is included in the tour.

The core of the formulation is

$$\begin{array}{lll} \text{minimize} & \sum\limits_{i,j} c_{ij} x_{ij} & \\ \text{subject to} & \sum\limits_{i} x_{ij} = 1 & \forall j, \\ & \sum\limits_{j} x_{ij} = 1 & \forall i, \\ & 0 \leq x_{ij} \leq 1, & x_{ij} \text{ integer.} \end{array} \tag{13.1}$$

These constraints are called the assignment constraints. The assignment constraints, however, do not constitute the entire formulation as groups of disjoint cycles, called subtours, as well as the complete tours are feasible.

To exclude the subtours two sets of constraints are considered.

The MTZ formulation The MTZ (Miller-Tucker-Zemlin) formulation of the TSP includes the following constraints

$$\begin{array}{rll} u_1 = 1, & & \\ 2 \leq u_i \leq n & \forall i \neq 1, & \\ u_i - u_j + 1 \leq (n-1)(1-x_{ij}) & \forall i \neq 1, j \neq 1. & \end{array} \tag{13.2}$$

The idea of this formulation is to assign the numbers 1 through n to the nodes with the extra variables u_i so that this numbering corresponds to the order of the nodes in the tour. It is obvious that this excludes subtours, as a subtour excluding the node 1 cannot have a feasible assignment of the corresponding u_i variables.

The subtour formulation An alternative approach is simply to take any potential subtour, i.e. any true subset of nodes, and declare that it is illegal.

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \quad (S \not\subseteq V, |S| > 1) \tag{13.3}$$

As the subtour inequality for $V \setminus S$ is a linear combination of the inequality for S and the assignment constraints, it is sufficient to use the subtour inequalities with S having size $n/2$ at most. Note that this formulation has the disadvantage of being exponential in size.

The MTZ formulation of the TSP is a very weak formulation so we will try to strengthen it by adding some of the subtour constraints from the stronger subtour formulation and then compare the solution times. For each problem we will try to identify some of the most relevant subtour constraints by solving the relaxed IP without the MTZ constraints and then choosing some of the violated subtour inequalities, corresponding to the subtours in the solution. The complete algorithm in pseudo-code is (the complete C implementation is included below in Section 13.1.3):

1. Let the IP formulation consist of the assignment constraints (13.1) only.
2. **for** $k = 1$ **to** *maxrounds*
   - 2a. Solve the IP over the current formulation. Assume that the optimal solution consists of r subtours $S_1, \ldots, S_r$.
   - 2b. If $r = 1$, stop; the solution is optimal to the TSP. Otherwise, add to the formulation 1000 subtour constraints at most, in which S is the union of several S_i sets and $|S| \leq \lfloor n/2 \rfloor$.
3. Add the MTZ arc constraints to the formulation, and solve the IP to optimality.

Each round we add 1000 constraints at most as the number of violated subtour inequalities is exponential in r.

Setting *maxrounds* equal to 0, 1, and 2, we obtain three formulations of increasing strength which we solve in 3.

Number of rounds	Zero rounds		One round		Two rounds	
Problem name	Time	B&B nodes	Time	B&B nodes	Time	B&B nodes
bays29	658	53809	85	1715	39	2739
berlin52	553	7944	56	198	10	2
br17	***	***	1	13	1	1
ft70	***	***	17	3	16	5
ftv33	23	1882	8	2	9	1
ftv55	864	12494	138	4853	53	2515

Table 13.1: Solving TSP using increasingly stronger formulations.

13.1.2 Comparing formulations

We have tested this method on six TSP instances from the TSPLIB library which can be found at

http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

The time and number of B&B nodes for each of the three formulations are recorded in Table 13.1. The entry "***" means that the problem was unsolvable within a time window of 5000 seconds. The time spent solving the relaxed IPs and identifying subtour constraints was negligible.

Not surprisingly a stronger formulation means shorter solution time (with a few exceptions where the second round of strengthening seemingly is superfluous), but it is worth noting the magnitude of the decrease in solution time arising from stronger formulations.

Therefore, it is often worthwhile to consider whether one can strengthen a given formulation when solving a mixed integer optimization problem.

13.1.3 Example code

The following example is included in the distribution in the file `msktsp.c`.

```
/*
   Copyright: Copyright (c) 1998-2011 MOSEK ApS, Denmark. All rights reserved.

   File:      msktsp.c

   Purpose:   Demonstrates the difference between weak
              and strong formulations when solving MIP's.
 */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <assert.h>

#include "mosek.h"

#define MAXCUTROUNDS 2
```

```
18 #define MAXADDPERROUND 1000
19
20 static void MSKAPI printstr(void *handle, char str[])
21 {
22     printf("MOSEK: %s",str);
23 } /* printstr */
24
25
26 /* conversion from n x n tsp city matrix indices to array index */
27 #define IJ(i,j) (n*(i)+(j))
28
29 /* mallocs and returns costmatrix, returns number of cities in ncities */
30 int* readtspfromfile(char* filename, int* ncities)
31 {
32     FILE *tspfile;
33     char sbuf[21];
34     tspfile = fopen(filename,"r");
35     if (!tspfile) return NULL;
36     do
37     {
38         if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
39     } while (strncmp(sbuf,"DIMENSION",9) != 0);
40     if (1 != fscanf(tspfile,"%d ",ncities)) return NULL;
41     do
42     {
43         if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
44     } while (strncmp(sbuf,"EDGE_WEIGHT_TYPE",16) != 0);
45     if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
46     if (strcmp(sbuf,"EXPLICIT") == 0)
47     {
48         do
49         {
50             if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
51         } while (strncmp(sbuf,"EDGE_WEIGHT_FORMAT",18) != 0);
52         if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
53         if (strcmp(sbuf,"FULL_MATRIX") == 0)
54         {
55             int* cost;
56             int ij, n2;
57             do
58             {
59                 if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
60             } while (strncmp(sbuf,"EDGE_WEIGHT_SECTION",19) != 0);
61             n2 = *ncities;
62             n2 *= n2;
63             cost = (int*) malloc(n2*sizeof(int));
64             assert(cost);
65             for (ij = 0; ij<n2; ij++)
66             {
67                 if (1 != fscanf(tspfile,"%d ",&cost[ij]))
68                 {
69                     free(cost);
70                     return NULL;
71                 }
72             }
73             return cost;
74         }
75         else if (strcmp(sbuf,"LOWER_DIAG_ROW") == 0)
```

```
76          {
77              int* cost;
78              int i, j, n;
79              do
80              {
81                  if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
82              } while (strncmp(sbuf,"EDGE_WEIGHT_SECTION",19) != 0);
83              n = *ncities;
84              cost = (int*) malloc(n*n*sizeof(int));
85              assert(cost);
86              for (i=0; i<n; i++) for (j=0; j<=i; j++)
87              {
88                  int c;
89                  if (1 != fscanf(tspfile,"%d ",&c))
90                  {
91                      free(cost);
92                      return NULL;
93                  }
94                  cost[IJ(i,j)] = c;
95                  cost[IJ(j,i)] = c;
96              }
97              return cost;
98          }
99          else
100         {
101             printf("Format not supported\n");
102             return NULL;
103         }
104     }
105     else if (strcmp(sbuf,"EUC_2D") == 0)
106     {
107         int* cost;
108         double *xcoord, *ycoord;
109         int i, j, n;
110         do
111         {
112             if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
113         } while (strncmp(sbuf,"NODE_COORD_SECTION",18) != 0);
114         n = *ncities;
115         xcoord = (double*) malloc(n*sizeof(double));
116         ycoord = (double*) malloc(n*sizeof(double));
117         cost = (int*) malloc(n*n*sizeof(int));
118         assert(xcoord); assert(ycoord); assert(cost);
119         for (i = 0; i<n; i++)
120         {
121             int dummy;
122             if (3 != fscanf(tspfile,"%d %lf %lf ",&dummy,&xcoord[i],&ycoord[i]))
123             {
124                 free(cost);
125                 return NULL;
126             }
127         }
128         for (i = 0; i<n; i++) for (j=0; j<n; j++)
129         {
130             double xd = xcoord[i] - xcoord[j];
131             double yd = ycoord[i] - ycoord[j];
132             cost[IJ(i,j)] = (int) (0.5 + sqrt(xd*xd + yd*yd));
133         }
```

```
134         return cost;
135     }
136     else
137     {
138         printf("E_W_Type not supported\n");
139         return NULL;
140     }
141 } /* readtspfromfile */
142 
143 /* add the x_ij variables */
144 void add_vars(MSKtask_t task, int n)
145 {
146     MSKrescodee r;
147     int ij;
148     int n2 = n*n;
149     r = MSK_append(task,MSK_ACC_VAR,n2); assert(r==MSK_RES_OK);
150     for(ij=0; ij<n2; ++ij)
151     {
152         r = MSK_putbound(task,MSK_ACC_VAR,ij,MSK_BK_RA,0,1);
153         assert(r==MSK_RES_OK);
154         r = MSK_putvartype(task,ij,MSK_VAR_TYPE_INT); assert(r==MSK_RES_OK);
155     }
156     for (ij=0; ij<n; ij++)
157     {
158         r = MSK_putbound(task,MSK_ACC_VAR,IJ(ij,ij),MSK_BK_FX,0,0);
159         assert(r==MSK_RES_OK);
160     }
161 } /* add_vars */
162 
163 /* adds the tsp objective function and frees cost */
164 void add_objective_function(MSKtask_t task, int n, int* cost)
165 {
166     MSKrescodee r;
167     int ij;
168     int n2 = n*n;
169     r = MSK_putcfix(task,0.0); assert(r==MSK_RES_OK);
170     for(ij=0; ij<n2; ++ij)
171     {
172         r = MSK_putcj(task,ij,cost[ij]); assert(r==MSK_RES_OK);
173     }
174     free(cost);
175 } /* add_objective_function */
176 
177 /* adds the tsp assignment constraints */
178 void add_assignment_constraints(MSKtask_t task, int n)
179 {
180     MSKrescodee r;
181     int i, j;
182     double* aval;
183     int *asub;
184     aval = (double*) malloc(n*sizeof(double)); assert(aval);
185     asub = (int*) malloc(n*sizeof(int)); assert(asub);
186     for (i=0; i<n; i++) aval[i] = 1;
187     r = MSK_append(task,MSK_ACC_CON,n*2); assert(r==MSK_RES_OK);
188     /* Constraint 0--(n-1) is \sum_j x_{ij} = 1 */
189     for (i=0; i<n; i++)
190     {
191         r = MSK_putbound(task,MSK_ACC_CON,i,MSK_BK_FX,1,1);
```

```
192         assert(r==MSK_RES_OK);
193         for (j=0; j<n; j++)
194             asub[j] = IJ(i,j);
195         r = MSK_putavec(task,MSK_ACC_CON,i,n,asub,aval); assert(r==MSK_RES_OK);
196     }
197     /* Constraint n--(2n-1) is \sum_i x_{ij} = 1 */
198     for (j=0; j<n; j++)
199     {
200         r = MSK_putbound(task,MSK_ACC_CON,j+n,MSK_BK_FX,1,1);
201         assert(r==MSK_RES_OK);
202         for (i=0; i<n; i++)
203             asub[i] = IJ(i,j);
204         r = MSK_putavec(task,MSK_ACC_CON,j+n,n,asub,aval);
205         assert(r==MSK_RES_OK);
206     }
207     free(aval);
208     free(asub);
209 } /* add_assignment_constraints */
210
211 /* adds the Miller-Tucker-Zemlin arc constraints */
212 void add_MTZ_arc_constraints(MSKtask_t task, int n)
213 {
214     MSKrescodee r;
215     int varidx, conidx, i, j;
216     r = MSK_getnumvar(task,&varidx); assert(r==MSK_RES_OK);
217     r = MSK_getnumcon(task,&conidx); assert(r==MSK_RES_OK);
218     /* add the vars u_k for k=1..(n-1) getting index
219      * from varidx to varidx+n-2 */
220     r = MSK_append(task,MSK_ACC_VAR,n-1); assert(r==MSK_RES_OK);
221     for(i=varidx; i<varidx+n-1; ++i)
222     {
223         /* set bound: 2 <= u_k <= n, k=1..(n-1) */
224         r = MSK_putbound(task,MSK_ACC_VAR,i,MSK_BK_RA,2,n);
225         assert(r==MSK_RES_OK);
226     }
227     /* add the (n-1)^2 constraints:
228      * u_i - u_j + 1 <= (n - 1)(1 - x_ij) or equivalently
229      * u_i - u_j + (n - 1)x_ij <= n - 2, for i,j != 0 */
230     r = MSK_append(task,MSK_ACC_CON,(n-1)*(n-1)); assert(r==MSK_RES_OK);
231     for (i=1; i<n; i++) for (j=1; j<n; j++)
232     {
233         double aval[3];
234         int asub[3];
235         aval[0] = 1; aval[1] = -1; aval[2] = n-1;
236         asub[0] = varidx + i - 1; /* u_i */
237         asub[1] = varidx + j - 1; /* u_j */
238         asub[2] = IJ(i,j);        /* x_ij */
239         r = MSK_putbound(task,MSK_ACC_CON,conidx,MSK_BK_UP,-MSK_INFINITY,n-2);
240         assert(r==MSK_RES_OK);
241         r = MSK_putavec(task,MSK_ACC_CON,conidx,3,asub,aval);
242         assert(r==MSK_RES_OK);
243         conidx++;
244     }
245 } /* add_MTZ_arc_constraints */
246
247 /* construct the list of cities in the chosen subtours */
248 int* subtourstolist(MSKtask_t task, int n, int nextnode[],
249         int subtour[], int chosen[], int k, int* size)
```

```
{
    int ncities, i, j;
    int *cities;
    cities = (int*) malloc(n*sizeof(int));
    assert(cities);
    ncities = 0;
    for (i=0; i<k; i++)
    {
        int subtourstart = subtour[chosen[i]];
        j = subtourstart;
        do
        {
            cities[ncities] = j;
            ncities++;
            j = nextnode[j];
        } while (j != subtourstart);
    }
    *size = ncities;
    return cities;
} /* subtourstolist */

/* adds the subtour constraint given by the list cities S:
 * \sum_{i,j \in S} x_{ij} \leq |S|-1 */
void addcut(MSKtask_t task, int n, int citylist[], int size)
{
    MSKrescodee r;
    int i, j, asubidx, conidx;
    double* aval;
    int *asub;
    int size2 = size*size;
    aval = (double*) malloc(size2*sizeof(double)); assert(aval);
    asub = (int*) malloc(size2*sizeof(int)); assert(asub);
    for (i=0; i<size2; i++) aval[i] = 1;
    r = MSK_getnumcon(task,&conidx); assert(r==MSK_RES_OK);
    r = MSK_append(task,MSK_ACC_CON,1); assert(r==MSK_RES_OK);
    r = MSK_putbound(task,MSK_ACC_CON,conidx,MSK_BK_UP,-MSK_INFINITY,size-1);
    assert(r==MSK_RES_OK);
    asubidx = 0;
    for (i=0; i<size; i++) for (j=0; j<size; j++)
    {
        asub[asubidx] = IJ(citylist[i],citylist[j]);
        asubidx++;
    }
    r = MSK_putavec(task,MSK_ACC_CON,conidx,size2,asub,aval);
    assert(r==MSK_RES_OK);
    free(aval);
    free(asub);
} /* addcut */

/* identifies subtours and adds a number of violated cuts */
void addcuts(MSKtask_t task, int n, int maxcuts, int* nsubtours, int* ncuts)
{
    MSKrescodee r;
    int i, j, k;
    int n2 = n*n;
    double *xx;
    int *nextnode, *visited, *subtour, *chosen;
    int nsubt = 0;
```

```
308     xx = (double*) malloc(n2*sizeof(double));
309     nextnode = (int*) malloc(n*sizeof(int));
310     assert(xx);
311     assert(nextnode);
312     r = MSK_getsolutionslice(task,MSK_SOL_ITG,MSK_SOL_ITEM_XX,0,n2,xx);
313     assert(r==MSK_RES_OK);
314     /* convert matrix representation of graph (xx) to
315      * adjacency(-list) (nextnode) */
316     for (i=0; i<n; i++) for (j=0; j<n; j++)
317     {
318         if (xx[IJ(i,j)]>0.5) /* i.e. x_ij = 1 */
319             nextnode[i] = j;
320     }
321     free(xx); xx = NULL;
322     visited = (int*) calloc(n,sizeof(int)); /* visited is initialized to 0 */
323     subtour = (int*) malloc(n*sizeof(int));
324     assert(visited);
325     assert(subtour);
326     /* identify subtours; keep count in nsubt, save starting
327      * pointers in subtour[0..(nsubt-1)] */
328     for (i=0; i<n; i++) if (!visited[i]) /* find an unvisited node;
329                                           * this starts a new subtour */
330     {
331         subtour[nsubt] = i;
332         nsubt++;
333         j = i;
334         do
335         {
336             assert(!visited[j]);
337             visited[j] = 1;
338             j = nextnode[j];
339         } while (j!=i);
340     }
341     free(visited); visited = NULL;
342     *nsubtours = nsubt;
343     *ncuts = 0;
344     chosen = (int*) malloc(nsubt*sizeof(int)); /* list of chosen subtours */
345     for (k=1; k<=nsubt; k++) /* choose k of nsubt subtours */
346     {
347         int nchosen = 1;
348         chosen[0] = nsubt - 1;
349         while (*ncuts < maxcuts)
350         {
351             if (nchosen == k)
352             {
353                 int *citylist;
354                 int size;
355                 citylist = subtourstolist(task,n,nextnode,subtour,
356                                           chosen,k,&size);
357                 if (size <= n/2) /* add only subtour constraints
358                                   * of size n/2 or less */
359                 {
360                     addcut(task,n,citylist,size);
361                     (*ncuts)++;
362                 }
363                 free(citylist);
364                 j=0;
365                 while (j<k && chosen[k - 1 - j] == j) j++;
```

```
366                 if (k==j) break; /* all k-size subsets done */
367                 nchosen = k - j;
368                 chosen[nchosen - 1]--;
369             }
370             else /* 0 < nchosen < k */
371             {
372                 chosen[nchosen] = chosen[nchosen - 1] - 1;
373                 nchosen++;
374             }
375         }
376     }
377     free(nextnode);
378     free(subtour);
379     free(chosen);
380 } /* addcuts */
381
382 int main(int argc, char *argv[])
383 {
384     int             *cost;     /* tsp cost matrix */
385     int             n;         /* number of cities */
386     MSKenv_t        env;       /* Mosek environment */
387     MSKtask_t       task;      /* Mosek task */
388     MSKrescodee     r;         /* Mosek return code */
389     double          ObjVal;    /* Value of the objective function */
390     int             maxrounds; /* number of cutting rounds */
391     int             maxcuts;   /* maximum number of cuts added per round */
392     int k;
393     int nsubtours, ncuts;
394     double t;
395     double cuttime = 0;
396
397     if (argc < 2)
398     {
399         printf("Usage: ./tsp filename.tsp [rounds] [maxcuts]\n\n"
400                "rounds is the maximum number of cutting rounds (default = %d)\n"
401                "maxcuts is the maximum number of cuts added per round "
402                "(default = %d)\n",
403                 MAXCUTROUNDS,MAXADDPERROUND);
404         return 1;
405     }
406     maxrounds = MAXCUTROUNDS;
407     if (argc >= 3) maxrounds = atoi(argv[2]);
408     maxcuts = MAXADDPERROUND;
409     if (argc >= 4) maxcuts = atoi(argv[3]);
410
411     cost = readtspfromfile(argv[1],&n);
412     if (!cost)
413     {
414         printf("Bad tsp file\n");
415         return 1;
416     }
417
418     r = MSK_makeenv(&env,NULL,NULL,NULL,NULL); assert(r==MSK_RES_OK);
419     MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
420     r = MSK_initenv(env);                      assert(r==MSK_RES_OK);
421     r = MSK_makeemptytask(env,&task);          assert(r==MSK_RES_OK);
422     MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
423
```

```
424     add_vars(task,n);
425     add_objective_function(task,n,cost);
426     add_assignment_constraints(task,n);
427
428     nsubtours = 2;
429     for (k=0; k<maxrounds; k++)
430     {
431         r = MSK_optimize(task);                          assert(r==MSK_RES_OK);
432         r = MSK_getprimalobj(task,MSK_SOL_ITG,&ObjVal); assert(r==MSK_RES_OK);
433         MSK_getdouinf(task,MSK_DINF_OPTIMIZER_TIME,&t);
434         cuttime += t;
435         addcuts(task,n,maxcuts,&nsubtours,&ncuts);
436         printf("\n"
437                "Round: %d\n"
438                "ObjValue: %e\n"
439                "Number of subtours: %d\n"
440                "Number of cuts added: %d\n\n",k+1,ObjVal,nsubtours,ncuts);
441         if (nsubtours == 1) break; /* problem solved! */
442     }
443
444     t = 0;
445     if (nsubtours > 1)
446     {
447         printf("Adding MTZ arc constraints\n\n");
448         add_MTZ_arc_constraints(task,n);
449         r = MSK_optimize(task);                          assert(r==MSK_RES_OK);
450         r = MSK_getprimalobj(task,MSK_SOL_ITG,&ObjVal); assert(r==MSK_RES_OK);
451         MSK_getdouinf(task,MSK_DINF_OPTIMIZER_TIME,&t);
452     }
453
454     printf("\n"
455            "Done solving.\n"
456            "Time spent cutting: %.2f\n"
457            "Total time spent: %.2f\n"
458            "ObjValue: %e\n",cuttime,cuttime+t,ObjVal);
459
460     MSK_deletetask(&task);
461     MSK_deleteenv(&env);
462     return 0;
463 } /* main */
```

13.2 Geometric (posynomial) optimization

13.2.1 The problem

A *geometric optimization* problem can be stated as follows

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{k\in J_0} c_k \prod_{j=0}^{n-1} t_j^{a_{kj}} & \\ \text{subject to} & \displaystyle\sum_{k\in J_i} c_k \prod_{j=0}^{n-1} t_j^{a_{kj}} & \leq \quad 1, \quad i = 1,\ldots,m, \\ & t > 0, & \end{array} \tag{13.4}$$

where it is assumed that

$$\cup_{k=0}^{m} J_k = \{1, \ldots, T\}$$

and if $i \neq j$, then

$$J_i \cap J_j = \emptyset.$$

Hence, A is a $T \times n$ matrix and c is a vector of length T. Given $c_k > 0$ then

$$c_k \prod_{j=0}^{n-1} t_j^{a_{kj}}$$

is called a *monomial* . A sum of monomials i.e.

$$\sum_{k \in J_i} c_k \prod_{j=0}^{n-1} t_j^{a_{kj}}$$

is called a *posynomial*. In general, the problem (13.4) is very hard to solve. However, the posynomial case where it is required that

$$c > 0$$

is relatively easy. The reason is that using a simple variable transformation a convex optimization problem can be obtained. Indeed using the variable transformation

$$t_j = e^{x_j} \tag{13.5}$$

we obtain the problem

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{k \in J_0} c_k e^{\sum_{j=0}^{n-1} a_{kj} x_j} & \\ \text{subject to} & \displaystyle\sum_{k \in J_i} c_k e^{\sum_{j=0}^{n-1} a_{kj} x_j} \leq 1, & i = 1, \ldots, m, \end{array} \tag{13.6}$$

which is a convex optimization problem that can be solved using MOSEK. We will call

$$c_t e^{\left(\sum_{j=0}^{n-1} a_{tj} x_j\right)} = e^{\left(\log(c_t) + \sum_{j=0}^{n-1} a_{tj} x_j\right)}$$

a *term* and hence the number of terms is T.

As stated, the problem (13.6) is non-separable. However, using

$$v_t = \log(c_t) + \sum_{j=0}^{n-1} a_{tj} x_j$$

we obtain the separable problem

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{t \in J_0} e^{v_t} & \\ \text{subject to} & \displaystyle\sum_{t \in J_i} e^{v_t} \leq 1, & i = 1, \ldots, m, \\ & \displaystyle\sum_{j=0}^{n-1} a_{tj} x_j - v_t = -\log(c_t), & t = 0, \ldots, T, \end{array} \tag{13.7}$$

which is a separable convex optimization problem.

A warning about this approach is that the exponential function e^x is only numerically well-defined for values of x in a small interval around 0 since e^x grows very rapidly as x becomes larger. Therefore numerical problems may arise when solving the problem on this form.

13.2.2 Applications

A large number of practical applications, particularly in electrical circuit design, can be cast as a geometric optimization problem. We will not review these applications here but rather refer the reader to [14] and the references therein.

13.2.3 Modeling tricks

A lot of tricks that can be used for modeling posynomial optimization problems are described in [14]. Therefore, in this section we cover only one important case.

13.2.3.1 Equalities

In general, equalities are not allowed in (13.4), i.e.

$$\sum_{k\in J_i} c_k \prod_{j=0}^{n-1} t_j^{a_{kj}} = 1$$

is not allowed. However, a monomial equality is not a problem. Indeed consider the example

$$xyz^{-1} = 1$$

of a monomial equality. The equality is identical to

$$1 \leq xyz^{-1} \leq 1$$

which in turn is identical to the two inequalities

$$\begin{array}{rcl} xyz^{-1} & \leq & 1, \\ \frac{1}{xyz^{-1}} = x^{-1}y^{-1}z & \leq & 1. \end{array}$$

Hence, it is possible to model a monomial equality using two inequalities.

13.2.4 Problematic formulations

Certain formulations of geometric optimization problems may cause problems for the algorithms implemented in MOSEK. Basically there are two kinds of problems that may occur:

- The solution vector is finite, but an optimal objective value can only be a approximated.
- The optimal objective value is finite but implies that a variable in the solution is infinite.

13.2.4.1 Finite unattainable solution

The following problem illustrates an unattainable solution:

$$\begin{array}{lcl} \text{minimize} & x^2y & \\ \text{subject to} & xy \leq 1, \\ & x,y>0. \end{array}$$

Clearly, the optimal objective value is 0 but because of the constraint the $x, y > 0$ constraint this value can never be attained: To see why this is a problem, remember that MOSEK substitutes $x = e^{t_x}$ and $y = e^{t_y}$ and solves the problem as

$$\begin{array}{lcl} \text{minimize} & e^{2t_x}e^{t_y} & \\ \text{subject to} & e^{t_x}e^{t_y} \leq 1, \\ & t_x, t_y \in \mathbb{R}. \end{array}$$

The optimal solution implies that $t_x = -\infty$ or $t_y = -\infty$, and thus it is unattainable.

Now, the issue should be clear: If a variable x appears only with nonnegative exponents, then fixing $x = 0$ will minimize all terms in which it appears — but such a solution cannot be attained.

13.2.4.2 Infinite solution

A similar problem will occur if a finite optimal objective value requires a variable to be infinite. This can be illustrated by the following example:

$$\begin{array}{lcl} \text{minimize} & x^{-2} & \\ \text{subject to} & x^{-1} \leq 1, \\ & x>0, \end{array}$$

which is a valid geometric programming problem. In this case the optimal objective is 0, but this requires $x = \infty$, which is unattainable.

Again, this specific case will appear if a variable x appears only with negative exponents in the problem, implying that each term in which it appears can be minimized for $x \to \infty$.

13.2.5 An example

Consider the example

$$\begin{array}{lcll} \text{minimize} & x^{-1}y & & \\ \text{subject to} & x^2y^{-\frac{1}{2}} + 3y^{\frac{1}{2}}z^{-1} & \leq & 1, \\ & xy^{-1} & = & z^2, \\ & -x & \leq & -\frac{1}{10}, \\ & x & \leq & 3, \\ & x,y,z>0, & & \end{array}$$

which is not a geometric optimization problem. However, using the obvious transformations we obtain the problem

$$\begin{array}{lccc} \text{minimize} & x^{-1}y & & \\ \text{subject to} & x^2y^{-\frac{1}{2}} + 3y^{\frac{1}{2}}z^{-1} & \leq & 1, \\ & xy^{-1}z^{-2} & \leq & 1, \\ & x^{-1}yz^2 & \leq & 1, \\ & \frac{1}{10}x^{-1} & \leq & 1, \\ & \frac{1}{3}x & \leq & 1, \\ & x,y,z > 0, & & \end{array} \tag{13.8}$$

which is a geometric optimization problem.

13.2.6 Solving from the command line tool

MOSEK provides the command line tool `mskexpopt` to solve a problem on the form (13.7). As demonstrated previously an optimal solution to this problem can be transformed into an optimal solution to the geometric optimization problem (13.4) by using the transform:

$$t_j = e^{x_j}.$$

A more detailed description of `mskexpopt` and the definition of the input format used is found in Section 6.2. The source code is also included in the MOSEK distribution.

13.2.6.1 An example

The problem (13.8) can be written in the `mskexpopt` format as follows:

```
5   * numcon
3   * numvar
7   * numter
* Coefficients of terms
1
1
3
1
1
0.1
0.333333
* Constraints each term belong to
0
1
1
2
3
4
5
```

```
* Section defining a_kj.
* Format: term var coef
0 0 -1
0 1  1
1 0  2
1 1  -0.5
2 1  0.5
2 2  -1
3 0 1
3 1 -1
3 2 -2
4 0 -1
4 1 1
4 2 2
5 0 -1
6 0 1
```

The command line:

```
mskexpopt go1.eo
```

solves the problem and writes the solution file:

```
PROBLEM STATUS      : PRIMAL_AND_DUAL_FEASIBLE
SOLUTION STATUS     : OPTIMAL
OBJECTIVE           : 1.001904e-03

PRIMAL VARIABLES
INDEX   ACTIVITY
1       -2.302585e+00
2       -9.208438e+00
3       3.452927e+00

DUAL VARIABLES
INDEX   ACTIVITY
1       1.000000e+00
2       2.003813e+00
3       1.906415e-03
4       5.272269e+00
5       5.273223e+00
6       3.006672e+00
7       8.758884e-12
```

The primal solution can be transformed into a solution to the geometric optimization problem as follows

$$t_0 = e^{-2.302585e+00} = 0.1 \tag{13.9}$$

$$t_1 = e^{-9.208438e+00} = 1.0019^{-4} \tag{13.10}$$

$$t_1 = e^{3.452927e+00} = 31.5927. \tag{13.11}$$

13.2.7 Further information

More information about geometric optimization problems is located in [11, 12, 14].

Chapter 14

Usage guidelines

The purpose of this chapter is to present some general guidelines to follow when using MOSEK.

14.1 Verifying the results

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

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

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

The first step in verifying that MOSEK reports the expected solution is to inspect the solution summary generated by MOSEK; see section 8.7 for details. The solution summary provides information about

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

By inspecting the solution summary it can be verified that MOSEK produces a feasible solution, and, in the continuous case, the optimality can be checked using the dual solution. Furthermore, the problem itself ca be inspected using the problem analyzer discussed in section 10.1.

If the summary reports conflicting information (e.g. a solution status that does not match the actual solution), or the cause for terminating the solver before a solution was found cannot be traced back to

the reasons stated above, it may be caused by a bug in the solver; in this case, please contact MOSEK support.

14.1.1 Verifying primal feasibility

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

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

and MOSEK reports the optimal solution

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

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

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

may have been inputted instead of

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

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

14.1.2 Verifying optimality

Verifying that a feasible solution is optimal can be harder. However, for continuous problems[1] optimality can verified using a dual solution. Normally, MOSEK will report a dual solution; if that is feasible and has the same objective value as the primal solution, then the primal solution must be optimal.

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

14.2 Turn on logging

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

Using the `MSK_linkfiletotaskstream` function a file can be linked to a task stream. This means that all messages sent to a task stream are also written to a file. As an example consider the code fragment

[1] A problem without any integer constraints.

```
MSK_linkfiletotaskstream(task,MSK_STREAM_LOG ,"moseklog.txt");
```

which shows how to link the file `moseklog.txt` to the log stream.

It is also possible to link a custom function to a stream using the MSK_linkfunctotaskstream function.

More log information can be obtained by modifying one or more of the parameters:

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

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

```
MSK_IPAR_LOG_CUT_SECOND_OPT 0
```

14.3 Turn on data checking

In the development phase it is useful to use the parameter setting

```
MSK_IPAR_DATA_CHECK MSK_ON
```

which forces MOSEK to check the input data. For instance, MOSEK looks for NANs in double values and outputs a warning if any are found.

14.4 Debugging an optimization task

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

```
MSK_writedata(task,"taskdump.opf");
MSK_optimize(task);
```

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

14.5 Error handling

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

14.6 Fatal error handling

If MOSEK encounter a fatal error caused by either an internal bug or a user error, an *exit function* is called. It is possible to tell MOSEK to use a custom exit function using the `MSK_putexitfunc` function. The user-defined exit function will then be called if a fatal error is detected.

The purpose of an exit function is to print out a suitable message that can help diagnose the cause of the error.

14.7 Checking for memory leaks and overwrites

If you suspect that MOSEK or your own application incorrectly overwrites memory or leaks memory, we suggest you use external tools such as Purify [2] or valgrind [3] to pinpoint the cause of the problem.

Alternatively, MOSEK has a memory check feature which can be enabled by letting the argument `dbugfile` be the name of a writable file when calling `MSK_makeenv`. If `dgbfile` is valid file name, then MOSEK will write memory debug information to this file. Assuming memory debugging is turned on, MOSEK will warn about MOSEK specific memory leaks when a MOSEK environment or task is deleted.

Moreover, the functions `MSK_checkmemenv` and `MSK_checkmemtask` can be used to check the memory allocated by a MOSEK environment or task at any time. If one these functions finds that the memory has been corrupted a fatal error is generated.

14.8 Important API limitations

14.8.1 Thread safety

The MOSEK API is thread-safe provided that a task is accessed from one thread only at any time.

14.8.2 Unicoded strings

The `C` API supports the usage of unicoded strings. Indeed all (`char *`) arguments are allowed to be UTF8 encoded strings.

[2] Purify is a commercial product available from IBM, which runs on Windows and various UNIXes.
[3] Valgrind is open source product available for Linux on X86 and other architectures.

14.8.2.1 Limitations

Please note that the MPS and LP file formats are ASCII formats. Therefore, it might be advantageous to limit all names of constraints, variables etc. to ASCII strings.

14.9 Bug reporting

If you think MOSEK is solving your problem incorrectly, please contact MOSEK support at support@mosek.com providing a detailed description of the problem. MOSEK support may ask for the task file which is produced as follows

```
MSK_writedata(task,"taskfile.mbt");
MSK_optimize(task);
```

The task data will then be written to the `taskfile.mbt` file in binary form which is very useful when reproducing a problem.

Chapter 15

API reference

This chapter lists all functionality in the MOSEK C API.

15.1 Type definitions

- `MSKbooleant`

  Description:
  A signed integer interpreted as a boolean value.

- `MSKenv_t`

  Description:
  The MOSEK Environment type.

- `MSKidxt`

  Description:
  A 32 bits signed integer used for indexing. This is used as indexer into arrays which are guaranteed to not exceed 2^{32} bits in length.

- `MSKint32t`

  Description:
  Signed 32bit integer.

- `MSKint64t`

  Description:
  Signed 64bit integer.

- `MSKintt`

Description:

A signed integer. This is a 32 bits signed integer.

- `MSKlidxt`

  Description:

  A signed integer used for indexing. This is used as indexer into arrays which on some platforms may exceed 2^32 bits in length. On 32-bit architectures it will always be a signed 32 bits integer, while on 64-bit architectures it may be either a 32 or 64 bits signed integer.

- `MSKlintt`

  Description:

  A signed large integer. On 32-bit architectures it is always 32 bits, while on 64-bit architectures it may be either 32 or 64 bits.

- `MSKoprt`

  Description:

  An unsigned integer interpreted as a code list element. Each operation in a code list consists of 4 elements.

- `MSKrealt`

  Description:

  The floating point type used by MOSEK.

- `MSKstring_t`

  Description:

  The string type used by MOSEK. This is an UTF-8 encoded zero-terminated char string.

- `MSKtask_t`

  Description:

  The MOSEK Task type.

- `MSKuint32t`

  Description:

  Unsigned 32bit integer.

- `MSKuint64t`

  Description:

  Unsigned 64bit integer.

- `MSKuserhandle_t`

  Description:

  A pointer to a generic user-defined structure.

- `MSKwchart`

  Description:
  Wide char type. The actual type may differ depending on the platform; it is either a 16 or 32 bits signed or unsigned integer.

- `MSKcallbackfunc`

  Description: Definition of the progress call-back function. The progress call-back function is a user-defined function which will be called by MOSEK occasionally during the optimization process. In particular, the call-back function is called at the beginning of each iteration in the interior-point optimizer. For the simplex optimizers `MSK_IPAR_LOG_SIM_FREQ` controls how frequently the call-back is called.

  Typically the user-defined call-back function displays information about the solution process. The call-back function can also be used to terminate the optimization process since if the progress call-back function returns a non-zero value, the optimization process is aborted.

  It is important that the user-defined call-back function does not modify the optimization task, this will lead to undefined and incorrect results. The only MOSEK functions that can be called safely from within the user-defined call-back function are `MSK_getdouinf` and `MSK_getintinf` which access the task information database. The items in task information database are updated during the optimization process.

  Syntax:
  ```
  MSKintt MSKcallbackfunc (
    MSKtask_t task,
    MSKuserhandle_t usrptr,
    MSKcallbackcodee caller);
  ```

  Arguments: `task` (input)
  An optimization task.

  `usrptr` (input/output)
  A pointer to a user-defined structure.

  `caller` (input)
  An integer which tells where the function was called from. See section 18.7 for the possible values of this argument.

- `MSKexitfunc`

  Description: A user-defined exit function which is called in case of fatal errors to handle an error message and terminate the program. The function should never return.

  Syntax:
  ```
  void MSKexitfunc (
    MSKuserhandle_t usrptr,
    MSKCONST char * file,
    MSKintt line,
    MSKCONST char * msg);
  ```

  Arguments: `usrptr` (input/output)
  A pointer to a user-defined structure.

  `file` (input)
  The name of the file where the fatal error occurred.

`line` (input)
The line number in the file where the fatal error occurred.

`msg` (input)
A message about the error.

- `MSKfreefunc`

  Description: A user-defined memory freeing function.

  Syntax:
  ```
  void MSKfreefunc (
    MSKuserhandle_t usrptr,
    void * buffer);
  ```

  Arguments: `usrptr` (input)
  A pointer to a user-defined structure.

  `buffer` (input/output)
  A pointer to the buffer which should be freed.

- `MSKmallocfunc`

  Description: A user-defined memory allocation function.

  Syntax:
  ```
  void * MSKmallocfunc (
    MSKuserhandle_t usrptr,
    MSKCONST size_t size);
  ```

  Arguments: `usrptr` (input)
  A pointer to a user-defined structure.

  `size` (input)
  The number of characters to allocate.

- `MSKnlgetspfunc`

  Description: Type definition of the call-back function which is used to provide structural information about the nonlinear functions f and g in the optimization problem.

  Hence, it is the user's responsibility to provide a function satisfying the definition. The function is inputted to MOSEK using the API function `MSK_putnlfunc`.

  Syntax:
  ```
  MSKintt MSKnlgetspfunc (
    MSKuserhandle_t nlhandle,
    MSKintt * numgrdobjnz,
    MSKidxt * grdobjsub,
    MSKidxt i,
    MSKbooleant * convali,
    MSKintt * grdconinz,
    MSKidxt * grdconisub,
    MSKintt yo,
    MSKintt numycnz,
    MSKCONST MSKidxt * ycsub,
    MSKintt maxnumhesnz,
    MSKintt * numhesnz,
  ```

```
MSKidxt * hessubi,
MSKidxt * hessubj);
```

Arguments: `nlhandle` (input/output)
A pointer to a user-defined data structure specified when the function is attached to a task using the function MSK_putnlfunc.

`numgrdobjnz` (output)
If requested, `numgrdobjnz` should be assigned the number of non-zero elements in the gradient of f.

`grdobjsub` (output)
If requested, put here the positions of the non-zero elements in the gradient of f. The elements are stored in

$$\texttt{grdobjsub}[0,..,\texttt{numgrdobjsub}-1.]$$

`i` (input)
Index of a constraint. If $i < 0$ or $i \geq$ `numcon`, no information about a constraint is requested.

`convali` (output)
If requested, assign a true/false value indicating if constraint `i` contains general non-linear terms.

`grdconinz` (output)
If requested, `grdconinz` shall be assigned the number of non-zero elements in $\nabla g_i(x)$.

`grdconisub` (output)
If requested, this array shall contain the indexes of the non-zeros in $\nabla g_i(x)$. The length of the array must be the same as given in `grdconinz`.

`yo` (input)
If non-zero, then the f shall be included when the gradient and the Hessian of the Lagrangian are computed.

`numycnz` (input)
Number of constraint functions which are included in the definition of the Lagrangian. See (15.1).

`ycsub` (input)
Index of constraint functions which are included in the definition of the Lagrangian. See (15.1).

`maxnumhesnz` (input)
Length of the arguments `hessubi` and `hessubj`.

`numhesnz` (output)
If requested, `numhesnz` should be assigned the number of non-zero elements in the lower triangular part of the Hessian of the Lagrangian:

$$L := \texttt{yo} f(x) - \sum_{k=0}^{\texttt{numycnz}-1} g_{\texttt{ycsub[k]}}(x) \tag{15.1}$$

`hessubi` (output)

If requested, `hessubi` and `hessubj` are used to convey the position of the non-zeros in the Hessian of the Lagrangian L (see (15.1)) as follows

$$\nabla^2 L_{\texttt{hessubi}[k],\texttt{hessubj}[k]}(x) \neq 0.0 \tag{15.2}$$

for $k = 0, \ldots, \texttt{numhesnz} - 1$. All other positions in L are assumed to be zero. Please note that *only* the lower *or* the upper triangular part of the Hessian should be return.

`hessubj` (output)
See the argument `hessubi`.

- `MSKnlgetvafunc`

Description: Type definition of the call-back function which is used to provide structural and numerical information about the nonlinear functions f and g in an optimization problem.

For later use we need the definition of the Lagrangian L which is given by

$$L := \texttt{yo} * f(\texttt{xx}) - \sum_{k=0}^{\texttt{numi}-1} \texttt{yc}_{\texttt{subi}[\texttt{k}]} g_{\texttt{subi}[\texttt{k}]}(\texttt{xx}). \tag{15.3}$$

Syntax:
```
MSKintt MSKnlgetvafunc (
MSKuserhandle_t nlhandle,
MSKCONST MSKrealt * xx,
MSKrealt yo,
MSKCONST MSKrealt * yc,
MSKrealt * objval,
MSKintt * numgrdobjnz,
MSKidxt * grdobjsub,
MSKrealt * grdobjval,
MSKintt numi,
MSKCONST MSKidxt * subi,
MSKrealt * conval,
MSKCONST MSKintt * grdconptrb,
MSKCONST MSKintt * grdconptre,
MSKCONST MSKidxt * grdconsub,
MSKrealt * grdconval,
MSKrealt * grdlag,
MSKintt maxnumhesnz,
MSKintt * numhesnz,
MSKidxt * hessubi,
MSKidxt * hessubj,
MSKrealt * hesval);
```

Arguments: `nlhandle` (input/output)
A pointer to a user-defined data structure. The pointer is passed to MOSEK when the function `MSK_putnlfunc` is called.

`xx` (input)
The point at which the nonlinear function must be evaluated. The length equals the number of variables in the task.

`yo` (input)
Multiplier on the objective function f.

`yc` (input)
Multipliers for the constraint functions g_i. The length is `numcon`.

`objval` (output)
If requested, `objval` shall be assigned the value of f evaluated at xx.

`numgrdobjnz` (output)
If requested, `numgrdobjnz` shall be assigned the number of non-zero elements in the gradient of f.

`grdobjsub` (output)
If requested, it shall contain the position of the non-zero elements in the gradient of f. The elements are stored in

$$\texttt{grdobjsub}[0,\ldots,\texttt{numgrdobjnz}-1].$$

`grdobjval` (output)
If requested, it shall contain the the gradient of f evaluated at `xx`. The following data structure

$$\texttt{grdobjval}[\texttt{k}] = \frac{\partial f}{\partial x_{\texttt{grdobjsub}[\texttt{k}]}}(\texttt{xx})$$

for $k = 0,\ldots,\texttt{numgrdobjnz}-1$ is used.

`numi` (input)
Number of elements in `subi`.

`subi` (input)
`subi`$[0,...,$`numi`$-1]$ contain the indexes of the constraints that has to be evaluated. The length is `numi`.

`conval` (output)
$g($`xx`$)$ for the required constraint functions i.e.

$$\texttt{conval}[\texttt{k}] = g_{\texttt{subi}[\texttt{k}]}(\texttt{xx})$$

for $k = 0,\ldots,\texttt{numi}-1$.

`grdconptrb` (input)
If given, it specifies the structure of the gradients of the constraint functions. See the argument `grdconval` for details.

`grdconptre` (input)
If given, it specifies the structure of the gradients of the constraint functions. See the argument `grdconval` for details.

`grdconsub` (input)
If requested, it shall specify the positions of the non-zeros in gradients of the constraints. See the argument `grdconval` for details.

`grdconval` (output)
If requested, it shall specify the values of the gradient of the nonlinear constraints. Together `grdconptrb`, `grdconptre`, `grdconsub` and `grdconval` are used to specify the gradients of the nonlinear constraint functions.

Please note that both `grdconsub` and `grdconval` should be computed when requested.
The gradient data is stored as follows

$$\texttt{grdconval}[\texttt{k}] = \frac{\partial g_{\texttt{subi}[\texttt{i}]}(xx)}{\partial xx_{\texttt{grdconsub}[\texttt{k}]}}, \quad \text{for}$$
$$k = \texttt{grdconptrb}[i], \ldots, \texttt{grdconptre}[i] - 1,$$
$$i = 0, \ldots, \texttt{numi} - 1.$$

`grdlag` (output)
If requested, `grdlag` shall contain the gradient of the Lagrangian function, i.e.

$$\texttt{grdlag} = \nabla L.$$

`maxnumhesnz` (input)
Maximum number of non-zeros in the Hessian of the Lagrangian, i.e. `maxnumhesnz` is the length of the arrays `hessubi`, `hessubj`, and `hesval`.

`numhesnz` (output)
If requested, `numhesnz` shall be assigned the number of non-zeros elements in the Hessian of the Lagrangian L. See (15.3).

`hessubi` (output)
See the argument `hesval`.

`hessubj` (output)
See the argument `hesval`.

`hesval` (output)
Together `hessubi`, `hessubj`, and `hesval` specify the Hessian of the Lagrangian function L defined in (15.3).
The Hessian is stored in the following format:

$$\texttt{hesval}[k] = \nabla^2 L_{\min(\texttt{hessubi}[k],\texttt{hessubj}[k]),\max(\texttt{hessubi}[k],\texttt{hessubj}[k])}$$

for $k = 0, \ldots, \texttt{numhesnz}[0]-1$. Please note that if an element is specified multiple times, then the elements are added together. Hence, *only* the lower *or* the upper triangular part of the Hessian should be returned.

- `MSKresponsefunc`

Description: Whenever MOSEK generate a warning or an error this function is called. The argument `r` contains the code of the error/warning and the argument `msg` contains the corresponding error/warning message. This function should always return `MSK_RES_OK`.

Syntax:
```
MSKrescodee MSKresponsefunc (
 MSKuserhandle_t handle,
 MSKrescodee r,
 MSKCONST char * msg);
```

Arguments: `handle` (input/output)
A pointer to a user-defined data structure or NULL.

`r` (input)
The response code corresponding to the exception.

`msg` (input)
A string containing the exception message.

- `MSKstreamfunc`

  Description: A function of this type can be linked to any of the MOSEK streams. This implies that if a message is send to the stream to which the function is linked, the function is called by MOSEK and the argument `str` will contain the message. Hence, the user can decide what should happen to message.

  Syntax:
  ```
  void MSKstreamfunc (
       MSKuserhandle_t handle,
       MSKCONST char * str);
  ```

  Arguments: `handle` (input/output)
  A pointer to a user-defined data structure (or a null pointer).

  `str` (input)
  A string containing a message to a stream.

15.2 API Functionality

Functions in the interface grouped by functionality.

15.2.1 Analyzing the problem and associated data

Analyzing the problem and associated data.

MSK_analyzeproblem (page 295)
Analyze the data of a task.

MSK_analyzesolution (page 296)
Print information related to the quality of the solution.

15.2.2 Reading and writing data files

Reading and writing data files.

MSK_readbranchpriorities (page 381)
Reads branching priority data from a file.

MSK_readdata (page 382)
Reads problem data from a file.

MSK_readparamfile (page 382)
Reads a parameter file.

MSK_readsolution (page 382)
Reads a solution from a file.

MSK_writebranchpriorities (page 391)
Writes branching priority data to a file.

MSK_writedata (page 391)
Writes problem data to a file.

MSK_writeparamfile (page 392)
Writes all the parameters to a parameter file.

MSK_writesolution (page 392)
Write a solution to a file.

15.2.3 Solutions

Obtain or define a solution.

MSK_deletesolution (page 302)
Undefines a solution and frees the memory it uses.

MSK_getdbi (page 313)
Obtains the dual bound infeasibility.

MSK_getdcni (page 313)
Obtains the dual cone infeasibility.

MSK_getdeqi (page 314)
Optains the dual equation infeasibility.

MSK_getdualobj (page 315)
Obtains the dual objective value.

MSK_getinti (page 317)
Obtains the primal equation infeasibility.

MSK_getpbi (page 331)
Obtains the primal bound infeasibility.

MSK_getpcni (page 332)
Obtains the primal cone infeasibility.

MSK_getpeqi (page 333)
Obtains the primal equation infeasibility.

MSK_getprimalobj (page 333)
Obtains the primal objective value.

MSK_getreducedcosts (page 336)
Obtains the difference of (slx-sux) for a sequence of variables.

MSK_getsolution (page 337)
Obtains the complete solution.

MSK_getsolutioni (page 338)
Obtains the solution for a single constraint or variable.

MSK_getsolutionincallback (page 339)
Obtains the whole or a part of the solution from the progress call-back function.

MSK_getsolutioninf (page 340)
Obtains information about a solution.

MSK_getsolutionslice (page 342)
Obtains a slice of the solution.

MSK_getsolutionstatus (page 343)
Obtains information about the problem and solution statuses.

MSK_getsolutionstatuskeyslice (page 344)
Obtains a slice of the solution status keys.

MSK_makesolutionstatusunknown (page 353)
Sets the solution status to unknown.

MSK_optimizersummary (page 357)
Prints a short summary with optimizer statistics for last optimization.

MSK_putsolution (page 378)
Inserts a solution.

MSK_putsolutioni (page 379)
Sets the primal and dual solution information for a single constraint or variable.

MSK_readsolution (page 382)
Reads a solution from a file.

MSK_solstatostr (page 387)
Obtains a solution status string.

MSK_solutiondef (page 387)
Checks whether a solution is defined.

MSK_solutionsummary (page 388)
Prints a short summary of the current solutions.

MSK_undefsolution (page 390)
Undefines a solution.

15.2.4 Call-backs (put/get)

Manipulating call-backs.

MSK_getcallbackfunc (page 310)
Obtains the call-back function and the associated user handle.

MSK_getnlfunc (page 326)
Gets nonlinear call-back functions.

MSK_getsolutionincallback (page 339)
Obtains the whole or a part of the solution from the progress call-back function.

MSK_linkfunctoenvstream (page 277)
Connects a user-defined function to a stream.

MSK_linkfunctotaskstream (page 352)
Connects a user-defined function to a task stream.

MSK_putcallbackfunc (page 367)
Input the progress call-back function.

MSK_putexitfunc (page 280)
Inputs a user-defined exit function which is called in case of fatal errors.

MSK_putnlfunc (page 373)
Inputs nonlinear function information.

MSK_putresponsefunc (page 378)
Inputs a user-defined error call-back function.

MSK_unlinkfuncfromenvstream (page 282)
Disconnects a user-defined function from a stream.

MSK_unlinkfuncfromtaskstream (page 391)
Disconnects a user-defined function from a task stream.

15.2.5 Memory allocation and deallocation

Memory allocation and deallocation.

MSK_callocdbgenv (page 271)
A replacement for the system calloc function.

MSK_callocdbgtask (page 299)
A replacement for the system calloc function.

MSK_callocenv (page 271)
A replacement for the system calloc function.

MSK_calloctask (page 299)
A replacement for the system calloc function.

MSK_checkmemenv (page 272)
Checks the memory allocated by the environment.

MSK_checkmemtask (page 300)
Checks the memory allocated by the task.

MSK_freedbgenv (page 274)
Frees space allocated by MOSEK.

MSK_freedbgtask (page 304)
Frees space allocated by MOSEK.

MSK_freeenv (page 274)
Frees space allocated by MOSEK.

MSK_freetask (page 304)
Frees space allocated by MOSEK.

MSK_getmemusagetask (page 322)
Obtains information about the amount of memory used by a task.

MSK_getmemusagetask64 (page 322)
Obtains information about the amount of memory used by a task.

15.2.6 Changing problem specification

Input or change problem specification.

MSK_append (page 296)
Appends a number of variables or constraints to the optimization task.

MSK_appendcone (page 297)
Appends a new cone constraint to the problem.

MSK_checkoutlicense (page 272)
Check out a license feature from the license server ahead of time.

MSK_chgbound (page 300)
Changes the bounds for one constraint or variable.

MSK_clonetask (page 301)
Creates a clone of an existing task.

MSK_commitchanges (page 301)
Commits all cached problem changes.

MSK_inputdata (page 349)
Input the linear part of an optimization task in one function call.

MSK_inputdata64 (page 350)
Input the linear part of an optimization task in one function call.

MSK_putaij (page 362)
Changes a single value in the linear coefficient matrix.

MSK_putaijlist (page 362)
Changes one or more coefficients in the linear constraint matrix.

MSK_putavec (page 363)
Replaces all elements in one row or column of the linear coefficient matrix.

MSK_putaveclist (page 363)
Replaces all elements in one or more rows or columns in the linear constraint matrix by new values.

MSK_putaveclist64 (page 364)
Replaces all elements in one or more rows or columns in the linear constraint matrix by new values.

MSK_putbound (page 365)
Changes the bound for either one constraint or one variable.

MSK_putboundlist (page 366)
Changes the bounds of constraints or variables.

MSK_putboundslice (page 367)
Modifies bounds.

MSK_putcfix (page 368)
Replaces the fixed term in the objective.

MSK_putcj (page 368)
Modifies one linear coefficient in the objective.

MSK_putclist (page 368)
Modifies a part of the linear objective coefficients.

MSK_putcone (page 369)
Replaces a conic constraint.

MSK_putobjsense (page 374)
Sets the objective sense.

MSK_putqcon (page 375)
Replaces all quadratic terms in constraints.

MSK_putqconk (page 375)
Replaces all quadratic terms in a single constraint.

MSK_putqobj (page 376)
Replaces all quadratic terms in the objective.

MSK_putqobjij (page 377)
Replaces one coefficient in the quadratic term in the objective.

MSK_putvartype (page 381)
Sets the variable type of one variable.

MSK_putvartypelist (page 381)
Sets the variable type for one or more variables.

15.2.7 Delete problem elements (variables,constraints,cones)

Functionality for deleting problem elements such as variables, constraints or cones.

MSK_remove (page 385)
The function removes a number of constraints or variables.

MSK_removecone (page 385)
Removes a conic constraint from the problem.

15.2.8 Add problem elements (variables,constraints,cones)

Functionality for adding problem elements such as variables, constraints or cones.

MSK_append (page 296)
Appends a number of variables or constraints to the optimization task.

MSK_appendcone (page 297)
Appends a new cone constraint to the problem.

15.2.9 Problem inspection

Functionality for inspecting the problem specification (A,Q, bounds, objective e.t.c).

MSK_getaij (page 304)
Obtains a single coefficient in linear constraint matrix.

MSK_getaslice (page 305)
Obtains a sequence of rows or columns from the coefficient matrix.

MSK_getaslice64 (page 306)
Obtains a sequence of rows or columns from the coefficient matrix.

MSK_getaslicetrip (page 307)
Obtains a sequence of rows or columns from the coefficient matrix in triplet format.

MSK_getavec (page 308)
Obtains one row or column of the linear constraint matrix.

MSK_getavecnumnz (page 309)
Obtains the number of non-zero elements in one row or column of the linear constraint matrix

MSK_getbound (page 309)
Obtains bound information for one constraint or variable.

MSK_getboundslice (page 310)
Obtains bounds information for a sequence of variables or constraints.

MSK_getc (page 310)
Obtains all objective coefficients.

MSK_getcfix (page 311)
Obtains the fixed term in the objective.

MSK_getcone (page 311)
Obtains a conic constraint.

MSK_getconeinfo (page 311)
Obtains information about a conic constraint.

MSK_getcslice (page 312)
Obtains a sequence of coefficients from the objective.

MSK_getintpntnumthreads (page 318)
Obtains the number of threads used by the interior-point optimizer.

MSK_getnumanz (page 327)
Obtains the number of non-zeros in the coefficient matrix.

MSK_getnumanz64 (page 327)
Obtains the number of non-zeros in the coefficient matrix.

MSK_getnumcon (page 327)
Obtains the number of constraints.

MSK_getnumcone (page 328)
Obtains the number of cones.

MSK_getnumconemem (page 328)
Obtains the number of members in a cone.

MSK_getnumintvar (page 328)
Obtains the number of integer-constrained variables.

MSK_getnumqconknz (page 329)
Obtains the number of non-zero quadratic terms in a constraint.

MSK_getnumqconknz64 (page 329)
Obtains the number of non-zero quadratic terms in a constraint.

MSK_getnumqobjnz (page 329)
Obtains the number of non-zero quadratic terms in the objective.

MSK_getnumqobjnz64 (page 329)
Obtains the number of non-zero quadratic terms in the objective.

MSK_getnumvar (page 330)
Obtains the number of variables.

MSK_getobjsense (page 331)
Gets the objective sense.

MSK_getprobtype (page 333)
Obtains the problem type.

MSK_getqconk (page 334)
Obtains all the quadratic terms in a constraint.

MSK_getqconk64 (page 334)
Obtains all the quadratic terms in a constraint.

MSK_getqobj (page 335)
Obtains all the quadratic terms in the objective.

MSK_getqobj64 (page 335)
Obtains all the quadratic terms in the objective.

MSK_getqobjij (page 336)
Obtains one coefficient from the quadratic term of the objective

MSK_getvartype (page 348)
Gets the variable type of one variable.

MSK_getvartypelist (page 348)
Obtains the variable type for one or more variables.

15.2.10 Conic constraints

Functionality related to conic terms in the problem.

MSK_appendcone (page 297)
Appends a new cone constraint to the problem.

MSK_getcone (page 311)
Obtains a conic constraint.

MSK_getconeinfo (page 311)
Obtains information about a conic constraint.

MSK_getnumcone (page 328)
Obtains the number of cones.

MSK_putcone (page 369)
Replaces a conic constraint.

MSK_removecone (page 385)
Removes a conic constraint from the problem.

15.2.11 Bounds

Functionality related to changing or inspecting bounds on variables or constraints.

MSK_chgbound (page 300)
Changes the bounds for one constraint or variable.

MSK_getbound (page 309)
Obtains bound information for one constraint or variable.

MSK_getboundslice (page 310)
Obtains bounds information for a sequence of variables or constraints.

MSK_putbound (page 365)
Changes the bound for either one constraint or one variable.

MSK_putboundlist (page 366)
Changes the bounds of constraints or variables.

MSK_putboundslice (page 367)
Modifies bounds.

15.2.12 Task initialization and deletion

Task initialization and deletion.

MSK_deletetask (page 302)
Deletes an optimization task.

MSK_makeemptytask (page 278)
Creates a new and empty optimization task.

MSK_maketask (page 279)
Creates a new optimization task.

15.2.13 Error handling

Error handling.

MSK_exceptiontask (page 303)
Echo a response code to a task stream.

MSK_getcodedesc (page 275)
Obtains a short description of a response code.

MSK_getcodedisc (page 275)
Obtains a short description of a response code.

MSK_getresponseclass (page 276)
Obtain the class of a response code.

MSK_putresponsefunc (page 378)
Inputs a user-defined error call-back function.

15.2.14 Output stream functions

Output stream functions.

MSK_echoenv (page 273)
Sends a message to a given environment stream.

MSK_echointro (page 274)
Prints an intro to message stream.

MSK_echotask (page 303)
Prints a format string to a task stream.

MSK_exceptiontask (page 303)
Echo a response code to a task stream.

MSK_linkfiletoenvstream (page 277)
Directs all output from a stream to a file.

MSK_linkfiletotaskstream (page 352)
Directs all output from a task stream to a file.

MSK_linkfunctoenvstream (page 277)
Connects a user-defined function to a stream.

MSK_linkfunctotaskstream (page 352)
Connects a user-defined function to a task stream.

MSK_optimizersummary (page 357)
Prints a short summary with optimizer statistics for last optimization.

MSK_printdata (page 360)
Prints a part of the problem data to a stream.

MSK_printparam (page 361)
Prints the current parameter settings.

MSK_readsummary (page 383)
Prints information about last file read.

MSK_solutionsummary (page 388)
Prints a short summary of the current solutions.

MSK_unlinkfuncfromenvstream (page 282)
Disconnects a user-defined function from a stream.

MSK_unlinkfuncfromtaskstream (page 391)
Disconnects a user-defined function from a task stream.

15.2.15 Objective function

Change or inspect objective function.

MSK_getc (page 310)
Obtains all objective coefficients.

MSK_getcfix (page 311)
Obtains the fixed term in the objective.

MSK_getcslice (page 312)
Obtains a sequence of coefficients from the objective.

MSK_getdualobj (page 315)
Obtains the dual objective value.

MSK_getnumqobjnz (page 329)
Obtains the number of non-zero quadratic terms in the objective.

MSK_getnumqobjnz64 (page 329)
Obtains the number of non-zero quadratic terms in the objective.

MSK_getobjname (page 330)
Obtains the name assigned to the objective function.

MSK_getobjname64 (page 330)
Obtains the name assigned to the objective function.

MSK_getobjsense (page 331)
Gets the objective sense.

MSK_getprimalobj (page 333)
Obtains the primal objective value.

MSK_getqobj (page 335)
Obtains all the quadratic terms in the objective.

MSK_getqobj64 (page 335)
Obtains all the quadratic terms in the objective.

MSK_getqobjij (page 336)
Obtains one coefficient from the quadratic term of the objective

MSK_putcfix (page 368)
Replaces the fixed term in the objective.

MSK_putcj (page 368)
Modifies one linear coefficient in the objective.

MSK_putclist (page 368)
Modifies a part of the linear objective coefficients.

MSK_putobjsense (page 374)
Sets the objective sense.

MSK_putqobj (page 376)
Replaces all quadratic terms in the objective.

MSK_putqobjij (page 377)
Replaces one coefficient in the quadratic term in the objective.

15.2.16 Optimizer statistics

Inspect statistics from the optimizer.

MSK_getdouinf (page 315)
Obtains a double information item.

MSK_getinfindex (page 316)
Obtains the index of a named information item.

MSK_getinfmax (page 316)
Obtains the maximum index of an information of a given type inftype plus 1.

MSK_getinfname (page 317)
Obtains the name of an information item.

MSK_getintinf (page 317)
Obtains an integer information item.

MSK_getlintinf (page 319)
Obtains an integer information item.

MSK_getnadouinf (page 322)
Obtains a double information item.

MSK_getnaintinf (page 323)
Obtains an integer information item.

MSK_getnaintparam (page 323)
Obtains an integer parameter.

15.2.17 Parameters (set/get)

Setting and inspecting solver parameters.

MSK_getdouparam (page 315)
Obtains a double parameter.

MSK_getintparam (page 318)
Obtains an integer parameter.

MSK_getnadouparam (page 322)
Obtains a double parameter.

MSK_getnastrparam (page 326)
Obtains a string parameter.

MSK_getnastrparamal (page 326)
Obtains the value of a string parameter.

MSK_getnumparam (page 328)
Obtains the number of parameters of a given type.

MSK_getparammax (page 331)
Obtains the maximum index of a parameter of a given type plus 1.

MSK_getparamname (page 331)
Obtains the name of a parameter.

MSK_getstrparam (page 344)
Obtains the value of a string parameter.

MSK_getstrparam64 (page 344)
Obtains the value of a string parameter.

MSK_getstrparamal (page 345)
Obtains the value a string parameter.

MSK_getsymbcondim (page 276)
Obtains dimensional information for the defined symbolic constants.

MSK_iparvaltosymnam (page 277)
Obtains the symbolic name corresponding to a value that can be assigned to an integer parameter.

MSK_isdouparname (page 351)
Checks a double parameter name.

MSK_isintparname (page 351)
Checks an integer parameter name.

MSK_isstrparname (page 352)
Checks a string parameter name.

MSK_putdouparam (page 369)
Sets a double parameter.

MSK_putintparam (page 369)
Sets an integer parameter.

MSK_putnadouparam (page 372)
Sets a double parameter.

MSK_putnaintparam (page 372)
Sets an integer parameter.

MSK_putnastrparam (page 373)
Sets a string parameter.

MSK_putparam (page 374)
Modifies the value of parameter.

MSK_putstrparam (page 380)
Sets a string parameter.

MSK_setdefaults (page 387)
Resets all parameters values.

MSK_symnamtovalue (page 282)
Obtains the value corresponding to a symbolic name defined by MOSEK.

MSK_whichparam (page 391)
Checks a parameter name.

15.2.18 Naming

Functionality related to naming.

MSK_getconname (page 312)
Obtains a name of a constraint.

MSK_getconname64 (page 312)
Obtains a name of a constraint.

MSK_getmaxnamelen (page 319)
Obtains the maximum length (not including terminating zero character) of any objective, constraint, variable or cone name.

MSK_getname (page 323)
Obtains the name of a cone, a variable or a constraint.

MSK_getname64 (page 324)
Obtains the name of a cone, a variable or a constraint.

MSK_getnameapi64 (page 324)
Obtains the name of a cone, a variable or a constraint.

MSK_getnameindex (page 325)
Checks whether a name has been assigned and returns the index corresponding to the name.

MSK_getnamelen64 (page 325)
Obtains the length of a problem item name.

MSK_getobjname (page 330)
Obtains the name assigned to the objective function.

MSK_getobjname64 (page 330)
Obtains the name assigned to the objective function.

MSK_gettaskname (page 346)
Obtains the task name.

MSK_gettaskname64 (page 346)
Obtains the task name.

MSK_getvarname (page 347)
Obtains a name of a variable.

MSK_getvarname64 (page 347)
Obtains a name of a variable.

MSK_putname (page 373)
Assigns a name to a problem item.

MSK_putobjname (page 374)
Assigns a new name to the objective.

MSK_puttaskname (page 380)
Assigns a new name to the task.

15.2.19 Preallocating space for problem data

Functionality related to preallocating space for problem data.

MSK_getmaxnumanz (page 320)
Obtains number of preallocated non-zeros in the linear constraint matrix.

MSK_getmaxnumanz64 (page 320)
Obtains number of preallocated non-zeros in the linear constraint matrix.

MSK_getmaxnumcon (page 320)
Obtains the number of preallocated constraints in the optimization task.

MSK_getmaxnumcone (page 320)
Obtains the number of preallocated cones in the optimization task.

MSK_getmaxnumqnz (page 321)
Obtains the number of preallocated non-zeros for all quadratic terms in objective and constraints.

MSK_getmaxnumqnz64 (page 321)
Obtains the number of preallocated non-zeros for all quadratic terms in objective and constraints.

MSK_getmaxnumvar (page 321)
Obtains the maximum number variables allowed.

MSK_putmaxnumanz (page 369)
The function changes the size of the preallocated storage for linear coefficients.

MSK_putmaxnumanz64 (page 370)
The function changes the size of the preallocated storage for linear coefficients.

MSK_putmaxnumcon (page 370)
Sets the number of preallocated constraints in the optimization task.

MSK_putmaxnumcone (page 371)
Sets the number of preallocated conic constraints in the optimization task.

MSK_putmaxnumqnz (page 371)
Changes the size of the preallocated storage for quadratic terms.

MSK_putmaxnumqnz64 (page 371)
Changes the size of the preallocated storage for quadratic terms.

MSK_putmaxnumvar (page 372)
Sets the number of preallocated variables in the optimization task.

15.2.20 Integer variables

Functionality related to integer variables.

MSK_getnumintvar (page 328)
Obtains the number of integer-constrained variables.

MSK_getvarbranchdir (page 346)
Obtains the branching direction for a variable.

MSK_getvarbranchorder (page 346)
Obtains the branching priority for a variable.

MSK_getvarbranchpri (page 347)
Obtains the branching priority for a variable.

MSK_getvartype (page 348)
Gets the variable type of one variable.

MSK_getvartypelist (page 348)
Obtains the variable type for one or more variables.

MSK_putvarbranchorder (page 380)
Assigns a branching priority and direction to a variable.

MSK_putvartype (page 381)
Sets the variable type of one variable.

MSK_putvartypelist (page 381)
Sets the variable type for one or more variables.

15.2.21 Quadratic terms

Functionality related to quadratic terms.

MSK_getqconk (page 334)
Obtains all the quadratic terms in a constraint.

MSK_getqconk64 (page 334)
Obtains all the quadratic terms in a constraint.

MSK_getqobj (page 335)
Obtains all the quadratic terms in the objective.

MSK_getqobj64 (page 335)
Obtains all the quadratic terms in the objective.

MSK_getqobjij (page 336)
Obtains one coefficient from the quadratic term of the objective

MSK_putqcon (page 375)
Replaces all quadratic terms in constraints.

MSK_putqconk (page 375)
Replaces all quadratic terms in a single constraint.

MSK_putqobj (page 376)
Replaces all quadratic terms in the objective.

MSK_putqobjij (page 377)
Replaces one coefficient in the quadratic term in the objective.

15.2.22 Diagnosing infeasibility

Functions for diagnosing infeasibility.

MSK_getinfeasiblesubproblem (page 316)
Obtains an infeasible sub problem.

MSK_relaxprimal (page 383)
Creates a problem that finds the minimal change to the bounds that makes an infeasible problem feasible.

15.2.23 Optimization

Functions for optimization.

MSK_checkdata (page 300)
Checks data of the task.

MSK_netoptimize (page 354)
Optimizes a pure network flow problem.

MSK_optimize (page 356)
Optimizes the problem.

MSK_optimizeconcurrent (page 356)
Optimize a given task with several optimizers concurrently.

MSK_optimizetrm (page 357)
Optimizes the problem.

15.2.24 Network optimization

Functions for network optimization.

MSK_netextraction (page 353)
Finds embedded network structure.

MSK_netoptimize (page 354)
Optimizes a pure network flow problem.

15.2.25 Sensitivity analysis

Functions for sensitivity analysis.

MSK_dualsensitivity (page 302)
Performs sensitivity analysis on objective coefficients.

MSK_primalsensitivity (page 358)
Perform sensitivity analysis on bounds.

MSK_sensitivityreport (page 386)
Creates a sensitivity report.

15.2.26 Testing data validity

Functions for testing data validity.

MSK_checkconvexity (page 299)
Checks if a quadratic optimization problem is convex.

15.2.27 Solving with the basis

Functions for solving linear systems with the basis matrix.

MSK_basiscond (page 298)
Computes conditioning information for the basis matrix.

MSK_initbasissolve (page 349)
Prepare a task for basis solver.

MSK_solvewithbasis (page 388)
Solve a linear equation system involving a basis matrix.

15.2.28 Initialization of environment

Creation and initialization of environment.

MSK_checkinlicense (page 272)
Check in a license feature from the license server ahead of time.

MSK_deleteenv (page 273)
Delete a MOSEK environment.

MSK_initenv (page 276)
Initialize a MOSEK environment.

MSK_makeenv (page 278)
Creates a new MOSEK environment.

MSK_putlicensedefaults (page 280)
Set defaults used by the license manager.

15.2.29 Change A

Change elements in the coefficient (A) matrix.

MSK_checkoutlicense (page 272)
Check out a license feature from the license server ahead of time.

MSK_commitchanges (page 301)
Commits all cached problem changes.

MSK_putaij (page 362)
Changes a single value in the linear coefficient matrix.

MSK_putaijlist (page 362)
Changes one or more coefficients in the linear constraint matrix.

MSK_putavec (page 363)
Replaces all elements in one row or column of the linear coefficient matrix.

MSK_putaveclist (page 363)
Replaces all elements in one or more rows or columns in the linear constraint matrix by new values.

MSK_putaveclist64 (page 364)
Replaces all elements in one or more rows or columns in the linear constraint matrix by new values.

15.3 Mosek Env

Description:

A Mosek Environment

15.3.1 Methods

- `MSK_callocdbgenv`

  Syntax:

  ```
  void * MSK_callocdbgenv (
      MSKenv_t env,
      MSKCONST size_t number,
      MSKCONST size_t size,
      MSKCONST char * file,
      MSKCONST unsigned line);
  ```

  `env` **(input)** The MOSEK environment.
  `number` **(input)** Number of elements.
  `size` **(input)** Size of each individual element.
  `file` **(input)** File from which the function is called.
  `line` **(input)** Line in the file from which the function is called.

  Description: Debug version of `MSK_callocenv`.

- `MSK_callocenv`

  Syntax:

```
void * MSK_callocenv (
    MSKenv_t env,
    MSKCONST size_t number,
    MSKCONST size_t size);
```

`env` **(input)** The MOSEK environment.

`number` **(input)** Number of elements.

`size` **(input)** Size of each individual element.

Description: Equivalent to `calloc` i.e. allocate space for an array of length number where each element is of size `size`.

- `MSK_checkinlicense`

  Syntax:

  ```
  MSKrescodee MSK_checkinlicense (
      MSKenv_t env,
      MSKfeaturee feature);
  ```

  `env` **(input)** The MOSEK environment.

  `feature` **(input)** Feature to check in to the license system.

  Description: Check in a license feature to the license server. By default all licenses consumed by functions using a single environment is kept checked out for the lifetime of the MOSEK environment. This function checks in a given license feature to the license server immidiatly.

  If the given license feature is not checked out or is in use by a call to `MSK_optimizetrm` calling this function has no effect.

  Please note that returning a license to the license server incurs a small overhead, so frequent calls to this function should be avoided.

- `MSK_checkmemenv`

  Syntax:

  ```
  MSKrescodee MSK_checkmemenv (
      MSKenv_t env,
      MSKCONST char * file,
      MSKintt line);
  ```

  `env` **(input)** The MOSEK environment.

  `file` **(input)** File from which the function is called.

  `line` **(input)** Line in the file from which the function is called.

  Description: Checks the memory allocated by the environment.

- `MSK_checkoutlicense`

  Syntax:

  ```
  MSKrescodee MSK_checkoutlicense (
      MSKenv_t env,
      MSKfeaturee feature);
  ```

`env` **(input)** The MOSEK environment.

`feature` **(input)** Feature to check out from the license system.

Description: Check out a license feature from the license server. Normally the required license features will be automatically checked out the first time it is needed by the function `MSK_optimizetrm`. This function can be used to check out one or more features ahead of time.

The license will remain checked out until the environment is deleted with `MSK_deleteenv` or the function `MSK_checkinlicense` is called.

If a given feature is already checked out when this function is called, only one feature will be checked out from the license server.

- `MSK_checkversion`

Syntax:

```
MSKrescodee MSK_checkversion (
    MSKenv_t env,
    MSKintt major,
    MSKintt minor,
    MSKintt build,
    MSKintt revision);
```

`env` **(input)** The MOSEK environment.

`major` **(input)** Major version number.

`minor` **(input)** Minor version number.

`build` **(input)** Build number.

`revision` **(input)** Revision number.

Description: Compares the version of the MOSEK DLL with a specified version. Normally the specified version is the version at the build time.

- `MSK_deleteenv`

Syntax:

```
MSKrescodee MSK_deleteenv (MSKenv_t * env)
```

`env` **(input/output)** The MOSEK environment.

Description: Deletes a MOSEK environment and all the data associated with it.

Before calling this function it is a good idea to call the function `MSK_unlinkfuncfromenvstream` for each stream that has have had function linked to it.

- `MSK_echoenv`

Syntax:

```
MSKrescodee MSK_echoenv (
    MSKenv_t env,
    MSKstreamtypee whichstream,
    MSKCONST char * format,
    ...);
```

`env` **(input)** The MOSEK environment.

`whichstream` **(input)** Index of the stream.

`format` **(input)** Is a valid C format string which matches the arguments in '`...`'.

`varnumarg` **(input)** A variable argument list.

Description: Sends a message to a given environment stream.

- `MSK_echointro`

  Syntax:

  ```
  MSKrescodee MSK_echointro (
      MSKenv_t env,
      MSKintt longver);
  ```

  `env` **(input)** The MOSEK environment.

  `longver` **(input)** If non-zero, then the intro is slightly longer.

  Description: Prints an intro to message stream.

- `MSK_freedbgenv`

  Syntax:

  ```
  void MSK_freedbgenv (
      MSKenv_t env,
      MSKCONST void * buffer,
      MSKCONST char * file,
      MSKCONST unsigned line);
  ```

  `env` **(input)** The MOSEK environment.

  `buffer` **(input)** A pointer.

  `file` **(input)** File from which the function is called.

  `line` **(input)** Line in the file from which the function is called.

  Description: Frees space allocated by a MOSEK function. Must not be applied to the MOSEK environment and task.

- `MSK_freeenv`

  Syntax:

  ```
  void MSK_freeenv (
      MSKenv_t env,
      MSKCONST void * buffer);
  ```

  `env` **(input)** The MOSEK environment.

  `buffer` **(input)** A pointer.

  Description: Frees space allocated by a MOSEK function. Must not be applied to the MOSEK environment and task.

- `MSK_getbuildinfo`

Syntax:

```
MSKrescodee MSK_getbuildinfo (
    char * buildstate,
    char * builddate,
    char * buildtool);
```

`buildstate` **(output)** State of binaries, i.e. a debug, release candidate or final release.
`builddate` **(output)** Date when the binaries were build.
`buildtool` **(output)** Tool(s) used to build the binaries.

Description: Obtains build information.

- `MSK_getcodedesc`

Syntax:

```
MSKrescodee MSK_getcodedesc (
    MSKrescodee code,
    char * symname,
    char * str);
```

`code` **(input)** A valid MOSEK response code.
`symname` **(output)** Symbolic name corresponding to `code`.
`str` **(output)** Obtains a short description of a response code.

Description: Obtains a short description of the meaning of the response code given by `code`.

- `MSK_getcodedisc`

Syntax:

```
MSKrescodee MSK_getcodedisc (
    MSKrescodee code,
    char * symname,
    char * str);
```

`code` **(input)** A valid MOSEK response code.
`symname` **(output)** Symbolic name corresponding to `code`.
`str` **(output)** Obtains a short description of a response code.

Description: Obtains a short description of the meaning of the response code given by `code`.

- `MSK_getglbdllname`

Syntax:

```
MSKrescodee MSK_getglbdllname (
    MSKenv_t env,
    MSKCONST size_t sizedllname,
    char * dllname);
```

`env` **(input)** The MOSEK environment.
`sizedllname` **(input)**

`dllname` **(output)** The DLL name.

Description: Obtains the name of the global optimizer DLL.

- `MSK_getresponseclass`

  Syntax:

  ```
  MSKrescodee MSK_getresponseclass (
      MSKrescodee r,
      MSKrescodetypee * rc);
  ```

  `r` **(input)** A response code indicating the result of function call.

  `rc` **(output)** The return response class

  Description: Obtain the class of a response code.

- `MSK_getsymbcondim`

  Syntax:

  ```
  MSKrescodee MSK_getsymbcondim (
      MSKenv_t env,
      MSKintt * num,
      size_t * maxlen);
  ```

  `env` **(input)** The MOSEK environment.

  `num` **(output)** Number of symbolic constants defined by MOSEK.

  `maxlen` **(output)** Maximum length of the name of any symbolic constants.

  Description: Obtains the number of symbolic constants defined by MOSEK and the maximum length of the name of any symbolic constant.

- `MSK_getversion`

  Syntax:

  ```
  MSKrescodee MSK_getversion (
      MSKintt * major,
      MSKintt * minor,
      MSKintt * build,
      MSKintt * revision);
  ```

  `major` **(output)** Major version number.

  `minor` **(output)** Minor version number.

  `build` **(output)** Build number.

  `revision` **(output)** Revision number.

  Description: Obtains MOSEK version information.

- `MSK_initenv`

  Syntax:

  ```
  MSKrescodee MSK_initenv (MSKenv_t env)
  ```

`env` **(input)** The MOSEK environment.

Description: This function initializes the MOSEK environment. Among other things the license server will be contacted. Error messages from the license manager can be captured by linking to the environment message stream before calling this function.

- `MSK_iparvaltosymnam`

  Syntax:

  ```
  MSKrescodee MSK_iparvaltosymnam (
      MSKenv_t env,
      MSKiparame whichparam,
      MSKintt whichvalue,
      char * symbolicname);
  ```

  `env` **(input)** The MOSEK environment.

  `whichparam` **(input)** Which parameter.

  `whichvalue` **(input)** Which value.

  `symbolicname` **(output)** The symbolic name corresponding to `whichvalue`.

  Description: Obtains the symbolic name corresponding to a value that can be assigned to an integer parameter.

- `MSK_isinfinity`

  Syntax:

  ```
  MSKbooleant MSK_isinfinity (MSKrealt value)
  ```

  `value`

  Description: Return true if `value` considered infinity by MOSEK

- `MSK_linkfiletoenvstream`

  Syntax:

  ```
  MSKrescodee MSK_linkfiletoenvstream (
      MSKenv_t env,
      MSKstreamtypee whichstream,
      MSKCONST char * filename,
      MSKintt append);
  ```

  `env` **(input)** The MOSEK environment.

  `whichstream` **(input)** Index of the stream.

  `filename` **(input)** Sends all output from the stream defined by `whichstream` to the file given by `filename`.

  `append` **(input)** If this argument is non-zero, the output is appended to the file.

  Description: Directs all output from a stream to a file.

- `MSK_linkfunctoenvstream`

Syntax:

```
MSKrescodee MSK_linkfunctoenvstream (
    MSKenv_t env,
    MSKstreamtypee whichstream,
    MSKuserhandle_t handle,
    MSKstreamfunc func);
```

`env` **(input)** The MOSEK environment.

`whichstream` **(input)** Index of the stream.

`handle` **(input)** A user-defined handle which is passed to the user-defined function `func`.

`func` **(input)** All output to the stream `whichstream` is passed to `func`.

Description: Connects a user-defined function to a stream.

- `MSK_makeemptytask`

Syntax:

```
MSKrescodee MSK_makeemptytask (
    MSKenv_t env,
    MSKtask_t * task);
```

`env` **(input)** The MOSEK environment.

`task` **(output)** An optimization task.

Description: Creates a new optimization task.

- `MSK_makeenv`

Syntax:

```
MSKrescodee MSK_makeenv (
    MSKenv_t * env,
    MSKuserhandle_t usrptr,
    MSKmallocfunc usrmalloc,
    MSKfreefunc usrfree,
    MSKCONST char * dbgfile);
```

`env` **(output)** The MOSEK environment.

`usrptr` **(input)** A `pointer` to user-defined data structure. The pointer is feed into `usrmalloc` and `usrfree`.

`usrmalloc` **(input)** A user-defined `malloc` function or a `NULL` pointer.

`usrfree` **(input)** A user-defined `free` function which is used deallocate space allocated by `usrmalloc`. This function must be defined if `usrmalloc!=NULL`.

`dbgfile` **(input)** A user-defined file debug file.

Description: Creates a new MOSEK environment. Before the created environment can be used to create a task, then the environment must be initialized using the function `MSK_initenv`.

See also:

`MSK_initenv` Initialize a MOSEK environment.

MSK_putdllpath Sets the path to the DLL/shared libraries that MOSEK is loading.

MSK_putlicensedefaults Set defaults used by the license manager.

MSK_putcpudefaults Set defaults default CPU type and cache sizes.

- MSK_maketask

  Syntax:

  ```
  MSKrescodee MSK_maketask (
      MSKenv_t env,
      MSKintt maxnumcon,
      MSKintt maxnumvar,
      MSKtask_t * task);
  ```

  env **(input)** The MOSEK environment.

  maxnumcon **(input)** An optional estimate on the maximum number of constraints in the task. Can e.g be 0 if no such estimate is known.

  maxnumvar **(input)** An optional estimate on the maximum number of variables in the task. Can be 0 if no such estimate is known.

  task **(output)** An optimization task.

  Description: Creates a new task.

- MSK_putcpudefaults

  Syntax:

  ```
  MSKrescodee MSK_putcpudefaults (
      MSKenv_t env,
      int cputype,
      MSKintt sizel1,
      MSKintt sizel2);
  ```

  env **(input)** The MOSEK environment.

  cputype **(input)** The CPU ID.

  sizel1 **(input)** Size of the L1 cache.

  sizel2 **(input)** Size of the L2 cache.

  Description: Sets default CPU type and cache sizes. This function should be called before MSK_initenv.

- MSK_putdllpath

  Syntax:

  ```
  MSKrescodee MSK_putdllpath (
      MSKenv_t env,
      MSKCONST char * dllpath);
  ```

  env **(input)** The MOSEK environment.

`dllpath` **(input)** A path to where the MOSEK dynamic link/shared libraries are located. If `dllpath` is `NULL`, then MOSEK assumes that the operating system can locate the libraries.

Description: Sets the path to the DLL/shared libraries that MOSEK are loading. If needed, then it should normally be called before `MSK_initenv`.

- `MSK_putexitfunc`

  Syntax:

  ```
  MSKrescodee MSK_putexitfunc (
      MSKenv_t env,
      MSKexitfunc exitfunc,
      MSKuserhandle_t handle);
  ```

  `env` **(input)** The MOSEK environment.

  `exitfunc` **(input)** A user-defined exit function.

  `handle` **(input)** A pointer to user-defined data structure which is passed to `exitfunc` when called.

  Description: In case MOSEK has a fatal error, then an exit function is called. The exit function should terminate MOSEK. In general it is not necessary to define an exit function.

- `MSK_putkeepdlls`

  Syntax:

  ```
  MSKrescodee MSK_putkeepdlls (
      MSKenv_t env,
      MSKintt keepdlls);
  ```

  `env` **(input)** Size of the L2 cache.

  `keepdlls` **(input)** Controls whether explicitly loaded DLLs should be kept.

  Description: Controls whether explicitly loaded DLLs should be kept when they no longer are in use.

- `MSK_putlicensedefaults`

  Syntax:

  ```
  MSKrescodee MSK_putlicensedefaults (
      MSKenv_t env,
      MSKCONST char * licensefile,
      MSKCONST MSKintt * licensebuf,
      MSKintt licwait,
      MSKintt licdebug);
  ```

  `env` **(input)** The MOSEK environment.

  `licensefile` **(input)** Either `NULL` or the path to a valid MOSEK license file.

  `licensebuf` **(input)** This is the license string authorizing the use of MOSEK in the runtime version of MOSEK. Therefore, most frequently this string is a NULL pointer.

`licwait` **(input)** If this argument is non-zero, then MOSEK will wait for a license if no license is available. Moreover, `licwait-1` is the number of milliseconds to wait between each check for an available license.

`licdebug` **(input)** If this argument is non-zero, then MOSEK will print debug info regarding the license checkout.

Description: Sets default values for the license manager. This function should be called before MSK_initenv.

- `MSK_replacefileext`

  Syntax:

  ```
  void MSK_replacefileext (
      char * filename,
      MSKCONST char * newextension);
  ```

  `filename` **(input/output)** The filename.

  `newextension` **(input)** The new extension.

  Description: Replaces the file extension in a file name by a new one.

- `MSK_strdupdbgenv`

  Syntax:

  ```
  char * MSK_strdupdbgenv (
      MSKenv_t env,
      MSKCONST char * str,
      MSKCONST char * file,
      MSKCONST unsigned line);
  ```

  `env` **(input)** The MOSEK environment.

  `str` **(input)** String that should be copied.

  `file` **(input)** File from which the function is called.

  `line` **(input)** Line in the file from which the function is called.

  Description: Make a copy of a string. The string created by this procedure must be freed by MSK_freeenv.

- `MSK_strdupenv`

  Syntax:

  ```
  char * MSK_strdupenv (
      MSKenv_t env,
      MSKCONST char * str);
  ```

  `env` **(input)** The MOSEK environment.

  `str` **(input)** String that should be copied.

  Description: Make a copy of a string. The string created by this procedure must be freed by MSK_freeenv.

- `MSK_symnamtovalue`

  Syntax:

  ```
  MSKbooleant MSK_symnamtovalue (
      MSKCONST char * name,
      char * value);
  ```

  `name` **(input)** Symbolic name.

  `value` **(output)** The corresponding value.

  Description: Obtains the value corresponding to a symbolic name defined by MOSEK.

- `MSK_unlinkfuncfromenvstream`

  Syntax:

  ```
  MSKrescodee MSK_unlinkfuncfromenvstream (
      MSKenv_t env,
      MSKstreamtypee whichstream);
  ```

  `env` **(input)** The MOSEK environment.

  `whichstream` **(input)** Index of the stream.

  Description: Disconnects a user-defined function from a stream.

- `MSK_utf8towchar`

  Syntax:

  ```
  MSKrescodee MSK_utf8towchar (
      MSKCONST size_t outputlen,
      size_t * len,
      size_t * conv,
      MSKwchart * output,
      MSKCONST char * input);
  ```

  `outputlen` **(input)** The length of the output buffer.

  `len` **(output)** The length of the string contained in the output buffer.

  `conv` **(output)** Returns the number of characters from converted, i.e. `input[conv]` is the first char which was not converted. If the whole string was converted, then `input[conv]=0`.

  `output` **(output)** The input string converted to a wchar string.

  `input` **(input)** The UTF8 input string.

  Description: Converts an UTF8 string to a wchar string.

- `MSK_wchartoutf8`

  Syntax:

  ```
  MSKrescodee MSK_wchartoutf8 (
      MSKCONST size_t outputlen,
      size_t * len,
      size_t * conv,
      char * output,
      MSKCONST MSKwchart * input);
  ```

`outputlen` **(input)** The length of the output buffer.

`len` **(output)** The length of the string contained in the output buffer.

`conv` **(output)** Returns the number of characters from converted, i.e. `input[conv]` is the first char which was not converted. If the whole string was converted, then `input[conv]=0`.

`output` **(output)** The input string converted to a wchar string.

`input` **(input)** The UTF8 input string.

Description: Converts an UTF8 string to a wchar string.

15.4 Mosek Task

Description:

A Mosek Optimization task

15.4.1 Methods

- `MSK_analyzeproblem`

  Syntax:

```
MSKrescodee MSK_analyzeproblem (
    MSKtask_t task,
    MSKstreamtypee whichstream);
```

`task` **(input)** An optimization task.

`whichstream` **(input)** Index of the stream.

Description: The function analyze the data of task and writes out a report.

- `MSK_analyzesolution`

  Syntax:

  ```
  MSKrescodee MSK_analyzesolution (
      MSKtask_t task,
      MSKstreamtypee whichstream,
      MSKsoltypee whichsol);
  ```

  `task` **(input)** An optimization task.

  `whichstream` **(input)** Index of the stream.

  `whichsol` **(input)** Selects a solution.

  Description: Print information related to the quality of the solution and other solution statistics.

  By default this function prints information about the largest infeasibilites in the solution, the primal (and possibly dual) objective value and the solution status.

  Following parameters can be used to configure the printed statistics:

  - `MSK_IPAR_ANA_SOL_BASIS`. Enables or disables printing of statistics specific to the basis solution (condition number, number of basic variables etc.). Default is on.
  - `MSK_IPAR_ANA_SOL_PRINT_VIOLATED`. Enables or disables listing names of all constraints (both primal and dual) which are violated by the solution. Default is off.
  - `MSK_DPAR_ANA_SOL_INFEAS_TOL`. The tolerance defining when a constraint is considered violated. If a constraint is violated more than this, it will be listed in the summary.

  See also:

  `MSK_getpeqi` Obtains the primal equation infeasibility.

  `MSK_getdeqi` Optains the dual equation infeasibility.

  `MSK_getpbi` Obtains the primal bound infeasibility.

  `MSK_getdbi` Obtains the dual bound infeasibility.

  `MSK_getdcni` Obtains the dual cone infeasibility.

  `MSK_getpcni` Obtains the primal cone infeasibility.

  `MSK_getsolutioninf` Obtains information about a solution.

  `MSK_getsolutionstatus` Obtains information about the problem and solution statuses.

- `MSK_append`

  Syntax:

  ```
  MSKrescodee MSK_append (
      MSKtask_t task,
      MSKaccmodee accmode,
      MSKintt num);
  ```

`task` **(input)** An optimization task.

`accmode` **(input)** Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

`num` **(input)** Number of constraints or variables which should be appended.

Description: Appends a number of constraints or variables to the model. Appended constraints will be declared free and appended variables will be fixed at the level zero. Please note that MOSEK will automatically expand the problem dimension to accommodate the additional constraints and variables.

See also:

`MSK_remove` The function removes a number of constraints or variables.

- `MSK_appendcone`

Syntax:

```
MSKrescodee MSK_appendcone (
    MSKtask_t task,
    MSKconetypee conetype,
    MSKrealt conepar,
    MSKintt nummem,
    MSKCONST MSKidxt * submem);
```

`task` **(input)** An optimization task.

`conetype` **(input)** Specifies the type of the cone.

`conepar` **(input)** This argument is currently not used. Can be set to 0.0.

`nummem` **(input)** Number of member variables in the cone.

`submem` **(input)** Variable subscripts of the members in the cone.

Description: Appends a new conic constraint to the problem. Hence, add a constraint

$$\bar{x} \in \mathcal{C}$$

to the problem where $\mathcal{C}$ is a convex cone. $\bar{x}$ is a subset of the variables which will be specified by the argument `submem`.

Depending on the value of `conetype` this function appends a normal (`MSK_CT_QUAD`) or rotated quadratic cone (`MSK_CT_RQUAD`). Define

$$\bar{x} = x_{\texttt{submem}[0]}, \ldots, x_{\texttt{submem}[\texttt{nummem}-1]}$$

. Depending on the value of `conetype` this function appends one of the constraints:

– Quadratic cone (`MSK_CT_QUAD`) :

$$\bar{x}_0 \geq \sqrt{\sum_{i=1}^{i<\texttt{nummem}} \bar{x}_i^2}$$

– Rotated quadratic cone (MSK_CT_RQUAD) :

$$2\bar{x}_0\bar{x}_1 \geq \sum_{i=2}^{i<\text{nummem}} \bar{x}_i^2, \quad \bar{x}_0, \bar{x}_1 \geq 0$$

Please note that the sets of variables appearing in different conic constraints must be disjoint.

For an explained code example see Section 5.4.

- MSK_basiscond

Syntax:

```
MSKrescodee MSK_basiscond (
    MSKtask_t task,
    MSKrealt * nrmbasis,
    MSKrealt * nrminvbasis);
```

task (input) An optimization task.

nrmbasis (output) An estimate for the 1 norm of the basis.

nrminvbasis (output) An estimate for the 1 norm of the inverse of the basis.

Description: If a basic solution is available and it defines a nonsingular basis, then this function computes the 1-norm estimate of the basis matrix and an 1-norm estimate for the inverse of the basis matrix. The 1-norm estimates are computed using the method outlined in [24, pp. 388-391].

By defintion the 1-norm condition number of a matrix B is defined as

$$\kappa_1(B) := \|B\|_1 \left\|B^{-1}\right\|.$$

Moreover, the larger the condition number is the harder it is to solve linear equation systems involving B. Given estimates for $\|B\|_1$ and $\left\|B^{-1}\right\|_1$ it is also possible to estimate $\kappa_1(B)$.

- MSK_bktostr

Syntax:

```
MSKrescodee MSK_bktostr (
    MSKtask_t task,
    MSKboundkeye bk,
    char * str);
```

task (input) An optimization task.

bk (input) Bound key.

str (output) String corresponding to the bound key code bk.

Description: Obtains an identifier string corresponding to a bound key.

- MSK_callbackcodetostr

Syntax:

```
MSKrescodee MSK_callbackcodetostr (
    MSKcallbackcodee code,
    char * callbackcodestr);
```

`code` **(input)** A call-back code.

`callbackcodestr` **(output)** String corresponding to the call-back code.

Description: Obtains a the string representation of a corresponding to a call-back code.

- `MSK_callocdbgtask`

  Syntax:

```
void * MSK_callocdbgtask (
    MSKtask_t task,
    MSKCONST size_t number,
    MSKCONST size_t size,
    MSKCONST char * file,
    MSKCONST unsigned line);
```

  `task` **(input)** An optimization task.

  `number` **(input)** Number of elements.

  `size` **(input)** Size of each individual element.

  `file` **(input)** File from which the function is called.

  `line` **(input)** Line in the file from which the function is called.

  Description: Debug version of `MSK_calloctask`.

- `MSK_calloctask`

  Syntax:

```
void * MSK_calloctask (
    MSKtask_t task,
    MSKCONST size_t number,
    MSKCONST size_t size);
```

  `task` **(input)** An optimization task.

  `number` **(input)** Number of elements.

  `size` **(input)** Size of each individual element.

  Description: Equivalent to `calloc` i.e. allocate space for an array of length number where each element is of size `size`.

- `MSK_checkconvexity`

  Syntax:

```
MSKrescodee MSK_checkconvexity (MSKtask_t task)
```

  `task` **(input)** An optimization task.

  Description: This function checks if a quadratic optimization problem is convex. The amount of checking is controlled by `MSK_IPAR_CHECK_CONVEXITY`.

  The function returns an error code other than `MSK_RES_OK` if the problem is not convex.

See also:

MSK_IPAR_CHECK_CONVEXITY

- MSK_checkdata

  Syntax:

  ```
  MSKrescodee MSK_checkdata (MSKtask_t task)
  ```

  `task` **(input)** An optimization task.

  Description: Checks the data of the optimization task.

- MSK_checkmemtask

  Syntax:

  ```
  MSKrescodee MSK_checkmemtask (
      MSKtask_t task,
      MSKCONST char * file,
      MSKintt line);
  ```

  `task` **(input)** An optimization task.

  `file` **(input)** File from which the function is called.

  `line` **(input)** Line in the file from which the function is called.

  Description: Checks the memory allocated by the task.

- MSK_chgbound

  Syntax:

  ```
  MSKrescodee MSK_chgbound (
      MSKtask_t task,
      MSKaccmodee accmode,
      MSKidxt i,
      MSKintt lower,
      MSKintt finite,
      MSKrealt value);
  ```

  `task` **(input)** An optimization task.

  `accmode` **(input)** Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

  `i` **(input)** Index of the constraint or variable for which the bounds should be changed.

  `lower` **(input)** If non-zero, then the lower bound is changed, otherwise the upper bound is changed.

  `finite` **(input)** If non-zero, then `value` is assumed to be finite.

  `value` **(input)** New value for the bound.

Description: Changes a bound for one constraint or variable. If `accmode` equals MSK_ACC_CON, a constraint bound is changed, otherwise a variable bound is changed.

If `lower` is non-zero, then the lower bound is changed as follows:

$$\text{new lower bound} = \begin{cases} -\infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Otherwise if `lower` is zero, then

$$\text{new upper bound} = \begin{cases} \infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Please note that this function automatically updates the bound key for bound, in particular, if the lower and upper bounds are identical, the bound key is changed to `fixed`.

See also:

MSK_putbound Changes the bound for either one constraint or one variable.
MSK_DPAR_DATA_TOL_BOUND_INF
MSK_DPAR_DATA_TOL_BOUND_WRN

- `MSK_clonetask`

Syntax:

```
MSKrescodee MSK_clonetask (
    MSKtask_t task,
    MSKtask_t * clonedtask);
```

`task` **(input)** An optimization task.

`clonedtask` **(output)** The cloned task.

Description: Creates a clone of an existing task copying all problem data and parameter settings to a new task. Call-back functions are not copied, so a task containing nonlinear functions cannot be cloned.

- `MSK_commitchanges`

Syntax:

```
MSKrescodee MSK_commitchanges (MSKtask_t task)
```

`task` **(input)** An optimization task.

Description: Commits all cached problem changes to the task. It is usually not necessary explicitly to call this function since changes will be committed automatically when required.

- `MSK_conetypetostr`

Syntax:

```
MSKrescodee MSK_conetypetostr (
    MSKtask_t task,
    MSKconetypee conetype,
    char * str);
```

`task` **(input)** An optimization task.

`conetype` **(input)** Specifies the type of the cone.

`str` **(output)** String corresponding to the cone type code `conetype`.

Description: Obtains the cone string identifier corresponding to a cone type.

- `MSK_deletesolution`

  Syntax:

  ```
  MSKrescodee MSK_deletesolution (
      MSKtask_t task,
      MSKsoltypee whichsol);
  ```

  `task` **(input)** An optimization task.

  `whichsol` **(input)** Selects a solution.

  Description: Undefines a solution and frees the memory it uses.

- `MSK_deletetask`

  Syntax:

  ```
  MSKrescodee MSK_deletetask (MSKtask_t * task)
  ```

  `task` **(input/output)** An optimization task.

  Description: Deletes a task.

- `MSK_dualsensitivity`

  Syntax:

  ```
  MSKrescodee MSK_dualsensitivity (
      MSKtask_t task,
      MSKintt numj,
      MSKCONST MSKidxt * subj,
      MSKrealt * leftpricej,
      MSKrealt * rightpricej,
      MSKrealt * leftrangej,
      MSKrealt * rightrangej);
  ```

  `task` **(input)** An optimization task.

  `numj` **(input)** Number of coefficients to be analyzed. Length of `subj`.

  `subj` **(input)** Index of objective coefficients to analyze.

  `leftpricej` **(output)** `leftpricej`$[j]$ is the left shadow price for the coefficients with index `subj`[j].

  `rightpricej` **(output)** `rightpricej`$[j]$ is the right shadow price for the coefficients with index `subj`[j].

  `leftrangej` **(output)** `leftrangej`$[j]$ is the left range β_1 for the coefficient with index `subj`[j].

  `rightrangej` **(output)** `rightrangej`$[j]$ is the right range β_2 for the coefficient with index `subj`[j].

Description: Calculates sensitivity information for objective coefficients. The indexes of the coefficients to analyze are

$$\{\texttt{subj}[i] | i \in 0, \ldots, \texttt{numj} - 1\}$$

The results are returned so that e.g `leftprice`$[j]$ is the left shadow price of the objective coefficient with index `subj`$[j]$.

The type of sensitivity analysis to perform (basis or optimal partition) is controlled by the parameter MSK_IPAR_SENSITIVITY_TYPE.

For an example, please see Section 12.5.

See also:

MSK_primalsensitivity Perform sensitivity analysis on bounds.

MSK_sensitivityreport Creates a sensitivity report.

MSK_IPAR_SENSITIVITY_TYPE

MSK_IPAR_LOG_SENSITIVITY

MSK_IPAR_LOG_SENSITIVITY_OPT

- MSK_echotask

Syntax:

```
MSKrescodee MSK_echotask (
    MSKtask_t task,
    MSKstreamtypee whichstream,
    MSKCONST char * format,
    ...);
```

`task` **(input)** An optimization task.

whichstream **(input)** Index of the stream.

`format` **(input)**

`varnumarg` **(input)**

Description: Prints a format string to a task stream.

- MSK_exceptiontask

Syntax:

```
MSKrescodee MSK_exceptiontask (
    MSKtask_t task,
    MSKrescodee code,
    ...);
```

`task` **(input)** An optimization task.

code **(input)**

`varnumarg` **(input)**

Description: Prints the `code` to the error task stream formatted "nicely". `code` must be a valid response code listed in Appendix 17. Moreover, the corresponding response string listed in Appendix 17 is printed. It is the users responsibility to provide appropriate arguments for the response string listed in Appendix 17 too.

- `MSK_freedbgtask`

  Syntax:

  ```
  void MSK_freedbgtask (
      MSKtask_t task,
      MSKCONST void * buffer,
      MSKCONST char * file,
      MSKCONST unsigned line);
  ```

  `task` **(input)** An optimization task.

  `buffer` **(input)** A pointer.

  `file` **(input)** File from which the function is called.

  `line` **(input)** Line in the file from which the function is called.

  Description: Frees space allocated by a MOSEK function. Must not be applied to the MOSEK environment and task.

- `MSK_freetask`

  Syntax:

  ```
  void MSK_freetask (
      MSKtask_t task,
      MSKCONST void * buffer);
  ```

  `task` **(input)** An optimization task.

  `buffer` **(input)** A pointer.

  Description: Frees space allocated by a MOSEK function. Must not be applied to the MOSEK environment and task.

- `MSK_getaij`

  Syntax:

  ```
  MSKrescodee MSK_getaij (
      MSKtask_t task,
      MSKidxt i,
      MSKidxt j,
      MSKrealt * aij);
  ```

  `task` **(input)** An optimization task.

  `i` **(input)** Row index of the coefficient to be returned.

  `j` **(input)** Column index of the coefficient to be returned.

  `aij` **(output)** The required coefficient $a_{i,j}$.

  Description: Obtains a single coefficient in A.

- `MSK_getapiecenumnz`

  Syntax:

```
MSKrescodee MSK_getapiecenumnz (
    MSKtask_t task,
    MSKidxt firsti,
    MSKidxt lasti,
    MSKidxt firstj,
    MSKidxt lastj,
    MSKintt * numnz);
```

`task` **(input)** An optimization task.

`firsti` **(input)** Index of the first row in the rectangular piece.

`lasti` **(input)** Index of the last row plus one in the rectangular piece.

`firstj` **(input)** Index of the first column in the rectangular piece.

`lastj` **(input)** Index of the last column plus one in the rectangular piece.

`numnz` **(output)** Number of non-zero A elements in the rectangular piece.

Description: Obtains the number non-zeros in a rectangular piece of A, i.e. the number

$$|\{(i,j):\ a_{i,j} \neq 0,\ \texttt{firsti} \leq i \leq \texttt{lasti} - 1,\ \texttt{firstj} \leq j \leq \texttt{lastj} - 1\}|$$

where $|\mathcal{I}|$ means the number of elements in the set $\mathcal{I}$.

This function is not an efficient way to obtain the number of non-zeros in one row or column. In that case use the function MSK_getavecnumnz.

See also:

MSK_getavecnumnz Obtains the number of non-zero elements in one row or column of the linear constraint matrix

MSK_getaslicenumnz Obtains the number of non-zeros in a row or column slice of the coefficient matrix.

- MSK_getaslice

Syntax:

```
MSKrescodee MSK_getaslice (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt first,
    MSKidxt last,
    MSKintt maxnumnz,
    MSKintt * surp,
    MSKintt * ptrb,
    MSKlidxt * ptre,
    MSKidxt * sub,
    MSKrealt * val);
```

`task` **(input)** An optimization task.

`accmode` **(input)** Defines whether a column-slice or a row-slice is requested.

`first` **(input)** Index of the first row or column in the sequence.

`last` **(input)** Index of the last row or column in the sequence **plus one**.

`maxnumnz` **(input)** Denotes the length of the arrays `sub` and `val`.

`surp` **(input/output)** The required rows and columns are stored sequentially in `sub` and `val` starting from position `maxnumnz-surp[0]`. Upon return `surp` has been decremented by the total number of non-zero elements in the rows and columns obtained.

`ptrb` **(output)** `ptrb[t]` is an index pointing to the first element in the tth row or column obtained.

`ptre` **(output)** `ptre[t]` is an index pointing to the last element plus one in the tth row or column obtained.

`sub` **(output)** Contains the row or column subscripts.

`val` **(output)** Contains the coefficient values.

Description: Obtains a sequence of rows or columns from A in sparse format.

See also:

MSK_getaslicenumnz Obtains the number of non-zeros in a row or column slice of the coefficient matrix.

- `MSK_getaslice64`

Syntax:

```
MSKrescodee MSK_getaslice64 (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt first,
    MSKidxt last,
    MSKint64t maxnumnz,
    MSKint64t * surp,
    MSKint64t * ptrb,
    MSKint64t * ptre,
    MSKidxt * sub,
    MSKrealt * val);
```

`task` **(input)** An optimization task.

`accmode` **(input)** Defines whether a column slice or a row slice is requested.

`first` **(input)** Index of the first row or column in the sequence.

`last` **(input)** Index of the last row or column in the sequence **plus one**.

`maxnumnz` **(input)** Denotes the length of the arrays `sub` and `val`.

`surp` **(input/output)** The required rows and columns are stored sequentially in `sub` and `val` starting from position `maxnumnz-surp[0]`. Upon return `surp` has been decremented by the total number of non-zero elements in the rows and columns obtained.

`ptrb` **(output)** `ptrb[t]` is an index pointing to the first element in the tth row or column obtained.

`ptre` **(output)** `ptre[t]` is an index pointing to the last element plus one in the tth row or column obtained.

`sub` **(output)** Contains the row or column subscripts.

`val` **(output)** Contains the coefficient values.

Description: Obtains a sequence of rows or columns from A in sparse format.

See also:

MSK_getaslicenumnz64 Obtains the number of non-zeros in a slice of rows or columns of the coefficient matrix.

- MSK_getaslicenumnz

  Syntax:

  ```
  MSKrescodee MSK_getaslicenumnz (
      MSKtask_t task,
      MSKaccmodee accmode,
      MSKidxt first,
      MSKidxt last,
      MSKintt * numnz);
  ```

  task **(input)** An optimization task.

  accmode **(input)** Defines whether non-zeros are counted in a column-slice or a row-slice.

  first **(input)** Index of the first row or column in the sequence.

  last **(input)** Index of the last row or column **plus one** in the sequence.

  numnz **(output)** Number of non-zeros in the slice.

  Description: Obtains the number of non-zeros in a row or column slice of A.

- MSK_getaslicenumnz64

  Syntax:

  ```
  MSKrescodee MSK_getaslicenumnz64 (
      MSKtask_t task,
      MSKaccmodee accmode,
      MSKidxt first,
      MSKidxt last,
      MSKint64t * numnz);
  ```

  task **(input)** An optimization task.

  accmode **(input)** Defines whether non-zeros are counted in a column slice or a row slice.

  first **(input)** Index of the first row or column in the sequence.

  last **(input)** Index of the last row or column **plus one** in the sequence.

  numnz **(output)** Number of non-zeros in the slice.

  Description: Obtains the number of non-zeros in a slice of rows or columns of A.

- MSK_getaslicetrip

  Syntax:

  ```
  MSKrescodee MSK_getaslicetrip (
      MSKtask_t task,
      MSKaccmodee accmode,
  ```

```
    MSKidxt first,
    MSKidxt last,
    MSKintt maxnumnz,
    MSKintt * surp,
    MSKidxt * subi,
    MSKidxt * subj,
    MSKrealt * val);
```

`task` **(input)** An optimization task.

`accmode` **(input)** Defines whether a column-slice or a row-slice is requested.

`first` **(input)** Index of the first row or column in the sequence.

`last` **(input)** Index of the last row or column in the sequence **plus one**.

`maxnumnz` **(input)** Denotes the length of the arrays `subi`, `subj`, and `aval`.

`surp` **(input/output)** The required rows and columns are stored sequentially in `subi` and `val` starting from position `maxnumnz-surp[0]`. On return `surp` has been decremented by the total number of non-zero elements in the rows and columns obtained.

`subi` **(output)** Constraint subscripts.

`subj` **(output)** Variable subscripts.

`val` **(output)** Values.

Description: Obtains a sequence of rows or columns from A in a sparse triplet format.

Define p^1 as

$$p^1 = \texttt{maxnumnz} - \texttt{surp}[0]$$

when the function is called and p^2 by

$$p^2 = \texttt{maxnumnz} - \texttt{surp}[0],$$

where `surp[0]` is the value upon termination. Using this notation then

$$\texttt{val}[k] = a_{\texttt{subi[k]},\texttt{subj[k]}}, \quad k = p^1, \ldots, p^2 - 1.$$

See also:

`MSK_getaslicenumnz` Obtains the number of non-zeros in a row or column slice of the coefficient matrix.

- `MSK_getavec`

Syntax:

```
MSKrescodee MSK_getavec (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKintt * nzi,
    MSKidxt * subi,
    MSKrealt * vali);
```

`task` **(input)** An optimization task.

accmode **(input)** Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

i **(input)** Index of the row or column.

nzi **(output)** Number of non-zeros in the vector obtained.

subi **(output)** Index of the non-zeros in the vector obtained.

vali **(output)** Numerical values of the vector to be obtained.

Description: Obtains one row or column of A in a sparse format. If accmode equals MSK_ACC_CON a row is returned and hence:

$$\texttt{vali}[k] = a_{i,\texttt{subi}[k]}, \quad k = 0, \ldots, \texttt{nzi}[0] - 1$$

If accmode equals MSK_ACC_VAR a column is returned, that is:

$$\texttt{vali}[k] = a_{\texttt{subi}[k],i}, \quad k = 0, \ldots, \texttt{nzi}[0] - 1.$$

- MSK_getavecnumnz

Syntax:

```
MSKrescodee MSK_getavecnumnz (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKintt * nzj);
```

task **(input)** An optimization task.

accmode **(input)** Defines whether non-zeros are counted by columns or by rows.

i **(input)** Index of the row or column.

nzj **(output)** Number of non-zeros in the ith row or column of A.

Description: Obtains the number of non-zero elements in one row or column of A.

- MSK_getbound

Syntax:

```
MSKrescodee MSK_getbound (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKboundkeye * bk,
    MSKrealt * bl,
    MSKrealt * bu);
```

task **(input)** An optimization task.

accmode **(input)** Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

i **(input)** Index of the constraint or variable for which the bound information should be obtained.

`bk` **(output)** Bound keys.

`bl` **(output)** Values for lower bounds.

`bu` **(output)** Values for upper bounds.

Description: Obtains bound information for one constraint or variable.

- `MSK_getboundslice`

  Syntax:

```
MSKrescodee MSK_getboundslice (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt first,
    MSKidxt last,
    MSKboundkeye * bk,
    MSKrealt * bl,
    MSKrealt * bu);
```

  `task` **(input)** An optimization task.

  `accmode` **(input)** Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

  `first` **(input)** First index in the sequence.

  `last` **(input)** Last index plus 1 in the sequence.

  `bk` **(output)** Bound keys.

  `bl` **(output)** Values for lower bounds.

  `bu` **(output)** Values for upper bounds.

  Description: Obtains bounds information for a sequence of variables or constraints.

- `MSK_getc`

  Syntax:

```
MSKrescodee MSK_getc (
    MSKtask_t task,
    MSKrealt * c);
```

  `task` **(input)** An optimization task.

  `c` **(output)** Linear terms of the objective as a dense vector. The lengths is the number of variables.

  Description: Obtains all objective coefficients c.

- `MSK_getcallbackfunc`

  Syntax:

```
MSKrescodee MSK_getcallbackfunc (
    MSKtask_t task,
    MSKcallbackfunc * func,
    MSKuserhandle_t * handle);
```

`task` **(input)** An optimization task.

`func` **(output)** Get the user-defined progress call-back function `MSK_callbackfunc` associated with `task`. If `func` is identical to `NULL`, then no call-back function is associated with the `task`.

`handle` **(output)** The user-defined pointer associated with the user-defined call-back function.

Description: Obtains the current user-defined call-back function and associated `userhandle`.

- `MSK_getcfix`

  Syntax:

  ```
  MSKrescodee MSK_getcfix (
      MSKtask_t task,
      MSKrealt * cfix);
  ```

  `task` **(input)** An optimization task.

  `cfix` **(output)** Fixed term in the objective.

  Description: Obtains the fixed term in the objective.

- `MSK_getcone`

  Syntax:

  ```
  MSKrescodee MSK_getcone (
      MSKtask_t task,
      MSKidxt k,
      MSKconetypee * conetype,
      MSKrealt * conepar,
      MSKintt * nummem,
      MSKidxt * submem);
  ```

  `task` **(input)** An optimization task.

  `k` **(input)** Index of the cone constraint.

  `conetype` **(output)** Specifies the type of the cone.

  `conepar` **(output)** This argument is currently not used. Can be set to 0.0.

  `nummem` **(output)** Number of member variables in the cone.

  `submem` **(output)** Variable subscripts of the members in the cone.

  Description: Obtains a conic constraint.

- `MSK_getconeinfo`

  Syntax:

  ```
  MSKrescodee MSK_getconeinfo (
      MSKtask_t task,
      MSKidxt k,
      MSKconetypee * conetype,
      MSKrealt * conepar,
      MSKintt * nummem);
  ```

`task` **(input)** An optimization task.

`k` **(input)** Index of the conic constraint.

`conetype` **(output)** Specifies the type of the cone.

`conepar` **(output)** This argument is currently not used. Can be set to 0.0.

`nummem` **(output)** Number of member variables in the cone.

Description: Obtains information about a conic constraint.

- `MSK_getconname`

  Syntax:

  ```
  MSKrescodee MSK_getconname (
      MSKtask_t task,
      MSKidxt i,
      MSKCONST size_t maxlen,
      char * name);
  ```

  `task` **(input)** An optimization task.

  `i` **(input)** Index.

  `maxlen` **(input)** Maximum length of name that can be stored in `name`.

  `name` **(output)** Is assigned the required name.

  Description: Obtains a name of a constraint.

  See also:

  `MSK_getmaxnamelen` Obtains the maximum length (not including terminating zero character) of any objective, constraint, variable or cone name.

- `MSK_getconname64`

  Syntax:

  ```
  MSKrescodee MSK_getconname64 (
      MSKtask_t task,
      MSKidxt i,
      MSKint64t maxlen,
      char * name);
  ```

  `task` **(input)** An optimization task.

  `i` **(input)** Index.

  `maxlen` **(input)** Maximum length of name that can be stored in `name`.

  `name` **(output)** Is assigned the required name.

  Description: Obtains a name of a constraint.

  See also:

  `MSK_getmaxnamelen` Obtains the maximum length (not including terminating zero character) of any objective, constraint, variable or cone name.

- `MSK_getcslice`

Syntax:

```
MSKrescodee MSK_getcslice (
    MSKtask_t task,
    MSKidxt first,
    MSKidxt last,
    MSKrealt * c);
```

`task` **(input)** An optimization task.

`first` **(input)** First index in the sequence.

`last` **(input)** Last index plus 1 in the sequence.

`c` **(output)** Linear terms of the objective as a dense vector. The lengths is the number of variables.

Description: Obtains a sequence of elements in c.

- `MSK_getdbi`

Syntax:

```
MSKrescodee MSK_getdbi (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKaccmodee accmode,
    MSKCONST MSKidxt * sub,
    MSKintt len,
    MSKrealt * dbi);
```

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

`accmode` **(input)** If set to `MSK_ACC_CON` then `sub` contains constraint indexes, otherwise variable indexes.

`sub` **(input)** Indexes of constraints or variables.

`len` **(input)** Length of `sub`

`dbi` **(output)** Dual bound infeasibility. If `acmode` is `MSK_ACC_CON` then

$$\texttt{dbi}[i] = \max(-(s_l^c)_{\texttt{sub}[i]}, -(s_u^c)_{\texttt{sub}[i]}, 0) \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1 \tag{15.4}$$

else

$$\texttt{dbi}[i] = \max(-(s_l^x)_{\texttt{sub}[i]}, -(s_u^x)_{\texttt{sub}[i]}, 0) \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1. \tag{15.5}$$

Description: Obtains the dual bound infeasibility.

See also:

`MSK_getsolutioninf` Obtains information about a solution.

- `MSK_getdcni`

Syntax:

```
MSKrescodee MSK_getdcni (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKCONST MSKidxt * sub,
    MSKintt len,
    MSKrealt * dcni);
```

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

`sub` **(input)** Constraint indexes to calculate equation infeasibility for.

`len` **(input)** Length of `sub` and `dcni`

`dcni` **(output)** `dcni[i]` contains dual cone infeasibility for the cone with index `sub[i]`.

Description: Obtains the dual cone infeasibility.

See also:

`MSK_getsolutioninf` Obtains information about a solution.

- `MSK_getdeqi`

Syntax:

```
MSKrescodee MSK_getdeqi (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKaccmodee accmode,
    MSKCONST MSKidxt * sub,
    MSKintt len,
    MSKrealt * deqi,
    MSKintt normalize);
```

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

`accmode` **(input)** If set to `MSK_ACC_CON` the dual equation infeasibilitys corresponding to constraints are retrieved. Otherwise for a variables.

`sub` **(input)** Indexes of constraints or variables.

`len` **(input)** Length of `sub` and `deqi`.

`deqi` **(output)** Dual equation infeasibilitys corresponding to constraints or variables.

`normalize` **(input)** If non-zero, normalize with largest absolute value of the input data used to compute the individual infeasibility.

Description: Optains the dual equation infeasibility. If `acmode` is `MSK_ACC_CON` then

$$\texttt{pbi}[i] = \left|(-y + s_l^c - s_u^c)_{\texttt{sub}[i]}\right| \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1 \tag{15.6}$$

If `acmode` is `MSK_ACC_VAR` then

$$\texttt{pbi}[i] = \left|(A^T y + s_l^x - s_u^x - c)_{\texttt{sub}[i]}\right| \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1 \tag{15.7}$$

See also:

MSK_getsolutioninf Obtains information about a solution.

- MSK_getdouinf

  Syntax:

  ```
  MSKrescodee MSK_getdouinf (
      MSKtask_t task,
      MSKdinfiteme whichdinf,
      MSKrealt * dvalue);
  ```

  task **(input)** An optimization task.
  whichdinf **(input)** A double information item. See section 18.13 for the possible values.
  dvalue **(output)** The value of the required double information item.

  Description: Obtains a double information item from the task information database.

- MSK_getdouparam

  Syntax:

  ```
  MSKrescodee MSK_getdouparam (
      MSKtask_t task,
      MSKdparame param,
      MSKrealt * parvalue);
  ```

  task **(input)** An optimization task.
  param **(input)** Which parameter.
  parvalue **(output)** Parameter value.

  Description: Obtains the value of a double parameter.

- MSK_getdualobj

  Syntax:

  ```
  MSKrescodee MSK_getdualobj (
      MSKtask_t task,
      MSKsoltypee whichsol,
      MSKrealt * dualobj);
  ```

  task **(input)** An optimization task.
  whichsol **(input)** Selects a solution.
  dualobj **(output)** Objective value corresponding to the dual solution.

  Description: Obtains the current objective value of the dual problem for whichsol.

- MSK_getenv

  Syntax:

  ```
  MSKrescodee MSK_getenv (
      MSKtask_t task,
      MSKenv_t * env);
  ```

`task` **(input)** An optimization task.

`env` **(output)** The MOSEK environment.

Description: Obtains the environment used to create the task.

- `MSK_getinfeasiblesubproblem`

  Syntax:

  ```
  MSKrescodee MSK_getinfeasiblesubproblem (
      MSKtask_t task,
      MSKsoltypee whichsol,
      MSKtask_t * inftask);
  ```

  `task` **(input)** An optimization task.

  `whichsol` **(input)** Which solution to use when determining the infeasible subproblem.

  `inftask` **(output)** A new task containing the infeasible subproblem.

  Description: Obtains an infeasible subproblem. The infeasible subproblem is a problem consisting of a subset of the original constraints such that the problem is still infeasible. For more information see Section 10.2.

  See also:

  `MSK_IPAR_INFEAS_PREFER_PRIMAL`

  `MSK_relaxprimal` Creates a problem that finds the minimal change to the bounds that makes an infeasible problem feasible.

- `MSK_getinfindex`

  Syntax:

  ```
  MSKrescodee MSK_getinfindex (
      MSKtask_t task,
      MSKinftypee inftype,
      MSKCONST char * infname,
      MSKintt * infindex);
  ```

  `task` **(input)** An optimization task.

  `inftype` **(input)** Type of the information item.

  `infname` **(input)** Name of the information item.

  `infindex` **(output)** The item index.

  Description: Obtains the index of a named information item.

- `MSK_getinfmax`

  Syntax:

  ```
  MSKrescodee MSK_getinfmax (
      MSKtask_t task,
      MSKinftypee inftype,
      MSKintt * infmax);
  ```

`task` **(input)** An optimization task.

`inftype` **(input)** Type of the information item.

`infmax` **(output)**

Description: Obtains the maximum index of an information of a given type `inftype` plus 1.

- `MSK_getinfname`

Syntax:

```
MSKrescodee MSK_getinfname (
    MSKtask_t task,
    MSKinftypee inftype,
    MSKintt whichinf,
    char * infname);
```

`task` **(input)** An optimization task.

`inftype` **(input)** Type of the information item.

`whichinf` **(input)** An information item. See section 18.13 and section 18.16 for the possible values.

`infname` **(output)** Name of the information item.

Description: Obtains the name of an information item.

- `MSK_getinti`

Syntax:

```
MSKrescodee MSK_getinti (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKCONST MSKidxt * sub,
    MSKintt len,
    MSKrealt * inti);
```

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

`sub` **(input)** Variable indexes for which to calculate the integer infeasibility.

`len` **(input)** Length of `sub` and `inti`

`inti` **(output)** `inti[i]` contains integer infeasibility of variable `sub[i]`.

Description: Obtains the primal equation infeasibility.

$$\texttt{peqi}[i] = |(|\,(Ax - x^c)_{\texttt{sub}[i]}) \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1. \tag{15.8}$$

See also:

`MSK_getsolutioninf` Obtains information about a solution.

- `MSK_getintinf`

Syntax:

```
MSKrescodee MSK_getintinf (
    MSKtask_t task,
    MSKiinfiteme whichiinf,
    MSKintt * ivalue);
```

`task` **(input)** An optimization task.

`whichiinf` **(input)** Specifies an information item.

`ivalue` **(output)** The value of the required integer information item.

Description: Obtains an integer information item from the task information database.

- `MSK_getintparam`

  Syntax:

```
MSKrescodee MSK_getintparam (
    MSKtask_t task,
    MSKiparame param,
    MSKintt * parvalue);
```

  `task` **(input)** An optimization task.

  `param` **(input)** Which parameter.

  `parvalue` **(output)** Parameter value.

  Description: Obtains the value of an integer parameter.

- `MSK_getintpntnumthreads`

  Syntax:

```
MSKrescodee MSK_getintpntnumthreads (
    MSKtask_t task,
    MSKintt * numthreads);
```

  `task` **(input)** An optimization task.

  `numthreads` **(output)** The number of threads.

  Description: Obtains the number of threads used by the interior-point optimizer. If `MSK_IPAR_INTPNT_NUM_THREADS` is set to zero this function will return the number of cores on the system. Otherwise it return the value of `MSK_IPAR_INTPNT_NUM_THREADS`.

- `MSK_getlasterror`

  Syntax:

```
MSKrescodee MSK_getlasterror (
    MSKtask_t task,
    MSKrescodee * lastrescode,
    MSKCONST size_t maxlen,
    size_t * lastmsglen,
    char * lastmsg);
```

  `task` **(input)** An optimization task.

lastrescode **(output)** Returns the last error code reported in the task.

maxlen The length if the lastmsg buffer.

lastmsglen **(output)** Returns the length of the last error message reported in the task.

lastmsg **(output)** Returns the the last error message reported in the task.

Description: Obtains the last response code and corresponding message reported in MOSEK. If there is no previous error, warning or termination code for this task, lastrescode returns MSK_RES_OK and lastmsg returns an empty string, otherwise the last response code different from MSK_RES_OK and the corresponding message are returned.

- MSK_getlasterror64

Syntax:

```
MSKrescodee MSK_getlasterror64 (
    MSKtask_t task,
    MSKrescodee * lastrescode,
    MSKint64t maxlen,
    MSKint64t * lastmsglen,
    char * lastmsg);
```

task **(input)** An optimization task.

lastrescode **(output)** Returns the last error code reported in the task.

maxlen The length if the lastmsg buffer.

lastmsglen **(output)** Returns the length of the last error message reported in the task.

lastmsg **(output)** Returns the the last error message reported in the task.

Description: Obtains the last response code and corresponding message reported in MOSEK. If there is no previous error, warning or termination code for this task, lastrescode returns MSK_RES_OK and lastmsg returns an empty string, otherwise the last response code different from MSK_RES_OK and the corresponding message are returned.

- MSK_getlintinf

Syntax:

```
MSKrescodee MSK_getlintinf (
    MSKtask_t task,
    MSKliinfiteme whichliinf,
    MSKint64t * ivalue);
```

task **(input)** An optimization task.

whichliinf **(input)** Specifies an information item.

ivalue **(output)** The value of the required integer information item.

Description: Obtains an integer information item from the task information database.

- MSK_getmaxnamelen

Syntax:

```
MSKrescodee MSK_getmaxnamelen (
    MSKtask_t task,
    size_t * maxlen);
```

`task` **(input)** An optimization task.

`maxlen` **(output)** The maximum length of any name.

Description: Obtains the maximum length (not including terminating zero character) of any objective, constraint, variable or cone name.

- `MSK_getmaxnumanz`

  Syntax:

```
MSKrescodee MSK_getmaxnumanz (
    MSKtask_t task,
    MSKintt * maxnumanz);
```

  `task` **(input)** An optimization task.

  `maxnumanz` **(output)** Number of preallocated non-zero elements in A.

  Description: Obtains number of preallocated non-zeros in A. When this number of non-zeros is reached MOSEK will automatically allocate more space for A.

- `MSK_getmaxnumanz64`

  Syntax:

```
MSKrescodee MSK_getmaxnumanz64 (
    MSKtask_t task,
    MSKint64t * maxnumanz);
```

  `task` **(input)** An optimization task.

  `maxnumanz` **(output)** Number of preallocated non-zero elements in A.

  Description: Obtains number of preallocated non-zeros in A. When this number of non-zeros is reached MOSEK will automatically allocate more space for A.

- `MSK_getmaxnumcon`

  Syntax:

```
MSKrescodee MSK_getmaxnumcon (
    MSKtask_t task,
    MSKintt * maxnumcon);
```

  `task` **(input)** An optimization task.

  `maxnumcon` **(output)** Number of preallocated constraints in the optimization task.

  Description: Obtains the number of preallocated constraints in the optimization task. When this number of constraints is reached MOSEK will automatically allocate more space for constraints.

- `MSK_getmaxnumcone`

Syntax:

```
MSKrescodee MSK_getmaxnumcone (
    MSKtask_t task,
    MSKintt * maxnumcone);
```

`task` **(input)** An optimization task.

`maxnumcone` **(output)** Number of preallocated conic constraints in the optimization task.

Description: Obtains the number of preallocated cones in the optimization task. When this number of cones is reached MOSEK will automatically allocate space for more cones.

- `MSK_getmaxnumqnz`

Syntax:

```
MSKrescodee MSK_getmaxnumqnz (
    MSKtask_t task,
    MSKintt * maxnumqnz);
```

`task` **(input)** An optimization task.

`maxnumqnz` **(output)** Number of non-zero elements preallocated in quadratic coefficient matrices.

Description: Obtains the number of preallocated non-zeros for Q (both objective and constraints). When this number of non-zeros is reached MOSEK will automatically allocate more space for Q.

- `MSK_getmaxnumqnz64`

Syntax:

```
MSKrescodee MSK_getmaxnumqnz64 (
    MSKtask_t task,
    MSKint64t * maxnumqnz);
```

`task` **(input)** An optimization task.

`maxnumqnz` **(output)** Number of non-zero elements preallocated in quadratic coefficient matrices.

Description: Obtains the number of preallocated non-zeros for Q (both objective and constraints). When this number of non-zeros is reached MOSEK will automatically allocate more space for Q.

- `MSK_getmaxnumvar`

Syntax:

```
MSKrescodee MSK_getmaxnumvar (
    MSKtask_t task,
    MSKintt * maxnumvar);
```

`task` **(input)** An optimization task.

`maxnumvar` **(output)** Number of preallocated variables in the optimization task.

Description: Obtains the number of preallocated variables in the optimization task. When this number of variables is reached MOSEK will automatically allocate more space for constraints.

- `MSK_getmemusagetask`

  Syntax:

  ```
  MSKrescodee MSK_getmemusagetask (
      MSKtask_t task,
      size_t * meminuse,
      size_t * maxmemuse);
  ```

  `task` **(input)** An optimization task.

  `meminuse` **(output)** Amount of memory currently used by the `task`.

  `maxmemuse` **(output)** Maximum amount of memory used by the `task` until now.

  Description: Obtains information about the amount of memory used by a task.

- `MSK_getmemusagetask64`

  Syntax:

  ```
  MSKrescodee MSK_getmemusagetask64 (
      MSKtask_t task,
      MSKint64t * meminuse,
      MSKint64t * maxmemuse);
  ```

  `task` **(input)** An optimization task.

  `meminuse` **(output)** Amount of memory currently used by the `task`.

  `maxmemuse` **(output)** Maximum amount of memory used by the `task` until now.

  Description: Obtains information about the amount of memory used by a task.

- `MSK_getnadouinf`

  Syntax:

  ```
  MSKrescodee MSK_getnadouinf (
      MSKtask_t task,
      MSKCONST char * whichdinf,
      MSKrealt * dvalue);
  ```

  `task` **(input)** An optimization task.

  `whichdinf` **(input)** A `double` information item. See section 18.13 for the possible values.

  `dvalue` **(output)** The value of the required double information item.

  Description: Obtains a double information item from task information database.

- `MSK_getnadouparam`

  Syntax:

```
MSKrescodee MSK_getnadouparam (
    MSKtask_t task,
    MSKCONST char * paramname,
    MSKrealt * parvalue);
```

`task` **(input)** An optimization task.

`paramname` **(input)** Name of a parameter.

`parvalue` **(output)** Parameter value.

Description: Obtains the value of a named double parameter.

- `MSK_getnaintinf`

  Syntax:

```
MSKrescodee MSK_getnaintinf (
    MSKtask_t task,
    MSKCONST char * infitemname,
    MSKintt * ivalue);
```

  `task` **(input)** An optimization task.

  `infitemname` **(input)**

  `ivalue` **(output)** The value of the required integer information item.

  Description: Obtains an integer information item from the task information database.

- `MSK_getnaintparam`

  Syntax:

```
MSKrescodee MSK_getnaintparam (
    MSKtask_t task,
    MSKCONST char * paramname,
    MSKintt * parvalue);
```

  `task` **(input)** An optimization task.

  `paramname` **(input)** Name of a parameter.

  `parvalue` **(output)** Parameter value.

  Description: Obtains the value of a named integer parameter.

- `MSK_getname`

  Syntax:

```
MSKrescodee MSK_getname (
    MSKtask_t task,
    MSKproblemiteme whichitem,
    MSKidxt i,
    MSKCONST size_t maxlen,
    size_t * len,
    char * name);
```

`task` **(input)** An optimization task.

`whichitem` **(input)** Problem item, i.e. a cone, a variable or a constraint name..

`i` **(input)** Index.

`maxlen` **(input)** Maximum length of name that can be stored in `name`. The buffer `name` must be be able to hold the name including a terminating zero character.

`len` **(output)** Is assigned the length of the required name.

`name` **(output)** Is assigned the required name.

Description: Obtains a name of a problem item, i.e. a cone, a variable or a constraint.

See also:

`MSK_getmaxnamelen` Obtains the maximum length (not including terminating zero character) of any objective, constraint, variable or cone name.

- `MSK_getname64`

Syntax:

```
MSKrescodee MSK_getname64 (
    MSKtask_t task,
    MSKproblemiteme whichitem,
    MSKidxt i,
    MSKint64t maxlen,
    MSKint64t * len,
    char * name);
```

`task` **(input)** An optimization task.

`whichitem` **(input)** Problem item, i.e. a cone, a variable or a constraint name..

`i` **(input)** Index.

`maxlen` **(input)** Maximum length of name that can be stored in `name`. The buffer `name` must be be able to hold the name including a terminating zero character.

`len` **(output)** Is assigned the length of the required name.

`name` **(output)** Is assigned the required name.

Description: Obtains a name of a problem item, i.e. a cone, a variable or a constraint.

See also:

`MSK_getmaxnamelen` Obtains the maximum length (not including terminating zero character) of any objective, constraint, variable or cone name.

- `MSK_getnameapi64`

Syntax:

```
MSKrescodee MSK_getnameapi64 (
    MSKtask_t task,
    MSKproblemiteme whichitem,
    MSKidxt i,
    MSKint64t maxlen,
    char * name);
```

`task` **(input)** An optimization task.

`whichitem` **(input)** Problem item, i.e. a cone, a variable or a constraint name..

`i` **(input)** Index.

`maxlen` **(input)** Maximum length of name that can be stored in `name`. The buffer `name` must be be able to hold the name including a terminating zero character.

`name` **(output)** Is assigned the required name.

Description: Obtains a name of a problem item, i.e. a cone, a variable or a constraint.

See also:

`MSK_getmaxnamelen` Obtains the maximum length (not including terminating zero character) of any objective, constraint, variable or cone name.

- `MSK_getnameindex`

Syntax:

```
MSKrescodee MSK_getnameindex (
    MSKtask_t task,
    MSKproblemiteme whichitem,
    MSKCONST char * name,
    MSKintt * asgn,
    MSKidxt * index);
```

`task` **(input)** An optimization task.

`whichitem` **(input)** Problem item, i.e. a cone, a variable or a constraint name..

`name` **(input)** The name which should be checked.

`asgn` **(output)** Is non-zero if `name` is assigned.

`index` **(output)** If the `name` identifies an item in the task, then `index` is assigned the index of that item.

Description: Checks if a given name identifies a cone, a constraint or a variable in the `task`. If it does, the index of that item is assigned to `index`, and a non-zero value is assigned to `asgn`. If the name does not identify a problem item, `asgn` is assigned a zero.

- `MSK_getnamelen64`

Syntax:

```
MSKrescodee MSK_getnamelen64 (
    MSKtask_t task,
    MSKproblemiteme whichitem,
    MSKidxt i,
    MSKint64t * len);
```

`task` **(input)** An optimization task.

`whichitem` **(input)** Problem item, i.e. a cone, a variable or a constraint name..

`i` **(input)** Index.

`len` **(output)** Is assigned the length of the required name.

Description: Obtains the length of a problem item name.

See also:

MSK_getmaxnamelen Obtains the maximum length (not including terminating zero character) of any objective, constraint, variable or cone name.

MSK_getname Obtains the name of a cone, a variable or a constraint.

MSK_getname64 Obtains the name of a cone, a variable or a constraint.

- MSK_getnastrparam

Syntax:

```
MSKrescodee MSK_getnastrparam (
    MSKtask_t task,
    MSKCONST char * paramname,
    MSKCONST size_t maxlen,
    size_t * len,
    char * parvalue);
```

task **(input)** An optimization task.

paramname **(input)** Name of a parameter.

maxlen **(input)** Length of parvalue.

len **(output)** Identical to length of string hold by parvalue.

parvalue **(output)** Parameter value.

Description: Obtains the value of a named string parameter.

- MSK_getnastrparamal

Syntax:

```
MSKrescodee MSK_getnastrparamal (
    MSKtask_t task,
    MSKCONST char * paramname,
    MSKCONST size_t numaddchr,
    MSKstring_t * value);
```

task **(input)** An optimization task.

paramname **(input)** Name of a parameter.

numaddchr **(input)** Number of additional characters that is made room for in value[0].

value **(input/output)** Is the value corresponding to string parameter param. value[0] is char buffer allocated MOSEK and it must be freed by MSK_freetask.

Description: Obtains the value of a string parameter.

- MSK_getnlfunc

Syntax:

```
MSKrescodee MSK_getnlfunc (
    MSKtask_t task,
    MSKuserhandle_t * nlhandle,
    MSKnlgetspfunc * nlgetsp,
    MSKnlgetvafunc * nlgetva);
```

`task` **(input)** An optimization task.

`nlhandle` **(input/output)** Retrieve the pointer to the user-defined data structure. This structure is passed to the functions `nlgetsp` and `nlgetva` whenever those two functions called.

`nlgetsp` **(output)** Retrieve the function which provide information about the structure of the nonlinear functions in the optimization problem.

`nlgetva` **(output)** Retrieve the function which is used to evaluate the nonlinear function in the optimization problem at a given point.

Description: This function is used to retrieve the nonlinear call-back functions. If NULL no nonlinear call-back function exists.

- `MSK_getnumanz`

  Syntax:

  ```
  MSKrescodee MSK_getnumanz (
      MSKtask_t task,
      MSKintt * numanz);
  ```

  `task` **(input)** An optimization task.

  `numanz` **(output)** Number of non-zero elements in A.

  Description: Obtains the number of non-zeros in A.

- `MSK_getnumanz64`

  Syntax:

  ```
  MSKrescodee MSK_getnumanz64 (
      MSKtask_t task,
      MSKint64t * numanz);
  ```

  `task` **(input)** An optimization task.

  `numanz` **(output)** Number of non-zero elements in A.

  Description: Obtains the number of non-zeros in A.

- `MSK_getnumcon`

  Syntax:

  ```
  MSKrescodee MSK_getnumcon (
      MSKtask_t task,
      MSKintt * numcon);
  ```

  `task` **(input)** An optimization task.

`numcon` **(output)** Number of constraints.

Description: Obtains the number of constraints.

- `MSK_getnumcone`

  Syntax:

  ```
  MSKrescodee MSK_getnumcone (
      MSKtask_t task,
      MSKintt * numcone);
  ```

  `task` **(input)** An optimization task.
  `numcone` **(output)** Number conic constraints.

  Description: Obtains the number of cones.

- `MSK_getnumconemem`

  Syntax:

  ```
  MSKrescodee MSK_getnumconemem (
      MSKtask_t task,
      MSKidxt k,
      MSKintt * nummem);
  ```

  `task` **(input)** An optimization task.
  `k` **(input)** Index of the cone.
  `nummem` **(output)** Number of member variables in the cone.

  Description: Obtains the number of members in a cone.

- `MSK_getnumintvar`

  Syntax:

  ```
  MSKrescodee MSK_getnumintvar (
      MSKtask_t task,
      MSKintt * numintvar);
  ```

  `task` **(input)** An optimization task.
  `numintvar` **(output)** Number of integer variables.

  Description: Obtains the number of integer-constrained variables.

- `MSK_getnumparam`

  Syntax:

  ```
  MSKrescodee MSK_getnumparam (
      MSKtask_t task,
      MSKparametertypee partype,
      MSKintt * numparam);
  ```

  `task` **(input)** An optimization task.

`partype` **(input)** Parameter type.

`numparam` **(output)** Identical to the number of parameters of the type `partype`.

Description: Obtains the number of parameters of a given type.

- `MSK_getnumqconknz`

  Syntax:

  ```
  MSKrescodee MSK_getnumqconknz (
      MSKtask_t task,
      MSKidxt k,
      MSKintt * numqcnz);
  ```

  `task` **(input)** An optimization task.

  `k` **(input)** Index of the constraint for which the number of non-zero quadratic terms should be obtained.

  `numqcnz` **(output)** Number of quadratic terms. See (5.32).

  Description: Obtains the number of non-zero quadratic terms in a constraint.

- `MSK_getnumqconknz64`

  Syntax:

  ```
  MSKrescodee MSK_getnumqconknz64 (
      MSKtask_t task,
      MSKidxt k,
      MSKint64t * numqcnz);
  ```

  `task` **(input)** An optimization task.

  `k` **(input)** Index of the constraint for which the number quadratic terms should be obtained.

  `numqcnz` **(output)** Number of quadratic terms. See (5.32).

  Description: Obtains the number of non-zero quadratic terms in a constraint.

- `MSK_getnumqobjnz`

  Syntax:

  ```
  MSKrescodee MSK_getnumqobjnz (
      MSKtask_t task,
      MSKintt * numqonz);
  ```

  `task` **(input)** An optimization task.

  `numqonz` **(output)** Number of non-zero elements in Q^o.

  Description: Obtains the number of non-zero quadratic terms in the objective.

- `MSK_getnumqobjnz64`

  Syntax:

```
MSKrescodee MSK_getnumqobjnz64 (
    MSKtask_t task,
    MSKint64t * numqonz);
```

`task` **(input)** An optimization task.

`numqonz` **(output)** Number of non-zero elements in Q^o.

Description: Obtains the number of non-zero quadratic terms in the objective.

- `MSK_getnumvar`

  Syntax:

```
MSKrescodee MSK_getnumvar (
    MSKtask_t task,
    MSKintt * numvar);
```

  `task` **(input)** An optimization task.

  `numvar` **(output)** Number of variables.

  Description: Obtains the number of variables.

- `MSK_getobjname`

  Syntax:

```
MSKrescodee MSK_getobjname (
    MSKtask_t task,
    MSKCONST size_t maxlen,
    size_t * len,
    char * objname);
```

  `task` **(input)** An optimization task.

  `maxlen` **(input)** Length of `objname`.

  `len` **(output)** Assigned the length of the objective name.

  `objname` **(output)** Assigned the objective name.

  Description: Obtains the name assigned to the objective function.

- `MSK_getobjname64`

  Syntax:

```
MSKrescodee MSK_getobjname64 (
    MSKtask_t task,
    MSKint64t maxlen,
    MSKint64t * len,
    char * objname);
```

  `task` **(input)** An optimization task.

  `maxlen` **(input)** Length of `objname`.

  `len` **(output)** Assigned the length of the objective name.

  `objname` **(output)** Assigned the objective name.

Description: Obtains the name assigned to the objective function.

- MSK_getobjsense

  Syntax:

  ```
  MSKrescodee MSK_getobjsense (
      MSKtask_t task,
      MSKobjsensee * sense);
  ```

  task **(input)** An optimization task.
  sense **(output)** The returned objective sense.

  Description: Gets the objective sense of the task.

  See also:

  MSK_putobjsense Sets the objective sense.

- MSK_getparammax

  Syntax:

  ```
  MSKrescodee MSK_getparammax (
      MSKtask_t task,
      MSKparametertypee partype,
      MSKCONST MSKintt * parammax);
  ```

  task **(input)** An optimization task.
  partype **(input)** Parameter type.
  parammax **(input)**

  Description: Obtains the maximum index of a parameter of a given type plus 1.

- MSK_getparamname

  Syntax:

  ```
  MSKrescodee MSK_getparamname (
      MSKtask_t task,
      MSKparametertypee partype,
      MSKintt param,
      char * parname);
  ```

  task **(input)** An optimization task.
  partype **(input)** Parameter type.
  param **(input)** Which parameter.
  parname **(output)** Parameter name.

  Description: Obtains the name for a parameter param of type partype.

- MSK_getpbi

  Syntax:

```
MSKrescodee MSK_getpbi (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKaccmodee accmode,
    MSKCONST MSKidxt * sub,
    MSKintt len,
    MSKrealt * pbi,
    MSKintt normalize);
```

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

`accmode` **(input)** If set to `MSK_ACC_VAR` return bound infeasibility for x otherwise for x^c.

`sub` **(input)** An array of constraint or variable indexes.

`len` **(input)** Length of `sub` and `pbi`

`pbi` **(output)** Bound infeasibility for x or x^c.

`normalize` **(input)** If non-zero, normalize with largest absolute value of the input data used to compute the individual infeasibility.

Description: Obtains the primal bound infeasibility. If `acmode` is `MSK_ACC_CON` then

$$\texttt{pbi}[i] = \max(x^c_{\texttt{sub[i]}} - u^c_{\texttt{sub[i]}}, l^c_{\texttt{sub[i]}} - x^c_{\texttt{sub[i]}}, 0) \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1 \tag{15.9}$$

If `acmode` is `MSK_ACC_VAR` then

$$\texttt{pbi}[i] = \max(x_{\texttt{sub[i]}} - u^x_{\texttt{sub[i]}}, l^x_{\texttt{sub[i]}} - x_{\texttt{sub[i]}}, 0) \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1 \tag{15.10}$$

See also:

`MSK_getsolutioninf` Obtains information about a solution.

- `MSK_getpcni`

Syntax:

```
MSKrescodee MSK_getpcni (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKCONST MSKidxt * sub,
    MSKintt len,
    MSKrealt * pcni);
```

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

`sub` **(input)** Constraint indexes for which to calculate the equation infeasibility.

`len` **(input)** Length of `sub` and `pcni`

`pcni` **(output)** `pcni[i]` contains primal cone infeasibility for the cone with index `sub[i]`.

Description: Obtains the primal cone infeasibility.

See also:

MSK_getsolutioninf Obtains information about a solution.

- MSK_getpeqi

  Syntax:

  ```
  MSKrescodee MSK_getpeqi (
      MSKtask_t task,
      MSKsoltypee whichsol,
      MSKCONST MSKidxt * sub,
      MSKintt len,
      MSKrealt * peqi,
      MSKintt normalize);
  ```

  task **(input)** An optimization task.

  whichsol **(input)** Selects a solution.

  sub **(input)** Constraint indexes for which to calculate the equation infeasibility.

  len **(input)** Length of sub and peqi

  peqi **(output)** peqi[i] contains equation infeasibility of constraint sub[i].

  normalize **(input)** If non-zero, normalize with largest absolute value of the input data used to compute the individual infeasibility.

  Description: Obtains the primal equation infeasibility.

  $$\texttt{peqi}[i] = |(|\,(Ax - x^c)_{\texttt{sub}[i]}) \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1. \tag{15.11}$$

  See also:

  MSK_getsolutioninf Obtains information about a solution.

- MSK_getprimalobj

  Syntax:

  ```
  MSKrescodee MSK_getprimalobj (
      MSKtask_t task,
      MSKsoltypee whichsol,
      MSKrealt * primalobj);
  ```

  task **(input)** An optimization task.

  whichsol **(input)** Selects a solution.

  primalobj **(output)** Objective value corresponding to the primal solution.

  Description: Obtains the primal objective value for a solution.

- MSK_getprobtype

  Syntax:

  ```
  MSKrescodee MSK_getprobtype (
      MSKtask_t task,
      MSKproblemtypee * probtype);
  ```

`task` **(input)** An optimization task.

`probtype` **(output)** The problem type.

Description: Obtains the problem type.

- `MSK_getqconk`

Syntax:

```
MSKrescodee MSK_getqconk (
    MSKtask_t task,
    MSKidxt k,
    MSKintt maxnumqcnz,
    MSKintt * qcsurp,
    MSKintt * numqcnz,
    MSKidxt * qcsubi,
    MSKidxt * qcsubj,
    MSKrealt * qcval);
```

`task` **(input)** An optimization task.

`k` **(input)** Which constraint.

`maxnumqcnz` **(input)** Length of the arrays `qcsubi`, `qcsubj`, and `qcval`.

`qcsurp` **(input/output)** When entering the function it is assumed that the last `qcsurp[0]` positions in `qcsubi`, `qcsubj`, and `qcval` are free. Hence, the quadratic terms are stored in this area, and upon return `qcsurp` is number of free positions left in `qcsubi`, `qcsubj`, and `qcval`.

`numqcnz` **(output)** Number of quadratic terms. See (5.32).

`qcsubi` **(output)** i subscripts for q_{ij}^k. See (5.32).

`qcsubj` **(output)** j subscripts for q_{ij}^k. See (5.32).

`qcval` **(output)** Numerical value for q_{ij}^k.

Description: Obtains all the quadratic terms in a constraint. The quadratic terms are stored sequentially `qcsubi`, `qcsubj`, and `qcval`.

- `MSK_getqconk64`

Syntax:

```
MSKrescodee MSK_getqconk64 (
    MSKtask_t task,
    MSKidxt k,
    MSKint64t maxnumqcnz,
    MSKint64t * qcsurp,
    MSKint64t * numqcnz,
    MSKidxt * qcsubi,
    MSKidxt * qcsubj,
    MSKrealt * qcval);
```

`task` **(input)** An optimization task.

`k` **(input)** Which constraint.

`maxnumqcnz` **(input)** Length of the arrays `qcsubi`, `qcsubj`, and `qcval`.

`qcsurp` **(input/output)** When entering the function it is assumed that the last `qcsurp[0]` positions in `qcsubi`, `qcsubj`, and `qcval` are free. Hence, the quadratic terms are stored in this area, and upon return `qcsurp` is number of free positions left in `qcsubi`, `qcsubj`, and `qcval`.

`numqcnz` **(output)** Number of quadratic terms. See (5.32).

`qcsubi` **(output)** i subscripts for q_{ij}^k. See (5.32).

`qcsubj` **(output)** j subscripts for q_{ij}^k. See (5.32).

`qcval` **(output)** Numerical value for q_{ij}^k.

Description: Obtains all the quadratic terms in a constraint. The quadratic terms are stored sequentially `qcsubi`, `qcsubj`, and `qcval`.

- `MSK_getqobj`

Syntax:

```
MSKrescodee MSK_getqobj (
    MSKtask_t task,
    MSKintt maxnumqonz,
    MSKintt * qosurp,
    MSKintt * numqonz,
    MSKidxt * qosubi,
    MSKidxt * qosubj,
    MSKrealt * qoval);
```

`task` **(input)** An optimization task.

`maxnumqonz` **(input)** The length of the arrays `qosubi`, `qosubj`, and `qoval`.

`qosurp` **(input/output)** When entering the function `qosurp[0]` is the number of free positions at the end of the arrays `qosubi`, `qosubj`, and `qoval`, and upon return `qosurp` is the updated number of free positions left in those arrays.

`numqonz` **(output)** Number of non-zero elements in Q^o.

`qosubi` **(output)** i subscript for q_{ij}^o.

`qosubj` **(output)** j subscript for q_{ij}^o.

`qoval` **(output)** Numerical value for q_{ij}^o.

Description: Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in `qosubi`, `qosubj`, and `qoval`.

- `MSK_getqobj64`

Syntax:

```
MSKrescodee MSK_getqobj64 (
    MSKtask_t task,
    MSKint64t maxnumqonz,
    MSKint64t * qosurp,
    MSKint64t * numqonz,
    MSKidxt * qosubi,
```

```
    MSKidxt * qosubj,
    MSKrealt * qoval);
```

`task` **(input)** An optimization task.

`maxnumqonz` **(input)** The length of the arrays `qosubi`, `qosubj`, and `qoval`.

`qosurp` **(input/output)** When entering the function `qosurp[0]` is the number of free positions at the end of the arrays `qosubi`, `qosubj`, and `qoval`, and upon return `qosurp` is the updated number of free positions left in those arrays.

`numqonz` **(output)** Number of non-zero elements in Q^o.

`qosubi` **(output)** i subscript for q^o_{ij}.

`qosubj` **(output)** j subscript for q^o_{ij}.

`qoval` **(output)** Numerical value for q^o_{ij}.

Description: Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in `qosubi`, `qosubj`, and `qoval`.

- `MSK_getqobjij`

  Syntax:

```
MSKrescodee MSK_getqobjij (
    MSKtask_t task,
    MSKidxt i,
    MSKidxt j,
    MSKrealt * qoij);
```

  `task` **(input)** An optimization task.

  `i` **(input)** Row index of the coefficient.

  `j` **(input)** Column index of coefficient.

  `qoij` **(output)** The required coefficient.

  Description: Obtains one coefficient q^o_{ij} in the quadratic term of the objective.

- `MSK_getreducedcosts`

  Syntax:

```
MSKrescodee MSK_getreducedcosts (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKidxt first,
    MSKidxt last,
    MSKrealt * redcosts);
```

  `task` **(input)** An optimization task.

  `whichsol` **(input)** Selects a solution.

  `first` **(input)** See formula (15.12) for the definition.

  `last` **(input)** See formula (15.12) for the definition.

  `redcosts` **(output)** The reduced costs in the required sequence of variables are stored sequentially in `redcosts` starting at `redcosts[0]`.

Description: Computes the reduced costs for a sequence of variables and return them in the variable `redcosts` i.e.

$$\texttt{redcosts}[j-\texttt{first}] = (s_l^x)_j - (s_u^x)_j,\ j = \texttt{first}, \ldots, \texttt{last}-1. \tag{15.12}$$

- `MSK_getsolution`

Syntax:

```
MSKrescodee MSK_getsolution (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKprostae * prosta,
    MSKsolstae * solsta,
    MSKstakeye * skc,
    MSKstakeye * skx,
    MSKstakeye * skn,
    MSKrealt * xc,
    MSKrealt * xx,
    MSKrealt * y,
    MSKrealt * slc,
    MSKrealt * suc,
    MSKrealt * slx,
    MSKrealt * sux,
    MSKrealt * snx);
```

`task` **(input)** An optimization task.
`whichsol` **(input)** Selects a solution.
`prosta` **(output)** Problem status.
`solsta` **(output)** Solution status.
`skc` **(output)** Status keys for the constraints.
`skx` **(output)** Status keys for the variables.
`skn` **(output)** Status keys for the conic constraints.
`xc` **(output)** Primal constraint solution.
`xx` **(output)** Primal variable solution (x).
`y` **(output)** Vector of dual variables corresponding to the constraints.
`slc` **(output)** Dual variables corresponding to the lower bounds on the constraints (s_l^c).
`suc` **(output)** Dual variables corresponding to the upper bounds on the constraints (s_u^c).
`slx` **(output)** Dual variables corresponding to the lower bounds on the variables (s_l^x).
`sux` **(output)** Dual variables corresponding to the upper bounds on the variables (appears as s_u^x).
`snx` **(output)** Dual variables corresponding to the conic constraints on the variables (s_n^x).

Description: Obtains the complete solution.

Consider the case of linear programming. The primal problem is given by

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x. \end{array} \tag{15.13}$$

and the corresponding dual problem is

$$\begin{array}{lccl} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & \end{array} \tag{15.14}$$

In this case the mapping between variables and arguments to the function is as follows:

`xx`: Corresponds to variable x.

`y`: Corresponds to variable y.

`slc`: Corresponds to variable s_l^c.

`suc`: Corresponds to variable s_u^c.

`slx`: Corresponds to variable s_l^x.

`sux`: Corresponds to variable s_u^x.

`xc`: Corresponds to Ax.

The meaning of the values returned by this function depend on the *solution status* returned in the argument `solsta`. The most important possible values of `solsta` are:

`MSK_SOL_STA_OPTIMAL` An optimal solution satisfying the optimality criteria for continuous problems is returned.

`MSK_SOL_STA_INTEGER_OPTIMAL` An optimal solution satisfying the optimality criteria for integer problems is returned.

`MSK_SOL_STA_PRIM_FEAS` A solution satisfying the feasibility criteria.

`MSK_SOL_STA_PRIM_INFEAS_CER` A primal certificate of infeasibility is returned.

`MSK_SOL_STA_DUAL_INFEAS_CER` A dual certificate of infeasibility is returned.

See also:

`MSK_getsolutioni` Obtains the solution for a single constraint or variable.

`MSK_getsolutionslice` Obtains a slice of the solution.

- `MSK_getsolutioni`

Syntax:

```
MSKrescodee MSK_getsolutioni (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKsoltypee whichsol,
    MSKstakeye * sk,
    MSKrealt * x,
    MSKrealt * sl,
    MSKrealt * su,
    MSKrealt * sn);
```

`task` **(input)** An optimization task.

accmode **(input)** If set to MSK_ACC_CON the solution information for a constraint is retrieved. Otherwise for a variable.

i **(input)** Index of the constraint or variable.

whichsol **(input)** Selects a solution.

sk **(output)** Status key of the constraint of variable.

x **(output)** Solution value of the primal variable.

sl **(output)** Solution value of the dual variable associated with the lower bound.

su **(output)** Solution value of the dual variable associated with the upper bound.

sn **(output)** Solution value of the dual variable associated with the cone constraint.

Description: Obtains the primal and dual solution information for a single constraint or variable.

See also:

MSK_getsolution Obtains the complete solution.

MSK_getsolutionslice Obtains a slice of the solution.

- MSK_getsolutionincallback

Syntax:

```
MSKrescodee MSK_getsolutionincallback (
    MSKtask_t task,
    MSKcallbackcodee where,
    MSKsoltypee whichsol,
    MSKprostae * prosta,
    MSKsolstae * solsta,
    MSKstakeye * skc,
    MSKstakeye * skx,
    MSKstakeye * skn,
    MSKrealt * xc,
    MSKrealt * xx,
    MSKrealt * y,
    MSKrealt * slc,
    MSKrealt * suc,
    MSKrealt * slx,
    MSKrealt * sux,
    MSKrealt * snx);
```

task **(input)** An optimization task.

where **(input)** The call-back-key from the current call-back

whichsol **(input)** Selects a solution.

prosta **(output)** Problem status.

solsta **(output)** Solution status.

skc **(output)** Status keys for the constraints.

skx **(output)** Status keys for the variables.

skn **(output)** Status keys for the conic constraints.

`xc` **(output)** Primal constraint solution.

`xx` **(output)** Primal variable solution (x).

`y` **(output)** Vector of dual variables corresponding to the constraints.

`slc` **(output)** Dual variables corresponding to the lower bounds on the constraints (s_l^c).

`suc` **(output)** Dual variables corresponding to the upper bounds on the constraints (s_u^c).

`slx` **(output)** Dual variables corresponding to the lower bounds on the variables (s_l^x).

`sux` **(output)** Dual variables corresponding to the upper bounds on the variables (appears as s_u^x).

`snx` **(output)** Dual variables corresponding to the conic constraints on the variables (s_n^x).

Description: Obtains the whole or a part of the solution from within a progress call-back. This function must only be called from a progress call-back function.

This is an experimental feature. Please contact MOSEK support before using this function.

- `MSK_getsolutioninf`

  Syntax:

```
MSKrescodee MSK_getsolutioninf (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKprostae * prosta,
    MSKsolstae * solsta,
    MSKrealt * primalobj,
    MSKrealt * maxpbi,
    MSKrealt * maxpcni,
    MSKrealt * maxpeqi,
    MSKrealt * maxinti,
    MSKrealt * dualobj,
    MSKrealt * maxdbi,
    MSKrealt * maxdcni,
    MSKrealt * maxdeqi);
```

  `task` **(input)** An optimization task.

  `whichsol` **(input)** Selects a solution.

  `prosta` **(output)** Problem status.

  `solsta` **(output)** Solution status.

  `primalobj` **(output)** Value of the primal objective.

$$c^T x + c^f \tag{15.15}$$

  `maxpbi` **(output)** Maximum infeasibility in primal bounds on variables.

$$\max\left\{0, \max_{i\in 1,\ldots,n-1}(x_i - u_i^x), \max_{i\in 1,\ldots,n-1}(l_i^x - x_i), \max_{i\in 1,\ldots,n-1}(x_i^c - u_i^c), \max_{i\in 1,\ldots,n-1}(l_i^c - x_i^c)\right\} \tag{15.16}$$

  `maxpcni` **(output)** Maximum infeasibility in the primal conic constraints.

`maxpeqi` **(output)** Maximum infeasibility in primal equality constraints.

$$\|Ax - x^c\|_\infty \tag{15.17}$$

`maxinti` **(output)** Maximum infeasibility in integer constraints.

$$\max_{i\in\{0,\ldots,n-1\}} (\min(x_i - \lfloor x_i \rfloor, \lceil x_i \rceil - x_i)). \tag{15.18}$$

`dualobj` **(output)** Value of the dual objective.

$$(l^c)^T s_l^c - (u^c)^T s_u^c + c^f \tag{15.19}$$

`maxdbi` **(output)** Maximum infeasibility in bounds on dual variables.

$$\max\{0, \max_{i\in\{0,\ldots,n-1\}} -(s_l^x)_i, \max_{i\in\{0,\ldots,n-1\}} -(s_u^x)_i, \max_{i\in\{0,\ldots,m-1\}} -(s_l^c)_i, \max_{i\in\{0,\ldots,m-1\}} -(s_u^c)_i\} \tag{15.20}$$

`maxdcni` **(output)** Maximum infeasibility in the dual conic constraints.

`maxdeqi` **(output)** Maximum infeasibility in the dual equality constraints.

$$\max\left\{\left\|A^T y + s_l^x - s_u^x - c\right\|_\infty, \|-y + s_l^c - s_u^c\|_\infty\right\} \tag{15.21}$$

Description: Obtains information about the quality of a solution. Part of the following documentation is restricted to the linear case, it should be clear how to extend to other problem classes.

When optimizing MOSEK solves a reformulated problem with only equality constraints.

$$\begin{array}{lrcccl} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & & & Ax - x^c & = & 0 \\ & l^x & \leq & x & \leq & u^x, \\ & l^c & \leq & x^c & \leq & u^c. \end{array} \tag{15.22}$$

where

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

and the corresponding dual problem is

$$\begin{array}{lcccl} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + c^f & & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & c, & \\ & -y + s_l^c - s_u^c & = & 0, & \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & & \end{array} \tag{15.23}$$

The values returned by this function refers to these problems.

Please note that this function computes the objective value and other values every time it is called. Hence, for efficiency reasons this function should not be used too frequently.

If only the problem status or the solution status is required then use `MSK_getsolutionstatus` instead.

See also:

MSK_getpeqi Obtains the primal equation infeasibility.

MSK_getdeqi Optains the dual equation infeasibility.

MSK_getpbi Obtains the primal bound infeasibility.

MSK_getdbi Obtains the dual bound infeasibility.

MSK_getdcni Obtains the dual cone infeasibility.

MSK_getpcni Obtains the primal cone infeasibility.

MSK_analyzesolution Print information related to the quality of the solution.

MSK_getsolutionstatus Obtains information about the problem and solution statuses.

- MSK_getsolutionslice

Syntax:

```
MSKrescodee MSK_getsolutionslice (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKsoliteme solitem,
    MSKidxt first,
    MSKidxt last,
    MSKrealt * values);
```

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

`solitem` **(input)** Which part of the solution is required.

`first` **(input)** Index of the first value in the slice.

`last` **(input)** Value of the last index+1 in the slice, e.g. if $xx[5,\ldots,9]$ is required `last` should be 10.

`values` **(output)** The values in the required sequence are stored sequentially in `values` starting at `values[0]`.

Description: Obtains a slice of the solution.

Consider the case of linear programming. The primal problem is given by

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x. \end{array} \tag{15.24}$$

and the corresponding dual problem is

$$\begin{array}{lcccc} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & c, & \\ & -y + s_l^c - s_u^c & = & 0, & \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & & \end{array} \tag{15.25}$$

The `solitem` argument determines which part of the solution is returned:

MSK_SOL_ITEM_XX: The variable `values` return x.

MSK_SOL_ITEM_Y: The variable `values` return y.

MSK_SOL_ITEM_SLC: The variable `values` return s_l^c.

MSK_SOL_ITEM_SUC: The variable `values` return s_u^c.

MSK_SOL_ITEM_SLX: The variable `values` return s_l^x.

MSK_SOL_ITEM_SUX: The variable `values` return s_u^x.

A conic optimization problem has the same primal variables as in the linear case. Recall that the dual of a conic optimization problem is given by:

$$\begin{array}{lccc} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x + s_n^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, \\ & s_n^x \in \mathcal{C}^* & & \end{array} \tag{15.26}$$

This introduces one additional dual variable s_n^x. This variable can be acceded by selecting `solitem` as MSK_SOL_ITEM_SNX.

The meaning of the values returned by this function also depends on the *solution status* which can be obtained with MSK_getsolutionstatus. Depending on the solution status `value` will be:

MSK_SOL_STA_OPTIMAL A part of the optimal solution satisfying the optimality criteria for continuous problems.

MSK_SOL_STA_INTEGER_OPTIMAL A part of the optimal solution satisfying the optimality criteria for integer problems.

MSK_SOL_STA_PRIM_FEAS A part of the solution satisfying the feasibility criteria.

MSK_SOL_STA_PRIM_INFEAS_CER A part of the primal certificate of infeasibility.

MSK_SOL_STA_DUAL_INFEAS_CER A part of the dual certificate of infeasibility.

See also:

MSK_getsolution Obtains the complete solution.

MSK_getsolutioni Obtains the solution for a single constraint or variable.

- MSK_getsolutionstatus

Syntax:

```
MSKrescodee MSK_getsolutionstatus (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKprostae * prosta,
    MSKsolstae * solsta);
```

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

`prosta` **(output)** Problem status.

`solsta` **(output)** Solution status.

Description: Obtains information about the problem and solution statuses.

- MSK_getsolutionstatuskeyslice

  Syntax:

  ```
  MSKrescodee MSK_getsolutionstatuskeyslice (
      MSKtask_t task,
      MSKaccmodee accmode,
      MSKsoltypee whichsol,
      MSKidxt first,
      MSKidxt last,
      MSKstakeye * sk);
  ```

  task **(input)** An optimization task.

  accmode **(input)** Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

  whichsol **(input)** Selects a solution.

  first **(input)** Index of the first value in the slice.

  last **(input)** Value of the last index+1 in the slice, e.g. if $xx[5,\ldots,9]$ is required last should be 10.

  sk **(output)** The status keys in the required sequence are stored sequentially in sk starting at sk[0].

  Description: Obtains a slice of the solution status keys.

  See also:

  MSK_getsolution Obtains the complete solution.

  MSK_getsolutioni Obtains the solution for a single constraint or variable.

- MSK_getstrparam

  Syntax:

  ```
  MSKrescodee MSK_getstrparam (
      MSKtask_t task,
      MSKsparame param,
      MSKCONST size_t maxlen,
      size_t * len,
      char * parvalue);
  ```

  task **(input)** An optimization task.

  param **(input)** Which parameter.

  maxlen **(input)** Length of the parvalue buffer.

  len **(output)** The length of the parameter value.

  parvalue **(output)** If this is not NULL, the parameter value is stored here.

  Description: Obtains the value of a string parameter.

- MSK_getstrparam64

Syntax:

```
MSKrescodee MSK_getstrparam64 (
    MSKtask_t task,
    MSKsparame param,
    MSKint64t maxlen,
    MSKint64t * len,
    char * parvalue);
```

`task` **(input)** An optimization task.

`param` **(input)** Which parameter.

`maxlen` **(input)** Length of the `parvalue` buffer.

`len` **(output)** The length of the parameter value.

`parvalue` **(output)** If this is not NULL, the parameter value is stored here.

Description: Obtains the value of a string parameter.

- `MSK_getstrparamal`

Syntax:

```
MSKrescodee MSK_getstrparamal (
    MSKtask_t task,
    MSKsparame param,
    MSKCONST size_t numaddchr,
    MSKstring_t * value);
```

`task` **(input)** An optimization task.

`param` **(input)** Which parameter.

`numaddchr` **(input)** Number of additional characters that is made room for in `value[0]`.

`value` **(input/output)** Is the value corresponding to string parameter `param`. `value[0]` is char buffer allocated MOSEK and it must be freed by `MSK_freetask`.

Description: Obtains the value of a string parameter.

- `MSK_getsymbcon`

Syntax:

```
MSKrescodee MSK_getsymbcon (
    MSKtask_t task,
    MSKidxt i,
    MSKCONST size_t maxlen,
    char * name,
    MSKintt * value);
```

`task` **(input)** An optimization task.

`i` **(input)** Index.

`maxlen` **(input)** The length of the buffer pointed to by the `value` argument.

`name` **(output)** Name of the ith symbolic constant.

`value` **(output)** The corresponding value.

Description: Obtains the name and corresponding value for the ith symbolic constant.

- `MSK_gettaskname`

  Syntax:

  ```
  MSKrescodee MSK_gettaskname (
      MSKtask_t task,
      MSKCONST size_t maxlen,
      size_t * len,
      char * taskname);
  ```

  `task` **(input)** An optimization task.
  `maxlen` **(input)** Length of the `taskname` array.
  `len` **(output)** Is assigned the length of the task name.
  `taskname` **(output)** Is assigned the task name.

  Description: Obtains the name assigned to the task.

- `MSK_gettaskname64`

  Syntax:

  ```
  MSKrescodee MSK_gettaskname64 (
      MSKtask_t task,
      MSKint64t maxlen,
      MSKint64t * len,
      char * taskname);
  ```

  `task` **(input)** An optimization task.
  `maxlen` **(input)** Length of the `taskname` array.
  `len` **(output)** Is assigned the length of the task name.
  `taskname` **(output)** Is assigned the task name.

  Description: Obtains the name assigned to the task.

- `MSK_getvarbranchdir`

  Syntax:

  ```
  MSKrescodee MSK_getvarbranchdir (
      MSKtask_t task,
      MSKidxt j,
      MSKbranchdire * direction);
  ```

  `task` **(input)** An optimization task.
  `j` **(input)** Index of the variable.
  `direction` **(output)** The branching direction assigned to variable j.

  Description: Obtains the branching direction for a given variable j.

- `MSK_getvarbranchorder`

Syntax:

```
MSKrescodee MSK_getvarbranchorder (
    MSKtask_t task,
    MSKidxt j,
    MSKintt * priority,
    MSKbranchdire * direction);
```

`task` **(input)** An optimization task.

`j` **(input)** Index of the variable.

`priority` **(output)** The branching priority assigned to variable j.

`direction` **(output)** The preferred branching direction for the j'th variable.

Description: Obtains the branching priority and direction for a given variable j.

- `MSK_getvarbranchpri`

Syntax:

```
MSKrescodee MSK_getvarbranchpri (
    MSKtask_t task,
    MSKidxt j,
    MSKintt * priority);
```

`task` **(input)** An optimization task.

`j` **(input)** Index of the variable.

`priority` **(output)** The branching priority assigned to variable j.

Description: Obtains the branching priority for a given variable j.

- `MSK_getvarname`

Syntax:

```
MSKrescodee MSK_getvarname (
    MSKtask_t task,
    MSKidxt i,
    MSKCONST size_t maxlen,
    char * name);
```

`task` **(input)** An optimization task.

`i` **(input)** Index.

`maxlen` **(input)** The length of the buffer pointed to by the `name` argument.

`name` **(output)** Is assigned the required name.

Description: Obtains a name of a variable.

See also:

`MSK_getmaxnamelen` Obtains the maximum length (not including terminating zero character) of any objective, constraint, variable or cone name.

- `MSK_getvarname64`

Syntax:

```
MSKrescodee MSK_getvarname64 (
    MSKtask_t task,
    MSKidxt i,
    MSKint64t maxlen,
    char * name);
```

`task` **(input)** An optimization task.

`i` **(input)** Index.

`maxlen` **(input)** The length of the buffer pointed to by the `name` argument.

`name` **(output)** Is assigned the required name.

Description: Obtains a name of a variable.

See also:

`MSK_getmaxnamelen` Obtains the maximum length (not including terminating zero character) of any objective, constraint, variable or cone name.

- `MSK_getvartype`

Syntax:

```
MSKrescodee MSK_getvartype (
    MSKtask_t task,
    MSKidxt j,
    MSKvariabletypee * vartype);
```

`task` **(input)** An optimization task.

`j` **(input)** Index of the variable.

`vartype` **(output)** Variable type of variable `j`.

Description: Gets the variable type of one variable.

- `MSK_getvartypelist`

Syntax:

```
MSKrescodee MSK_getvartypelist (
    MSKtask_t task,
    MSKintt num,
    MSKCONST MSKidxt * subj,
    MSKvariabletypee * vartype);
```

`task` **(input)** An optimization task.

`num` **(input)** Number of variables for which the variable type should be obtained.

`subj` **(input)** A list of variable indexes.

`vartype` **(output)** The variables types corresponding to the variables specified by `subj`.

Description: Obtains the variable type of one or more variables.

Upon return `vartype[k]` is the variable type of variable `subj[k]`.

- `MSK_initbasissolve`

  Syntax:

  ```
  MSKrescodee MSK_initbasissolve (
      MSKtask_t task,
      MSKidxt * basis);
  ```

  `task` **(input)** An optimization task.

  `basis` **(output)** The array of basis indexes to use.
  The array is interpreted as follows: If $\texttt{basis}[i] \leq \texttt{numcon} - 1$, then $x^c_{\texttt{basis}[i]}$ is in the basis at position i, otherwise $x_{\texttt{basis}[i]-\texttt{numcon}}$ is in the basis at position i.

  Description: Prepare a task for use with the `MSK_solvewithbasis` function.

  This function should be called

  – immediately before the first call to `MSK_solvewithbasis`, and
  – immediately before any subsequent call to `MSK_solvewithbasis` if the task has been modified.

  If the basis is singular i.e. not invertible, then
  the response code `MSK_RES_ERR_BASIS_SINGULAR`.

- `MSK_inputdata`

  Syntax:

  ```
  MSKrescodee MSK_inputdata (
      MSKtask_t task,
      MSKintt maxnumcon,
      MSKintt maxnumvar,
      MSKintt numcon,
      MSKintt numvar,
      MSKCONST MSKrealt * c,
      MSKrealt cfix,
      MSKCONST MSKintt * aptrb,
      MSKCONST MSKintt * aptre,
      MSKCONST MSKidxt * asub,
      MSKCONST MSKrealt * aval,
      MSKCONST MSKboundkeye * bkc,
      MSKCONST MSKrealt * blc,
      MSKCONST MSKrealt * buc,
      MSKCONST MSKboundkeye * bkx,
      MSKCONST MSKrealt * blx,
      MSKCONST MSKrealt * bux);
  ```

  `task` **(input)** An optimization task.

  `maxnumcon` **(input)** Number of preallocated constraints in the optimization task.

  `maxnumvar` **(input)** Number of preallocated variables in the optimization task.

  `numcon` **(input)** Number of constraints.

`numvar` **(input)** Number of variables.

`c` **(input)** Linear terms of the objective as a dense vector. The lengths is the number of variables.

`cfix` **(input)** Fixed term in the objective.

`aptrb` **(input)** Pointer to the first element in the rows or the columns of A. See (5.33) and Section 5.8.3.

`aptre` **(input)** Pointers to the last element + 1 in the rows or the columns of A. See (5.33) and Section 5.8.3

`asub` **(input)** Coefficient subscripts. See (5.33) and Section 5.8.3.

`aval` **(input)** Coefficient values. See (5.33) and Section 5.8.3.

`bkc` **(input)** Bound keys for the constraints.

`blc` **(input)** Lower bounds for the constraints.

`buc` **(input)** Upper bounds for the constraints.

`bkx` **(input)** Bound keys for the variables.

`blx` **(input)** Lower bounds for the variables.

`bux` **(input)** Upper bounds for the variables.

Description: Input the linear part of an optimization problem.

The non-zeros of A are inputted column-wise in the format described in Section 5.8.3.2.

For an explained code example see Section 5.2 and Section 5.8.3.

- `MSK_inputdata64`

Syntax:

```
MSKrescodee MSK_inputdata64 (
    MSKtask_t task,
    MSKintt maxnumcon,
    MSKintt maxnumvar,
    MSKintt numcon,
    MSKintt numvar,
    MSKCONST MSKrealt * c,
    MSKrealt cfix,
    MSKCONST MSKint64t * aptrb,
    MSKCONST MSKint64t * aptre,
    MSKCONST MSKidxt * asub,
    MSKCONST MSKrealt * aval,
    MSKCONST MSKboundkeye * bkc,
    MSKCONST MSKrealt * blc,
    MSKCONST MSKrealt * buc,
    MSKCONST MSKboundkeye * bkx,
    MSKCONST MSKrealt * blx,
    MSKCONST MSKrealt * bux);
```

`task` **(input)** An optimization task.

`maxnumcon` **(input)** Number of preallocated constraints in the optimization task.

`maxnumvar` **(input)** Number of preallocated variables in the optimization task.

`numcon` **(input)** Number of constraints.

`numvar` **(input)** Number of variables.

`c` **(input)** Linear terms of the objective as a dense vector. The lengths is the number of variables.

`cfix` **(input)** Fixed term in the objective.

`aptrb` **(input)** Pointer to the first element in the rows or the columns of A. See (5.33) and Section 5.8.3.

`aptre` **(input)** Pointers to the last element + 1 in the rows or the columns of A. See (5.33) and Section 5.8.3

`asub` **(input)** Coefficient subscripts. See (5.33) and Section 5.8.3.

`aval` **(input)** Coefficient values. See (5.33) and Section 5.8.3.

`bkc` **(input)** Bound keys for the constraints.

`blc` **(input)** Lower bounds for the constraints.

`buc` **(input)** Upper bounds for the constraints.

`bkx` **(input)** Bound keys for the variables.

`blx` **(input)** Lower bounds for the variables.

`bux` **(input)** Upper bounds for the variables.

Description: Input the linear part of an optimization problem.

The non-zeros of A are inputted column-wise in the format described in Section 5.8.3.2.

For an explained code example see Section 5.2 and Section 5.8.3.

- `MSK_isdouparname`

Syntax:

```
MSKrescodee MSK_isdouparname (
    MSKtask_t task,
    MSKCONST char * parname,
    MSKdparame * param);
```

`task` **(input)** An optimization task.

`parname` **(input)** Parameter name.

`param` **(output)** Which parameter.

Description: Checks whether `parname` is a valid double parameter name.

- `MSK_isintparname`

Syntax:

```
MSKrescodee MSK_isintparname (
    MSKtask_t task,
    MSKCONST char * parname,
    MSKiparame * param);
```

`task` **(input)** An optimization task.

`parname` **(input)** Parameter name.

`param` **(output)** Which parameter.

Description: Checks whether `parname` is a valid integer parameter name.

- `MSK_isstrparname`

  Syntax:

  ```
  MSKrescodee MSK_isstrparname (
      MSKtask_t task,
      MSKCONST char * parname,
      MSKsparame * param);
  ```

  `task` **(input)** An optimization task.

  `parname` **(input)** Parameter name.

  `param` **(output)** Which parameter.

  Description: Checks whether `parname` is a valid string parameter name.

- `MSK_linkfiletotaskstream`

  Syntax:

  ```
  MSKrescodee MSK_linkfiletotaskstream (
      MSKtask_t task,
      MSKstreamtypee whichstream,
      MSKCONST char * filename,
      MSKintt append);
  ```

  `task` **(input)** An optimization task.

  `whichstream` **(input)** Index of the stream.

  `filename` **(input)** The name of the file where text from the stream defined by `whichstream` is written.

  `append` **(input)** If this argument is 0 the output file will be overwritten, otherwise text is append to the output file.

  Description: Directs all output from a task stream to a file.

- `MSK_linkfunctotaskstream`

  Syntax:

  ```
  MSKrescodee MSK_linkfunctotaskstream (
      MSKtask_t task,
      MSKstreamtypee whichstream,
      MSKuserhandle_t handle,
      MSKstreamfunc func);
  ```

  `task` **(input)** An optimization task.

  `whichstream` **(input)** Index of the stream.

  `handle` **(input)** A user-defined handle which is passed to the user-defined function `func`.

`func` **(input)** All output to the stream `whichstream` is passed to `func`.

Description: Connects a user-defined function to a task stream.

- `MSK_makesolutionstatusunknown`

  Syntax:

  ```
  MSKrescodee MSK_makesolutionstatusunknown (
      MSKtask_t task,
      MSKsoltypee whichsol);
  ```

  `task` **(input)** An optimization task.

  `whichsol` **(input)** Selects a solution.

  Description: Sets the solution status to unknown. Also all the status keys for the constraints and the variables are set to unknown.

- `MSK_netextraction`

  Syntax:

  ```
  MSKrescodee MSK_netextraction (
      MSKtask_t task,
      MSKintt * numcon,
      MSKintt * numvar,
      MSKidxt * netcon,
      MSKidxt * netvar,
      MSKrealt * scalcon,
      MSKrealt * scalvar,
      MSKrealt * cx,
      MSKboundkeye * bkc,
      MSKrealt * blc,
      MSKrealt * buc,
      MSKboundkeye * bkx,
      MSKrealt * blx,
      MSKrealt * bux,
      MSKidxt * from,
      MSKidxt * to);
  ```

  `task` **(input)** Task containing the linear problem the network extraction heuristics should be performed on.

  `numcon` **(output)** Number of network constraints (nodes) in the embedded network.

  `numvar` **(output)** Number of network variables (arcs) in the embedded network.

  `netcon` **(output)** Indexes of network constraints (nodes) in the embedded network.

  `netvar` **(output)** Indexes of network variables (arcs) in the embedded network.

  `scalcon` **(output)** Scaling values on constraints, used to convert the original part of the problem into network form.

  `scalvar` **(output)** Scaling values on variables, used to convert the original part of the problem into network form.

`cx` **(output)** Linear terms of the objective for variables (arcs) in the embedded network as a dense vector.

`bkc` **(output)** Bound keys for the constraints (nodes) in the embedded network.

`blc` **(output)** Lower bounds for the constraints (nodes) in the embedded network.

`buc` **(output)** Upper bounds for the constraints (nodes) in the embedded network.

`bkx` **(output)** Bound keys for the variables (arcs) in the embedded network.

`blx` **(output)** Lower bounds for the variables (arcs) in the embedded network.

`bux` **(output)** Upper bounds for the variables (arcs) in the embedded network.

`from` **(output)** Defines the origins of the arcs in the embedded network.

`to` **(output)** Defines the destinations of the arcs in the embedded network.

Description: Uses a heuristic to find an embedded network in the problem specified in task. The returned network is a pure network flow problem and can be solved with a direct call to the network optimizer `MSK_netoptimize`.
Each arc a in the network corresponds to a scaled subset of elements in column $j = \texttt{netvar}[a]$ from the constraint matrix A stored in `task`. Each node n in the network corresponds to a scaled subset of elements in constraint $i = netcon[n]$ from A. Data structures for network problems is explained in 5.2 and 6.6. The relation between A and the extracted embedded network can be explained as follows:

- $A_{\texttt{netcon[from}[a]\texttt{],netvar}[a]} * \texttt{scalcon[netcon[from}[a]\texttt{]]} * \texttt{scalvar[netvar}[a]\texttt{]} = -1$
- $A_{\texttt{netcon[to}[a]\texttt{],netvar}[a]} * \texttt{scalcon[netcon[to}[a]\texttt{]]} * \texttt{scalvar[netvar}[a]\texttt{]} = 1$
- The scaled matrix has at most two non-zeroes in each column in `netvar` over the indexes in `netcon` (i.e defines a pure network flow matrix).

Please note if a column $j = netvar[a]$ is only represented by one non-zero in the embedded network, then either $from[a] = netcon$ or $to[a] = netcon$.

See also:

`MSK_netoptimize` Optimizes a pure network flow problem.

- `MSK_netoptimize`

Syntax:

```
MSKrescodee MSK_netoptimize (
    MSKtask_t task,
    MSKintt numcon,
    MSKintt numvar,
    MSKCONST MSKrealt * cc,
    MSKCONST MSKrealt * cx,
    MSKCONST MSKboundkeye * bkc,
    MSKCONST MSKrealt * blc,
    MSKCONST MSKrealt * buc,
    MSKCONST MSKboundkeye * bkx,
    MSKCONST MSKrealt * blx,
    MSKCONST MSKrealt * bux,
    MSKCONST MSKidxt * from,
```

```
    MSKCONST MSKidxt * to,
    MSKprostae * prosta,
    MSKsolstae * solsta,
    MSKbooleant hotstart,
    MSKstakeye * skc,
    MSKstakeye * skx,
    MSKrealt * xc,
    MSKrealt * xx,
    MSKrealt * y,
    MSKrealt * slc,
    MSKrealt * suc,
    MSKrealt * slx,
    MSKrealt * sux);
```

`task` **(input)** An optimization task.

`numcon` **(input)** Number of network constraints (nodes) in the network.

`numvar` **(input)** Number of network variables (arcs) in the network.

`cc` **(input)** Linear terms of the objective for constraints (nodes) as a dense vector.

`cx` **(input)** Linear terms of the objective for variables (arcs) as a dense vector.

`bkc` **(input)** Bound keys for the constraints (nodes).

`blc` **(input)** Lower bounds for the constraints (nodes).

`buc` **(input)** Upper bounds for the constraints (nodes).

`bkx` **(input)** Bound keys for the variables (arcs).

`blx` **(input)** Lower bounds for the variables (arcs).

`bux` **(input)** Upper bounds for the variables (arcs).

`from` **(input)** Defines the origins of the arcs in the network.

`to` **(input)** Defines the destinations of the arcs in the network.

`prosta` **(output)** Problem status.

`solsta` **(output)** Solution status.

`hotstart` **(input)** If zero the network optimizer will not use hot-starts, if non-zero a solution must be defined in the solution variables below, which will be used to hot-start the network optimizer.

`skc` **(input/output)** Status keys for the constraints (nodes).

`skx` **(input/output)** Status keys for the variables (arcs).

`xc` **(input/output)** Primal constraint solution (nodes).

`xx` **(input/output)** Primal variable solution (arcs).

`y` **(input/output)** Vector of dual variables corresponding to the constraints (nodes).

`slc` **(input/output)** Dual variables corresponding to the lower bounds on the constraints (nodes).

`suc` **(input/output)** Dual variables corresponding to the upper bounds on the constraints (nodes).

`slx` **(input/output)** Dual variables corresponding to the lower bounds on the constraints (arcs).

`sux` **(input/output)** Dual variables corresponding to the upper bounds on the constraints (arcs).

Description: Uses the network optimizer to solve the given network problem. The problem must be a pure network flow problem. If `hotstart` is zero the network optimizer will not use hot-starts, if non-zero a solution must be defined in the solution variables `skc`,`skx`,`xc`,`xx`,`y`,`slc`,`suc`,`slx` and `sux`, which will be used to hot-start the network optimizer. Please note `task` only acts as a dummy task, where parameters and streams can be set for the network optimizer. No other data in `task` is used.

See also:

`MSK_netextraction` Finds embedded network structure.

- `MSK_optimize`

Syntax:

```
MSKrescodee MSK_optimize (MSKtask_t task)
```

`task` **(input)** An optimization task.

Description: Calls the optimizer. Depending on the problem type and the selected optimizer this will call one of the optimizers in MOSEK. By default the interior point optimizer will be selected for continuous problems. The optimizer may be selected manually by setting the parameter `MSK_IPAR_OPTIMIZER`.

See also:

`MSK_optimizeconcurrent` Optimize a given task with several optimizers concurrently.

`MSK_getsolution` Obtains the complete solution.

`MSK_getsolutioni` Obtains the solution for a single constraint or variable.

`MSK_getsolutioninf` Obtains information about a solution.

`MSK_IPAR_OPTIMIZER`

- `MSK_optimizeconcurrent`

Syntax:

```
MSKrescodee MSK_optimizeconcurrent (
    MSKtask_t task,
    MSKCONST MSKtask_t * taskarray,
    MSKintt num);
```

`task` **(input)** An optimization task.

`taskarray` **(input)** An array of `num` tasks.

`num` **(input)** Length of `taskarray`.

Description: Solves several instances of the same problem in parallel, with unique parameter settings for each task. The argument `task` contains the problem to be solved. `taskarray` is a pointer to an array of `num` empty tasks. The task `task` and the `num` tasks pointed to by `taskarray` are solved in parallel. That is `num` + 1 threads are started with one optimizer in each. Each of the tasks can be initialized with different parameters, e.g different selection of solver.

All the concurrently running tasks are stopped when the optimizer successfully terminates for one of the tasks. After the function returns `task` contains the solution found by the task that finished first.

After MSK_optimizeconcurrent returns `task` holds the optimal solution of the task which finished first. If all the concurrent optimizations finished without providing an optimal solution the error code from the solution of the task `task` is returned.

In summary a call to MSK_optimizeconcurrent does the following:

1. All data except task parameters (MSKiparam, MSKdparam and MSKsparam) in `task` is copied to each of the tasks in `taskarray`. In particular this means that any solution in `task` is copied to the other tasks. Call-back functions are not copied.
2. The tasks `task` and the `num` tasks in `taskarray` are started in parallel.
3. When a task finishes providing an optimal solution (or a certificate of infeasibility) its solution is copied to `task` and all other tasks are stopped.

For an explained code example see Section 8.6.4.

- MSK_optimizersummary

  Syntax:

  ```
  MSKrescodee MSK_optimizersummary (
      MSKtask_t task,
      MSKstreamtypee whichstream);
  ```

  `task` **(input)** An optimization task.

  whichstream **(input)** Index of the stream.

  Description: Prints a short summary with optimizer statistics for last optimization.

- MSK_optimizetrm

  Syntax:

  ```
  MSKrescodee MSK_optimizetrm (
      MSKtask_t task,
      MSKrescodee * trmcode);
  ```

  `task` **(input)** An optimization task.

  trmcode **(output)** Is either MSK_RES_OK or a termination response code.

  Description: Calls the optimizer. Depending on the problem type and the selected optimizer this will call one of the optimizers in MOSEK. By default the interior point optimizer will be selected for continuous problems. The optimizer may be selected manually by setting the parameter MSK_IPAR_OPTIMIZER.

  This function is equivalent to MSK_optimize except in the case where MSK_optimize would have returned a termination response code such as

  – MSK_RES_TRM_MAX_ITERATIONS or

  – MSK_RES_TRM_STALL.

  Response codes comes in three categories:

- Errors: Indicate that an error has occurred during the optimization. E.g that the optimizer has run out of memory (MSK_RES_ERR_SPACE).
- Warnings: Less fatal than errors. E.g MSK_RES_WRN_LARGE_CJ indicating possibly problematic problem data.
- Termination codes: Relaying information about the conditions under which the optimizer terminated. E.g MSK_RES_TRM_MAX_ITERATIONS indicates that the optimizer finished because it reached the maximum number of iterations specified by the user.

This function returns errors on the left hand side. Warnings are not returned and termination codes are returned in the separate argument `trmcode`.

See also:

MSK_optimize Optimizes the problem.

MSK_optimizeconcurrent Optimize a given task with several optimizers concurrently.

MSK_getsolution Obtains the complete solution.

MSK_getsolutioni Obtains the solution for a single constraint or variable.

MSK_getsolutioninf Obtains information about a solution.

MSK_IPAR_OPTIMIZER

- MSK_primalsensitivity

Syntax:

```
MSKrescodee MSK_primalsensitivity (
    MSKtask_t task,
    MSKintt numi,
    MSKCONST MSKidxt * subi,
    MSKCONST MSKmarke * marki,
    MSKintt numj,
    MSKCONST MSKidxt * subj,
    MSKCONST MSKmarke * markj,
    MSKrealt * leftpricei,
    MSKrealt * rightpricei,
    MSKrealt * leftrangei,
    MSKrealt * rightrangei,
    MSKrealt * leftpricej,
    MSKrealt * rightpricej,
    MSKrealt * leftrangej,
    MSKrealt * rightrangej);
```

`task` **(input)** An optimization task.

`numi` **(input)** Number of bounds on constraints to be analyzed. Length of `subi` and `marki`.

`subi` **(input)** Indexes of bounds on constraints to analyze.

`marki` **(input)** The value of `marki[i]` specifies for which bound (upper or lower) on constraint `subi[i]` sensitivity analysis should be performed.

`numj` **(input)** Number of bounds on variables to be analyzed. Length of `subj` and `markj`.

`subj` **(input)** Indexes of bounds on variables to analyze.

markj **(input)** The value of `markj[j]` specifies for which bound (upper or lower) on variable `subj[j]` sensitivity analysis should be performed.

`leftpricei` **(output)** `leftpricei[i]` is the left shadow price for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

`rightpricei` **(output)** `rightpricei[i]` is the right shadow price for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

`leftrangei` **(output)** `leftrangei[i]` is the left range for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

`rightrangei` **(output)** `rightrangei[i]` is the right range for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

`leftpricej` **(output)** `leftpricej[j]` is the left shadow price for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]`.

`rightpricej` **(output)** `rightpricej[j]` is the right shadow price for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]` .

`leftrangej` **(output)** `leftrangej[j]` is the left range for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]`.

`rightrangej` **(output)** `rightrangej[j]` is the right range for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]`.

Description: Calculates sensitivity information for bounds on variables and constraints.

For details on sensitivity analysis and the definitions of *shadow price* and *linearity interval* see chapter 12.

The constraints for which sensitivity analysis is performed are given by the data structures:

1. `subi` Index of constraint to analyze.
2. `marki` Indicate for which bound of constraint `subi[i]` sensitivity analysis is performed. If `marki[i] = MSK_MARK_UP` the upper bound of constraint `subi[i]` is analyzed, and if `marki[i] = MSK_MARK_LO` the lower bound is analyzed. If `subi[i]` is an equality constraint, either `MSK_MARK_LO` or `MSK_MARK_UP` can be used to select the constraint for sensitivity analysis.

Consider the problem:

$$\begin{array}{lrcl} \text{minimize} & x_1 + x_2 & & \\ \text{subject to} -1 \leq & x_1 - x_2 & \leq & 1, \\ & x_1 & = & 0, \\ & x_1 \geq 0, x_2 \geq 0 & & \end{array} \tag{15.27}$$

Suppose that

```
numi = 1;
subi = [0];
marki = [MSK_MARK_UP]
```

then

`leftpricei[0]`, `rightpricei[0]`, `leftrangei[0]` and `rightrangei[0]` will contain the sensitivity information for the upper bound on constraint 0 given by the expression:

$$x_1 - x_2 \leq 1 \tag{15.28}$$

Similarly, the variables for which to perform sensitivity analysis are given by the structures:

1. `subj` Index of variables to analyze.
2. `markj` Indicate for which bound of variable `subi[j]` sensitivity analysis is performed. If `markj[j] = MSK_MARK_UP` the upper bound of constraint `subi[j]` is analyzed, and if `markj[j] = MSK_MARK_LO` the lower bound is analyzed. If `subi[j]` is an equality constraint, either `MSK_MARK_LO` or `MSK_MARK_UP` can be used to select the constraint for sensitivity analysis.

For an example, please see Section 12.5.

The type of sensitivity analysis to be performed (basis or optimal partition) is controlled by the parameter `MSK_IPAR_SENSITIVITY_TYPE`.

See also:

`MSK_dualsensitivity` Performs sensitivity analysis on objective coefficients.

`MSK_sensitivityreport` Creates a sensitivity report.

`MSK_IPAR_SENSITIVITY_TYPE`

`MSK_IPAR_LOG_SENSITIVITY`

`MSK_IPAR_LOG_SENSITIVITY_OPT`

- `MSK_printdata`

Syntax:

```
MSKrescodee MSK_printdata (
    MSKtask_t task,
    MSKstreamtypee whichstream,
    MSKidxt firsti,
    MSKidxt lasti,
    MSKidxt firstj,
    MSKidxt lastj,
    MSKidxt firstk,
    MSKidxt lastk,
    MSKintt c,
    MSKintt qo,
    MSKintt a,
    MSKintt qc,
    MSKintt bc,
    MSKintt bx,
    MSKintt vartype,
    MSKintt cones);
```

`task` **(input)** An optimization task.

`whichstream` **(input)** Index of the stream.

`firsti` **(input)** Index of first constraint for which data should be printed.

`lasti` **(input)** Index of last constraint plus 1 for which data should be printed.

`firstj` **(input)** Index of first variable for which data should be printed.

`lastj` **(input)** Index of last variable plus 1 for which data should be printed.

`firstk` **(input)** Index of first cone for which data should be printed.

`lastk` **(input)** Index of last cone plus 1 for which data should be printed.

`c` **(input)** If non-zero c is printed.

`qo` **(input)** If non-zero Q^o is printed.

`a` **(input)** If non-zero A is printed.

`qc` **(input)** If non-zero Q^k is printed for the relevant constraints.

`bc` **(input)** If non-zero the constraints bounds are printed.

`bx` **(input)** If non-zero the variable bounds are printed.

`vartype` **(input)** If non-zero the variable types are printed.

`cones` **(input)** If non-zero the conic data is printed.

Description: Prints a part of the problem data to a stream. This function is normally used for debugging purposes only, e.g. to verify that the correct data has been inputted.

- `MSK_printparam`

  Syntax:

  ```
  MSKrescodee MSK_printparam (MSKtask_t task)
  ```

  `task` **(input)** An optimization task.

  Description: Prints the current parameter settings to the message stream.

- `MSK_probtypetostr`

  Syntax:

  ```
  MSKrescodee MSK_probtypetostr (
      MSKtask_t task,
      MSKproblemtypee probtype,
      char * str);
  ```

  `task` **(input)** An optimization task.

  `probtype` **(input)** Problem type.

  `str` **(output)** String corresponding to the problem type key `probtype`.

  Description: Obtains a string containing the name of a problem type given.

- `MSK_prostatostr`

  Syntax:

  ```
  MSKrescodee MSK_prostatostr (
      MSKtask_t task,
      MSKprostae prosta,
      char * str);
  ```

  `task` **(input)** An optimization task.

  `prosta` **(input)** Problem status.

  `str` **(output)** String corresponding to the status key `prosta`.

Description: Obtains a string containing the name of a problem status given.

- MSK_putaij

Syntax:

```
MSKrescodee MSK_putaij (
    MSKtask_t task,
    MSKidxt i,
    MSKidxt j,
    MSKrealt aij);
```

`task` **(input)** An optimization task.
`i` **(input)** Index of the constraint in which the change should occur.
`j` **(input)** Index of the variable in which the change should occur.
`aij` **(input)** New coefficient for $a_{i,j}$.

Description: Changes a coefficient in A using the method

$$a_{\texttt{ij}} = \texttt{aij}.$$

See also:

MSK_putavec Replaces all elements in one row or column of the linear coefficient matrix.
MSK_putaij Changes a single value in the linear coefficient matrix.
MSK_putmaxnumanz The function changes the size of the preallocated storage for linear coefficients.

- MSK_putaijlist

Syntax:

```
MSKrescodee MSK_putaijlist (
    MSKtask_t task,
    MSKintt num,
    MSKCONST MSKidxt * subi,
    MSKCONST MSKidxt * subj,
    MSKCONST MSKrealt * valij);
```

`task` **(input)** An optimization task.
`num` **(input)** Number of coefficients that should be changed.
`subi` **(input)** Constraint indexes in which the change should occur.
`subj` **(input)** Variable indexes in which the change should occur.
`valij` **(input)** New coefficient values for $a_{i,j}$.

Description: Changes one or more coefficients in A using the method

$$a_{\texttt{subi[k]},\texttt{subj[k]}} = \texttt{valij[k]}, \quad k = 0, \ldots, \texttt{num} - 1.$$

If the same $a_{i,j}$ entry appears multiple times only the last one will be used.

See also:

MSK_putavec Replaces all elements in one row or column of the linear coefficient matrix.

MSK_putaij Changes a single value in the linear coefficient matrix.

MSK_putmaxnumanz The function changes the size of the preallocated storage for linear coefficients.

- MSK_putavec

Syntax:

```
MSKrescodee MSK_putavec (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKintt nzi,
    MSKCONST MSKidxt * asub,
    MSKCONST MSKrealt * aval);
```

task **(input)** An optimization task.

accmode **(input)** Defines whether to replace a column or a row.

i **(input)** If accmode equals MSK_ACC_CON, then i is a constraint index. Otherwise it is a column index.

nzi **(input)** Number of non-zeros in the vector.

asub **(input)** Index of the $a_{i,j}$ values that should be changed.

aval **(input)** New $a_{i,j}$ values.

Description: Replaces all elements in one row or column of A.

Assuming that there are no duplicate subscripts in asub, assignment is performed as follows:

- If accmode is MSK_ACC_CON, then

$$a_{\mathtt{i},\mathtt{asub}[k]} = \mathtt{aval}[k], \quad k = 0, \ldots, \mathtt{nzi} - 1$$

  and all other $a_{\mathtt{i},\cdot} = 0$.
- If accmode is MSK_ACC_VAR, then

$$a_{\mathtt{asub}[k],\mathtt{i}} = \mathtt{aval}[k], \quad k = 0, \ldots, \mathtt{nzi} - 1$$

  and all other $a_{\cdot,\mathtt{i}} = 0$.

If asub contains duplicates, the corresponding coefficients will be added together.

For an explanation of the meaning of ptrb and ptre see Section 5.8.3.2.

See also:

MSK_putavec Replaces all elements in one row or column of the linear coefficient matrix.

MSK_putaij Changes a single value in the linear coefficient matrix.

MSK_putmaxnumanz The function changes the size of the preallocated storage for linear coefficients.

- MSK_putaveclist

Syntax:

```
MSKrescodee MSK_putaveclist (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKintt num,
    MSKCONST MSKidxt * sub,
    MSKCONST MSKintt * ptrb,
    MSKCONST MSKintt * ptre,
    MSKCONST MSKidxt * asub,
    MSKCONST MSKrealt * aval);
```

`task` **(input)** An optimization task.

`accmode` **(input)** Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

`num` **(input)** Number of rows or columns of A to replace.

`sub` **(input)** Indexes of rows or columns that should be replaced. `sub` should not contain duplicate values.

`ptrb` **(input)** Array of pointers to the first element in the rows or columns stored in `asub` and `aval`. For an explanation of the meaning of `ptrb` see Section 5.8.3.2.

`ptre` **(input)** Array of pointers to the last element plus one in the rows or columns stored in `asub` and `aval`. For an explanation of the meaning of `ptre` see Section 5.8.3.2.

`asub` **(input)** If `accmode` is `MSK_ACC_CON`, then `asub` contains the new variable indexes, otherwise it contains the new constraint indexes.

`aval` **(input)** Coefficient values. See (5.33) and Section 5.8.3.

Description: Replaces all elements in a set of rows or columns of A.

The elements are replaced as follows.

- If `accmode` is `MSK_ACC_CON`, then for $i = 0, \ldots, \texttt{num} - 1$

$$a_{\texttt{sub}[i],\texttt{asub}[k]} = \texttt{aval}[k], \quad k = \texttt{aptrb}[i], \ldots, \texttt{aptre}[i] - 1.$$

- If `accmode` is `MSK_ACC_VAR`, then for $i = 0, \ldots, \texttt{num} - 1$

$$a_{\texttt{asub}[k],\texttt{sub}[i]} = \texttt{aval}[k], \quad k = \texttt{aptrb}[i], \ldots, \texttt{aptre}[i] - 1.$$

If the same row or column appears multiple times only the last one will be used.

See also:

`MSK_putavec` Replaces all elements in one row or column of the linear coefficient matrix.

`MSK_putmaxnumanz` The function changes the size of the preallocated storage for linear coefficients.

- `MSK_putaveclist64`

Syntax:

```
MSKrescodee MSK_putaveclist64 (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKintt num,
    MSKCONST MSKidxt * sub,
    MSKCONST MSKint64t * ptrb,
    MSKCONST MSKint64t * ptre,
    MSKCONST MSKidxt * asub,
    MSKCONST MSKrealt * aval);
```

`task` **(input)** An optimization task.

`accmode` **(input)** Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

`num` **(input)** Number of rows or columns of A to replace.

`sub` **(input)** Indexes of rows or columns that should be replaced. `sub` should not contain duplicate values.

`ptrb` **(input)** Array of pointers to the first element in the rows or columns stored in `asub` and `aval`. For an explanation of the meaning of `ptrb` see Section 5.8.3.2.

`ptre` **(input)** Array of pointers to the last element plus one in the rows or columns stored in `asub` and `aval`. For an explanation of the meaning of `ptre` see Section 5.8.3.2.

`asub` **(input)** If `accmode` is `MSK_ACC_CON`, then `asub` contains the new variable indexes, otherwise it contains the new constraint indexes.

`aval` **(input)** Coefficient values. See (5.33) and Section 5.8.3.

Description: Replaces all elements in a set of rows or columns of A.

The elements are replaced as follows.

- If `accmode` is `MSK_ACC_CON`, then for $i = 0, \ldots, \texttt{num} - 1$

$$a_{\texttt{sub}[i],\texttt{asub}[k]} = \texttt{aval}[k], \quad k = \texttt{aptrb}[i], \ldots, \texttt{aptre}[i] - 1.$$

- If `accmode` is `MSK_ACC_VAR`, then for $i = 0, \ldots, \texttt{num} - 1$

$$a_{\texttt{asub}[k],\texttt{sub}[i]} = \texttt{aval}[k], \quad k = \texttt{aptrb}[i], \ldots, \texttt{aptre}[i] - 1.$$

If the same row or column appears multiple times only the last one will be used.

See also:

`MSK_putavec` Replaces all elements in one row or column of the linear coefficient matrix.

`MSK_putmaxnumanz` The function changes the size of the preallocated storage for linear coefficients.

- `MSK_putbound`

Syntax:

```
MSKrescodee MSK_putbound (
    MSKtask_t task,
    MSKaccmodee accmode,
```

```
    MSKidxt i,
    MSKboundkeye bk,
    MSKrealt bl,
    MSKrealt bu);
```

`task` **(input)** An optimization task.

`accmode` **(input)** Defines whether the bound for a constraint or a variable is changed.

`i` **(input)** Index of the constraint or variable.

`bk` **(input)** New bound key.

`bl` **(input)** New lower bound.

`bu` **(input)** New upper bound.

Description: Changes the bounds for either one constraint or one variable.

If the a bound value specified is numerically larger than `MSK_DPAR_DATA_TOL_BOUND_INF` it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than `MSK_DPAR_DATA_TOL_BOUND_WRN`, a warning will be displayed, but the bound is inputted as specified.

See also:

`MSK_putboundlist` Changes the bounds of constraints or variables.

- `MSK_putboundlist`

Syntax:

```
MSKrescodee MSK_putboundlist (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKintt num,
    MSKCONST MSKidxt * sub,
    MSKCONST MSKboundkeye * bk,
    MSKCONST MSKrealt * bl,
    MSKCONST MSKrealt * bu);
```

`task` **(input)** An optimization task.

`accmode` **(input)** Defines whether bounds for constraints (`MSK_ACC_CON`) or variables (`MSK_ACC_VAR`) are changed.

`num` **(input)** Number of bounds that should be changed.

`sub` **(input)** Subscripts of the bounds that should be changed.

`bk` **(input)** Constraint or variable index `sub[t]` is assigned the bound key `bk[t]`.

`bl` **(input)** Constraint or variable index `sub[t]` is assigned the lower bound `bl[t]`.

`bu` **(input)** Constraint or variable index `sub[t]` is assigned the upper bound `bu[t]`.

Description: Changes the bounds for either some constraints or variables. If multiple bound changes are specified for a constraint or a variable, only the last change takes effect.

See also:

`MSK_putbound` Changes the bound for either one constraint or one variable.

`MSK_DPAR_DATA_TOL_BOUND_INF`

MSK_DPAR_DATA_TOL_BOUND_WRN

- MSK_putboundslice

Syntax:

```
MSKrescodee MSK_putboundslice (
    MSKtask_t task,
    MSKaccmodee con,
    MSKidxt first,
    MSKidxt last,
    MSKCONST MSKboundkeye * bk,
    MSKCONST MSKrealt * bl,
    MSKCONST MSKrealt * bu);
```

`task` **(input)** An optimization task.

`con` **(input)** Defines whether bounds for constraints (MSK_ACC_CON) or variables (MSK_ACC_VAR) are changed.

`first` **(input)** First index in the sequence.

`last` **(input)** Last index plus 1 in the sequence.

`bk` **(input)** Bound keys.

`bl` **(input)** Values for lower bounds.

`bu` **(input)** Values for upper bounds.

Description: Changes the bounds for a sequence of variables or constraints.

See also:

MSK_putbound Changes the bound for either one constraint or one variable.

MSK_DPAR_DATA_TOL_BOUND_INF

MSK_DPAR_DATA_TOL_BOUND_WRN

- MSK_putcallbackfunc

Syntax:

```
MSKrescodee MSK_putcallbackfunc (
    MSKtask_t task,
    MSKcallbackfunc func,
    MSKuserhandle_t handle);
```

`task` **(input)** An optimization task.

`func` **(input)** A user-defined function which will be called occasionally from within the MOSEK optimizers. If the argument is a `NULL` pointer, then a previous inputted call-back function removed. The progress function has the type MSK_callbackfunc.

`handle` **(input)** A pointer to a user-defined data structure. Whenever the function `callbackfunc` is called, then `handle` is passed to the function.

Description: The function is used to input a user-defined progress call-back function of type MSK_callbackfunc. The call-back function is called frequently during the optimization process.

See also:

MSK_IPAR_LOG_SIM_FREQ

- MSK_putcfix

  Syntax:

  ```
  MSKrescodee MSK_putcfix (
      MSKtask_t task,
      MSKrealt cfix);
  ```

  `task` **(input)** An optimization task.
  `cfix` **(input)** Fixed term in the objective.

  Description: Replaces the fixed term in the objective by a new one.

- MSK_putcj

  Syntax:

  ```
  MSKrescodee MSK_putcj (
      MSKtask_t task,
      MSKidxt j,
      MSKrealt cj);
  ```

  `task` **(input)** An optimization task.
  `j` **(input)** Index of the variable for which c should be changed.
  `cj` **(input)** New value of c_j.

  Description: Modifies one coefficient in the linear objective vector c, i.e.

  $$c_{\texttt{j}} = \texttt{cj}.$$

- MSK_putclist

  Syntax:

  ```
  MSKrescodee MSK_putclist (
      MSKtask_t task,
      MSKintt num,
      MSKCONST MSKidxt * subj,
      MSKCONST MSKrealt * val);
  ```

  `task` **(input)** An optimization task.
  `num` **(input)** Number of coefficients that should be changed.
  `subj` **(input)** Index of variables for which c should be changed.
  `val` **(input)** New numerical values for coefficients in c that should be modified.

  Description: Modifies elements in the linear term c in the objective using the principle

  $$c_{\texttt{subj[t]}} = \texttt{val[t]}, \quad t = 0, \ldots, \texttt{num} - 1.$$

  If a variable index is specified multiple times in `subj` only the last entry is used.

- MSK_putcone

  Syntax:

  ```
  MSKrescodee MSK_putcone (
      MSKtask_t task,
      MSKidxt k,
      MSKconetypee conetype,
      MSKrealt conepar,
      MSKintt nummem,
      MSKCONST MSKidxt * submem);
  ```

  task **(input)** An optimization task.

  k **(input)** Index of the cone.

  conetype **(input)** Specifies the type of the cone.

  conepar **(input)** This argument is currently not used. Can be set to 0.0.

  nummem **(input)** Number of member variables in the cone.

  submem **(input)** Variable subscripts of the members in the cone.

  Description: Replaces a conic constraint.

- MSK_putdouparam

  Syntax:

  ```
  MSKrescodee MSK_putdouparam (
      MSKtask_t task,
      MSKdparame param,
      MSKrealt parvalue);
  ```

  task **(input)** An optimization task.

  param **(input)** Which parameter.

  parvalue **(input)** Parameter value.

  Description: Sets the value of a double parameter.

- MSK_putintparam

  Syntax:

  ```
  MSKrescodee MSK_putintparam (
      MSKtask_t task,
      MSKiparame param,
      MSKintt parvalue);
  ```

  task **(input)** An optimization task.

  param **(input)** Which parameter.

  parvalue **(input)** Parameter value.

  Description: Sets the value of an integer parameter.

- MSK_putmaxnumanz

Syntax:

```
MSKrescodee MSK_putmaxnumanz (
    MSKtask_t task,
    MSKintt maxnumanz);
```

`task` **(input)** An optimization task.

`maxnumanz` **(input)** New size of the storage reserved for storing A.

Description: MOSEK stores only the non-zero elements in A. Therefore, MOSEK cannot predict how much storage is required to store A. Using this function it is possible to specify the number of non-zeros to preallocate for storing A.

If the number of non-zeros in the problem is known, it is a good idea to set `maxnumanz` slightly larger than this number, otherwise a rough estimate can be used. In general, if A is inputted in many small chunks, setting this value may speed up the the data input phase.

It is not mandatory to call this function, since MOSEK will reallocate internal structures whenever it is necessary.

See also:

MSK_IINF_STO_NUM_A_REALLOC

- `MSK_putmaxnumanz64`

Syntax:

```
MSKrescodee MSK_putmaxnumanz64 (
    MSKtask_t task,
    MSKint64t maxnumanz);
```

`task` **(input)** An optimization task.

`maxnumanz` **(input)** New size of the storage reserved for storing A.

Description: The function changes the size of the preallocated storage for linear coefficients.

See also:

MSK_IINF_STO_NUM_A_REALLOC

- `MSK_putmaxnumcon`

Syntax:

```
MSKrescodee MSK_putmaxnumcon (
    MSKtask_t task,
    MSKintt maxnumcon);
```

`task` **(input)** An optimization task.

`maxnumcon` **(input)** Number of preallocated constraints in the optimization task.

Description: Sets the number of preallocated constraints in the optimization task. When this number of constraints is reached MOSEK will automatically allocate more space for constraints.

It is never mandatory to call this function, since MOSEK will reallocate any internal structures whenever it is required.

Please note that `maxnumcon` must be larger than the current number of constraints in the task.

- `MSK_putmaxnumcone`

  Syntax:

  ```
  MSKrescodee MSK_putmaxnumcone (
      MSKtask_t task,
      MSKintt maxnumcone);
  ```

  `task` **(input)** An optimization task.

  `maxnumcone` **(input)** Number of preallocated conic constraints in the optimization task.

  Description: Sets the number of preallocated conic constraints in the optimization task. When this number of conic constraints is reached MOSEK will automatically allocate more space for conic constraints.

  It is never mandatory to call this function, since MOSEK will reallocate any internal structures whenever it is required.

  Please note that `maxnumcon` must be larger than the current number of constraints in the task.

- `MSK_putmaxnumqnz`

  Syntax:

  ```
  MSKrescodee MSK_putmaxnumqnz (
      MSKtask_t task,
      MSKintt maxnumqnz);
  ```

  `task` **(input)** An optimization task.

  `maxnumqnz` **(input)** Number of non-zero elements preallocated in quadratic coefficient matrices.

  Description: MOSEK stores only the non-zero elements in Q. Therefore, MOSEK cannot predict how much storage is required to store Q. Using this function it is possible to specify the number non-zeros to preallocate for storing Q (both objective and constraints).

  It may be advantageous to reserve more non-zeros for Q than actually needed since it may improve the internal efficiency of MOSEK, however, it is never worthwhile to specify more than the double of the anticipated number of non-zeros in Q.

  It is never mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

- `MSK_putmaxnumqnz64`

  Syntax:

  ```
  MSKrescodee MSK_putmaxnumqnz64 (
      MSKtask_t task,
      MSKint64t maxnumqnz);
  ```

  `task` **(input)** An optimization task.

  `maxnumqnz` **(input)** Number of non-zero elements preallocated in quadratic coefficient matrices.

Description: MOSEK stores only the non-zero elements in Q. Therefore, MOSEK cannot predict how much storage is required to store Q. Using this function it is possible to specify the number non-zeros to preallocate for storing Q (both objective and constraints).

It may be advantageous to reserve more non-zeros for Q than actually needed since it may improve the internal efficiency of MOSEK, however, it is never worthwhile to specify more than the double of the anticipated number of non-zeros in Q.

It is never mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

- `MSK_putmaxnumvar`

  Syntax:

  ```
  MSKrescodee MSK_putmaxnumvar (
      MSKtask_t task,
      MSKintt maxnumvar);
  ```

  `task` **(input)** An optimization task.

  `maxnumvar` **(input)** Number of preallocated variables in the optimization task.

  Description: Sets the number of preallocated variables in the optimization task. When this number of variables is reached MOSEK will automatically allocate more space for variables.

  It is never mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

  Please note that `maxnumvar` must be larger than the current number of variables in the task.

- `MSK_putnadouparam`

  Syntax:

  ```
  MSKrescodee MSK_putnadouparam (
      MSKtask_t task,
      MSKCONST char * paramname,
      MSKrealt parvalue);
  ```

  `task` **(input)** An optimization task.

  `paramname` **(input)** Name of a parameter.

  `parvalue` **(input)** Parameter value.

  Description: Sets the value of a named double parameter.

- `MSK_putnaintparam`

  Syntax:

  ```
  MSKrescodee MSK_putnaintparam (
      MSKtask_t task,
      MSKCONST char * paramname,
      MSKintt parvalue);
  ```

  `task` **(input)** An optimization task.

  `paramname` **(input)** Name of a parameter.

`parvalue` **(input)** Parameter value.

Description: Sets the value of a named integer parameter.

- `MSK_putname`

  Syntax:

  ```
  MSKrescodee MSK_putname (
      MSKtask_t task,
      MSKproblemiteme whichitem,
      MSKidxt i,
      MSKCONST char * name);
  ```

  `task` **(input)** An optimization task.

  `whichitem` **(input)** Problem item, i.e. a cone, a variable or a constraint name..

  `i` **(input)** Index.

  `name` **(input)** New name to be assigned to the item.

  Description: Assigns the name defined by `name` to a problem item (a variable, a constraint or a cone).

- `MSK_putnastrparam`

  Syntax:

  ```
  MSKrescodee MSK_putnastrparam (
      MSKtask_t task,
      MSKCONST char * paramname,
      MSKCONST char * parvalue);
  ```

  `task` **(input)** An optimization task.

  `paramname` **(input)** Name of a parameter.

  `parvalue` **(input)** Parameter value.

  Description: Sets the value of a named string parameter.

- `MSK_putnlfunc`

  Syntax:

  ```
  MSKrescodee MSK_putnlfunc (
      MSKtask_t task,
      MSKuserhandle_t nlhandle,
      MSKnlgetspfunc nlgetsp,
      MSKnlgetvafunc nlgetva);
  ```

  `task` **(input)** An optimization task.

  `nlhandle` **(input)** A pointer to a user-defined data structure. It is passed to the functions `nlgetsp` and `nlgetva` whenever those two functions called.

  `nlgetsp` **(input)** A user-defined function which provide information about the structure of the nonlinear functions in the optimization problem.

`nlgetva` **(input)** A user-defined function which is used to evaluate the nonlinear function in the optimization problem at a given point.

Description: This function is used to communicate the nonlinear function information to MOSEK.

- `MSK_putobjname`

  Syntax:

  ```
  MSKrescodee MSK_putobjname (
      MSKtask_t task,
      MSKCONST char * objname);
  ```

  `task` **(input)** An optimization task.

  `objname` **(input)** Name of the objective.

  Description: Assigns the name given by `objname` to the objective function.

- `MSK_putobjsense`

  Syntax:

  ```
  MSKrescodee MSK_putobjsense (
      MSKtask_t task,
      MSKobjsensee sense);
  ```

  `task` **(input)** An optimization task.

  `sense` **(input)** The objective sense of the task. The values `MSK_OBJECTIVE_SENSE_MAXIMIZE` and `MSK_OBJECTIVE_SENSE_MINIMIZE` means that the the problem is maximized or minimized respectively. The value `MSK_OBJECTIVE_SENSE_UNDEFINED` means that the objective sense is taken from the parameter `MSK_IPAR_OBJECTIVE_SENSE`.

  Description: Sets the objective sense of the task.

  See also:

  `MSK_getobjsense` Gets the objective sense.

- `MSK_putparam`

  Syntax:

  ```
  MSKrescodee MSK_putparam (
      MSKtask_t task,
      MSKCONST char * parname,
      MSKCONST char * parvalue);
  ```

  `task` **(input)** An optimization task.

  `parname` **(input)** Parameter name.

  `parvalue` **(input)** Parameter value.

  Description: Checks if a `parname` is valid parameter name. If it is, the parameter is assigned the value specified by `parvalue`.

- MSK_putqcon

  Syntax:

  ```
  MSKrescodee MSK_putqcon (
      MSKtask_t task,
      MSKintt numqcnz,
      MSKCONST MSKidxt * qcsubk,
      MSKCONST MSKidxt * qcsubi,
      MSKCONST MSKidxt * qcsubj,
      MSKCONST MSKrealt * qcval);
  ```

  task (input) An optimization task.

  numqcnz (input) Number of quadratic terms. See (5.32).

  qcsubk (input) k subscripts for q_{ij}^k. See (5.32).

  qcsubi (input) i subscripts for q_{ij}^k. See (5.32).

  qcsubj (input) j subscripts for q_{ij}^k. See (5.32).

  qcval (input) Numerical value for q_{ij}^k.

  Description: Replaces all quadratic entries in the constraints. Consider constraints on the form:

  $$l_k^c \leq \frac{1}{2} \sum_{i=0}^{\texttt{numvar}-1} \sum_{j=0}^{\texttt{numvar}-1} q_{ij}^k x_i x_j + \sum_{j=0}^{\texttt{numvar}-1} a_{kj} x_j \leq u_k^c, \ k = 0, \ldots, m-1. \tag{15.29}$$

  The function assigns values to q such that:

  $$q^{\texttt{qcsubk[t]}}_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} = \texttt{qcval[t]}, \ t = 0, \ldots, \texttt{numqcnz} - 1. \tag{15.30}$$

  and

  $$q^{\texttt{qcsubk[t]}}_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} = \texttt{qcval[t]}, \ t = 0, \ldots, \texttt{numqcnz} - 1. \tag{15.31}$$

  Values not assigned are set to zero.

  Please note that duplicate entries are added together.

  See also:

  MSK_putqconk Replaces all quadratic terms in a single constraint.

  MSK_putmaxnumqnz Changes the size of the preallocated storage for quadratic terms.

- MSK_putqconk

  Syntax:

  ```
  MSKrescodee MSK_putqconk (
      MSKtask_t task,
      MSKidxt k,
      MSKintt numqcnz,
      MSKCONST MSKidxt * qcsubi,
      MSKCONST MSKintt * qcsubj,
      MSKCONST MSKrealt * qcval);
  ```

`task` **(input)** An optimization task.

`k` **(input)** The constraint in which the new Q elements are inserted.

`numqcnz` **(input)** Number of quadratic terms. See (5.32).

`qcsubi` **(input)** i subscripts for q_{ij}^k. See (5.32).

`qcsubj` **(input)** j subscripts for q_{ij}^k. See (5.32).

`qcval` **(input)** Numerical value for q_{ij}^k.

Description: Replaces all the quadratic entries in one constraint k of the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{\texttt{numvar}-1} \sum_{j=0}^{\texttt{numvar}-1} q_{ij}^k x_i x_j + \sum_{j=0}^{\texttt{numvar}-1} a_{kj} x_j \leq u_k^c. \tag{15.32}$$

It is assumed that Q^k is symmetric, i.e. $q_{ij}^k = q_{ji}^k$, and therefore, only the values of q_{ij}^k for which $i \geq j$ should be inputted to MOSEK. To be precise, MOSEK uses the following procedure

1. $Q^k = 0$
2. for $t = 0$ to `numqonz` $- 1$
3. $\quad q^k_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} = q^k_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} + \texttt{qcval[t]}$
3. $\quad q^k_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} = q^k_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} + \texttt{qcval[t]}$

Please note that:

- For large problems it is essential for the efficiency that the function MSK_putmaxnumqnz64 is employed to specify an appropriate `maxnumqnz`.
- Only the lower triangular part should be specified because Q^k is symmetric. Specifying values for q_{ij}^k where $i < j$ will result in an error.
- Only non-zero elements should be specified.
- The order in which the non-zero elements are specified is insignificant.
- Duplicate elements are added together. Hence, it is recommended not to specify the same element multiple times in `qosubi`, `qosubj`, and `qoval`.

For a code example see Section 5.3.2.

See also:

MSK_putqcon Replaces all quadratic terms in constraints.

MSK_putmaxnumqnz Changes the size of the preallocated storage for quadratic terms.

- MSK_putqobj

Syntax:

```
MSKrescodee MSK_putqobj (
    MSKtask_t task,
    MSKintt numqonz,
    MSKCONST MSKidxt * qosubi,
    MSKCONST MSKidxt * qosubj,
    MSKCONST MSKrealt * qoval);
```

`task` **(input)** An optimization task.

`numqonz` **(input)** Number of non-zero elements in Q^o.

`qosubi` **(input)** i subscript for q^o_{ij}.

`qosubj` **(input)** j subscript for q^o_{ij}.

`qoval` **(input)** Numerical value for q^o_{ij}.

Description: Replaces all the quadratic terms in the objective

$$\frac{1}{2}\sum_{i=0}^{\texttt{numvar}-1}\sum_{j=0}^{\texttt{numvar}-1} q^o_{ij}x_ix_j + \sum_{j=0}^{\texttt{numvar}-1} c_jx_j + c^f. \tag{15.33}$$

It is assumed that Q^o is symmetric, i.e. $q^o_{ij} = q^o_{ji}$, and therefore, only the values of q^o_{ij} for which $i \geq j$ should be specified. To be precise, MOSEK uses the following procedure

1. $Q^o = 0$
2. for $t = 0$ to $\texttt{numqonz} - 1$
3. $\quad q^o_{\texttt{qosubi[t]},\texttt{qosubj[t]}} = q^o_{\texttt{qosubi[t]},\texttt{qosubj[t]}} + \texttt{qoval[t]}$
3. $\quad q^o_{\texttt{qosubj[t]},\texttt{qosubi[t]}} = q^o_{\texttt{qosubj[t]},\texttt{qosubi[t]}} + \texttt{qoval[t]}$

Please note that:

- Only the lower triangular part should be specified because Q^o is symmetric. Specifying values for q^o_{ij} where $i < j$ will result in an error.
- Only non-zero elements should be specified.
- The order in which the non-zero elements are specified is insignificant.
- Duplicate entries are added to together.

For a code example see Section 5.3.1.

- `MSK_putqobjij`

Syntax:

```
MSKrescodee MSK_putqobjij (
    MSKtask_t task,
    MSKidxt i,
    MSKidxt j,
    MSKrealt qoij);
```

`task` **(input)** An optimization task.

`i` **(input)** Row index for the coefficient to be replaced.

`j` **(input)** Column index for the coefficient to be replaced.

`qoij` **(input)** The new value for q^o_{ij}.

Description: Replaces one coefficient in the quadratic term in the objective. The function performs the assignment

$$q^o_{\texttt{ij}} = \texttt{qoij}.$$

Only the elements in the lower triangular part are accepted. Setting q_{ij} with $j > i$ will cause an error.

Please note that replacing all quadratic element, one at a time, is more computationally expensive than replacing all elements at once. Use `MSK_putqobj` instead whenever possible.

- MSK_putresponsefunc

  Syntax:

  ```
  MSKrescodee MSK_putresponsefunc (
      MSKtask_t task,
      MSKresponsefunc responsefunc,
      MSKuserhandle_t handle);
  ```

  task **(input)** An optimization task.

  responsefunc **(input)** A user-defined response handling function.

  handle **(input)** A user-defined data structure that is passed to the function responsefunc whenever it is called.

  Description: Inputs a user-defined error call-back which is called when an error or warning occurs.

- MSK_putsolution

  Syntax:

  ```
  MSKrescodee MSK_putsolution (
      MSKtask_t task,
      MSKsoltypee whichsol,
      MSKCONST MSKstakeye * skc,
      MSKCONST MSKstakeye * skx,
      MSKCONST MSKstakeye * skn,
      MSKCONST MSKrealt * xc,
      MSKCONST MSKrealt * xx,
      MSKCONST MSKrealt * y,
      MSKCONST MSKrealt * slc,
      MSKCONST MSKrealt * suc,
      MSKCONST MSKrealt * slx,
      MSKCONST MSKrealt * sux,
      MSKCONST MSKrealt * snx);
  ```

  task **(input)** An optimization task.

  whichsol **(input)** Selects a solution.

  skc **(input)** Status keys for the constraints.

  skx **(input)** Status keys for the variables.

  skn **(input)** Status keys for the conic constraints.

  xc **(input)** Primal constraint solution.

  xx **(input)** Primal variable solution (x).

  y **(input)** Vector of dual variables corresponding to the constraints.

  slc **(input)** Dual variables corresponding to the lower bounds on the constraints (s_l^c).

  suc **(input)** Dual variables corresponding to the upper bounds on the constraints (s_u^c).

  slx **(input)** Dual variables corresponding to the lower bounds on the variables (s_l^x).

  sux **(input)** Dual variables corresponding to the upper bounds on the variables (appears as s_u^x).

snx **(input)** Dual variables corresponding to the conic constraints on the variables (s_n^x).

Description: Inserts a solution into the task.

- MSK_putsolutioni

  Syntax:

  ```
  MSKrescodee MSK_putsolutioni (
      MSKtask_t task,
      MSKaccmodee accmode,
      MSKidxt i,
      MSKsoltypee whichsol,
      MSKstakeye sk,
      MSKrealt x,
      MSKrealt sl,
      MSKrealt su,
      MSKrealt sn);
  ```

  task **(input)** An optimization task.

  accmode **(input)** If set to MSK_ACC_CON the solution information for a constraint is modified. Otherwise for a variable.

  i **(input)** Index of the constraint or variable.

  whichsol **(input)** Selects a solution.

  sk **(input)** Status key of the constraint or variable.

  x **(input)** Solution value of the primal constraint or variable.

  sl **(input)** Solution value of the dual variable associated with the lower bound.

  su **(input)** Solution value of the dual variable associated with the upper bound.

  sn **(input)** Solution value of the dual variable associated with the cone constraint.

  Description: Sets the primal and dual solution information for a single constraint or variable.

  To define a solution or a significant part of a solution, first call the MSK_makesolutionstatusunknown function, then for each relevant constraint and variable call MSK_putsolutioni to set the solution information.

  See also:

  MSK_makesolutionstatusunknown Sets the solution status to unknown.

- MSK_putsolutionyi

  Syntax:

  ```
  MSKrescodee MSK_putsolutionyi (
      MSKtask_t task,
      MSKidxt i,
      MSKsoltypee whichsol,
      MSKrealt y);
  ```

  task **(input)** An optimization task.

  i **(input)** Index of the dual variable.

whichsol **(input)** Selects a solution.

y **(input)** Solution value of the dual variable.

Description: Inputs the dual variable of a solution.

See also:

MSK_makesolutionstatusunknown Sets the solution status to unknown.

MSK_putsolutioni Sets the primal and dual solution information for a single constraint or variable.

- MSK_putstrparam

  Syntax:

  ```
  MSKrescodee MSK_putstrparam (
      MSKtask_t task,
      MSKsparame param,
      MSKCONST char * parvalue);
  ```

  task **(input)** An optimization task.

  param **(input)** Which parameter.

  parvalue **(input)** Parameter value.

  Description: Sets the value of a string parameter.

- MSK_puttaskname

  Syntax:

  ```
  MSKrescodee MSK_puttaskname (
      MSKtask_t task,
      MSKCONST char * taskname);
  ```

  task **(input)** An optimization task.

  taskname **(input)** Name assigned to the task.

  Description: Assigns the name taskname to the task.

- MSK_putvarbranchorder

  Syntax:

  ```
  MSKrescodee MSK_putvarbranchorder (
      MSKtask_t task,
      MSKidxt j,
      MSKintt priority,
      int direction);
  ```

  task **(input)** An optimization task.

  j **(input)** Index of the variable.

  priority **(input)** The branching priority that should be assigned to variable j.

  direction **(input)** Specifies the preferred branching direction for variable j.

Description: The purpose of the function is to assign a branching priority and direction. The higher priority that is assigned to an integer variable the earlier the mixed integer optimizer will branch on the variable. The branching direction controls if the optimizer branches up or down on the variable.

- `MSK_putvartype`

  Syntax:

  ```
  MSKrescodee MSK_putvartype (
      MSKtask_t task,
      MSKidxt j,
      MSKvariabletypee vartype);
  ```

  `task` **(input)** An optimization task.

  `j` **(input)** Index of the variable.

  `vartype` **(input)** The new variable type.

  Description: Sets the variable type of one variable.

  See also:

  `MSK_putvartypelist` Sets the variable type for one or more variables.

- `MSK_putvartypelist`

  Syntax:

  ```
  MSKrescodee MSK_putvartypelist (
      MSKtask_t task,
      MSKintt num,
      MSKCONST MSKidxt * subj,
      MSKCONST MSKvariabletypee * vartype);
  ```

  `task` **(input)** An optimization task.

  `num` **(input)** Number of variables for which the variable type should be set.

  `subj` **(input)** A list of variable indexes for which the variable type should be changed.

  `vartype` **(input)** A list of variable types that should be assigned to the variables specified by `subj`. See section 18.55 for the possible values of `vartype`.

  Description: Sets the variable type for one or more variables, i.e. variable number `subj`$[k]$ is assigned the variable type `vartype`$[k]$.

  If the same index is specified multiple times in `subj` only the last entry takes effect.

  See also:

  `MSK_putvartype` Sets the variable type of one variable.

- `MSK_readbranchpriorities`

  Syntax:

  ```
  MSKrescodee MSK_readbranchpriorities (
      MSKtask_t task,
      MSKCONST char * filename);
  ```

`task` **(input)** An optimization task.

`filename` **(input)** Data is read from the file `filename`.

Description: Reads branching priority data from a file.

See also:

MSK_writebranchpriorities Writes branching priority data to a file.

- `MSK_readdata`

  Syntax:

  ```
  MSKrescodee MSK_readdata (
      MSKtask_t task,
      MSKCONST char * filename);
  ```

  `task` **(input)** An optimization task.

  `filename` **(input)** Data is read from the file `filename` if it is a nonempty string. Otherwise data is read from the file specified by MSK_SPAR_DATA_FILE_NAME.

  Description: Reads an optimization data and associated data from a file.

  The data file format is determined by the MSK_IPAR_READ_DATA_FORMAT parameter. By default the parameter has the value MSK_DATA_FORMAT_EXTENSION indicating that the extension of the input file should determine the file type, where the extension is interpreted as follows:

  – "`.lp`" and "`.lp.gz`" are interpreted as an LP file and a compressed LP file respectively.
  – "`.opf`" and "`.opf.gz`" are interpreted as an OPF file and a compressed OPF file respectively.
  – "`.mps`" and "`.mps.gz`" are interpreted as an MPS file and a compressed MPS file respectively.
  – "`.mbt`" and "`.mbt.gz`" are interpreted as an MBT file and a compressed MBT file respectively.

  See also:

  MSK_writedata Writes problem data to a file.
  MSK_IPAR_READ_DATA_FORMAT

- `MSK_readparamfile`

  Syntax:

  ```
  MSKrescodee MSK_readparamfile (MSKtask_t task)
  ```

  `task` **(input)** An optimization task.

  Description: Reads a parameter file.

- `MSK_readsolution`

  Syntax:

  ```
  MSKrescodee MSK_readsolution (
      MSKtask_t task,
      MSKsoltypee whichsol,
      MSKCONST char * filename);
  ```

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

`filename` **(input)** A valid file name.

Description: Reads a solution file and inserts the solution into the solution `whichsol`.

- `MSK_readsummary`

  Syntax:

```
MSKrescodee MSK_readsummary (
    MSKtask_t task,
    MSKstreamtypee whichstream);
```

  `task` **(input)** An optimization task.

  `whichstream` **(input)** Index of the stream.

  Description: Prints a short summary of last file that was read.

- `MSK_relaxprimal`

  Syntax:

```
MSKrescodee MSK_relaxprimal (
    MSKtask_t task,
    MSKtask_t * relaxedtask,
    MSKrealt * wlc,
    MSKrealt * wuc,
    MSKrealt * wlx,
    MSKrealt * wux);
```

  `task` **(input)** An optimization task.

  `relaxedtask` **(output)** The returned task.

  `wlc` **(input/output)** Weights associated with lower bounds on the activity of constraints. If negative, the bound is strictly enforced, i.e. if $(w_l^c)_i < 0$, then $(v_l^c)_i$ is fixed to zero. On return `wlc[i]` contains the relaxed bound.

  `wuc` **(input/output)** Weights associated with upper bounds on the activity of constraints. If negative, the bound is strictly enforced, i.e. if $(w_u^c)_i < 0$, then $(v_u^c)_i$ is fixed to zero. On return `wuc[i]` contains the relaxed bound.

  `wlx` **(input/output)** Weights associated with lower bounds on the activity of variables. If negative, the bound is strictly enforced, i.e. if $(w_l^x)_j < 0$ then $(v_l^x)_j$ is fixed to zero. On return `wlx[i]` contains the relaxed bound.

  `wux` **(input/output)** Weights associated with upper bounds on the activity of variables. If negative, the bound is strictly enforced, i.e. if $(w_u^x)_j < 0$ then $(v_u^x)_j$ is fixed to zero. On return `wux[i]` contains the relaxed bound.

  Description: Creates a problem that computes the minimal (weighted) relaxation of the bounds that will make an infeasible problem feasible.

  Given an existing task describing the problem

$$\begin{array}{lccccc} \text{minimize} & & & c^T x & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \tag{15.34}$$

the function forms a new task `relaxedtask` having the form

$$\begin{array}{lrcc} \text{minimize} & & p & & \\ \text{subject to} & l^c & \leq Ax + v_l^c - v_u^c & \leq & u^c, \\ & l^x & \leq x + v_l^x - v_u^x & \leq & u^x, \\ & & (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq & 0, \\ & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0. & & \end{array} \tag{15.35}$$

Hence, the function adds so-called elasticity variables to all the constraints which relax the constraints, for instance $(v_l^c)_i$ and $(v_u^c)_i$ relax $(l^c)_i$ and $(u^c)_i$ respectively. It should be obvious that (15.35) is feasible. Moreover, the function adds the constraint

$$(w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p \leq 0$$

to the problem which makes the variable p bigger than the total weighted sum of the relaxation to the bounds. w_l^c, w_u^c, w_l^x and w_u^x are user-defined weights which normally should be nonnegative. If a weight is negative, then the corresponding elasticity variable is fixed to zero.

Hence, when the problem (15.35) is optimized, the weighted minimal change to the bounds such that the problem is feasible is computed.

One can specify that a bound should be strictly enforced by assigning a negative value to the corresponding weight, i.e if $(w_l^c)_i < 0$ then $(v_l^c)_i$ is fixed to zero.

Now let p^* be the optimal objective value to (15.35), then a natural thing to do is to solve the optimization problem

$$\begin{array}{lrcc} \text{minimize} & & c^T x & & \\ \text{subject to} & l^c & \leq Ax + v_l^c - v_u^c & \leq & u^c, \\ & l^x & \leq x + v_l^x - v_u^x & \leq & u^x, \\ & & (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq & 0, \\ & & p & = & p^*, \\ & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0, & & \end{array} \tag{15.36}$$

where the original objective function is minimized subject to the constraint that the total weighted relaxation is minimal.

The parameter `MSK_IPAR_FEASREPAIR_OPTIMIZE` controls whether the function returns the problem (15.35) or the problem (15.36).The parameter can take one of the following values.

`MSK_FEASREPAIR_OPTIMIZE_NONE` : The returned task `relaxedtask` contains problem (15.35) and is not optimized.

`MSK_FEASREPAIR_OPTIMIZE_PENALTY` : The returned task `relaxedtask` contains problem (15.35) and is optimized.

`MSK_FEASREPAIR_OPTIMIZE_COMBINED` : The returned task `relaxedtask` contains problem (15.36) and is optimized.

Please note that the v variables are appended to the x variables ordered as

$$(v_u^c)_1, (v_l^c)_1, (v_u^c)_2, (v_l^c)_2, \ldots, (v_u^c)_m, (v_l^c)_m, \quad (v_u^x)_1, (v_l^x)_1, (v_u^x)_2, (v_l^x)_2, \ldots, (v_u^x)_n, (v_l^x)_n$$

in the returned task.

If `NAME_CON` (`NAME_VAR`) is the name of the ith constraint (variable) then the new variables are named as follows:

- The variable corresponding to $(v_u^c)_i$ ($(v_u^x)_i$) is named "NAME_CON*up" ("NAME_VAR*up").
- The variable corresponding to $(v_l^c)_i$ ($(v_l^x)_i$) is named "NAME_CON*lo" ("NAME_VAR*lo").

where "*" can be replaced by a user-defined string by setting the MSK_SPAR_FEASREPAIR_NAME_SEPARATOR parameter.

Please note that if $u_i^c < l_i^c$ or $u_i^x < l_i^x$ then the feasibility repair problem becomes infeasible. Such trivial conflicts must therefore be removed manually before using MSK_relaxprimal.

The above discussion shows how the function works for a linear optimization problem. However, the function also works for quadratic and conic optimization problems but it cannot be used for general nonlinear optimization problems.

See also:

MSK_DPAR_FEASREPAIR_TOL

MSK_IPAR_FEASREPAIR_OPTIMIZE

MSK_SPAR_FEASREPAIR_NAME_SEPARATOR

MSK_SPAR_FEASREPAIR_NAME_PREFIX

- MSK_remove

Syntax:

```
MSKrescodee MSK_remove (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKintt num,
    MSKCONST MSKintt * sub);
```

task **(input)** An optimization task.

accmode **(input)** Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

num **(input)** Number of constraints or variables which should be removed.

sub **(input)** Indexes of constraints or variables which should be removed.

Description: The function removes a number of constraints or variables from the optimization task. This implies that the existing constraints and variables are renumbered, for instance if constraint 5 is removed then constraint 6 becomes constraint 5 and so forth.

See also:

MSK_append Appends a number of variables or constraints to the optimization task.

- MSK_removecone

Syntax:

```
MSKrescodee MSK_removecone (
    MSKtask_t task,
    MSKidxt k);
```

task **(input)** An optimization task.

k **(input)** Index of the conic constraint that should be removed.

Description: Removes a conic constraint from the problem. This implies that all the conic constraints appearing after cone number `k` are renumbered, decresing their indexes by one.

In general, it is much more efficient to remove a cone with a high index than a low index.

- `MSK_resizetask`

Syntax:

```
MSKrescodee MSK_resizetask (
    MSKtask_t task,
    MSKintt maxnumcon,
    MSKintt maxnumvar,
    MSKintt maxnumcone,
    MSKintt maxnumanz,
    MSKintt maxnumqnz);
```

`task` **(input)** An optimization task.

`maxnumcon` **(input)** New maximum number of constraints.

`maxnumvar` **(input)** New maximum number of variables.

`maxnumcone` **(input)** New maximum number of cones.

`maxnumanz` **(input)** New maximum number of non-zeros in A.

`maxnumqnz` **(input)** New maximum number of non-zeros in all Q matrices.

Description: Sets the amount of preallocated space assigned for each type of data in an optimization task.

It is never mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

Please note that the procedure is **destructive** in the sense that all existing data stored in the task is destroyed.

See also:

`MSK_putmaxnumvar` Sets the number of preallocated variables in the optimization task.

`MSK_putmaxnumcon` Sets the number of preallocated constraints in the optimization task.

`MSK_putmaxnumcone` Sets the number of preallocated conic constraints in the optimization task.

`MSK_putmaxnumanz` The function changes the size of the preallocated storage for linear coefficients.

`MSK_putmaxnumqnz` Changes the size of the preallocated storage for quadratic terms.

- `MSK_sensitivityreport`

Syntax:

```
MSKrescodee MSK_sensitivityreport (
    MSKtask_t task,
    MSKstreamtypee whichstream);
```

`task` **(input)** An optimization task.

`whichstream` **(input)** Index of the stream.

Description: Reads a sensitivity format file from a location given by MSK_SPAR_SENSITIVITY_FILE_NAME and writes the result to the stream `whichstream`. If MSK_SPAR_SENSITIVITY_RES_FILE_NAME is set to a non-empty string, then the sensitivity report is also written to a file of this name.

See also:

MSK_dualsensitivity Performs sensitivity analysis on objective coefficients.

MSK_primalsensitivity Perform sensitivity analysis on bounds.

MSK_IPAR_LOG_SENSITIVITY

MSK_IPAR_LOG_SENSITIVITY_OPT

MSK_IPAR_SENSITIVITY_TYPE

- `MSK_setdefaults`

  Syntax:

  ```
  MSKrescodee MSK_setdefaults (MSKtask_t task)
  ```

  `task` **(input)** An optimization task.

  Description: Resets all the parameters to their default values.

- `MSK_sktostr`

  Syntax:

  ```
  MSKrescodee MSK_sktostr (
      MSKtask_t task,
      MSKintt sk,
      char * str);
  ```

  `task` **(input)** An optimization task.
  `sk` **(input)** A valid status key.
  `str` **(output)** String corresponding to the status key `sk`.

  Description: Obtains an explanatory string corresponding to a status key.

- `MSK_solstatostr`

  Syntax:

  ```
  MSKrescodee MSK_solstatostr (
      MSKtask_t task,
      MSKsolstae solsta,
      char * str);
  ```

  `task` **(input)** An optimization task.
  `solsta` **(input)** Solution status.
  `str` **(output)** String corresponding to the solution status `solsta`.

  Description: Obtains an explanatory string corresponding to a solution status.

- `MSK_solutiondef`

  Syntax:

```
MSKrescodee MSK_solutiondef (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKintt * isdef);
```

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

`isdef` **(output)** Is non-zero if the requested solution is defined.

Description: Checks whether a solution is defined.

- `MSK_solutionsummary`

  Syntax:

```
MSKrescodee MSK_solutionsummary (
    MSKtask_t task,
    MSKstreamtypee whichstream);
```

  `task` **(input)** An optimization task.

  `whichstream` **(input)** Index of the stream.

  Description: Prints a short summary of the current solutions. Please see Section 8.7 for more details.

- `MSK_solvewithbasis`

  Syntax:

```
MSKrescodee MSK_solvewithbasis (
    MSKtask_t task,
    MSKintt transp,
    MSKintt * numnz,
    MSKidxt * sub,
    MSKrealt * val);
```

  `task` **(input)** An optimization task.

  `transp` **(input)** If this argument is non-zero, then (15.38) is solved. Otherwise the system (15.37) is solved.

  `numnz` **(input/output)** As input it is the number of non-zeros in b. As output it is the number of non-zeros in $\bar{x}$.

  `sub` **(input/output)** As input it contains the positions of the non-zeros in b, i.e.

$$b[\texttt{sub}[k]] \neq 0, \ \ k = 0, \ldots, \texttt{numnz}[0] - 1.$$

  As output it contains the positions of the non-zeros in $\bar{x}$. It is important that `sub` has room for `numcon` elements.

  `val` **(input/output)** As input it is the vector b. Although the positions of the non-zero elements are specified in `sub` it is required that $\texttt{val}[i] = 0$ if $b[i] = 0$. As output `val` is the vector $\bar{x}$.
  Please note that `val` is a dense vector — not a packed sparse vector. This implies that `val` has room for `numcon` elements.

Description: If a basic solution is available, then exactly `numcon` basis variables are defined. These `numcon` basis variables are denoted the basis. Associated with the basis is a basis matrix denoted B. This function solves either the linear equation system

$$B\bar{x} = b \tag{15.37}$$

or the system

$$B^T\bar{x} = b \tag{15.38}$$

for the unknowns $\bar{x}$, with b being a user-defined vector.

In order to make sense of the solution $\bar{x}$ it is important to know the ordering of the variables in the basis because the ordering specifies how B is constructed. When calling `MSK_initbasissolve` an ordering of the basis variables is obtained, whicd can be used to deduce how MOSEK has constructed B. Indeed if the kth basis variable is variable x_j it implies that

$$B_{i,k} = A_{i,j},\ i = 0,\ldots,\texttt{numcon} - 1.$$

Otherwise if the kth basis variable is variable x_j^c it implies that‘

$$B_{i,k} = \begin{cases} -1, & i = j, \\ 0, & i \neq j. \end{cases}$$

Given the knowledge of how B is constructed it is possible to interpret the solution $\bar{x}$ correctly.

Please note that this function exploits the sparsity in the vector b to speed up the computations.

See also:

`MSK_initbasissolve` Prepare a task for basis solver.

`MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE`

- `MSK_strdupdbgtask`

Syntax:

```
char * MSK_strdupdbgtask (
    MSKtask_t task,
    MSKCONST char * str,
    MSKCONST char * file,
    MSKCONST unsigned line);
```

`task` **(input)** An optimization task.

`str` **(input)** String that should be copied.

`file` **(input)** File from which the function is called.

`line` **(input)** Line in the file from which the function is called.

Description: Make a copy of a string. The string created by this procedure must be freed by `MSK_freetask`.

- `MSK_strduptask`

Syntax:

```
char * MSK_strduptask (
    MSKtask_t task,
    MSKCONST char * str);
```

`task` **(input)** An optimization task.

`str` **(input)** String that should be copied.

Description: Make a copy of a string. The string created by this procedure must be freed by `MSK_freetask`.

- `MSK_strtoconetype`

Syntax:

```
MSKrescodee MSK_strtoconetype (
    MSKtask_t task,
    MSKCONST char * str,
    MSKconetypee * conetype);
```

`task` **(input)** An optimization task.

`str` **(input)** String corresponding to the cone type code `codetype`.

`conetype` **(output)** The cone type corresponding to the string `str`.

Description: Obtains cone type code corresponding to a cone type string.

- `MSK_strtosk`

Syntax:

```
MSKrescodee MSK_strtosk (
    MSKtask_t task,
    MSKCONST char * str,
    MSKintt * sk);
```

`task` **(input)** An optimization task.

`str` **(input)** Status key string.

`sk` **(output)** Status key corresponding to the string.

Description: Obtains the status key corresponding to an explanatory string.

- `MSK_undefsolution`

Syntax:

```
MSKrescodee MSK_undefsolution (
    MSKtask_t task,
    MSKsoltypee whichsol);
```

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

Description: Undefines a solution. Purges all information regarding `whichsol`.

- MSK_unlinkfuncfromtaskstream

  Syntax:

  ```
  MSKrescodee MSK_unlinkfuncfromtaskstream (
      MSKtask_t task,
      MSKstreamtypee whichstream);
  ```

  task **(input)** An optimization task.

  whichstream **(input)** Index of the stream.

  Description: Disconnects a user-defined function from a task stream.

- MSK_whichparam

  Syntax:

  ```
  MSKrescodee MSK_whichparam (
      MSKtask_t task,
      MSKCONST char * parname,
      MSKparametertypee * partype,
      MSKintt * param);
  ```

  task **(input)** An optimization task.

  parname **(input)** Parameter name.

  partype **(output)** Parameter type.

  param **(output)** Which parameter.

  Description: Checks if parname is valid parameter name. If yes then, partype and param denotes the type and the index of parameter respectively.

- MSK_writebranchpriorities

  Syntax:

  ```
  MSKrescodee MSK_writebranchpriorities (
      MSKtask_t task,
      MSKCONST char * filename);
  ```

  task **(input)** An optimization task.

  filename **(input)** Data is written to the file filename.

  Description: Writes branching priority data to a file.

  See also:

  MSK_readbranchpriorities Reads branching priority data from a file.

- MSK_writedata

  Syntax:

  ```
  MSKrescodee MSK_writedata (
      MSKtask_t task,
      MSKCONST char * filename);
  ```

`task` **(input)** An optimization task.

`filename` **(input)** Data is written to the file `filename` if it is a nonempty string. Otherwise data is written to the file specified by MSK_SPAR_DATA_FILE_NAME.

Description: Writes problem data associated with the optimization task to a file in one of four formats:

LP : A text based row oriented format. File extension `.lp`. See Appendix D.

MPS : A text based column oriented format. File extension `.mps`. See Appendix C.

OPF : A text based row oriented format. File extension `.opf`. Supports more problem types than MPS and LP. See Appendix E.

MBT : A binary format for fast reading and writing. File extension `.mbt`.

By default the data file format is determined by the file name extension. This behaviour can be overridden by setting the MSK_IPAR_WRITE_DATA_FORMAT parameter.

MOSEK is able to read and write files in a compressed format (gzip). To write in the compressed format append the extension "`.gz`". E.g to write a gzip compressed MPS file use the extension `mps.gz`.

Please note that MPS, LP and OPF files require all variables to have unique names. If a task contains no names, it is possible to write the file with automaticly generated anonymous names by setting the MSK_IPAR_WRITE_GENERIC_NAMES parameter to MSK_ON.

See also:

MSK_readdata Reads problem data from a file.

MSK_IPAR_WRITE_DATA_FORMAT

- `MSK_writeparamfile`

  Syntax:

  ```
  MSKrescodee MSK_writeparamfile (
      MSKtask_t task,
      MSKCONST char * filename);
  ```

  `task` **(input)** An optimization task.

  `filename` **(input)** The name of parameter file.

  Description: Writes all the parameters to a parameter file.

- `MSK_writesolution`

  Syntax:

  ```
  MSKrescodee MSK_writesolution (
      MSKtask_t task,
      MSKsoltypee whichsol,
      MSKCONST char * filename);
  ```

  `task` **(input)** An optimization task.

  `whichsol` **(input)** Selects a solution.

  `filename` **(input)** A valid file name.

  Description: Saves the current basic, interior-point, or integer solution to a file.

Chapter 16

Parameter reference

16.1 Parameter groups

Parameters grouped by meaning and functionality.

16.1.1 Logging parameters.

16.1.2 Basis identification parameters.

- MSK_IPAR_BI_CLEAN_OPTIMIZER ... 457
  Controls which simplex optimizer is used in the clean-up phase.
- MSK_IPAR_BI_IGNORE_MAX_ITER ... 457
  Turns on basis identification in case the interior-point optimizer is terminated due to maximum number of iterations.
- MSK_IPAR_BI_IGNORE_NUM_ERROR ... 458
  Turns on basis identification in case the interior-point optimizer is terminated due to a numerical problem.
- MSK_IPAR_BI_MAX_ITERATIONS ... 458
  Maximum number of iterations after basis identification.
- MSK_IPAR_INTPNT_BASIS ... 463
  Controls whether basis identification is performed.
- MSK_IPAR_LOG_BI ... 470
  Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.
- MSK_IPAR_LOG_BI_FREQ ... 471
  Controls the logging frequency.
- MSK_DPAR_SIM_LU_TOL_REL_PIV ... 442
  Relative pivot tolerance employed when computing the LU factorization of the basis matrix.

16.1.3 The Interior-point method parameters.

Parameters defining the behavior of the interior-point method for linear, conic and convex problems.

- MSK_IPAR_BI_IGNORE_MAX_ITER ... 457
  Turns on basis identification in case the interior-point optimizer is terminated due to maximum number of iterations.
- MSK_IPAR_BI_IGNORE_NUM_ERROR ... 458
  Turns on basis identification in case the interior-point optimizer is terminated due to a numerical problem.
- MSK_DPAR_CHECK_CONVEXITY_REL_TOL ... 426
  Convexity check tolerance.
- MSK_IPAR_INTPNT_BASIS ... 463
  Controls whether basis identification is performed.
- MSK_DPAR_INTPNT_CO_TOL_DFEAS ... 429
  Dual feasibility tolerance used by the conic interior-point optimizer.

- MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL . . . 441
  This parameter determines when columns are dropped in incomplete cholesky factorization doing reformulation of quadratic problems.

- MSK_IPAR_QO_SEPARABLE_REFORMULATION . . . 492
  Determine if Quadratic programing problems should be reformulated to separable form.

16.1.4 Simplex optimizer parameters.

Parameters defining the behavior of the simplex optimizer for linear problems.

- MSK_DPAR_BASIS_REL_TOL_S . . . 425
  Maximum relative dual bound violation allowed in an optimal basic solution.

- MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE . . . 457
  Controls the sign of the columns in the basis matrix corresponding to slack variables.

- MSK_DPAR_BASIS_TOL_S . . . 425
  Maximum absolute dual bound violation in an optimal basic solution.

- MSK_DPAR_BASIS_TOL_X . . . 425
  Maximum absolute primal bound violation allowed in an optimal basic solution.

- MSK_IPAR_LOG_SIM . . . 476
  Controls the amount of log information from the simplex optimizers.

- MSK_IPAR_LOG_SIM_FREQ . . . 476
  Controls simplex logging frequency.

- MSK_IPAR_LOG_SIM_MINOR . . . 477
  Currently not in use.

- MSK_IPAR_SENSITIVITY_OPTIMIZER . . . 499
  Controls which optimizer is used for optimal partition sensitivity analysis.

- MSK_IPAR_SIM_BASIS_FACTOR_USE . . . 500
  Controls whether a (LU) factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penanlty.

- MSK_IPAR_SIM_DEGEN . . . 500
  Controls how aggressively degeneration is handled.

- MSK_IPAR_SIM_DUAL_PHASEONE_METHOD . . . 501
  An exprimental feature.

- MSK_IPAR_SIM_EXPLOIT_DUPVEC . . . 502
  Controls if the simplex optimizers are allowed to exploit duplicated columns.

16.1.5 Primal simplex optimizer parameters.

Parameters defining the behavior of the primal simplex optimizer for linear problems.

- MSK_IPAR_SIM_PRIMAL_CRASH 505
  Controls the simplex crash.
- MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION 505
  Controls how aggressively restricted selection is used.
- MSK_IPAR_SIM_PRIMAL_SELECTION 506
  Controls the primal simplex strategy.

16.1.6 Dual simplex optimizer parameters.

Parameters defining the behavior of the dual simplex optimizer for linear problems.

- MSK_IPAR_SIM_DUAL_CRASH 501
  Controls whether crashing is performed in the dual simplex optimizer.
- MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION 501
  Controls how aggressively restricted selection is used.
- MSK_IPAR_SIM_DUAL_SELECTION 501
  Controls the dual simplex strategy.

16.1.7 Network simplex optimizer parameters.

Parameters defining the behavior of the network simplex optimizer for linear problems.

- MSK_IPAR_LOG_SIM_NETWORK_FREQ 477
  Controls the network simplex logging frequency.
- MSK_IPAR_SIM_NETWORK_DETECT 504
  Level of aggressiveness of network detection.
- MSK_IPAR_SIM_NETWORK_DETECT_HOTSTART 504
  Level of aggressiveness of network detection in a simplex hot-start.
- MSK_IPAR_SIM_REFACTOR_FREQ 506
  Controls the basis refactoring frequency.

16.1.8 Nonlinear convex method parameters.

Parameters defining the behavior of the interior-point method for nonlinear convex problems.

- MSK_IPAR_CHECK_CONVEXITY . . . 459
  Specify the level of convexity check on quadratic problems
- MSK_DPAR_INTPNT_NL_MERIT_BAL . . . 431
  Controls if the complementarity and infeasibility is converging to zero at about equal rates.
- MSK_DPAR_INTPNT_NL_TOL_DFEAS . . . 431
  Dual feasibility tolerance used when a nonlinear model is solved.
- MSK_DPAR_INTPNT_NL_TOL_MU_RED . . . 431
  Relative complementarity gap tolerance.
- MSK_DPAR_INTPNT_NL_TOL_NEAR_REL . . . 431
  Nonlinear solver optimality tolerance parameter.
- MSK_DPAR_INTPNT_NL_TOL_PFEAS . . . 432
  Primal feasibility tolerance used when a nonlinear model is solved.
- MSK_DPAR_INTPNT_NL_TOL_REL_GAP . . . 432
  Relative gap termination tolerance for nonlinear problems.
- MSK_DPAR_INTPNT_NL_TOL_REL_STEP . . . 432
  Relative step size to the boundary for general nonlinear optimization problems.
- MSK_DPAR_INTPNT_TOL_INFEAS . . . 433
  Nonlinear solver infeasibility tolerance parameter.
- MSK_IPAR_LOG_CHECK_CONVEXITY . . . 471
  Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on.

  If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

16.1.9 The conic interior-point method parameters.

Parameters defining the behavior of the interior-point method for conic problems.

- MSK_DPAR_INTPNT_CO_TOL_DFEAS . . . 429
  Dual feasibility tolerance used by the conic interior-point optimizer.
- MSK_DPAR_INTPNT_CO_TOL_INFEAS . . . 429
  Infeasibility tolerance for the conic solver.
- MSK_DPAR_INTPNT_CO_TOL_MU_RED . . . 429
  Optimality tolerance for the conic solver.
- MSK_DPAR_INTPNT_CO_TOL_NEAR_REL . . . 430
  Optimality tolerance for the conic solver.

- MSK_DPAR_INTPNT_CO_TOL_PFEAS . . . 430
  Primal feasibility tolerance used by the conic interior-point optimizer.

- MSK_DPAR_INTPNT_CO_TOL_REL_GAP . . . 430
  Relative gap termination tolerance used by the conic interior-point optimizer.

16.1.10 The mixed-integer optimization parameters.

- MSK_IPAR_LOG_MIO . . . 473
  Controls the amount of log information from the mixed-integer optimizers.

- MSK_IPAR_LOG_MIO_FREQ . . . 474
  The mixed-integer solver logging frequency.

- MSK_IPAR_MIO_BRANCH_DIR . . . 478
  Controls whether the mixed-integer optimizer is branching up or down by default.

- MSK_IPAR_MIO_BRANCH_PRIORITIES_USE . . . 478
  Controls whether branching priorities are used by the mixed-integer optimizer.

- MSK_IPAR_MIO_CONSTRUCT_SOL . . . 478
  Controls if an initial mixed integer solution should be constructed from the values of the integer variables.

- MSK_IPAR_MIO_CONT_SOL . . . 479
  Controls the meaning of interior-point and basic solutions in mixed integer problems.

- MSK_IPAR_MIO_CUT_LEVEL_ROOT . . . 479
  Controls the cut level employed by the mixed-integer optimizer at the root node.

- MSK_IPAR_MIO_CUT_LEVEL_TREE . . . 480
  Controls the cut level employed by the mixed-integer optimizer in the tree.

- MSK_DPAR_MIO_DISABLE_TERM_TIME . . . 435
  Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

- MSK_IPAR_MIO_FEASPUMP_LEVEL . . . 480
  Controls the feasibility pump heuristic which is used to construct a good initial feasible solution.

- MSK_IPAR_MIO_HEURISTIC_LEVEL . . . 480
  Controls the heuristic employed by the mixed-integer optimizer to locate an initial integer feasible solution.

- MSK_DPAR_MIO_HEURISTIC_TIME . . . 436
  Time limit for the mixed-integer heuristics.

- MSK_IPAR_MIO_HOTSTART . . . 481
  Controls whether the integer optimizer is hot-started.

16.1.11 Presolve parameters.

16.1.12 Termination criterion parameters.

Parameters which define termination and optimality criteria and related information.

16.1.13 Progress call-back parameters.

- MSK_DPAR_CALLBACK_FREQ 425
  Controls progress call-back frequency.
- MSK_IPAR_SOLUTION_CALLBACK 510
  Indicates whether solution call-backs will be performed during the optimization.

16.1.14 Non-convex solver parameters.

- MSK_IPAR_LOG_NONCONVEX 474
  Controls amount of output printed by the nonconvex optimizer.
- MSK_IPAR_NONCONVEX_MAX_ITERATIONS 486
  Maximum number of iterations that can be used by the nonconvex optimizer.
- MSK_DPAR_NONCONVEX_TOL_FEAS 440
  Feasibility tolerance used by the nonconvex optimizer.
- MSK_DPAR_NONCONVEX_TOL_OPT 440
  Optimality tolerance used by the nonconvex optimizer.

16.1.15 Feasibility repair parameters.

- MSK_DPAR_FEASREPAIR_TOL 429
  Tolerance for constraint enforcing upper bound on sum of weighted violations in feasibility repair.

16.1.16 Optimization system parameters.

Parameters defining the overall solver system environment. This includes system and platform related information and behavior.

- MSK_IPAR_AUTO_UPDATE_SOL_INFO 456
  Controls whether the solution information items are automatically updated after an optimization is performed.
- MSK_IPAR_CACHE_LICENSE 458
  Control license caching.
- MSK_IPAR_CACHE_SIZE_L1 459
  Specifies the size of the level 1 cache of the processor.
- MSK_IPAR_CACHE_SIZE_L2 459
  Specifies the size of the level 2 cache of the processor.
- MSK_IPAR_CPU_TYPE 461
  Specifies the CPU type.

- MSK_IPAR_INTPNT_NUM_THREADS 466
  Controls the number of threads employed by the interior-point optimizer. If set to a positive number MOSEK will use this number of threads. If zero the number of threads used will equal the number of cores detected on the machine.

- MSK_IPAR_LICENSE_CACHE_TIME 468
  Setting this parameter no longer has any effect.

- MSK_IPAR_LICENSE_CHECK_TIME 469
  Controls the license manager client behavior.

- MSK_IPAR_LICENSE_WAIT 470
  Controls if MOSEK should queue for a license if none is available.

- MSK_IPAR_LOG_STORAGE 477
  Controls the memory related log information.

- MSK_IPAR_TIMING_LEVEL 511
  Controls the a amount of timing performed inside MOSEK.

16.1.17 Output information parameters.

- MSK_IPAR_INFEAS_REPORT_LEVEL 463
  Controls the contents of the infeasibility report.

- MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS 469
  Controls license manager client behavior.

- MSK_IPAR_LOG 470
  Controls the amount of log information.

- MSK_IPAR_LOG_BI 470
  Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.

- MSK_IPAR_LOG_BI_FREQ 471
  Controls the logging frequency.

- MSK_IPAR_LOG_CUT_SECOND_OPT 472
  Controls the reduction in the log levels for the second and any subsequent optimizations.

- MSK_IPAR_LOG_FACTOR 472
  If turned on, then the factor log lines are added to the log.

- MSK_IPAR_LOG_FEASREPAIR 472
  Controls the amount of output printed when performing feasibility repair.

- MSK_IPAR_LOG_FILE 472
  If turned on, then some log info is printed when a file is written or read.

- MSK_IPAR_MAX_NUM_WARNINGS 478
  Waning level. A higher value results in more warnings.
- MSK_IPAR_WARNING_LEVEL 511
  Warning level.

16.1.18 Extra information about the optimization problem.

- MSK_IPAR_OBJECTIVE_SENSE 486
  If the objective sense for the task is undefined, then the value of this parameter is used as the default objective sense.

16.1.19 Overall solver parameters.

- MSK_IPAR_BI_CLEAN_OPTIMIZER 457
  Controls which simplex optimizer is used in the clean-up phase.
- MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS 460
  The maximum number of simultaneous optimizations that will be started by the concurrent optimizer.
- MSK_IPAR_CONCURRENT_PRIORITY_DUAL_SIMPLEX 460
  Priority of the dual simplex algorithm when selecting solvers for concurrent optimization.
- MSK_IPAR_CONCURRENT_PRIORITY_FREE_SIMPLEX 460
  Priority of the free simplex optimizer when selecting solvers for concurrent optimization.
- MSK_IPAR_CONCURRENT_PRIORITY_INTPNT 461
  Priority of the interior-point algorithm when selecting solvers for concurrent optimization.
- MSK_IPAR_CONCURRENT_PRIORITY_PRIMAL_SIMPLEX 461
  Priority of the primal simplex algorithm when selecting solvers for concurrent optimization.
- MSK_IPAR_DATA_CHECK 462
  Enable data checking for debug purposes.
- MSK_IPAR_FEASREPAIR_OPTIMIZE 462
  Controls which type of feasibility analysis is to be performed.
- MSK_IPAR_INFEAS_PREFER_PRIMAL 463
  Controls which certificate is used if both primal- and dual- certificate of infeasibility is available.
- MSK_IPAR_LICENSE_WAIT 470
  Controls if MOSEK should queue for a license if none is available.
- MSK_IPAR_MIO_CONT_SOL 479
  Controls the meaning of interior-point and basic solutions in mixed integer problems.
- MSK_IPAR_MIO_LOCAL_BRANCH_NUMBER 481
  Controls the size of the local search space when doing local branching.

16.1.20 Behavior of the optimization task.

Parameters defining the behavior of an optimization task when loading data.

- MSK_IPAR_READ_ADD_VAR . . . 493
  Additional number of variables that is made room for in the problem.
- MSK_IPAR_READ_ANZ . . . 493
  Controls the expected number of constraint non-zeros.
- MSK_IPAR_READ_CON . . . 494
  Controls the expected number of constraints.
- MSK_IPAR_READ_CONE . . . 494
  Controls the expected number of conic constraints.
- MSK_IPAR_READ_QNZ . . . 498
  Controls the expected number of quadratic non-zeros.
- MSK_IPAR_READ_TASK_IGNORE_PARAM . . . 498
  Controls what information is used from the task files.
- MSK_IPAR_READ_VAR . . . 498
  Controls the expected number of variables.
- MSK_IPAR_WRITE_TASK_INC_SOL . . . 518
  Controls whether the solutions are stored in the task file too.

16.1.21 Data input/output parameters.

Parameters defining the behavior of data readers and writers.

- MSK_SPAR_BAS_SOL_FILE_NAME . . . 520
  Name of the bas solution file.
- MSK_SPAR_DATA_FILE_NAME . . . 520
  Data are read and written to this file.
- MSK_SPAR_DEBUG_FILE_NAME . . . 520
  MOSEK debug file.
- MSK_IPAR_INFEAS_REPORT_AUTO . . . 463
  Turns the feasibility report on or off.
- MSK_SPAR_INT_SOL_FILE_NAME . . . 521
  Name of the int solution file.
- MSK_SPAR_ITR_SOL_FILE_NAME . . . 522
  Name of the itr solution file.
- MSK_IPAR_LOG_FILE . . . 472
  If turned on, then some log info is printed when a file is written or read.
- MSK_IPAR_LP_WRITE_IGNORE_INCOMPATIBLE_ITEMS . . . 477
  Controls the result of writing a problem containing incompatible items to an LP file.

- MSK_SPAR_READ_MPS_RHS_NAME . . . 524
  Name of the RHS used. An empty name means that the first RHS vector is used.
- MSK_IPAR_READ_MPS_WIDTH . . . 497
  Controls the maximal number of characters allowed in one line of the MPS file.
- MSK_IPAR_READ_Q_MODE . . . 497
  Controls how the Q matrices are read from the MPS file.
- MSK_IPAR_READ_QNZ . . . 498
  Controls the expected number of quadratic non-zeros.
- MSK_IPAR_READ_TASK_IGNORE_PARAM . . . 498
  Controls what information is used from the task files.
- MSK_IPAR_READ_VAR . . . 498
  Controls the expected number of variables.
- MSK_SPAR_SENSITIVITY_FILE_NAME . . . 524
  Sensitivity report file name.
- MSK_SPAR_SENSITIVITY_RES_FILE_NAME . . . 524
  Name of the sensitivity report output file.
- MSK_SPAR_SOL_FILTER_XC_LOW . . . 524
  Solution file filter.
- MSK_SPAR_SOL_FILTER_XC_UPR . . . 525
  Solution file filter.
- MSK_SPAR_SOL_FILTER_XX_LOW . . . 525
  Solution file filter.
- MSK_SPAR_SOL_FILTER_XX_UPR . . . 525
  Solution file filter.
- MSK_IPAR_SOL_QUOTED_NAMES . . . 509
  Controls the solution file format.
- MSK_IPAR_SOL_READ_NAME_WIDTH . . . 510
  Controls the input solution file format.
- MSK_IPAR_SOL_READ_WIDTH . . . 510
  Controls the input solution file format.
- MSK_SPAR_STAT_FILE_NAME . . . 526
  Statistics file name.
- MSK_SPAR_STAT_KEY . . . 526
  Key used when writing the summary file.
- MSK_SPAR_STAT_NAME . . . 526
  Name used when writing the statistics file.

16.1.22 Analysis parameters.

Parameters controling the behaviour of the problem and solution analyzers.

16.1.23 Solution input/output parameters.

Parameters defining the behavior of solution reader and writer.

- MSK_IPAR_WRITE_SOL_CONSTRAINTS . . . 517
  Controls the solution file format.
- MSK_IPAR_WRITE_SOL_HEAD . . . 517
  Controls solution file format.
- MSK_IPAR_WRITE_SOL_VARIABLES . . . 518
  Controls the solution file format.

16.1.24 Infeasibility report parameters.

- MSK_IPAR_INFEAS_GENERIC_NAMES . . . 462
  Controls the contents of the infeasibility report.
- MSK_IPAR_INFEAS_REPORT_LEVEL . . . 463
  Controls the contents of the infeasibility report.
- MSK_IPAR_LOG_INFEAS_ANA . . . 473
  Controls log level for the infeasibility analyzer.

16.1.25 License manager parameters.

- MSK_IPAR_LICENSE_ALLOW_OVERUSE . . . 468
  Controls if license overuse is allowed when caching licenses
- MSK_IPAR_LICENSE_CACHE_TIME . . . 468
  Setting this parameter no longer has any effect.
- MSK_IPAR_LICENSE_CHECK_TIME . . . 469
  Controls the license manager client behavior.
- MSK_IPAR_LICENSE_DEBUG . . . 469
  Controls the license manager client debugging behavior.
- MSK_IPAR_LICENSE_PAUSE_TIME . . . 469
  Controls license manager client behavior.
- MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS . . . 469
  Controls license manager client behavior.
- MSK_IPAR_LICENSE_WAIT . . . 470
  Controls if MOSEK should queue for a license if none is available.

16.1.26 Data check parameters.

These parameters defines data checking settings and problem data tolerances, i.e. which values are rounded to 0 or infinity, and which values are large or small enough to produce a warning.

- MSK_IPAR_CHECK_CONVEXITY . . . 459
  Specify the level of convexity check on quadratic problems
- MSK_IPAR_CHECK_TASK_DATA . . . 460
  If this feature is turned on, then the task data is checked for bad values i.e. NaNs. before an optimization is performed.
- MSK_DPAR_DATA_TOL_AIJ . . . 426
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_AIJ_HUGE . . . 426
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_AIJ_LARGE . . . 427
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_BOUND_INF . . . 427
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_BOUND_WRN . . . 427
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_C_HUGE . . . 427
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_CJ_LARGE . . . 428
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_QIJ . . . 428
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_X . . . 428
  Data tolerance threshold.
- MSK_IPAR_LOG_CHECK_CONVEXITY . . . 471
  Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on.

  If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

16.1.27 Debugging parameters.

These parameters defines that can be used when debugging a problem.

- MSK_IPAR_AUTO_SORT_A_BEFORE_OPT . . . 456
  Controls whether the elements in each column of A are sorted before an optimization is performed.
- MSK_IPAR_CHECK_TASK_DATA . . . 460
  If this feature is turned on, then the task data is checked for bad values i.e. NaNs. before an optimization is performed.

16.2 Double parameters

Corresponding constant:
MSK_DPAR_ANA_SOL_INFEAS_TOL

Description:
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.

Possible Values:
Any number between 0.0 and +inf.

Default value:
+1e-8

- basis_rel_tol_s

Corresponding constant:
MSK_DPAR_BASIS_REL_TOL_S

Description:
Maximum relative dual bound violation allowed in an optimal basic solution.

Possible Values:
Any number between 0.0 and +inf.

Default value:
1.0e-12

- basis_tol_s

Corresponding constant:
MSK_DPAR_BASIS_TOL_S

Description:
Maximum absolute dual bound violation in an optimal basic solution.

Possible Values:
Any number between 1.0e-9 and +inf.

Default value:
1.0e-6

- basis_tol_x

Corresponding constant:
MSK_DPAR_BASIS_TOL_X

Description:
Maximum absolute primal bound violation allowed in an optimal basic solution.

Possible Values:
Any number between 1.0e-9 and +inf.

Default value:
1.0e-6

- callback_freq

Corresponding constant:
`MSK_DPAR_CALLBACK_FREQ`

Description:
Controls the time between calls to the progress call-back function. Hence, if the value of this parameter is for example 10, then the call-back is called approximately each 10 seconds. A negative value is equivalent to infinity.

In general frequent call-backs may hurt the performance.

Possible Values:
Any number between -inf and +inf.

Default value:
`-1.0`

- `check_convexity_rel_tol`

Corresponding constant:
`MSK_DPAR_CHECK_CONVEXITY_REL_TOL`

Description:
This parameter controls when the full convexity check declares a problem to be non-convex. Increasing this tolerance relaxes the criteria for declaring the problem non-convex.

A problem is declared non-convex if negative (positive) pivot elements are detected in the cholesky factor of a matrix which is required to be PSD (NSD). This parameter controles how much this non-negativity requirement may be violated.

If d_i is the pivot element for column i, then the matrix Q is considered to not be PSD if:

$$d_i \leq -|Q_{ii}| * \texttt{check\_convexity\_rel\_tol}$$

Possible Values:
Any number between 0 and +inf.

Default value:
`1e-10`

- `data_tol_aij`

Corresponding constant:
`MSK_DPAR_DATA_TOL_AIJ`

Description:
Absolute zero tolerance for elements in A. If any value A_{ij} is smaller than this parameter in absolute terms MOSEK will treat the values as zero and generate a warning.

Possible Values:
Any number between 1.0e-16 and 1.0e-6.

Default value:
`1.0e-12`

- `data_tol_aij_huge`

Corresponding constant:
MSK_DPAR_DATA_TOL_AIJ_HUGE

Description:
An element in A which is larger than this value in absolute size causes an error.

Possible Values:
Any number between 0.0 and +inf.

Default value:
1.0e20

- data_tol_aij_large

Corresponding constant:
MSK_DPAR_DATA_TOL_AIJ_LARGE

Description:
An element in A which is larger than this value in absolute size causes a warning message to be printed.

Possible Values:
Any number between 0.0 and +inf.

Default value:
1.0e10

- data_tol_bound_inf

Corresponding constant:
MSK_DPAR_DATA_TOL_BOUND_INF

Description:
Any bound which in absolute value is greater than this parameter is considered infinite.

Possible Values:
Any number between 0.0 and +inf.

Default value:
1.0e16

- data_tol_bound_wrn

Corresponding constant:
MSK_DPAR_DATA_TOL_BOUND_WRN

Description:
If a bound value is larger than this value in absolute size, then a warning message is issued.

Possible Values:
Any number between 0.0 and +inf.

Default value:
1.0e8

- data_tol_c_huge

Corresponding constant:
MSK_DPAR_DATA_TOL_C_HUGE

Description:
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.

Possible Values:
Any number between 0.0 and +inf.

Default value:
1.0e16

- data_tol_cj_large

Corresponding constant:
MSK_DPAR_DATA_TOL_CJ_LARGE

Description:
An element in c which is larger than this value in absolute terms causes a warning message to be printed.

Possible Values:
Any number between 0.0 and +inf.

Default value:
1.0e8

- data_tol_qij

Corresponding constant:
MSK_DPAR_DATA_TOL_QIJ

Description:
Absolute zero tolerance for elements in Q matrices.

Possible Values:
Any number between 0.0 and +inf.

Default value:
1.0e-16

- data_tol_x

Corresponding constant:
MSK_DPAR_DATA_TOL_X

Description:
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 lower bound is considered identical.

Possible Values:
Any number between 0.0 and +inf.

Default value:
1.0e-8

- `feasrepair_tol`

  Corresponding constant:
  `MSK_DPAR_FEASREPAIR_TOL`

  Description:
  Tolerance for constraint enforcing upper bound on sum of weighted violations in feasibility repair.

  Possible Values:
  Any number between 1.0e-16 and 1.0e+16.

  Default value:
  `1.0e-10`

- `intpnt_co_tol_dfeas`

  Corresponding constant:
  `MSK_DPAR_INTPNT_CO_TOL_DFEAS`

  Description:
  Dual feasibility tolerance used by the conic interior-point optimizer.

  Possible Values:
  Any number between 0.0 and 1.0.

  Default value:
  `1.0e-8`

  See also:
  `MSK_DPAR_INTPNT_CO_TOL_NEAR_REL` Optimality tolerance for the conic solver.

- `intpnt_co_tol_infeas`

  Corresponding constant:
  `MSK_DPAR_INTPNT_CO_TOL_INFEAS`

  Description:
  Controls when the conic interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

  Possible Values:
  Any number between 0.0 and 1.0.

  Default value:
  `1.0e-8`

- `intpnt_co_tol_mu_red`

  Corresponding constant:
  `MSK_DPAR_INTPNT_CO_TOL_MU_RED`

  Description:
  Relative complementarity gap tolerance feasibility tolerance used by the conic interior-point optimizer.

Possible Values:
Any number between 0.0 and 1.0.

Default value:
`1.0e-8`

- `intpnt_co_tol_near_rel`

Corresponding constant:
`MSK_DPAR_INTPNT_CO_TOL_NEAR_REL`

Description:
If MOSEK cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Possible Values:
Any number between 1.0 and +inf.

Default value:
`100`

- `intpnt_co_tol_pfeas`

Corresponding constant:
`MSK_DPAR_INTPNT_CO_TOL_PFEAS`

Description:
Primal feasibility tolerance used by the conic interior-point optimizer.

Possible Values:
Any number between 0.0 and 1.0.

Default value:
`1.0e-8`

See also:

`MSK_DPAR_INTPNT_CO_TOL_NEAR_REL` Optimality tolerance for the conic solver.

- `intpnt_co_tol_rel_gap`

Corresponding constant:
`MSK_DPAR_INTPNT_CO_TOL_REL_GAP`

Description:
Relative gap termination tolerance used by the conic interior-point optimizer.

Possible Values:
Any number between 0.0 and 1.0.

Default value:
`1.0e-8`

See also:

`MSK_DPAR_INTPNT_CO_TOL_NEAR_REL` Optimality tolerance for the conic solver.

- `intpnt_nl_merit_bal`

  Corresponding constant:
  `MSK_DPAR_INTPNT_NL_MERIT_BAL`

  Description:
  Controls if the complementarity and infeasibility is converging to zero at about equal rates.

  Possible Values:
  Any number between 0.0 and 0.99.

  Default value:
  `1.0e-4`

- `intpnt_nl_tol_dfeas`

  Corresponding constant:
  `MSK_DPAR_INTPNT_NL_TOL_DFEAS`

  Description:
  Dual feasibility tolerance used when a nonlinear model is solved.

  Possible Values:
  Any number between 0.0 and 1.0.

  Default value:
  `1.0e-8`

- `intpnt_nl_tol_mu_red`

  Corresponding constant:
  `MSK_DPAR_INTPNT_NL_TOL_MU_RED`

  Description:
  Relative complementarity gap tolerance.

  Possible Values:
  Any number between 0.0 and 1.0.

  Default value:
  `1.0e-12`

- `intpnt_nl_tol_near_rel`

  Corresponding constant:
  `MSK_DPAR_INTPNT_NL_TOL_NEAR_REL`

  Description:
  If the MOSEK nonlinear interior-point optimizer cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

  Possible Values:
  Any number between 1.0 and +inf.

Default value:
`1000.0`

- `intpnt_nl_tol_pfeas`

  Corresponding constant:
  `MSK_DPAR_INTPNT_NL_TOL_PFEAS`

  Description:
  Primal feasibility tolerance used when a nonlinear model is solved.

  Possible Values:
  Any number between 0.0 and 1.0.

  Default value:
  `1.0e-8`

- `intpnt_nl_tol_rel_gap`

  Corresponding constant:
  `MSK_DPAR_INTPNT_NL_TOL_REL_GAP`

  Description:
  Relative gap termination tolerance for nonlinear problems.

  Possible Values:
  Any number between 1.0e-14 and +inf.

  Default value:
  `1.0e-6`

- `intpnt_nl_tol_rel_step`

  Corresponding constant:
  `MSK_DPAR_INTPNT_NL_TOL_REL_STEP`

  Description:
  Relative step size to the boundary for general nonlinear optimization problems.

  Possible Values:
  Any number between 1.0e-4 and 0.9999999.

  Default value:
  `0.995`

- `intpnt_tol_dfeas`

  Corresponding constant:
  `MSK_DPAR_INTPNT_TOL_DFEAS`

  Description:
  Dual feasibility tolerance used for linear and quadratic optimization problems.

  Possible Values:
  Any number between 0.0 and 1.0.

  Default value:
  `1.0e-8`

- `intpnt_tol_dsafe`

  Corresponding constant:
  `MSK_DPAR_INTPNT_TOL_DSAFE`

  Description:
  Controls the initial dual starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly.

  Possible Values:
  Any number between 1.0e-4 and +inf.

  Default value:
  `1.0`

- `intpnt_tol_infeas`

  Corresponding constant:
  `MSK_DPAR_INTPNT_TOL_INFEAS`

  Description:
  Controls when the optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

  Possible Values:
  Any number between 0.0 and 1.0.

  Default value:
  `1.0e-8`

- `intpnt_tol_mu_red`

  Corresponding constant:
  `MSK_DPAR_INTPNT_TOL_MU_RED`

  Description:
  Relative complementarity gap tolerance.

  Possible Values:
  Any number between 0.0 and 1.0.

  Default value:
  `1.0e-16`

- `intpnt_tol_path`

  Corresponding constant:
  `MSK_DPAR_INTPNT_TOL_PATH`

  Description:
  Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central is followed very closely. On numerical unstable problems it may be worthwhile to increase this parameter.

  Possible Values:
  Any number between 0.0 and 0.9999.

Default value:
`1.0e-8`

- `intpnt_tol_pfeas`

  Corresponding constant:
  `MSK_DPAR_INTPNT_TOL_PFEAS`

  Description:
  Primal feasibility tolerance used for linear and quadratic optimization problems.

  Possible Values:
  Any number between 0.0 and 1.0.

  Default value:
  `1.0e-8`

- `intpnt_tol_psafe`

  Corresponding constant:
  `MSK_DPAR_INTPNT_TOL_PSAFE`

  Description:
  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.

  Possible Values:
  Any number between 1.0e-4 and +inf.

  Default value:
  `1.0`

- `intpnt_tol_rel_gap`

  Corresponding constant:
  `MSK_DPAR_INTPNT_TOL_REL_GAP`

  Description:
  Relative gap termination tolerance.

  Possible Values:
  Any number between 1.0e-14 and +inf.

  Default value:
  `1.0e-8`

- `intpnt_tol_rel_step`

  Corresponding constant:
  `MSK_DPAR_INTPNT_TOL_REL_STEP`

  Description:
  Relative step size to the boundary for linear and quadratic optimization problems.

  Possible Values:
  Any number between 1.0e-4 and 0.999999.

Default value:
`0.9999`

- `intpnt_tol_step_size`

  Corresponding constant:
  `MSK_DPAR_INTPNT_TOL_STEP_SIZE`

  Description:
  If the step size falls below the value of this parameter, then the interior-point optimizer assumes that it is stalled. It it does not not make any progress.

  Possible Values:
  Any number between 0.0 and 1.0.

  Default value:
  `1.0e-10`

- `lower_obj_cut`

  Corresponding constant:
  `MSK_DPAR_LOWER_OBJ_CUT`

  Description:
  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.

  Possible Values:
  Any number between -inf and +inf.

  Default value:
  `-1.0e30`

  See also:
  MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH Objective bound.

- `lower_obj_cut_finite_trh`

  Corresponding constant:
  `MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH`

  Description:
  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 $-\infty$.

  Possible Values:
  Any number between -inf and +inf.

  Default value:
  `-0.5e30`

- `mio_disable_term_time`

  Corresponding constant:
  `MSK_DPAR_MIO_DISABLE_TERM_TIME`

Description:
The termination criteria governed by

- MSK_IPAR_MIO_MAX_NUM_RELAXS
- MSK_IPAR_MIO_MAX_NUM_BRANCHES
- MSK_DPAR_MIO_NEAR_TOL_ABS_GAP
- MSK_DPAR_MIO_NEAR_TOL_REL_GAP

is disabled the first n seconds. This parameter specifies the number n. A negative value is identical to infinity i.e. the termination criteria are never checked.

Possible Values:
Any number between -inf and +inf.

Default value:
`-1.0`

See also:

MSK_IPAR_MIO_MAX_NUM_RELAXS Maximum number of relaxations in branch and bound search.

MSK_IPAR_MIO_MAX_NUM_BRANCHES Maximum number of branches allowed during the branch and bound search.

MSK_DPAR_MIO_NEAR_TOL_ABS_GAP Relaxed absolute optimality tolerance employed by the mixed-integer optimizer.

MSK_DPAR_MIO_NEAR_TOL_REL_GAP The mixed-integer optimizer is terminated when this tolerance is satisfied.

- `mio_heuristic_time`

Corresponding constant:
MSK_DPAR_MIO_HEURISTIC_TIME

Description:
Minimum amount of time to be used in the heuristic search for a good feasible integer solution. A negative values implies that the optimizer decides the amount of time to be spent in the heuristic.

Possible Values:
Any number between -inf and +inf.

Default value:
`-1.0`

- `mio_max_time`

Corresponding constant:
MSK_DPAR_MIO_MAX_TIME

Description:
This parameter limits the maximum time spent by the mixed-integer optimizer. A negative number means infinity.

Possible Values:
Any number between -inf and +inf.

Default value:
`-1.0`

- `mio_max_time_aprx_opt`

  Corresponding constant:
  `MSK_DPAR_MIO_MAX_TIME_APRX_OPT`

  Description:
  Number of seconds spent by the mixed-integer optimizer before the MSK_DPAR_MIO_TOL_REL_RELAX_INT is applied.

  Possible Values:
  Any number between 0.0 and +inf.

  Default value:
  `60`

- `mio_near_tol_abs_gap`

  Corresponding constant:
  `MSK_DPAR_MIO_NEAR_TOL_ABS_GAP`

  Description:
  Relaxed absolute optimality tolerance employed by the mixed-integer optimizer. This termination criteria is delayed. See MSK_DPAR_MIO_DISABLE_TERM_TIME for details.

  Possible Values:
  Any number between 0.0 and +inf.

  Default value:
  `0.0`

  See also:

  MSK_DPAR_MIO_DISABLE_TERM_TIME Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

- `mio_near_tol_rel_gap`

  Corresponding constant:
  `MSK_DPAR_MIO_NEAR_TOL_REL_GAP`

  Description:
  The mixed-integer optimizer is terminated when this tolerance is satisfied. This termination criteria is delayed. See MSK_DPAR_MIO_DISABLE_TERM_TIME for details.

  Possible Values:
  Any number between 0.0 and +inf.

  Default value:
  `1.0e-3`

  See also:

  MSK_DPAR_MIO_DISABLE_TERM_TIME Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

- `mio_rel_add_cut_limited`

  Corresponding constant:
  `MSK_DPAR_MIO_REL_ADD_CUT_LIMITED`

  Description:
  Controls how many cuts the mixed-integer optimizer is allowed to add to the problem. Let α be the value of this parameter and m the number constraints, then mixed-integer optimizer is allowed to αm cuts.

  Possible Values:
  Any number between 0.0 and 2.0.

  Default value:
  `0.75`

- `mio_rel_gap_const`

  Corresponding constant:
  `MSK_DPAR_MIO_REL_GAP_CONST`

  Description:
  This value is used to compute the relative gap for the solution to an integer optimization problem.

  Possible Values:
  Any number between 1.0e-15 and +inf.

  Default value:
  `1.0e-10`

- `mio_tol_abs_gap`

  Corresponding constant:
  `MSK_DPAR_MIO_TOL_ABS_GAP`

  Description:
  Absolute optimality tolerance employed by the mixed-integer optimizer.

  Possible Values:
  Any number between 0.0 and +inf.

  Default value:
  `0.0`

- `mio_tol_abs_relax_int`

  Corresponding constant:
  `MSK_DPAR_MIO_TOL_ABS_RELAX_INT`

  Description:
  Absolute relaxation tolerance of the integer constraints. I.e. $\min(|x| - \lfloor x \rfloor, \lceil x \rceil - |x|)$ is less than the tolerance then the integer restrictions assumed to be satisfied.

  Possible Values:
  Any number between 0.0 and +inf.

Default value:
`1.0e-5`

- `mio_tol_feas`

  Corresponding constant:
  `MSK_DPAR_MIO_TOL_FEAS`

  Description:
  Feasibility tolerance for mixed integer solver. Any solution with maximum infeasibility below this value will be considered feasible.

  Possible Values:
  Any number between 0.0 and +inf.

  Default value:
  `1.0e-7`

- `mio_tol_rel_gap`

  Corresponding constant:
  `MSK_DPAR_MIO_TOL_REL_GAP`

  Description:
  Relative optimality tolerance employed by the mixed-integer optimizer.

  Possible Values:
  Any number between 0.0 and +inf.

  Default value:
  `1.0e-4`

- `mio_tol_rel_relax_int`

  Corresponding constant:
  `MSK_DPAR_MIO_TOL_REL_RELAX_INT`

  Description:
  Relative relaxation tolerance of the integer constraints. I.e $(\min(|x| - \lfloor x \rfloor, \lceil x \rceil - |x|))$ is less than the tolerance times $|x|$ then the integer restrictions assumed to be satisfied.

  Possible Values:
  Any number between 0.0 and +inf.

  Default value:
  `1.0e-6`

- `mio_tol_x`

  Corresponding constant:
  `MSK_DPAR_MIO_TOL_X`

  Description:
  Absolute solution tolerance used in mixed-integer optimizer.

  Possible Values:
  Any number between 0.0 and +inf.

Default value:
`1.0e-6`

- `nonconvex_tol_feas`

  Corresponding constant:
  `MSK_DPAR_NONCONVEX_TOL_FEAS`

  Description:
  Feasibility tolerance used by the nonconvex optimizer.

  Possible Values:
  Any number between 0.0 and +inf.

  Default value:
  `1.0e-6`

- `nonconvex_tol_opt`

  Corresponding constant:
  `MSK_DPAR_NONCONVEX_TOL_OPT`

  Description:
  Optimality tolerance used by the nonconvex optimizer.

  Possible Values:
  Any number between 0.0 and +inf.

  Default value:
  `1.0e-7`

- `optimizer_max_time`

  Corresponding constant:
  `MSK_DPAR_OPTIMIZER_MAX_TIME`

  Description:
  Maximum amount of time the optimizer is allowed to spent on the optimization. A negative number means infinity.

  Possible Values:
  Any number between -inf and +inf.

  Default value:
  `-1.0`

- `presolve_tol_aij`

  Corresponding constant:
  `MSK_DPAR_PRESOLVE_TOL_AIJ`

  Description:
  Absolute zero tolerance employed for a_{ij} in the presolve.

  Possible Values:
  Any number between 1.0e-15 and +inf.

Default value:
`1.0e-12`

- `presolve_tol_lin_dep`

  Corresponding constant:
  `MSK_DPAR_PRESOLVE_TOL_LIN_DEP`

  Description:
  Controls when a constraint is determined to be linearly dependent.

  Possible Values:
  Any number between 0.0 and +inf.

  Default value:
  `1.0e-6`

- `presolve_tol_s`

  Corresponding constant:
  `MSK_DPAR_PRESOLVE_TOL_S`

  Description:
  Absolute zero tolerance employed for s_i in the presolve.

  Possible Values:
  Any number between 0.0 and +inf.

  Default value:
  `1.0e-8`

- `presolve_tol_x`

  Corresponding constant:
  `MSK_DPAR_PRESOLVE_TOL_X`

  Description:
  Absolute zero tolerance employed for x_j in the presolve.

  Possible Values:
  Any number between 0.0 and +inf.

  Default value:
  `1.0e-8`

- `qcqo_reformulate_rel_drop_tol`

  Corresponding constant:
  `MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL`

  Description:
  This parameter determines when columns are dropped in incomplete cholesky factorization doing reformulation of quadratic problems.

  Possible Values:
  Any number between 0 and +inf.

Default value:
`1e-15`

- `sim_lu_tol_rel_piv`

  Corresponding constant:
  `MSK_DPAR_SIM_LU_TOL_REL_PIV`

  Description:
  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.

  Possible Values:
  Any number between 1.0e-6 and 0.999999.

  Default value:
  `0.01`

- `simplex_abs_tol_piv`

  Corresponding constant:
  `MSK_DPAR_SIMPLEX_ABS_TOL_PIV`

  Description:
  Absolute pivot tolerance employed by the simplex optimizers.

  Possible Values:
  Any number between 1.0e-12 and +inf.

  Default value:
  `1.0e-7`

- `upper_obj_cut`

  Corresponding constant:
  `MSK_DPAR_UPPER_OBJ_CUT`

  Description:
  If either a primal or dual feasible solution is found proving that the optimal objective value is outside, [`MSK_DPAR_LOWER_OBJ_CUT`, `MSK_DPAR_UPPER_OBJ_CUT`], then MOSEK is terminated.

  Possible Values:
  Any number between -inf and +inf.

  Default value:
  `1.0e30`

  See also:
  `MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH` Objective bound.

- `upper_obj_cut_finite_trh`

  Corresponding constant:
  `MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH`

Description:
If the upper objective cut is greater than the value of this value parameter, then the the upper objective cut MSK_DPAR_UPPER_OBJ_CUT is treated as ∞.

Possible Values:
Any number between -inf and +inf.

Default value:
`0.5e30`

16.3 Integer parameters

- `alloc_add_qnz`

  Corresponding constant:
  `MSK_IPAR_ALLOC_ADD_QNZ`

  Description:
  Additional number of Q non-zeros that are allocated space for when `numanz` exceeds `maxnumqnz` during addition of new Q entries.

  Possible Values:
  Any number between 0 and +inf.

  Default value:
  `5000`

- `ana_sol_basis`

  Corresponding constant:
  `MSK_IPAR_ANA_SOL_BASIS`

  Description:
  Controls whether the basis matrix is analyzed in solaution analyzer.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- ana_sol_print_violated

Corresponding constant:
MSK_IPAR_ANA_SOL_PRINT_VIOLATED

Description:

Controls whether a list of violated constraints is printed when calling MSK_analyzesolution. All constraints violated by more than the value set by the parameter MSK_DPAR_ANA_SOL_INFEAS_TOL will be printed.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- auto_sort_a_before_opt

Corresponding constant:
MSK_IPAR_AUTO_SORT_A_BEFORE_OPT

Description:
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.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- auto_update_sol_info

Corresponding constant:
MSK_IPAR_AUTO_UPDATE_SOL_INFO

Description:
Controls whether the solution information items are automatically updated after an optimization is performed.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- basis_solve_use_plus_one

Corresponding constant:
MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE

Description:
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.

This has siginificance for the results returned by the MSK_solvewithbasis function.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- bi_clean_optimizer

Corresponding constant:
MSK_IPAR_BI_CLEAN_OPTIMIZER

Description:
Controls which simplex optimizer is used in the clean-up phase.

Possible values:

MSK_OPTIMIZER_INTPNT The interior-point optimizer is used.

MSK_OPTIMIZER_CONCURRENT The optimizer for nonconvex nonlinear problems.

MSK_OPTIMIZER_MIXED_INT The mixed-integer optimizer.

MSK_OPTIMIZER_DUAL_SIMPLEX The dual simplex optimizer is used.

MSK_OPTIMIZER_FREE The optimizer is chosen automatically.

MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX The primal dual simplex optimizer is used.

MSK_OPTIMIZER_CONIC The optimizer for problems having conic constraints.

MSK_OPTIMIZER_NONCONVEX The optimizer for nonconvex nonlinear problems.

MSK_OPTIMIZER_QCONE For internal use only.

MSK_OPTIMIZER_PRIMAL_SIMPLEX The primal simplex optimizer is used.

MSK_OPTIMIZER_FREE_SIMPLEX One of the simplex optimizers is used.

Default value:
MSK_OPTIMIZER_FREE

- bi_ignore_max_iter

Corresponding constant:
MSK_IPAR_BI_IGNORE_MAX_ITER

Description:
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.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- bi_ignore_num_error

Corresponding constant:
MSK_IPAR_BI_IGNORE_NUM_ERROR

Description:
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.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- bi_max_iterations

Corresponding constant:
MSK_IPAR_BI_MAX_ITERATIONS

Description:
Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification.

Possible Values:
Any number between 0 and +inf.

Default value:
1000000

- cache_license

Corresponding constant:
MSK_IPAR_CACHE_LICENSE

Description:
Specifies if the license is kept checked out for the lifetime of the mosek environment (on) or returned to the server immediately after the optimization (off).

By default the license is checked out for the lifetime of the MOSEK environment by the first call to MSK_optimizetrm. The license is checked in when MSK_deleteenv is called.

A specific license feature may be checked in when not in use with the function MSK_checkinlicense. Check-in and check-out of licenses have an overhead. Frequent communication with the license server should be avoided.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- cache_size_l1

Corresponding constant:
MSK_IPAR_CACHE_SIZE_L1

Description:
Specifies the size of the cache of the computer. This parameter is potentially very important for the efficiency on computers if MOSEK cannot determine the cache size automatically. If the cache size is negative, then MOSEK tries to determine the value automatically.

Possible Values:
Any number between -inf and +inf.

Default value:
-1

- cache_size_l2

Corresponding constant:
MSK_IPAR_CACHE_SIZE_L2

Description:
Specifies the size of the cache of the computer. This parameter is potentially very important for the efficiency on computers where MOSEK cannot determine the cache size automatically. If the cache size is negative, then MOSEK tries to determine the value automatically.

Possible Values:
Any number between -inf and +inf.

Default value:
-1

- check_convexity

Corresponding constant:
MSK_IPAR_CHECK_CONVEXITY

Description:
Specify the level of convexity check on quadratic problems

Possible values:

MSK_CHECK_CONVEXITY_SIMPLE Perform simple and fast convexity check.

MSK_CHECK_CONVEXITY_NONE No convexity check.

MSK_CHECK_CONVEXITY_FULL Perform a full convexity check.

Default value:
MSK_CHECK_CONVEXITY_FULL

- check_task_data

Corresponding constant:
MSK_IPAR_CHECK_TASK_DATA

Description:
If this feature is turned on, then the task data is checked for bad values i.e. NaNs. before an optimization is performed.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- concurrent_num_optimizers

Corresponding constant:
MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS

Description:
The maximum number of simultaneous optimizations that will be started by the concurrent optimizer.

Possible Values:
Any number between 0 and +inf.

Default value:
2

- concurrent_priority_dual_simplex

Corresponding constant:
MSK_IPAR_CONCURRENT_PRIORITY_DUAL_SIMPLEX

Description:
Priority of the dual simplex algorithm when selecting solvers for concurrent optimization.

Possible Values:
Any number between 0 and +inf.

Default value:
2

- concurrent_priority_free_simplex

Corresponding constant:
MSK_IPAR_CONCURRENT_PRIORITY_FREE_SIMPLEX

Description:
Priority of the free simplex optimizer when selecting solvers for concurrent optimization.

Possible Values:
Any number between 0 and +inf.

Default value:
3

- concurrent_priority_intpnt

Corresponding constant:
MSK_IPAR_CONCURRENT_PRIORITY_INTPNT

Description:
Priority of the interior-point algorithm when selecting solvers for concurrent optimization.

Possible Values:
Any number between 0 and +inf.

Default value:
4

- concurrent_priority_primal_simplex

Corresponding constant:
MSK_IPAR_CONCURRENT_PRIORITY_PRIMAL_SIMPLEX

Description:
Priority of the primal simplex algorithm when selecting solvers for concurrent optimization.

Possible Values:
Any number between 0 and +inf.

Default value:
1

- cpu_type

Corresponding constant:
MSK_IPAR_CPU_TYPE

Description:
Specifies the CPU type. By default MOSEK tries to auto detect the CPU type. Therefore, we recommend to change this parameter only if the auto detection does not work properly.

Possible values:

MSK_CPU_POWERPC_G5 A G5 PowerPC CPU.

MSK_CPU_INTEL_PM An Intel PM cpu.

MSK_CPU_GENERIC An generic CPU type for the platform

MSK_CPU_UNKNOWN An unknown CPU.

MSK_CPU_AMD_OPTERON An AMD Opteron (64 bit).

MSK_CPU_INTEL_ITANIUM2 An Intel Itanium2.

MSK_CPU_AMD_ATHLON An AMD Athlon.

MSK_CPU_HP_PARISC20 An HP PA RISC version 2.0 CPU.

MSK_CPU_INTEL_P4 An Intel Pentium P4 or Intel Xeon.

MSK_CPU_INTEL_P3 An Intel Pentium P3.

MSK_CPU_INTEL_CORE2 An Intel CORE2 cpu.

Default value:

MSK_CPU_UNKNOWN

- data_check

Corresponding constant:

MSK_IPAR_DATA_CHECK

Description:

If this option is turned on, then extensive data checking is enabled. It will slow down MOSEK but on the other hand help locating bugs.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:

MSK_ON

- feasrepair_optimize

Corresponding constant:

MSK_IPAR_FEASREPAIR_OPTIMIZE

Description:

Controls which type of feasibility analysis is to be performed.

Possible values:

MSK_FEASREPAIR_OPTIMIZE_NONE Do not optimize the feasibility repair problem.

MSK_FEASREPAIR_OPTIMIZE_COMBINED Minimize with original objective subject to minimal weighted violation of bounds.

MSK_FEASREPAIR_OPTIMIZE_PENALTY Minimize weighted sum of violations.

Default value:

MSK_FEASREPAIR_OPTIMIZE_NONE

- infeas_generic_names

Corresponding constant:

MSK_IPAR_INFEAS_GENERIC_NAMES

Description:

Controls whether generic names are used when an infeasible subproblem is created.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- infeas_prefer_primal

Corresponding constant:
MSK_IPAR_INFEAS_PREFER_PRIMAL

Description:
If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- infeas_report_auto

Corresponding constant:
MSK_IPAR_INFEAS_REPORT_AUTO

Description:
Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- infeas_report_level

Corresponding constant:
MSK_IPAR_INFEAS_REPORT_LEVEL

Description:
Controls the amount of information presented in an infeasibility report. Higher values imply more information.

Possible Values:
Any number between 0 and +inf.

Default value:
1

- intpnt_basis

Corresponding constant:
MSK_IPAR_INTPNT_BASIS

Description:
Controls whether the interior-point optimizer also computes an optimal basis.

Possible values:

`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_NEVER` Never do basis identification.

`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_OTHER` Try another BI method.

Default value:
`MSK_BI_ALWAYS`

See also:

`MSK_IPAR_BI_IGNORE_MAX_ITER` Turns on basis identification in case the interior-point optimizer is terminated due to maximum number of iterations.

`MSK_IPAR_BI_IGNORE_NUM_ERROR` Turns on basis identification in case the interior-point optimizer is terminated due to a numerical problem.

- `intpnt_diff_step`

Corresponding constant:
`MSK_IPAR_INTPNT_DIFF_STEP`

Description:
Controls whether different step sizes are allowed in the primal and dual space.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `intpnt_factor_debug_lvl`

Corresponding constant:
`MSK_IPAR_INTPNT_FACTOR_DEBUG_LVL`

Description:
Controls factorization debug level.

Possible Values:
Any number between 0 and +inf.

Default value:
`0`

- `intpnt_factor_method`

Corresponding constant:
`MSK_IPAR_INTPNT_FACTOR_METHOD`

Description:
Controls the method used to factor the Newton equation system.

Possible Values:
Any number between 0 and +inf.

Default value:
`0`

- `intpnt_max_iterations`

Corresponding constant:
`MSK_IPAR_INTPNT_MAX_ITERATIONS`

Description:
Controls the maximum number of iterations allowed in the interior-point optimizer.

Possible Values:
Any number between 0 and +inf.

Default value:
`400`

- `intpnt_max_num_cor`

Corresponding constant:
`MSK_IPAR_INTPNT_MAX_NUM_COR`

Description:
Controls the maximum number of correctors allowed by the multiple corrector procedure. A negative value means that MOSEK is making the choice.

Possible Values:
Any number between -1 and +inf.

Default value:
`-1`

- `intpnt_max_num_refinement_steps`

Corresponding constant:
`MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS`

Description:
Maximum number of steps to be used by the iterative refinement of the search direction. A negative value implies that the optimizer Chooses the maximum number of iterative refinement steps.

Possible Values:
Any number between -inf and +inf.

Default value:
`-1`

- `intpnt_num_threads`

  Corresponding constant:
  `MSK_IPAR_INTPNT_NUM_THREADS`

  Description:
  Controls the number of threads employed by the interior-point optimizer. If set to a positive number MOSEK will use this number of threads. If zero the number of threads used will equal the number of cores detected on the machine.

  Possible Values:
  Any integer greater or equal to 0.

  Default value:
  `1`

- `intpnt_off_col_trh`

  Corresponding constant:
  `MSK_IPAR_INTPNT_OFF_COL_TRH`

  Description:
  Controls how many offending columns are detected in the Jacobian of the constraint matrix.

  1 means aggressive detection, higher values mean less aggressive detection.

  0 means no detection.

  Possible Values:
  Any number between 0 and +inf.

  Default value:
  `40`

- `intpnt_order_method`

  Corresponding constant:
  `MSK_IPAR_INTPNT_ORDER_METHOD`

  Description:
  Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.

  Possible values:

  `MSK_ORDER_METHOD_NONE` No ordering is used.

  `MSK_ORDER_METHOD_APPMINLOC2` A variant of the approximate minimum local-fill-in ordering is used.

  `MSK_ORDER_METHOD_APPMINLOC1` Approximate minimum local-fill-in ordering is used.

  `MSK_ORDER_METHOD_GRAPHPAR2` An alternative graph partitioning based ordering.

  `MSK_ORDER_METHOD_FREE` The ordering method is chosen automatically.

  `MSK_ORDER_METHOD_GRAPHPAR1` Graph partitioning based ordering.

  Default value:
  `MSK_ORDER_METHOD_FREE`

- intpnt_regularization_use

  Corresponding constant:
  MSK_IPAR_INTPNT_REGULARIZATION_USE

  Description:
  Controls whether regularization is allowed.

  Possible values:

  MSK_ON Switch the option on.

  MSK_OFF Switch the option off.

  Default value:
  MSK_ON

- intpnt_scaling

  Corresponding constant:
  MSK_IPAR_INTPNT_SCALING

  Description:
  Controls how the problem is scaled before the interior-point optimizer is used.

  Possible values:

  MSK_SCALING_NONE No scaling is performed.

  MSK_SCALING_MODERATE A conservative scaling is performed.

  MSK_SCALING_AGGRESSIVE A very aggressive scaling is performed.

  MSK_SCALING_FREE The optimizer chooses the scaling heuristic.

  Default value:
  MSK_SCALING_FREE

- intpnt_solve_form

  Corresponding constant:
  MSK_IPAR_INTPNT_SOLVE_FORM

  Description:
  Controls whether the primal or the dual problem is solved.

  Possible values:

  MSK_SOLVE_PRIMAL The optimizer should solve the primal problem.

  MSK_SOLVE_DUAL The optimizer should solve the dual problem.

  MSK_SOLVE_FREE The optimizer is free to solve either the primal or the dual problem.

  Default value:
  MSK_SOLVE_FREE

- intpnt_starting_point

  Corresponding constant:
  MSK_IPAR_INTPNT_STARTING_POINT

Description:
Starting point used by the interior-point optimizer.

Possible values:

`MSK_STARTING_POINT_GUESS` The optimizer guesses a starting point.

`MSK_STARTING_POINT_SATISFY_BOUNDS` The starting point is choosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

`MSK_STARTING_POINT_CONSTANT` The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.

`MSK_STARTING_POINT_FREE` The starting point is chosen automatically.

Default value:
`MSK_STARTING_POINT_FREE`

- `lic_trh_expiry_wrn`

Corresponding constant:
`MSK_IPAR_LIC_TRH_EXPIRY_WRN`

Description:
If a license feature expires in a numbers days less than the value of this parameter then a warning will be issued.

Possible Values:
Any number between 0 and +inf.

Default value:
7

- `license_allow_overuse`

Corresponding constant:
`MSK_IPAR_LICENSE_ALLOW_OVERUSE`

Description:
Controls if license overuse is allowed when caching licenses

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `license_cache_time`

Corresponding constant:
`MSK_IPAR_LICENSE_CACHE_TIME`

Description:
Setting this parameter no longer has any effect. Please see `MSK_IPAR_CACHE_LICENSE` for an alternative.

Possible Values:
Any number between 0 and 65555.

Default value:
5

- `license_check_time`

  Corresponding constant:
  `MSK_IPAR_LICENSE_CHECK_TIME`

  Description:
  The parameter specifies the number of seconds between the checks of all the active licenses in the MOSEK environment license cache. These checks are performed to determine if the licenses should be returned to the server.

  Possible Values:
  Any number between 1 and 120.

  Default value:
  1

- `license_debug`

  Corresponding constant:
  `MSK_IPAR_LICENSE_DEBUG`

  Description:
  This option is used to turn on debugging of the incense manager.

  Possible values:

  `MSK_ON` Switch the option on.

  `MSK_OFF` Switch the option off.

  Default value:
  `MSK_OFF`

- `license_pause_time`

  Corresponding constant:
  `MSK_IPAR_LICENSE_PAUSE_TIME`

  Description:
  If `MSK_IPAR_LICENSE_WAIT`=`MSK_ON` and no license is available, then MOSEK sleeps a number of milliseconds between each check of whether a license has become free.

  Possible Values:
  Any number between 0 and 1000000.

  Default value:
  100

- `license_suppress_expire_wrns`

  Corresponding constant:
  `MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS`

Description:
Controls whether license features expire warnings are suppressed.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_OFF`

- `license_wait`

Corresponding constant:
`MSK_IPAR_LICENSE_WAIT`

Description:
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.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_OFF`

- `log`

Corresponding constant:
`MSK_IPAR_LOG`

Description:
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.

Possible Values:
Any number between 0 and +inf.

Default value:
`10`

See also:

`MSK_IPAR_LOG_CUT_SECOND_OPT` Controls the reduction in the log levels for the second and any subsequent optimizations.

- `log_bi`

Corresponding constant:
`MSK_IPAR_LOG_BI`

Description:
Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.

Possible Values:
Any number between 0 and +inf.

Default value:
`4`

- `log_bi_freq`

Corresponding constant:
`MSK_IPAR_LOG_BI_FREQ`

Description:
Controls how frequent the optimizer outputs information about the basis identification and how frequent the user-defined call-back function is called.

Possible Values:
Any number between 0 and +inf.

Default value:
`2500`

- `log_check_convexity`

Corresponding constant:
`MSK_IPAR_LOG_CHECK_CONVEXITY`

Description:
Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on.

If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

Possible Values:
Any number between 0 and +inf.

Default value:
`0`

- `log_concurrent`

Corresponding constant:
`MSK_IPAR_LOG_CONCURRENT`

Description:
Controls amount of output printed by the concurrent optimizer.

Possible Values:
Any number between 0 and +inf.

Default value:
`1`

- log_cut_second_opt

  Corresponding constant:
  MSK_IPAR_LOG_CUT_SECOND_OPT

  Description:
  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.

  Possible Values:
  Any number between 0 and +inf.

  Default value:
  1

  See also:

  MSK_IPAR_LOG Controls the amount of log information.

  MSK_IPAR_LOG_INTPNT Controls the amount of log information from the interior-point optimizers.

  MSK_IPAR_LOG_MIO Controls the amount of log information from the mixed-integer optimizers.

  MSK_IPAR_LOG_SIM Controls the amount of log information from the simplex optimizers.

- log_factor

  Corresponding constant:
  MSK_IPAR_LOG_FACTOR

  Description:
  If turned on, then the factor log lines are added to the log.

  Possible Values:
  Any number between 0 and +inf.

  Default value:
  1

- log_feasrepair

  Corresponding constant:
  MSK_IPAR_LOG_FEASREPAIR

  Description:
  Controls the amount of output printed when performing feasibility repair.

  Possible Values:
  Any number between 0 and +inf.

  Default value:
  0

- log_file

Corresponding constant:
MSK_IPAR_LOG_FILE

Description:
If turned on, then some log info is printed when a file is written or read.

Possible Values:
Any number between 0 and +inf.

Default value:
1

- log_head

Corresponding constant:
MSK_IPAR_LOG_HEAD

Description:
If turned on, then a header line is added to the log.

Possible Values:
Any number between 0 and +inf.

Default value:
1

- log_infeas_ana

Corresponding constant:
MSK_IPAR_LOG_INFEAS_ANA

Description:
Controls amount of output printed by the infeasibility analyzer procedures. A higher level implies that more information is logged.

Possible Values:
Any number between 0 and +inf.

Default value:
1

- log_intpnt

Corresponding constant:
MSK_IPAR_LOG_INTPNT

Description:
Controls amount of output printed printed by the interior-point optimizer. A higher level implies that more information is logged.

Possible Values:
Any number between 0 and +inf.

Default value:
4

- log_mio

Corresponding constant:
MSK_IPAR_LOG_MIO

Description:
Controls the log level for the mixed-integer optimizer. A higher level implies that more information is logged.

Possible Values:
Any number between 0 and +inf.

Default value:
4

- log_mio_freq

Corresponding constant:
MSK_IPAR_LOG_MIO_FREQ

Description:
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.

Possible Values:
A integer value.

Default value:
1000

- log_nonconvex

Corresponding constant:
MSK_IPAR_LOG_NONCONVEX

Description:
Controls amount of output printed by the nonconvex optimizer.

Possible Values:
Any number between 0 and +inf.

Default value:
1

- log_optimizer

Corresponding constant:
MSK_IPAR_LOG_OPTIMIZER

Description:
Controls the amount of general optimizer information that is logged.

Possible Values:
Any number between 0 and +inf.

Default value:
1

- log_order

Corresponding constant:
MSK_IPAR_LOG_ORDER

Description:
If turned on, then factor lines are added to the log.

Possible Values:
Any number between 0 and +inf.

Default value:
1

- log_param

Corresponding constant:
MSK_IPAR_LOG_PARAM

Description:
Controls the amount of information printed out about parameter changes.

Possible Values:
Any number between 0 and +inf.

Default value:
0

- log_presolve

Corresponding constant:
MSK_IPAR_LOG_PRESOLVE

Description:
Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.

Possible Values:
Any number between 0 and +inf.

Default value:
1

- log_response

Corresponding constant:
MSK_IPAR_LOG_RESPONSE

Description:
Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.

Possible Values:
Any number between 0 and +inf.

Default value:
0

- log_sensitivity

Corresponding constant:
MSK_IPAR_LOG_SENSITIVITY

Description:
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.

Possible Values:
Any number between 0 and +inf.

Default value:
1

- log_sensitivity_opt

Corresponding constant:
MSK_IPAR_LOG_SENSITIVITY_OPT

Description:
Controls the amount of logging from the optimizers employed during the sensitivity analysis. 0 means no logging information is produced.

Possible Values:
Any number between 0 and +inf.

Default value:
0

- log_sim

Corresponding constant:
MSK_IPAR_LOG_SIM

Description:
Controls amount of output printed by the simplex optimizer. A higher level implies that more information is logged.

Possible Values:
Any number between 0 and +inf.

Default value:
4

- log_sim_freq

Corresponding constant:
MSK_IPAR_LOG_SIM_FREQ

Description:
Controls how frequent the simplex optimizer outputs information about the optimization and how frequent the user-defined call-back function is called.

Possible Values:
Any number between 0 and +inf.

Default value:
500

- `log_sim_minor`

  Corresponding constant:
  `MSK_IPAR_LOG_SIM_MINOR`

  Description:
  Currently not in use.

  Possible Values:
  Any number between 0 and +inf.

  Default value:
  `1`

- `log_sim_network_freq`

  Corresponding constant:
  `MSK_IPAR_LOG_SIM_NETWORK_FREQ`

  Description:
  Controls how frequent the network simplex optimizer outputs information about the optimization and how frequent the user-defined call-back function is called. The network optimizer will use a logging frequency equal to `MSK_IPAR_LOG_SIM_FREQ` times `MSK_IPAR_LOG_SIM_NETWORK_FREQ`.

  Possible Values:
  Any number between 0 and +inf.

  Default value:
  `50`

- `log_storage`

  Corresponding constant:
  `MSK_IPAR_LOG_STORAGE`

  Description:
  When turned on, MOSEK prints messages regarding the storage usage and allocation.

  Possible Values:
  Any number between 0 and +inf.

  Default value:
  `0`

- `lp_write_ignore_incompatible_items`

  Corresponding constant:
  `MSK_IPAR_LP_WRITE_IGNORE_INCOMPATIBLE_ITEMS`

  Description:
  Controls the result of writing a problem containing incompatible items to an LP file.

  Possible values:

  `MSK_ON` Switch the option on.

  `MSK_OFF` Switch the option off.

Default value:
MSK_OFF

- max_num_warnings

Corresponding constant:
MSK_IPAR_MAX_NUM_WARNINGS

Description:
Waning level. A higher value results in more warnings.

Possible Values:
Any number between 0 and +inf.

Default value:
10

- mio_branch_dir

Corresponding constant:
MSK_IPAR_MIO_BRANCH_DIR

Description:
Controls whether the mixed-integer optimizer is branching up or down by default.

Possible values:

MSK_BRANCH_DIR_DOWN The mixed-integer optimizer always chooses the down branch first.

MSK_BRANCH_DIR_UP The mixed-integer optimizer always chooses the up branch first.

MSK_BRANCH_DIR_FREE The mixed-integer optimizer decides which branch to choose.

Default value:
MSK_BRANCH_DIR_FREE

- mio_branch_priorities_use

Corresponding constant:
MSK_IPAR_MIO_BRANCH_PRIORITIES_USE

Description:
Controls whether branching priorities are used by the mixed-integer optimizer.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- mio_construct_sol

Corresponding constant:
MSK_IPAR_MIO_CONSTRUCT_SOL

Description:
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.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- mio_cont_sol

Corresponding constant:
MSK_IPAR_MIO_CONT_SOL

Description:
Controls the meaning of the interior-point and basic solutions in mixed integer problems.

Possible values:

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_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_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.

Default value:
MSK_MIO_CONT_SOL_NONE

- mio_cut_level_root

Corresponding constant:
MSK_IPAR_MIO_CUT_LEVEL_ROOT

Description:
Controls the cut level employed by the mixed-integer optimizer at the root node. A negative value means a default value determined by the mixed-integer optimizer is used. By adding the appropriate values from the following table the employed cut types can be controlled.

GUB cover	+2
Flow cover	+4
Lifting	+8
Plant location	+16
Disaggregation	+32
Knapsack cover	+64
Lattice	+128
Gomory	+256
Coefficient reduction	+512
GCD	+1024
Obj. integrality	+2048

Possible Values:
Any value.

Default value:
`-1`

- `mio_cut_level_tree`

Corresponding constant:
`MSK_IPAR_MIO_CUT_LEVEL_TREE`

Description:
Controls the cut level employed by the mixed-integer optimizer at the tree. See MSK_IPAR_MIO_CUT_LEVEL_ROOT for an explanation of the parameter values.

Possible Values:
Any value.

Default value:
`-1`

- `mio_feaspump_level`

Corresponding constant:
`MSK_IPAR_MIO_FEASPUMP_LEVEL`

Description:
Feasibility pump is a heuristic designed to compute an initial feasible solution. A value of 0 implies that the feasibility pump heuristic is not used. A value of -1 implies that the mixed-integer optimizer decides how the feasibility pump heuristic is used. A larger value than 1 implies that the feasibility pump is employed more aggressively. Normally a value beyond 3 is not worthwhile.

Possible Values:
Any number between -inf and 3.

Default value:
`-1`

- `mio_heuristic_level`

Corresponding constant:
`MSK_IPAR_MIO_HEURISTIC_LEVEL`

Description:
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.

Possible Values:
Any value.

Default value:
`-1`

- `mio_hotstart`

Corresponding constant:
`MSK_IPAR_MIO_HOTSTART`

Description:
Controls whether the integer optimizer is hot-started.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `mio_keep_basis`

Corresponding constant:
`MSK_IPAR_MIO_KEEP_BASIS`

Description:
Controls whether the integer presolve keeps bases in memory. This speeds on the solution process at cost of bigger memory consumption.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `mio_local_branch_number`

Corresponding constant:
`MSK_IPAR_MIO_LOCAL_BRANCH_NUMBER`

Description:
Controls the size of the local search space when doing local branching.

Possible Values:
Any number between -inf and +inf.

Default value:
`-1`

- `mio_max_num_branches`

  Corresponding constant:
  `MSK_IPAR_MIO_MAX_NUM_BRANCHES`

  Description:
  Maximum number of branches allowed during the branch and bound search. A negative value means infinite.

  Possible Values:
  Any number between -inf and +inf.

  Default value:
  `-1`

  See also:
  `MSK_DPAR_MIO_DISABLE_TERM_TIME` Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

- `mio_max_num_relaxs`

  Corresponding constant:
  `MSK_IPAR_MIO_MAX_NUM_RELAXS`

  Description:
  Maximum number of relaxations allowed during the branch and bound search. A negative value means infinite.

  Possible Values:
  Any number between -inf and +inf.

  Default value:
  `-1`

  See also:
  `MSK_DPAR_MIO_DISABLE_TERM_TIME` Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

- `mio_max_num_solutions`

  Corresponding constant:
  `MSK_IPAR_MIO_MAX_NUM_SOLUTIONS`

  Description:
  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 and n is strictly positive, then the mixed-integer optimizer will be terminated when n feasible solutions have been located.

  Possible Values:
  Any number between -inf and +inf.

  Default value:
  `-1`

See also:

MSK_DPAR_MIO_DISABLE_TERM_TIME Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

- mio_mode

Corresponding constant:
MSK_IPAR_MIO_MODE

Description:
Controls whether the optimizer includes the integer restrictions when solving a (mixed) integer optimization problem.

Possible values:

MSK_MIO_MODE_IGNORED The integer constraints are ignored and the problem is solved as a continuous problem.

MSK_MIO_MODE_LAZY Integer restrictions should be satisfied if an optimizer is available for the problem.

MSK_MIO_MODE_SATISFIED Integer restrictions should be satisfied.

Default value:
MSK_MIO_MODE_SATISFIED

- mio_node_optimizer

Corresponding constant:
MSK_IPAR_MIO_NODE_OPTIMIZER

Description:
Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.

Possible values:

MSK_OPTIMIZER_INTPNT The interior-point optimizer is used.

MSK_OPTIMIZER_CONCURRENT The optimizer for nonconvex nonlinear problems.

MSK_OPTIMIZER_MIXED_INT The mixed-integer optimizer.

MSK_OPTIMIZER_DUAL_SIMPLEX The dual simplex optimizer is used.

MSK_OPTIMIZER_FREE The optimizer is chosen automatically.

MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX The primal dual simplex optimizer is used.

MSK_OPTIMIZER_CONIC The optimizer for problems having conic constraints.

MSK_OPTIMIZER_NONCONVEX The optimizer for nonconvex nonlinear problems.

MSK_OPTIMIZER_QCONE For internal use only.

MSK_OPTIMIZER_PRIMAL_SIMPLEX The primal simplex optimizer is used.

MSK_OPTIMIZER_FREE_SIMPLEX One of the simplex optimizers is used.

Default value:
MSK_OPTIMIZER_FREE

- mio_node_selection

Corresponding constant:
MSK_IPAR_MIO_NODE_SELECTION

Description:
Controls the node selection strategy employed by the mixed-integer optimizer.

Possible values:

MSK_MIO_NODE_SELECTION_PSEUDO The optimizer employs selects the node based on a pseudo cost estimate.

MSK_MIO_NODE_SELECTION_HYBRID The optimizer employs a hybrid strategy.

MSK_MIO_NODE_SELECTION_FREE The optimizer decides the node selection strategy.

MSK_MIO_NODE_SELECTION_WORST The optimizer employs a worst bound node selection strategy.

MSK_MIO_NODE_SELECTION_BEST The optimizer employs a best bound node selection strategy.

MSK_MIO_NODE_SELECTION_FIRST The optimizer employs a depth first node selection strategy.

Default value:
MSK_MIO_NODE_SELECTION_FREE

- mio_optimizer_mode

Corresponding constant:
MSK_IPAR_MIO_OPTIMIZER_MODE

Description:
An exprimental feature.

Possible Values:
Any number between 0 and 1.

Default value:
0

- mio_presolve_aggregate

Corresponding constant:
MSK_IPAR_MIO_PRESOLVE_AGGREGATE

Description:
Controls whether the presolve used by the mixed-integer optimizer tries to aggregate the constraints.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- mio_presolve_probing

Corresponding constant:
MSK_IPAR_MIO_PRESOLVE_PROBING

Description:
Controls whether the mixed-integer presolve performs probing. Probing can be very time consuming.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- mio_presolve_use

Corresponding constant:
MSK_IPAR_MIO_PRESOLVE_USE

Description:
Controls whether presolve is performed by the mixed-integer optimizer.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- mio_root_optimizer

Corresponding constant:
MSK_IPAR_MIO_ROOT_OPTIMIZER

Description:
Controls which optimizer is employed at the root node in the mixed-integer optimizer.

Possible values:

MSK_OPTIMIZER_INTPNT The interior-point optimizer is used.

MSK_OPTIMIZER_CONCURRENT The optimizer for nonconvex nonlinear problems.

MSK_OPTIMIZER_MIXED_INT The mixed-integer optimizer.

MSK_OPTIMIZER_DUAL_SIMPLEX The dual simplex optimizer is used.

MSK_OPTIMIZER_FREE The optimizer is chosen automatically.

MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX The primal dual simplex optimizer is used.

MSK_OPTIMIZER_CONIC The optimizer for problems having conic constraints.

MSK_OPTIMIZER_NONCONVEX The optimizer for nonconvex nonlinear problems.

MSK_OPTIMIZER_QCONE For internal use only.

MSK_OPTIMIZER_PRIMAL_SIMPLEX The primal simplex optimizer is used.

MSK_OPTIMIZER_FREE_SIMPLEX One of the simplex optimizers is used.

Default value:
MSK_OPTIMIZER_FREE

- mio_strong_branch

Corresponding constant:
MSK_IPAR_MIO_STRONG_BRANCH

Description:
The value specifies the depth from the root in which strong branching is used. A negative value means that the optimizer chooses a default value automatically.

Possible Values:
Any number between -inf and +inf.

Default value:
-1

- nonconvex_max_iterations

Corresponding constant:
MSK_IPAR_NONCONVEX_MAX_ITERATIONS

Description:
Maximum number of iterations that can be used by the nonconvex optimizer.

Possible Values:
Any number between 0 and +inf.

Default value:
100000

- objective_sense

Corresponding constant:
MSK_IPAR_OBJECTIVE_SENSE

Description:
If the objective sense for the task is undefined, then the value of this parameter is used as the default objective sense.

Possible values:

MSK_OBJECTIVE_SENSE_MINIMIZE The problem should be minimized.

MSK_OBJECTIVE_SENSE_UNDEFINED The objective sense is undefined.

MSK_OBJECTIVE_SENSE_MAXIMIZE The problem should be maximized.

Default value:
MSK_OBJECTIVE_SENSE_MINIMIZE

- opf_max_terms_per_line

Corresponding constant:
MSK_IPAR_OPF_MAX_TERMS_PER_LINE

Description:
The maximum number of terms (linear and quadratic) per line when an OPF file is written.

Possible Values:
Any number between 0 and +inf.

Default value:
5

- opf_write_header

Corresponding constant:
MSK_IPAR_OPF_WRITE_HEADER

Description:
Write a text header with date and MOSEK version in an OPF file.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- opf_write_hints

Corresponding constant:
MSK_IPAR_OPF_WRITE_HINTS

Description:
Write a hint section with problem dimensions in the beginning of an OPF file.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- opf_write_parameters

Corresponding constant:
MSK_IPAR_OPF_WRITE_PARAMETERS

Description:
Write a parameter section in an OPF file.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- opf_write_problem

Corresponding constant:
MSK_IPAR_OPF_WRITE_PROBLEM

Description:
Write objective, constraints, bounds etc. to an OPF file.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `opf_write_sol_bas`

Corresponding constant:
`MSK_IPAR_OPF_WRITE_SOL_BAS`

Description:
If `MSK_IPAR_OPF_WRITE_SOLUTIONS` is `MSK_ON` and a basic solution is defined, include the basic solution in OPF files.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `opf_write_sol_itg`

Corresponding constant:
`MSK_IPAR_OPF_WRITE_SOL_ITG`

Description:
If `MSK_IPAR_OPF_WRITE_SOLUTIONS` is `MSK_ON` and an integer solution is defined, write the integer solution in OPF files.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `opf_write_sol_itr`

Corresponding constant:
`MSK_IPAR_OPF_WRITE_SOL_ITR`

Description:
If `MSK_IPAR_OPF_WRITE_SOLUTIONS` is `MSK_ON` and an interior solution is defined, write the interior solution in OPF files.

Possible values:

`MSK_ON` Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- opf_write_solutions

Corresponding constant:
MSK_IPAR_OPF_WRITE_SOLUTIONS

Description:
Enable inclusion of solutions in the OPF files.

Possible values:

MSK_ON Switch the option on.
MSK_OFF Switch the option off.

Default value:
MSK_OFF

- optimizer

Corresponding constant:
MSK_IPAR_OPTIMIZER

Description:
The paramter controls which optimizer is used to optimize the task.

Possible values:

MSK_OPTIMIZER_INTPNT The interior-point optimizer is used.
MSK_OPTIMIZER_CONCURRENT The optimizer for nonconvex nonlinear problems.
MSK_OPTIMIZER_MIXED_INT The mixed-integer optimizer.
MSK_OPTIMIZER_DUAL_SIMPLEX The dual simplex optimizer is used.
MSK_OPTIMIZER_FREE The optimizer is chosen automatically.
MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX The primal dual simplex optimizer is used.
MSK_OPTIMIZER_CONIC The optimizer for problems having conic constraints.
MSK_OPTIMIZER_NONCONVEX The optimizer for nonconvex nonlinear problems.
MSK_OPTIMIZER_QCONE For internal use only.
MSK_OPTIMIZER_PRIMAL_SIMPLEX The primal simplex optimizer is used.
MSK_OPTIMIZER_FREE_SIMPLEX One of the simplex optimizers is used.

Default value:
MSK_OPTIMIZER_FREE

- param_read_case_name

Corresponding constant:
MSK_IPAR_PARAM_READ_CASE_NAME

Description:
If turned on, then names in the parameter file are case sensitive.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `param_read_ign_error`

  Corresponding constant:
  `MSK_IPAR_PARAM_READ_IGN_ERROR`

  Description:
  If turned on, then errors in paramter settings is ignored.

  Possible values:

  `MSK_ON` Switch the option on.

  `MSK_OFF` Switch the option off.

  Default value:
  `MSK_OFF`

- `presolve_elim_fill`

  Corresponding constant:
  `MSK_IPAR_PRESOLVE_ELIM_FILL`

  Description:
  Controls the maximum amount of fill-in that can be created during the elimination phase of the presolve. This parameter times (`numcon`+`numvar`) denotes the amount of fill-in.

  Possible Values:
  Any number between 0 and +inf.

  Default value:
  `1`

- `presolve_eliminator_max_num_tries`

  Corresponding constant:
  `MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES`

  Description:
  Control the maximum number of times the eliminator is tried.

  Possible Values:
  A negative value implies MOSEK decides maximum number of times.

  Default value:
  `-1`

- `presolve_eliminator_use`

  Corresponding constant:
  `MSK_IPAR_PRESOLVE_ELIMINATOR_USE`

Description:
Controls whether free or implied free variables are eliminated from the problem.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `presolve_level`

Corresponding constant:
`MSK_IPAR_PRESOLVE_LEVEL`

Description:
Currently not used.

Possible Values:
Any number between -inf and +inf.

Default value:
`-1`

- `presolve_lindep_use`

Corresponding constant:
`MSK_IPAR_PRESOLVE_LINDEP_USE`

Description:
Controls whether the linear constraints are checked for linear dependencies.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `presolve_lindep_work_lim`

Corresponding constant:
`MSK_IPAR_PRESOLVE_LINDEP_WORK_LIM`

Description:
Is used to limit the amount of work that can done to locate linear dependencies. In general the higher value this parameter is given the less work can be used. However, a value of 0 means no limit on the amount work that can be used.

Possible Values:
Any number between 0 and +inf.

Default value:
`1`

- presolve_use

  Corresponding constant:
  MSK_IPAR_PRESOLVE_USE

  Description:
  Controls whether the presolve is applied to a problem before it is optimized.

  Possible values:

  MSK_PRESOLVE_MODE_ON The problem is presolved before it is optimized.

  MSK_PRESOLVE_MODE_OFF The problem is not presolved before it is optimized.

  MSK_PRESOLVE_MODE_FREE It is decided automatically whether to presolve before the problem is optimized.

  Default value:
  MSK_PRESOLVE_MODE_FREE

- qo_separable_reformulation

  Corresponding constant:
  MSK_IPAR_QO_SEPARABLE_REFORMULATION

  Description:
  Determine if Quadratic programing problems should be reformulated to separable form.

  Possible values:

  MSK_ON Switch the option on.

  MSK_OFF Switch the option off.

  Default value:
  MSK_OFF

- read_add_anz

  Corresponding constant:
  MSK_IPAR_READ_ADD_ANZ

  Description:
  Additional number of non-zeros in A that is made room for in the problem.

  Possible Values:
  Any number between 0 and +inf.

  Default value:
  0

- read_add_con

  Corresponding constant:
  MSK_IPAR_READ_ADD_CON

  Description:
  Additional number of constraints that is made room for in the problem.

Possible Values:
Any number between 0 and +inf.

Default value:
0

- read_add_cone

Corresponding constant:
MSK_IPAR_READ_ADD_CONE

Description:
Additional number of conic constraints that is made room for in the problem.

Possible Values:
Any number between 0 and +inf.

Default value:
0

- read_add_qnz

Corresponding constant:
MSK_IPAR_READ_ADD_QNZ

Description:
Additional number of non-zeros in the Q matrices that is made room for in the problem.

Possible Values:
Any number between 0 and +inf.

Default value:
0

- read_add_var

Corresponding constant:
MSK_IPAR_READ_ADD_VAR

Description:
Additional number of variables that is made room for in the problem.

Possible Values:
Any number between 0 and +inf.

Default value:
0

- read_anz

Corresponding constant:
MSK_IPAR_READ_ANZ

Description:
Expected maximum number of A non-zeros to be read. The option is used only by fast MPS and LP file readers.

Possible Values:
Any number between 0 and +inf.

Default value:
100000

- read_con

Corresponding constant:
MSK_IPAR_READ_CON

Description:
Expected maximum number of constraints to be read. The option is only used by fast MPS and LP file readers.

Possible Values:
Any number between 0 and +inf.

Default value:
10000

- read_cone

Corresponding constant:
MSK_IPAR_READ_CONE

Description:
Expected maximum number of conic constraints to be read. The option is used only by fast MPS and LP file readers.

Possible Values:
Any number between 0 and +inf.

Default value:
2500

- read_data_compressed

Corresponding constant:
MSK_IPAR_READ_DATA_COMPRESSED

Description:
If this option is turned on,it is assumed that the data file is compressed.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- read_data_format

Corresponding constant:
MSK_IPAR_READ_DATA_FORMAT

Description:
Format of the data file to be read.

Possible values:

`MSK_DATA_FORMAT_XML` The data file is an XML formatted file.

`MSK_DATA_FORMAT_FREE_MPS` The data data a free MPS formatted file.

`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_MBT` The data file is a MOSEK binary task file.

`MSK_DATA_FORMAT_OP` The data file is an optimization problem formatted file.

Default value:
`MSK_DATA_FORMAT_EXTENSION`

- `read_keep_free_con`

Corresponding constant:
`MSK_IPAR_READ_KEEP_FREE_CON`

Description:
Controls whether the free constraints are included in the problem.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_OFF`

- `read_lp_drop_new_vars_in_bou`

Corresponding constant:
`MSK_IPAR_READ_LP_DROP_NEW_VARS_IN_BOU`

Description:
If this option is turned on, MOSEK will drop variables that are defined for the first time in the bounds section.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_OFF`

- `read_lp_quoted_names`

Corresponding constant:
`MSK_IPAR_READ_LP_QUOTED_NAMES`

Description:
If a name is in quotes when reading an LP file, the quotes will be removed.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `read_mps_format`

Corresponding constant:
`MSK_IPAR_READ_MPS_FORMAT`

Description:
Controls how strictly the MPS file reader interprets the MPS format.

Possible values:

`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.

Default value:
`MSK_MPS_FORMAT_RELAXED`

- `read_mps_keep_int`

Corresponding constant:
`MSK_IPAR_READ_MPS_KEEP_INT`

Description:
Controls whether MOSEK should keep the integer restrictions on the variables while reading the MPS file.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `read_mps_obj_sense`

Corresponding constant:
`MSK_IPAR_READ_MPS_OBJ_SENSE`

Description:
If turned on, the MPS reader uses the objective sense section. Otherwise the MPS reader ignores it.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `read_mps_quoted_names`

  Corresponding constant:
  `MSK_IPAR_READ_MPS_QUOTED_NAMES`

  Description:
  If a name is in quotes when reading an MPS file, then the quotes will be removed.

  Possible values:

  `MSK_ON` Switch the option on.

  `MSK_OFF` Switch the option off.

  Default value:
  `MSK_ON`

- `read_mps_relax`

  Corresponding constant:
  `MSK_IPAR_READ_MPS_RELAX`

  Description:
  If this option is turned on, then mixed integer constraints are ignored when a problem is read.

  Possible values:

  `MSK_ON` Switch the option on.

  `MSK_OFF` Switch the option off.

  Default value:
  `MSK_ON`

- `read_mps_width`

  Corresponding constant:
  `MSK_IPAR_READ_MPS_WIDTH`

  Description:
  Controls the maximal number of characters allowed in one line of the MPS file.

  Possible Values:
  Any positive number greater than 80.

  Default value:
  `1024`

- `read_q_mode`

Corresponding constant:
MSK_IPAR_READ_Q_MODE

Description:
Controls how the Q matrices are read from the MPS file.

Possible values:

MSK_Q_READ_ADD All elements in a Q matrix are assumed to belong to the lower triangular part. Duplicate elements in a Q matrix are added together.

MSK_Q_READ_DROP_LOWER All elements in the strict lower triangular part of the Q matrices are dropped.

MSK_Q_READ_DROP_UPPER All elements in the strict upper triangular part of the Q matrices are dropped.

Default value:
MSK_Q_READ_ADD

- read_qnz

Corresponding constant:
MSK_IPAR_READ_QNZ

Description:
Expected maximum number of Q non-zeros to be read. The option is used only by MPS and LP file readers.

Possible Values:
Any number between 0 and +inf.

Default value:
20000

- read_task_ignore_param

Corresponding constant:
MSK_IPAR_READ_TASK_IGNORE_PARAM

Description:
Controls whether MOSEK should ignore the parameter setting defined in the task file and use the default parameter setting instead.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- read_var

Corresponding constant:
MSK_IPAR_READ_VAR

Description:
Expected maximum number of variable to be read. The option is used only by MPS and LP file readers.

Possible Values:
Any number between 0 and +inf.

Default value:
10000

- sensitivity_all

Corresponding constant:
MSK_IPAR_SENSITIVITY_ALL

Description:

If set to MSK_ON, then MSK_sensitivityreport analyzes all bounds and variables instead of reading a specification from the file.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- sensitivity_optimizer

Corresponding constant:
MSK_IPAR_SENSITIVITY_OPTIMIZER

Description:
Controls which optimizer is used for optimal partition sensitivity analysis.

Possible values:

MSK_OPTIMIZER_INTPNT The interior-point optimizer is used.

MSK_OPTIMIZER_CONCURRENT The optimizer for nonconvex nonlinear problems.

MSK_OPTIMIZER_MIXED_INT The mixed-integer optimizer.

MSK_OPTIMIZER_DUAL_SIMPLEX The dual simplex optimizer is used.

MSK_OPTIMIZER_FREE The optimizer is chosen automatically.

MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX The primal dual simplex optimizer is used.

MSK_OPTIMIZER_CONIC The optimizer for problems having conic constraints.

MSK_OPTIMIZER_NONCONVEX The optimizer for nonconvex nonlinear problems.

MSK_OPTIMIZER_QCONE For internal use only.

MSK_OPTIMIZER_PRIMAL_SIMPLEX The primal simplex optimizer is used.

MSK_OPTIMIZER_FREE_SIMPLEX One of the simplex optimizers is used.

Default value:
MSK_OPTIMIZER_FREE_SIMPLEX

- `sensitivity_type`

  Corresponding constant:
  `MSK_IPAR_SENSITIVITY_TYPE`

  Description:
  Controls which type of sensitivity analysis is to be performed.

  Possible values:

  `MSK_SENSITIVITY_TYPE_OPTIMAL_PARTITION` Optimal partition sensitivity analysis is performed.

  `MSK_SENSITIVITY_TYPE_BASIS` Basis sensitivity analysis is performed.

  Default value:
  `MSK_SENSITIVITY_TYPE_BASIS`

- `sim_basis_factor_use`

  Corresponding constant:
  `MSK_IPAR_SIM_BASIS_FACTOR_USE`

  Description:
  Controls whether a (LU) factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penanlty.

  Possible values:

  `MSK_ON` Switch the option on.

  `MSK_OFF` Switch the option off.

  Default value:
  `MSK_ON`

- `sim_degen`

  Corresponding constant:
  `MSK_IPAR_SIM_DEGEN`

  Description:
  Controls how aggressively degeneration is handled.

  Possible values:

  `MSK_SIM_DEGEN_NONE` The simplex optimizer should use no 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.

  `MSK_SIM_DEGEN_AGGRESSIVE` The simplex optimizer should use an aggressive degeneration strategy.

  `MSK_SIM_DEGEN_FREE` The simplex optimizer chooses the degeneration strategy.

Default value:
MSK_SIM_DEGEN_FREE

- sim_dual_crash

Corresponding constant:
MSK_IPAR_SIM_DUAL_CRASH

Description:
Controls whether crashing is performed in the dual simplex optimizer.

In general if a basis consists of more than (100-this parameter value)% fixed variables, then a crash will be performed.

Possible Values:
Any number between 0 and +inf.

Default value:
90

- sim_dual_phaseone_method

Corresponding constant:
MSK_IPAR_SIM_DUAL_PHASEONE_METHOD

Description:
An exprimental feature.

Possible Values:
Any number between 0 and 10.

Default value:
0

- sim_dual_restrict_selection

Corresponding constant:
MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION

Description:
The dual simplex optimizer can use a so-called restricted selection/pricing strategy to chooses the outgoing variable. Hence, if restricted selection is applied, then the dual simplex optimizer first choose a subset of all the potential outgoing variables. Next, for some time it will choose the outgoing variable only among the subset. From time to time the subset is redefined.

A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Possible Values:
Any number between 0 and 100.

Default value:
50

- sim_dual_selection

Corresponding constant:
MSK_IPAR_SIM_DUAL_SELECTION

Description:
Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer.

Possible values:

MSK_SIM_SELECTION_FULL The optimizer uses full pricing.

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.

MSK_SIM_SELECTION_FREE The optimizer chooses the pricing strategy.

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).

Default value:
MSK_SIM_SELECTION_FREE

- sim_exploit_dupvec

Corresponding constant:
MSK_IPAR_SIM_EXPLOIT_DUPVEC

Description:
Controls if the simplex optimizers are allowed to exploit duplicated columns.

Possible values:

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.

Default value:
MSK_SIM_EXPLOIT_DUPVEC_OFF

- sim_hotstart

Corresponding constant:
MSK_IPAR_SIM_HOTSTART

Description:
Controls the type of hot-start that the simplex optimizer perform.

Possible values:

MSK_SIM_HOTSTART_NONE The simplex optimizer performs a coldstart.

MSK_SIM_HOTSTART_STATUS_KEYS Only the status keys of the constraints and variables are used to choose the type of hot-start.

MSK_SIM_HOTSTART_FREE The simplex optimize chooses the hot-start type.

Default value:
MSK_SIM_HOTSTART_FREE

- sim_hotstart_lu

Corresponding constant:
MSK_IPAR_SIM_HOTSTART_LU

Description:
Determines if the simplex optimizer should exploit the initial factorization.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- sim_integer

Corresponding constant:
MSK_IPAR_SIM_INTEGER

Description:
An exprimental feature.

Possible Values:
Any number between 0 and 10.

Default value:
0

- sim_max_iterations

Corresponding constant:
MSK_IPAR_SIM_MAX_ITERATIONS

Description:
Maximum number of iterations that can be used by a simplex optimizer.

Possible Values:
Any number between 0 and +inf.

Default value:
10000000

- sim_max_num_setbacks

Corresponding constant:
MSK_IPAR_SIM_MAX_NUM_SETBACKS

Description:
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.

Possible Values:
Any number between 0 and +inf.

Default value:
`250`

- `sim_network_detect`

  Corresponding constant:
  `MSK_IPAR_SIM_NETWORK_DETECT`

  Description:
  The simplex optimizer is capable of exploiting a network flow component in a problem. However it is only worthwhile to exploit the network flow component if it is sufficiently large. This parameter controls how large the network component has to be in "relative" terms before it is exploited. For instance a value of 20 means at least 20% of the model should be a network before it is exploited. If this value is larger than 100 the network flow component is never detected or exploited.

  Possible Values:
  Any number between 0 and +inf.

  Default value:
  `101`

- `sim_network_detect_hotstart`

  Corresponding constant:
  `MSK_IPAR_SIM_NETWORK_DETECT_HOTSTART`

  Description:
  This parameter controls has large the network component in "relative" terms has to be before it is exploited in a simplex hot-start. The network component should be equal or larger than

  ```
  max(MSK_IPAR_SIM_NETWORK_DETECT,MSK_IPAR_SIM_NETWORK_DETECT_HOTSTART)
  ```

  before it is exploited. If this value is larger than 100 the network flow component is never detected or exploited.

  Possible Values:
  Any number between 0 and +inf.

  Default value:
  `100`

- `sim_network_detect_method`

  Corresponding constant:
  `MSK_IPAR_SIM_NETWORK_DETECT_METHOD`

  Description:
  Controls which type of detection method the network extraction should use.

  Possible values:

MSK_NETWORK_DETECT_SIMPLE The network detection should use a very simple heuristic.

MSK_NETWORK_DETECT_ADVANCED The network detection should use a more advanced heuristic.

MSK_NETWORK_DETECT_FREE The network detection is free.

Default value:
MSK_NETWORK_DETECT_FREE

- sim_non_singular

Corresponding constant:
MSK_IPAR_SIM_NON_SINGULAR

Description:
Controls if the simplex optimizer ensures a non-singular basis, if possible.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- sim_primal_crash

Corresponding constant:
MSK_IPAR_SIM_PRIMAL_CRASH

Description:
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.

Possible Values:
Any nonnegative integer value.

Default value:
90

- sim_primal_phaseone_method

Corresponding constant:
MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD

Description:
An exprimental feature.

Possible Values:
Any number between 0 and 10.

Default value:
0

- sim_primal_restrict_selection

Corresponding constant:
MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION

Description:
The primal simplex optimizer can use a so-called restricted selection/pricing strategy to chooses the outgoing variable. Hence, if restricted selection is applied, then the primal simplex optimizer first choose a subset of all the potential incoming variables. Next, for some time it will choose the incoming variable only among the subset. From time to time the subset is redefined.

A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Possible Values:
Any number between 0 and 100.

Default value:
50

- sim_primal_selection

Corresponding constant:
MSK_IPAR_SIM_PRIMAL_SELECTION

Description:
Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.

Possible values:

MSK_SIM_SELECTION_FULL The optimizer uses full pricing.

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.

MSK_SIM_SELECTION_FREE The optimizer chooses the pricing strategy.

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).

Default value:
MSK_SIM_SELECTION_FREE

- sim_refactor_freq

Corresponding constant:
MSK_IPAR_SIM_REFACTOR_FREQ

Description:
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.

Possible Values:
Any number between 0 and +inf.

Default value:
0

- sim_reformulation

Corresponding constant:
MSK_IPAR_SIM_REFORMULATION

Description:
Controls if the simplex optimizers are allowed to reformulate the problem.

Possible values:

MSK_SIM_REFORMULATION_ON Allow the simplex optimizer to reformulate the problem.

MSK_SIM_REFORMULATION_AGGRESSIVE The simplex optimizer should use an aggressive reformulation strategy.

MSK_SIM_REFORMULATION_OFF Disallow the simplex optimizer to reformulate the problem.

MSK_SIM_REFORMULATION_FREE The simplex optimizer can choose freely.

Default value:
MSK_SIM_REFORMULATION_OFF

- sim_save_lu

Corresponding constant:
MSK_IPAR_SIM_SAVE_LU

Description:
Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- sim_scaling

Corresponding constant:
MSK_IPAR_SIM_SCALING

Description:
Controls how much effort is used in scaling the problem before a simplex optimizer is used.

Possible values:

MSK_SCALING_NONE No scaling is performed.

MSK_SCALING_MODERATE A conservative scaling is performed.

MSK_SCALING_AGGRESSIVE A very aggressive scaling is performed.

MSK_SCALING_FREE The optimizer chooses the scaling heuristic.

Default value:
MSK_SCALING_FREE

- sim_scaling_method

Corresponding constant:
MSK_IPAR_SIM_SCALING_METHOD

Description:
Controls how the problem is scaled before a simplex optimizer is used.

Possible values:

MSK_SCALING_METHOD_POW2 Scales only with power of 2 leaving the mantissa untouched.
MSK_SCALING_METHOD_FREE The optimizer chooses the scaling heuristic.

Default value:
MSK_SCALING_METHOD_POW2

- sim_solve_form

Corresponding constant:
MSK_IPAR_SIM_SOLVE_FORM

Description:
Controls whether the primal or the dual problem is solved by the primal-/dual- simplex optimizer.

Possible values:

MSK_SOLVE_PRIMAL The optimizer should solve the primal problem.
MSK_SOLVE_DUAL The optimizer should solve the dual problem.
MSK_SOLVE_FREE The optimizer is free to solve either the primal or the dual problem.

Default value:
MSK_SOLVE_FREE

- sim_stability_priority

Corresponding constant:
MSK_IPAR_SIM_STABILITY_PRIORITY

Description:
Controls how high priority the numerical stability should be given.

Possible Values:
Any number between 0 and 100.

Default value:
50

- sim_switch_optimizer

Corresponding constant:
MSK_IPAR_SIM_SWITCH_OPTIMIZER

Description:
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).

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_OFF`

- `sol_filter_keep_basic`

Corresponding constant:
`MSK_IPAR_SOL_FILTER_KEEP_BASIC`

Description:
If turned on, then basic and super basic constraints and variables are written to the solution file independent of the filter setting.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_OFF`

- `sol_filter_keep_ranged`

Corresponding constant:
`MSK_IPAR_SOL_FILTER_KEEP_RANGED`

Description:
If turned on, then ranged constraints and variables are written to the solution file independent of the filter setting.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_OFF`

- `sol_quoted_names`

Corresponding constant:
`MSK_IPAR_SOL_QUOTED_NAMES`

Description:
If this options is turned on, then MOSEK will quote names that contains blanks while writing the solution file. Moreover when reading leading and trailing quotes will be stripped of.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_OFF`

- `sol_read_name_width`

Corresponding constant:
`MSK_IPAR_SOL_READ_NAME_WIDTH`

Description:
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.

Possible Values:
Any number between -inf and +inf.

Default value:
`-1`

- `sol_read_width`

Corresponding constant:
`MSK_IPAR_SOL_READ_WIDTH`

Description:
Controls the maximal acceptable width of line in the solutions when read by MOSEK.

Possible Values:
Any positive number greater than 80.

Default value:
`1024`

- `solution_callback`

Corresponding constant:
`MSK_IPAR_SOLUTION_CALLBACK`

Description:
Indicates whether solution call-backs will be performed during the optimization.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_OFF`

- `timing_level`

Corresponding constant:
`MSK_IPAR_TIMING_LEVEL`

Description:
Controls the a amount of timing performed inside MOSEK.

Possible Values:
Any integer greater or equal to 0.

Default value:
`1`

- `warning_level`

Corresponding constant:
`MSK_IPAR_WARNING_LEVEL`

Description:
Warning level.

Possible Values:
Any number between 0 and +inf.

Default value:
`1`

- `write_bas_constraints`

Corresponding constant:
`MSK_IPAR_WRITE_BAS_CONSTRAINTS`

Description:
Controls whether the constraint section is written to the basic solution file.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `write_bas_head`

Corresponding constant:
`MSK_IPAR_WRITE_BAS_HEAD`

Description:
Controls whether the header section is written to the basic solution file.

Possible values:

`MSK_ON` Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- write_bas_variables

Corresponding constant:
MSK_IPAR_WRITE_BAS_VARIABLES

Description:
Controls whether the variables section is written to the basic solution file.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- write_data_compressed

Corresponding constant:
MSK_IPAR_WRITE_DATA_COMPRESSED

Description:
Controls whether the data file is compressed while it is written. 0 means no compression while higher values mean more compression.

Possible Values:
Any number between 0 and +inf.

Default value:
0

- write_data_format

Corresponding constant:
MSK_IPAR_WRITE_DATA_FORMAT

Description:

Controls the data format when a task is written using MSK_writedata.

Possible values:

MSK_DATA_FORMAT_XML The data file is an XML formatted file.

MSK_DATA_FORMAT_FREE_MPS The data data a free MPS formatted file.

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_MBT The data file is a MOSEK binary task file.

MSK_DATA_FORMAT_OP The data file is an optimization problem formatted file.

Default value:
MSK_DATA_FORMAT_EXTENSION

- write_data_param

Corresponding constant:
MSK_IPAR_WRITE_DATA_PARAM

Description:
If this option is turned on the parameter settings are written to the data file as parameters.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- write_free_con

Corresponding constant:
MSK_IPAR_WRITE_FREE_CON

Description:
Controls whether the free constraints are written to the data file.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- write_generic_names

Corresponding constant:
MSK_IPAR_WRITE_GENERIC_NAMES

Description:
Controls whether the generic names or user-defined names are used in the data file.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- write_generic_names_io

Corresponding constant:
MSK_IPAR_WRITE_GENERIC_NAMES_IO

Description:
Index origin used in generic names.

Possible Values:
Any number between 0 and +inf.

Default value:
`1`

- `write_int_constraints`

Corresponding constant:
`MSK_IPAR_WRITE_INT_CONSTRAINTS`

Description:
Controls whether the constraint section is written to the integer solution file.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `write_int_head`

Corresponding constant:
`MSK_IPAR_WRITE_INT_HEAD`

Description:
Controls whether the header section is written to the integer solution file.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `write_int_variables`

Corresponding constant:
`MSK_IPAR_WRITE_INT_VARIABLES`

Description:
Controls whether the variables section is written to the integer solution file.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `write_lp_line_width`

Corresponding constant:
MSK_IPAR_WRITE_LP_LINE_WIDTH

Description:
Maximum width of line in an LP file written by MOSEK.

Possible Values:
Any positive number.

Default value:
80

- write_lp_quoted_names

Corresponding constant:
MSK_IPAR_WRITE_LP_QUOTED_NAMES

Description:
If this option is turned on, then MOSEK will quote invalid LP names when writing an LP file.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- write_lp_strict_format

Corresponding constant:
MSK_IPAR_WRITE_LP_STRICT_FORMAT

Description:
Controls whether LP output files satisfy the LP format strictly.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_OFF

- write_lp_terms_per_line

Corresponding constant:
MSK_IPAR_WRITE_LP_TERMS_PER_LINE

Description:
Maximum number of terms on a single line in an LP file written by MOSEK. 0 means unlimited.

Possible Values:
Any number between 0 and +inf.

Default value:
10

- write_mps_int

Corresponding constant:
MSK_IPAR_WRITE_MPS_INT

Description:
Controls if marker records are written to the MPS file to indicate whether variables are integer restricted.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- write_mps_obj_sense

Corresponding constant:
MSK_IPAR_WRITE_MPS_OBJ_SENSE

Description:
If turned off, the objective sense section is not written to the MPS file.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- write_mps_quoted_names

Corresponding constant:
MSK_IPAR_WRITE_MPS_QUOTED_NAMES

Description:
If a name contains spaces (blanks) when writing an MPS file, then the quotes will be removed.

Possible values:

MSK_ON Switch the option on.

MSK_OFF Switch the option off.

Default value:
MSK_ON

- write_mps_strict

Corresponding constant:
MSK_IPAR_WRITE_MPS_STRICT

Description:
Controls whether the written MPS file satisfies the MPS format strictly or not.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_OFF`

- `write_precision`

Corresponding constant:
`MSK_IPAR_WRITE_PRECISION`

Description:
Controls the precision with which `double` numbers are printed in the MPS data file. In general it is not worthwhile to use a value higher than 15.

Possible Values:
Any number between 0 and +inf.

Default value:
`8`

- `write_sol_constraints`

Corresponding constant:
`MSK_IPAR_WRITE_SOL_CONSTRAINTS`

Description:
Controls whether the constraint section is written to the solution file.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `write_sol_head`

Corresponding constant:
`MSK_IPAR_WRITE_SOL_HEAD`

Description:
Controls whether the header section is written to the solution file.

Possible values:

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

Default value:
`MSK_ON`

- `write_sol_variables`

  Corresponding constant:
  `MSK_IPAR_WRITE_SOL_VARIABLES`

  Description:
  Controls whether the variables section is written to the solution file.

  Possible values:

  `MSK_ON` Switch the option on.

  `MSK_OFF` Switch the option off.

  Default value:
  `MSK_ON`

- `write_task_inc_sol`

  Corresponding constant:
  `MSK_IPAR_WRITE_TASK_INC_SOL`

  Description:
  Controls whether the solutions are stored in the task file too.

  Possible values:

  `MSK_ON` Switch the option on.

  `MSK_OFF` Switch the option off.

  Default value:
  `MSK_ON`

- `write_xml_mode`

  Corresponding constant:
  `MSK_IPAR_WRITE_XML_MODE`

  Description:
  Controls if linear coefficients should be written by row or column when writing in the XML file format.

  Possible values:

  `MSK_WRITE_XML_MODE_COL` Write in column order.

  `MSK_WRITE_XML_MODE_ROW` Write in row order.

  Default value:
  `MSK_WRITE_XML_MODE_ROW`

16.4 String parameter types

- `bas_sol_file_name`

  Corresponding constant:
  `MSK_SPAR_BAS_SOL_FILE_NAME`

  Description:
  Name of the `bas` solution file.

  Possible Values:
  Any valid file name.

  Default value:
  `""`

- `data_file_name`

  Corresponding constant:
  `MSK_SPAR_DATA_FILE_NAME`

  Description:
  Data are read and written to this file.

  Possible Values:
  Any valid file name.

  Default value:
  `""`

- `debug_file_name`

  Corresponding constant:
  `MSK_SPAR_DEBUG_FILE_NAME`

Description:
MOSEK debug file.

Possible Values:
Any valid file name.

Default value:
""

- `feasrepair_name_prefix`

Corresponding constant:
`MSK_SPAR_FEASREPAIR_NAME_PREFIX`

Description:

If the function `MSK_relaxprimal` adds new constraints to the problem, then they are prefixed by the value of this parameter.

Possible Values:
Any valid string.

Default value:
`"MSK-"`

- `feasrepair_name_separator`

Corresponding constant:
`MSK_SPAR_FEASREPAIR_NAME_SEPARATOR`

Description:

Separator string for names of constraints and variables generated by `MSK_relaxprimal`.

Possible Values:
Any valid string.

Default value:
`"-"`

- `feasrepair_name_wsumviol`

Corresponding constant:
`MSK_SPAR_FEASREPAIR_NAME_WSUMVIOL`

Description:
The constraint and variable associated with the total weighted sum of violations are each given the name of this parameter postfixed with `CON` and `VAR` respectively.

Possible Values:
Any valid string.

Default value:
`"WSUMVIOL"`

- `int_sol_file_name`

Corresponding constant:
MSK_SPAR_INT_SOL_FILE_NAME

Description:
Name of the `int` solution file.

Possible Values:
Any valid file name.

Default value:
""

- `itr_sol_file_name`

Corresponding constant:
MSK_SPAR_ITR_SOL_FILE_NAME

Description:
Name of the `itr` solution file.

Possible Values:
Any valid file name.

Default value:
""

- `param_comment_sign`

Corresponding constant:
MSK_SPAR_PARAM_COMMENT_SIGN

Description:
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.

Possible Values:
Any valid string.

Default value:
"%%"

- `param_read_file_name`

Corresponding constant:
MSK_SPAR_PARAM_READ_FILE_NAME

Description:
Modifications to the parameter database is read from this file.

Possible Values:
Any valid file name.

Default value:
""

- `param_write_file_name`

Corresponding constant:
MSK_SPAR_PARAM_WRITE_FILE_NAME

Description:
The parameter database is written to this file.

Possible Values:
Any valid file name.

Default value:
""

- read_mps_bou_name

Corresponding constant:
MSK_SPAR_READ_MPS_BOU_NAME

Description:
Name of the BOUNDS vector used. An empty name means that the first BOUNDS vector is used.

Possible Values:
Any valid MPS name.

Default value:
""

- read_mps_obj_name

Corresponding constant:
MSK_SPAR_READ_MPS_OBJ_NAME

Description:
Name of the free constraint used as objective function. An empty name means that the first constraint is used as objective function.

Possible Values:
Any valid MPS name.

Default value:
""

- read_mps_ran_name

Corresponding constant:
MSK_SPAR_READ_MPS_RAN_NAME

Description:
Name of the RANGE vector used. An empty name means that the first RANGE vector is used.

Possible Values:
Any valid MPS name.

Default value:
""

- `read_mps_rhs_name`

  Corresponding constant:
  `MSK_SPAR_READ_MPS_RHS_NAME`

  Description:
  Name of the RHS used. An empty name means that the first RHS vector is used.

  Possible Values:
  Any valid MPS name.

  Default value:
  `""`

- `sensitivity_file_name`

  Corresponding constant:
  `MSK_SPAR_SENSITIVITY_FILE_NAME`

  Description:

  If defined `MSK_sensitivityreport` reads this file as a sensitivity analysis data file specifying the type of analysis to be done.

  Possible Values:
  Any valid string.

  Default value:
  `""`

- `sensitivity_res_file_name`

  Corresponding constant:
  `MSK_SPAR_SENSITIVITY_RES_FILE_NAME`

  Description:

  If this is a nonempty string, then `MSK_sensitivityreport` writes results to this file.

  Possible Values:
  Any valid string.

  Default value:
  `""`

- `sol_filter_xc_low`

  Corresponding constant:
  `MSK_SPAR_SOL_FILTER_XC_LOW`

  Description:
  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.

Possible Values:
Any valid filter.

Default value:
""

- sol_filter_xc_upr

Corresponding constant:
MSK_SPAR_SOL_FILTER_XC_UPR

Description:
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.

Possible Values:
Any valid filter.

Default value:
""

- sol_filter_xx_low

Corresponding constant:
MSK_SPAR_SOL_FILTER_XX_LOW

Description:
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.

Possible Values:
Any valid filter..

Default value:
""

- sol_filter_xx_upr

Corresponding constant:
MSK_SPAR_SOL_FILTER_XX_UPR

Description:
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.

Possible Values:
Any valid file name.

Default value:
""

- `stat_file_name`

  Corresponding constant:
  `MSK_SPAR_STAT_FILE_NAME`

  Description:
  Statistics file name.

  Possible Values:
  Any valid file name.

  Default value:
  ""

- `stat_key`

  Corresponding constant:
  `MSK_SPAR_STAT_KEY`

  Description:
  Key used when writing the summary file.

  Possible Values:
  Any valid XML string.

  Default value:
  ""

- `stat_name`

  Corresponding constant:
  `MSK_SPAR_STAT_NAME`

  Description:
  Name used when writing the statistics file.

  Possible Values:
  Any valid XML string.

  Default value:
  ""

- `write_lp_gen_var_name`

  Corresponding constant:
  `MSK_SPAR_WRITE_LP_GEN_VAR_NAME`

  Description:
  Sometimes when an LP file is written additional variables must be inserted. They will have the prefix denoted by this parameter.

  Possible Values:
  Any valid string.

  Default value:
  `"xmskgen"`

Chapter 17

Response codes

(0) MSK_RES_OK
No error occurred.

(50) MSK_RES_WRN_OPEN_PARAM_FILE
The parameter file could not be opened.

(51) MSK_RES_WRN_LARGE_BOUND
A numerically large bound value is specified.

(52) MSK_RES_WRN_LARGE_LO_BOUND
A numerically large lower bound value is specified.

(53) MSK_RES_WRN_LARGE_UP_BOUND
A numerically large upper bound value is specified.

(54) MSK_RES_WRN_LARGE_CON_FX
An equality constraint is fixed to a numerically large value. This can cause numerical problems.

(57) MSK_RES_WRN_LARGE_CJ
A numerically large value is specified for one c_j.

(62) MSK_RES_WRN_LARGE_AIJ
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.

(63) MSK_RES_WRN_ZERO_AIJ
One or more zero elements are specified in A.

(65) MSK_RES_WRN_NAME_MAX_LEN
A name is longer than the buffer that is supposed to hold it.

(66) MSK_RES_WRN_SPAR_MAX_LEN
A value for a string parameter is longer than the buffer that is supposed to hold it.

(70) MSK_RES_WRN_MPS_SPLIT_RHS_VECTOR
An RHS vector is split into several nonadjacent parts in an MPS file.

(71) MSK_RES_WRN_MPS_SPLIT_RAN_VECTOR
A RANGE vector is split into several nonadjacent parts in an MPS file.

(72) MSK_RES_WRN_MPS_SPLIT_BOU_VECTOR
A BOUNDS vector is split into several nonadjacent parts in an MPS file.

(80) MSK_RES_WRN_LP_OLD_QUAD_FORMAT
Missing '/2' after quadratic expressions in bound or objective.

(85) MSK_RES_WRN_LP_DROP_VARIABLE
Ignored a variable because the variable was not previously defined. Usually this implies that a variable appears in the bound section but not in the objective or the constraints.

(200) MSK_RES_WRN_NZ_IN_UPR_TRI
Non-zero elements specified in the upper triangle of a matrix were ignored.

(201) MSK_RES_WRN_DROPPED_NZ_QOBJ
One or more non-zero elements were dropped in the Q matrix in the objective.

(250) MSK_RES_WRN_IGNORE_INTEGER
Ignored integer constraints.

(251) MSK_RES_WRN_NO_GLOBAL_OPTIMIZER
No global optimizer is available.

(270) MSK_RES_WRN_MIO_INFEASIBLE_FINAL
The final mixed-integer problem with all the integer variables fixed at their optimal values is infeasible.

(300) MSK_RES_WRN_SOL_FILTER
Invalid solution filter is specified.

(350) MSK_RES_WRN_UNDEF_SOL_FILE_NAME
Undefined name occurred in a solution.

(351) MSK_RES_WRN_SOL_FILE_IGNORED_CON
One or more lines in the constraint section were ignored when reading a solution file.

(352) MSK_RES_WRN_SOL_FILE_IGNORED_VAR
One or more lines in the variable section were ignored when reading a solution file.

(400) MSK_RES_WRN_TOO_FEW_BASIS_VARS
An incomplete basis has been specified. Too few basis variables are specified.

(405) MSK_RES_WRN_TOO_MANY_BASIS_VARS
A basis with too many variables has been specified.

(500) MSK_RES_WRN_LICENSE_EXPIRE
The license expires.

(501) MSK_RES_WRN_LICENSE_SERVER
The license server is not responding.

(502) MSK_RES_WRN_EMPTY_NAME
A variable or constraint name is empty. The output file may be invalid.

(503) MSK_RES_WRN_USING_GENERIC_NAMES
The file writer reverts to generic names because a name is blank.

(505) MSK_RES_WRN_LICENSE_FEATURE_EXPIRE
The license expires.

(705) MSK_RES_WRN_ZEROS_IN_SPARSE_ROW
One or more (near) zero elements are specified in a sparse row of a matrix. It is redundant to specify zero elements. Hence it may indicate an error.

(710) MSK_RES_WRN_ZEROS_IN_SPARSE_COL
One or more (near) zero elements are specified in a sparse column of a matrix. It is redundant to specify zero elements. Hence, it may indicate an error.

(800) MSK_RES_WRN_INCOMPLETE_LINEAR_DEPENDENCY_CHECK
The linear dependency check(s) was not completed and therefore the A matrix may contain linear dependencies.

(801) MSK_RES_WRN_ELIMINATOR_SPACE
The eliminator is skipped at least once due to lack of space.

(802) MSK_RES_WRN_PRESOLVE_OUTOFSPACE
The presolve is incomplete due to lack of space.

(803) MSK_RES_WRN_PRESOLVE_BAD_PRECISION
The presolve estimates that the model is specified with insufficient precision.

(804) MSK_RES_WRN_WRITE_DISCARDED_CFIX
The fixed objective term could not be converted to a variable and was discarded in the output file.

(805) MSK_RES_WRN_CONSTRUCT_SOLUTION_INFEAS
After fixing the integer variables at the suggested values then the problem is infeasible.

(807) MSK_RES_WRN_CONSTRUCT_INVALID_SOL_ITG
The intial value for one or more of the integer variables is not feasible.

(810) MSK_RES_WRN_CONSTRUCT_NO_SOL_ITG
The construct solution requires an integer solution.

(900) MSK_RES_WRN_ANA_LARGE_BOUNDS
This warning is issued by the problem analyzer, if one or more constraint or variable bounds are very large. One should consider omitting these bounds entirely by setting them to +inf or -inf.

(901) MSK_RES_WRN_ANA_C_ZERO
This warning is issued by the problem analyzer, if the coefficients in the linear part of the objective are all zero.

(902) MSK_RES_WRN_ANA_EMPTY_COLS
This warning is issued by the problem analyzer, if columns, in which all coefficients are zero, are found.

(903) MSK_RES_WRN_ANA_CLOSE_BOUNDS
This warning is issued by problem analyzer, if ranged constraints or variables with very close upper and lower bounds are detected. One should consider treating such constraints as equalities and such variables as constants.

(904) MSK_RES_WRN_ANA_ALMOST_INT_BOUNDS
This warning is issued by the problem analyzer if a constraint is bound nearly integral.

(1000) MSK_RES_ERR_LICENSE
Invalid license.

(1001) MSK_RES_ERR_LICENSE_EXPIRED
The license has expired.

(1002) MSK_RES_ERR_LICENSE_VERSION
The license is valid for another version of MOSEK.

(1005) MSK_RES_ERR_SIZE_LICENSE
The problem is bigger than the license.

(1006) MSK_RES_ERR_PROB_LICENSE
The software is not licensed to solve the problem.

(1007) MSK_RES_ERR_FILE_LICENSE
Invalid license file.

(1008) MSK_RES_ERR_MISSING_LICENSE_FILE
MOSEK cannot find the license file or license server. Usually this happens if the operating system variable MOSEKLM_LICENSE_FILE is not set up appropriately. Please see the MOSEK installation manual for details.

(1010) MSK_RES_ERR_SIZE_LICENSE_CON
The problem has too many constraints to be solved with the available license.

(1011) MSK_RES_ERR_SIZE_LICENSE_VAR
The problem has too many variables to be solved with the available license.

(1012) MSK_RES_ERR_SIZE_LICENSE_INTVAR
The problem contains too many integer variables to be solved with the available license.

(1013) MSK_RES_ERR_OPTIMIZER_LICENSE
The optimizer required is not licensed.

(1014) MSK_RES_ERR_FLEXLM
The FLEXlm license manager reported an error.

(1015) MSK_RES_ERR_LICENSE_SERVER
The license server is not responding.

(1016) MSK_RES_ERR_LICENSE_MAX
Maximum number of licenses is reached.

(1017) MSK_RES_ERR_LICENSE_MOSEKLM_DAEMON
The MOSEKLM license manager daemon is not up and running.

(1018) MSK_RES_ERR_LICENSE_FEATURE
A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.

(1019) MSK_RES_ERR_PLATFORM_NOT_LICENSED
A requested license feature is not available for the required platform.

(1020) MSK_RES_ERR_LICENSE_CANNOT_ALLOCATE
The license system cannot allocate the memory required.

(1021) MSK_RES_ERR_LICENSE_CANNOT_CONNECT
MOSEK cannot connect to the license server. Most likely the license server is not up and running.

(1025) MSK_RES_ERR_LICENSE_INVALID_HOSTID
The host ID specified in the license file does not match the host ID of the computer.

(1026) MSK_RES_ERR_LICENSE_SERVER_VERSION
The version specified in the checkout request is greater than the highest version number the daemon supports.

(1027) MSK_RES_ERR_LICENSE_NO_SERVER_SUPPORT
The license server does not support the requested feature. Possible reasons for this error include:

- The feature has expired.
- The feature's start date is later than today's date.
- The version requested is higher than feature's the highest supported version.
- A corrupted license file.

Try restarting the license and inspect the license server debug file, usually called `lmgrd.log`.

(1030) MSK_RES_ERR_OPEN_DL
A dynamic link library could not be opened.

(1035) MSK_RES_ERR_OLDER_DLL
The dynamic link library is older than the specified version.

(1036) MSK_RES_ERR_NEWER_DLL
The dynamic link library is newer than the specified version.

(1040) MSK_RES_ERR_LINK_FILE_DLL
A file cannot be linked to a stream in the DLL version.

(1045) MSK_RES_ERR_THREAD_MUTEX_INIT
Could not initialize a mutex.

(1046) MSK_RES_ERR_THREAD_MUTEX_LOCK
Could not lock a mutex.

(1047) MSK_RES_ERR_THREAD_MUTEX_UNLOCK
Could not unlock a mutex.

(1048) MSK_RES_ERR_THREAD_CREATE
Could not create a thread. This error may occur if a large number of environments are created and not deleted again. In any case it is a good practice to minimize the number of environments created.

(1049) MSK_RES_ERR_THREAD_COND_INIT
Could not initialize a condition.

(1050) MSK_RES_ERR_UNKNOWN
Unknown error.

(1051) MSK_RES_ERR_SPACE
Out of space.

(1052) MSK_RES_ERR_FILE_OPEN
Error while opening a file.

(1053) MSK_RES_ERR_FILE_READ
File read error.

(1054) MSK_RES_ERR_FILE_WRITE
File write error.

(1055) MSK_RES_ERR_DATA_FILE_EXT
The data file format cannot be determined from the file name.

(1056) MSK_RES_ERR_INVALID_FILE_NAME
An invalid file name has been specified.

(1057) MSK_RES_ERR_INVALID_SOL_FILE_NAME
An invalid file name has been specified.

(1058) MSK_RES_ERR_INVALID_MBT_FILE
A MOSEK binary task file is invalid.

(1059) MSK_RES_ERR_END_OF_FILE
End of file reached.

(1060) MSK_RES_ERR_NULL_ENV
`env` is a NULL pointer.

(1061) MSK_RES_ERR_NULL_TASK
`task` is a NULL pointer.

(1062) MSK_RES_ERR_INVALID_STREAM
An invalid stream is referenced.

(1063) MSK_RES_ERR_NO_INIT_ENV
`env` is not initialized.

(1064) MSK_RES_ERR_INVALID_TASK
The `task` is invalid.

(1065) MSK_RES_ERR_NULL_POINTER
An argument to a function is unexpectedly a NULL pointer.

(1066) MSK_RES_ERR_LIVING_TASKS
All tasks associated with an enviroment must be deleted before the environment is deleted. There are still some undeleted tasks.

(1070) MSK_RES_ERR_BLANK_NAME
An all blank name has been specified.

(1071) MSK_RES_ERR_DUP_NAME
The same name was used multiple times for the same problem item type.

(1075) MSK_RES_ERR_INVALID_OBJ_NAME
An invalid objective name is specified.

(1080) MSK_RES_ERR_SPACE_LEAKING
MOSEK is leaking memory. This can be due to either an incorrect use of MOSEK or a bug.

(1081) MSK_RES_ERR_SPACE_NO_INFO
No available information about the space usage.

(1090) MSK_RES_ERR_READ_FORMAT
The specified format cannot be read.

(1100) MSK_RES_ERR_MPS_FILE
An error occurred while reading an MPS file.

(1101) MSK_RES_ERR_MPS_INV_FIELD
A field in the MPS file is invalid. Probably it is too wide.

(1102) MSK_RES_ERR_MPS_INV_MARKER
An invalid marker has been specified in the MPS file.

(1103) MSK_RES_ERR_MPS_NULL_CON_NAME
An empty constraint name is used in an MPS file.

(1104) MSK_RES_ERR_MPS_NULL_VAR_NAME
An empty variable name is used in an MPS file.

(1105) MSK_RES_ERR_MPS_UNDEF_CON_NAME
An undefined constraint name occurred in an MPS file.

(1106) MSK_RES_ERR_MPS_UNDEF_VAR_NAME
An undefined variable name occurred in an MPS file.

(1107) MSK_RES_ERR_MPS_INV_CON_KEY
An invalid constraint key occurred in an MPS file.

(1108) MSK_RES_ERR_MPS_INV_BOUND_KEY
An invalid bound key occurred in an MPS file.

(1109) MSK_RES_ERR_MPS_INV_SEC_NAME
An invalid section name occurred in an MPS file.

(1110) MSK_RES_ERR_MPS_NO_OBJECTIVE
No objective is defined in an MPS file.

(1111) MSK_RES_ERR_MPS_SPLITTED_VAR
All elements in a column of the A matrix must be specified consecutively. Hence, it is illegal to specify non-zero elements in A for variable 1, then for variable 2 and then variable 1 again.

(1112) MSK_RES_ERR_MPS_MUL_CON_NAME
A constraint name was specified multiple times in the ROWS section.

(1113) MSK_RES_ERR_MPS_MUL_QSEC
Multiple QSECTIONs are specified for a constraint in the MPS data file.

(1114) MSK_RES_ERR_MPS_MUL_QOBJ
The Q term in the objective is specified multiple times in the MPS data file.

(1115) MSK_RES_ERR_MPS_INV_SEC_ORDER
The sections in the MPS data file are not in the correct order.

(1116) MSK_RES_ERR_MPS_MUL_CSEC
Multiple CSECTIONs are given the same name.

(1117) MSK_RES_ERR_MPS_CONE_TYPE
Invalid cone type specified in a CSECTION.

(1118) MSK_RES_ERR_MPS_CONE_OVERLAP
A variable is specified to be a member of several cones.

(1119) MSK_RES_ERR_MPS_CONE_REPEAT
A variable is repeated within the CSECTION.

(1122) MSK_RES_ERR_MPS_INVALID_OBJSENSE
An invalid objective sense is specified.

(1125) MSK_RES_ERR_MPS_TAB_IN_FIELD2
A tab char occurred in field 2.

(1126) MSK_RES_ERR_MPS_TAB_IN_FIELD3
A tab char occurred in field 3.

(1127) MSK_RES_ERR_MPS_TAB_IN_FIELD5
A tab char occurred in field 5.

(1128) MSK_RES_ERR_MPS_INVALID_OBJ_NAME
An invalid objective name is specified.

(1130) MSK_RES_ERR_ORD_INVALID_BRANCH_DIR
An invalid branch direction key is specified.

(1131) MSK_RES_ERR_ORD_INVALID
Invalid content in branch ordering file.

(1150) MSK_RES_ERR_LP_INCOMPATIBLE
The problem cannot be written to an LP formatted file.

(1151) MSK_RES_ERR_LP_EMPTY
The problem cannot be written to an LP formatted file.

(1152) MSK_RES_ERR_LP_DUP_SLACK_NAME
The name of the slack variable added to a ranged constraint already exists.

(1153) MSK_RES_ERR_WRITE_MPS_INVALID_NAME
An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.

(1154) MSK_RES_ERR_LP_INVALID_VAR_NAME
A variable name is invalid when used in an LP formatted file.

(1155) MSK_RES_ERR_LP_FREE_CONSTRAINT
Free constraints cannot be written in LP file format.

(1156) MSK_RES_ERR_WRITE_OPF_INVALID_VAR_NAME
Empty variable names cannot be written to OPF files.

(1157) MSK_RES_ERR_LP_FILE_FORMAT
Syntax error in an LP file.

(1158) MSK_RES_ERR_WRITE_LP_FORMAT
Problem cannot be written as an LP file.

(1159) MSK_RES_ERR_READ_LP_MISSING_END_TAG
Missing End tag in LP file.

(1160) MSK_RES_ERR_LP_FORMAT
Syntax error in an LP file.

(1161) MSK_RES_ERR_WRITE_LP_NON_UNIQUE_NAME
An auto-generated name is not unique.

(1162) MSK_RES_ERR_READ_LP_NONEXISTING_NAME
A variable never occurred in objective or constraints.

(1163) MSK_RES_ERR_LP_WRITE_CONIC_PROBLEM
The problem contains cones that cannot be written to an LP formatted file.

(1164) MSK_RES_ERR_LP_WRITE_GECO_PROBLEM
The problem contains general convex terms that cannot be written to an LP formatted file.

(1166) MSK_RES_ERR_WRITING_FILE
An error occurred while writing file

(1168) MSK_RES_ERR_OPF_FORMAT
Syntax error in an OPF file

(1169) MSK_RES_ERR_OPF_NEW_VARIABLE
Introducing new variables is now allowed. When a [variables] section is present, it is not allowed to introduce new variables later in the problem.

(1170) MSK_RES_ERR_INVALID_NAME_IN_SOL_FILE
An invalid name occurred in a solution file.

(1171) MSK_RES_ERR_LP_INVALID_CON_NAME
A constraint name is invalid when used in an LP formatted file.

(1172) MSK_RES_ERR_OPF_PREMATURE_EOF
Premature end of file in an OPF file.

(1197) MSK_RES_ERR_ARGUMENT_LENNEQ
Incorrect length of arguments.

(1198) MSK_RES_ERR_ARGUMENT_TYPE
Incorrect argument type.

(1199) MSK_RES_ERR_NR_ARGUMENTS
Incorrect number of function arguments.

(1200) MSK_RES_ERR_IN_ARGUMENT
A function argument is incorrect.

(1201) MSK_RES_ERR_ARGUMENT_DIMENSION
A function argument is of incorrect dimension.

(1203) MSK_RES_ERR_INDEX_IS_TOO_SMALL
An index in an argument is too small.

(1204) MSK_RES_ERR_INDEX_IS_TOO_LARGE
An index in an argument is too large.

(1205) MSK_RES_ERR_PARAM_NAME
The parameter name is not correct.

(1206) MSK_RES_ERR_PARAM_NAME_DOU
The parameter name is not correct for a double parameter.

(1207) MSK_RES_ERR_PARAM_NAME_INT
The parameter name is not correct for an integer parameter.

(1208) MSK_RES_ERR_PARAM_NAME_STR
The parameter name is not correct for a string parameter.

(1210) MSK_RES_ERR_PARAM_INDEX
Parameter index is out of range.

(1215) MSK_RES_ERR_PARAM_IS_TOO_LARGE
The parameter value is too large.

(1216) MSK_RES_ERR_PARAM_IS_TOO_SMALL
The parameter value is too small.

(1217) MSK_RES_ERR_PARAM_VALUE_STR
The parameter value string is incorrect.

(1218) MSK_RES_ERR_PARAM_TYPE
The parameter type is invalid.

(1219) MSK_RES_ERR_INF_DOU_INDEX
A double information index is out of range for the specified type.

(1220) MSK_RES_ERR_INF_INT_INDEX
An integer information index is out of range for the specified type.

(1221) MSK_RES_ERR_INDEX_ARR_IS_TOO_SMALL
An index in an array argument is too small.

(1222) MSK_RES_ERR_INDEX_ARR_IS_TOO_LARGE
An index in an array argument is too large.

(1225) MSK_RES_ERR_INF_LINT_INDEX
A long integer information index is out of range for the specified type.

(1230) MSK_RES_ERR_INF_DOU_NAME
A double information name is invalid.

(1231) MSK_RES_ERR_INF_INT_NAME
An integer information name is invalid.

(1232) MSK_RES_ERR_INF_TYPE
The information type is invalid.

(1234) MSK_RES_ERR_INF_LINT_NAME
A long integer information name is invalid.

(1235) MSK_RES_ERR_INDEX
An index is out of range.

(1236) MSK_RES_ERR_WHICHSOL
The solution defined by compwhichsol does not exists.

(1237) MSK_RES_ERR_SOLITEM
The solution item number `solitem` is invalid. Please note that MSK_SOL_ITEM_SNX is invalid for the basic solution.

(1238) MSK_RES_ERR_WHICHITEM_NOT_ALLOWED
`whichitem` is unacceptable.

(1240) MSK_RES_ERR_MAXNUMCON
The maximum number of constraints specified is smaller than the number of constraints in the task.

(1241) MSK_RES_ERR_MAXNUMVAR
The maximum number of variables specified is smaller than the number of variables in the task.

(1243) MSK_RES_ERR_MAXNUMQNZ
The maximum number of non-zeros specified for the Q matrices is smaller than the number of non-zeros in the current Q matrices.

(1250) MSK_RES_ERR_NUMCONLIM
Maximum number of constraints limit is exceeded.

(1251) MSK_RES_ERR_NUMVARLIM
Maximum number of variables limit is exceeded.

(1252) MSK_RES_ERR_TOO_SMALL_MAXNUMANZ
The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A.

(1253) MSK_RES_ERR_INV_APTRE
`aptre[j]` is strictly smaller than `aptrb[j]` for some `j`.

(1254) MSK_RES_ERR_MUL_A_ELEMENT
An element in A is defined multiple times.

(1255) MSK_RES_ERR_INV_BK
Invalid bound key.

(1256) MSK_RES_ERR_INV_BKC
Invalid bound key is specified for a constraint.

(1257) MSK_RES_ERR_INV_BKX
An invalid bound key is specified for a variable.

(1258) MSK_RES_ERR_INV_VAR_TYPE
An invalid variable type is specified for a variable.

(1259) MSK_RES_ERR_SOLVER_PROBTYPE
Problem type does not match the chosen optimizer.

(1260) MSK_RES_ERR_OBJECTIVE_RANGE
Empty objective range.

(1261) MSK_RES_ERR_FIRST
Invalid `first`.

(1262) MSK_RES_ERR_LAST
Invalid index `last`. A given index was out of expected range.

(1263) MSK_RES_ERR_NEGATIVE_SURPLUS
Negative surplus.

(1264) MSK_RES_ERR_NEGATIVE_APPEND
Cannot append a negative number.

(1265) MSK_RES_ERR_UNDEF_SOLUTION
MOSEK has the following solution types:

- an interior-point solution,
- an basic solution,
- and an integer solution.

Each optimizer may set one or more of these solutions; e.g by default a successful optimization with the interior-point optimizer defines the interior-point solution, and, for linear problems, also the basic solution. This error occurs when asking for a solution or for information about a solution that is not defined.

(1266) MSK_RES_ERR_BASIS
An invalid basis is specified. Either too many or too few basis variables are specified.

(1267) MSK_RES_ERR_INV_SKC
Invalid value in `skc`.

(1268) MSK_RES_ERR_INV_SKX
Invalid value in `skx`.

(1269) MSK_RES_ERR_INV_SK_STR
Invalid status key string encountered.

(1270) MSK_RES_ERR_INV_SK
Invalid status key code.

(1271) MSK_RES_ERR_INV_CONE_TYPE_STR
Invalid cone type string encountered.

(1272) MSK_RES_ERR_INV_CONE_TYPE
Invalid cone type code is encountered.

(1274) MSK_RES_ERR_INV_SKN
Invalid value in `skn`.

(1275) MSK_RES_ERR_INVALID_SURPLUS
Invalid surplus.

(1280) MSK_RES_ERR_INV_NAME_ITEM
An invalid name item code is used.

(1281) MSK_RES_ERR_PRO_ITEM
An invalid problem is used.

(1283) MSK_RES_ERR_INVALID_FORMAT_TYPE
Invalid format type.

(1285) MSK_RES_ERR_FIRSTI
Invalid `firsti`.

(1286) MSK_RES_ERR_LASTI
Invalid `lasti`.

(1287) MSK_RES_ERR_FIRSTJ
Invalid `firstj`.

(1288) MSK_RES_ERR_LASTJ
Invalid `lastj`.

(1290) MSK_RES_ERR_NONLINEAR_EQUALITY
The model contains a nonlinear equality which defines a nonconvex set.

(1291) MSK_RES_ERR_NONCONVEX
The optimization problem is nonconvex.

(1292) MSK_RES_ERR_NONLINEAR_RANGED
The model contains a nonlinear ranged constraint which by definition defines a nonconvex set.

(1293) MSK_RES_ERR_CON_Q_NOT_PSD
The quadratic constraint matrix is not positive semi-definite as expected for a constraint with finite upper bound. This results in a nonconvex problem.

(1294) MSK_RES_ERR_CON_Q_NOT_NSD
The quadratic constraint matrix is not negative semi-definite as expected for a constraint with finite lower bound. This results in a nonconvex problem.

(1295) MSK_RES_ERR_OBJ_Q_NOT_PSD
The quadratic coefficient matrix in the objective is not positive semi-definite as expected for a minimization problem.

(1296) MSK_RES_ERR_OBJ_Q_NOT_NSD
The quadratic coefficient matrix in the objective is not negative semi-definite as expected for a maximization problem.

(1299) MSK_RES_ERR_ARGUMENT_PERM_ARRAY
An invalid permutation array is specified.

(1300) MSK_RES_ERR_CONE_INDEX
An index of a non-existing cone has been specified.

(1301) MSK_RES_ERR_CONE_SIZE
A cone with too few members is specified.

(1302) MSK_RES_ERR_CONE_OVERLAP
A new cone which variables overlap with an existing cone has been specified.

(1303) MSK_RES_ERR_CONE_REP_VAR
A variable is included multiple times in the cone.

(1304) MSK_RES_ERR_MAXNUMCONE
The value specified for `maxnumcone` is too small.

(1305) MSK_RES_ERR_CONE_TYPE
Invalid cone type specified.

(1306) MSK_RES_ERR_CONE_TYPE_STR
Invalid cone type specified.

(1310) MSK_RES_ERR_REMOVE_CONE_VARIABLE
A variable cannot be removed because it will make a cone invalid.

(1350) MSK_RES_ERR_SOL_FILE_INVALID_NUMBER
An invalid number is specified in a solution file.

(1375) MSK_RES_ERR_HUGE_C
A huge value in absolute size is specified for one c_j.

(1380) MSK_RES_ERR_HUGE_AIJ
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.

(1400) MSK_RES_ERR_INFINITE_BOUND
A numerically huge bound value is specified.

(1401) MSK_RES_ERR_INV_QOBJ_SUBI
Invalid value in `qosubi`.

(1402) MSK_RES_ERR_INV_QOBJ_SUBJ
Invalid value in `qosubj`.

(1403) MSK_RES_ERR_INV_QOBJ_VAL
Invalid value in `qoval`.

(1404) MSK_RES_ERR_INV_QCON_SUBK
Invalid value in `qcsubk`.

(1405) MSK_RES_ERR_INV_QCON_SUBI
Invalid value in `qcsubi`.

(1406) MSK_RES_ERR_INV_QCON_SUBJ
Invalid value in qcsubj.

(1407) MSK_RES_ERR_INV_QCON_VAL
Invalid value in qcval.

(1408) MSK_RES_ERR_QCON_SUBI_TOO_SMALL
Invalid value in qcsubi.

(1409) MSK_RES_ERR_QCON_SUBI_TOO_LARGE
Invalid value in qcsubi.

(1415) MSK_RES_ERR_QOBJ_UPPER_TRIANGLE
An element in the upper triangle of Q^o is specified. Only elements in the lower triangle should be specified.

(1417) MSK_RES_ERR_QCON_UPPER_TRIANGLE
An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.

(1425) MSK_RES_ERR_FIXED_BOUND_VALUES
A fixed constraint/variable has been specified using the bound keys but the numerical value of the lower and upper bound is different.

(1430) MSK_RES_ERR_USER_FUNC_RET
An user function reported an error.

(1431) MSK_RES_ERR_USER_FUNC_RET_DATA
An user function returned invalid data.

(1432) MSK_RES_ERR_USER_NLO_FUNC
The user-defined nonlinear function reported an error.

(1433) MSK_RES_ERR_USER_NLO_EVAL
The user-defined nonlinear function reported an error.

(1440) MSK_RES_ERR_USER_NLO_EVAL_HESSUBI
The user-defined nonlinear function reported an invalid subscript in the Hessian.

(1441) MSK_RES_ERR_USER_NLO_EVAL_HESSUBJ
The user-defined nonlinear function reported an invalid subscript in the Hessian.

(1445) MSK_RES_ERR_INVALID_OBJECTIVE_SENSE
An invalid objective sense is specified.

(1446) MSK_RES_ERR_UNDEFINED_OBJECTIVE_SENSE
The objective sense has not been specified before the optimization.

(1449) MSK_RES_ERR_Y_IS_UNDEFINED
The solution item y is undefined.

(1450) MSK_RES_ERR_NAN_IN_DOUBLE_DATA
An invalid floating point value was used in some double data.

(1461) MSK_RES_ERR_NAN_IN_BLC
l^c contains an invalid floating point value, i.e. a NaN.

(1462) MSK_RES_ERR_NAN_IN_BUC
u^c contains an invalid floating point value, i.e. a NaN.

(1470) MSK_RES_ERR_NAN_IN_C
c contains an invalid floating point value, i.e. a NaN.

(1471) MSK_RES_ERR_NAN_IN_BLX
l^x contains an invalid floating point value, i.e. a NaN.

(1472) MSK_RES_ERR_NAN_IN_BUX
u^x contains an invalid floating point value, i.e. a NaN.

(1473) MSK_RES_ERR_NAN_IN_AIJ
$a_{i,j}$ contains an invalid floating point value, i.e. a NaN.

(1500) MSK_RES_ERR_INV_PROBLEM
Invalid problem type. Probably a nonconvex problem has been specified.

(1501) MSK_RES_ERR_MIXED_PROBLEM
The problem contains both conic and nonlinear constraints.

(1550) MSK_RES_ERR_INV_OPTIMIZER
An invalid optimizer has been chosen for the problem. This means that the simplex or the conic optimizer is chosen to optimize a nonlinear problem.

(1551) MSK_RES_ERR_MIO_NO_OPTIMIZER
No optimizer is available for the current class of integer optimization problems.

(1552) MSK_RES_ERR_NO_OPTIMIZER_VAR_TYPE
No optimizer is available for this class of optimization problems.

(1553) MSK_RES_ERR_MIO_NOT_LOADED
The mixed-integer optimizer is not loaded.

(1580) MSK_RES_ERR_POSTSOLVE
An error occurred during the postsolve. Please contact MOSEK support.

(1590) MSK_RES_ERR_OVERFLOW
A computation produced an overflow i.e. a very large number.

(1600) MSK_RES_ERR_NO_BASIS_SOL
No basic solution is defined.

(1610) MSK_RES_ERR_BASIS_FACTOR
The factorization of the basis is invalid.

(1615) MSK_RES_ERR_BASIS_SINGULAR
The basis is singular and hence cannot be factored.

(1650) MSK_RES_ERR_FACTOR
An error occurred while factorizing a matrix.

(1700) MSK_RES_ERR_FEASREPAIR_CANNOT_RELAX
An optimization problem cannot be relaxed. This is the case e.g. for general nonlinear optimization problems.

(1701) MSK_RES_ERR_FEASREPAIR_SOLVING_RELAXED
The relaxed problem could not be solved to optimality. Please consult the log file for further details.

(1702) MSK_RES_ERR_FEASREPAIR_INCONSISTENT_BOUND
The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.

(1750) MSK_RES_ERR_NAME_MAX_LEN
A name is longer than the buffer that is supposed to hold it.

(1760) MSK_RES_ERR_NAME_IS_NULL
The name buffer is a NULL pointer.

(1800) MSK_RES_ERR_INVALID_COMPRESSION
Invalid compression type.

(1801) MSK_RES_ERR_INVALID_IOMODE
Invalid io mode.

(2000) MSK_RES_ERR_NO_PRIMAL_INFEAS_CER
A certificate of primal infeasibility is not available.

(2001) MSK_RES_ERR_NO_DUAL_INFEAS_CER
A certificate of infeasibility is not available.

(2500) MSK_RES_ERR_NO_SOLUTION_IN_CALLBACK
The required solution is not available.

(2501) MSK_RES_ERR_INV_MARKI
Invalid value in marki.

(2502) MSK_RES_ERR_INV_MARKJ
Invalid value in markj.

(2503) MSK_RES_ERR_INV_NUMI
Invalid numi.

(2504) MSK_RES_ERR_INV_NUMJ
Invalid numj.

(2505) MSK_RES_ERR_CANNOT_CLONE_NL
A task with a nonlinear function call-back cannot be cloned.

(2506) MSK_RES_ERR_CANNOT_HANDLE_NL
A function cannot handle a task with nonlinear function call-backs.

(2520) MSK_RES_ERR_INVALID_ACCMODE
An invalid access mode is specified.

(2550) MSK_RES_ERR_MBT_INCOMPATIBLE
The MBT file is incompatible with this platform. This results from reading a file on a 32 bit platform generated on a 64 bit platform.

(2800) MSK_RES_ERR_LU_MAX_NUM_TRIES
Could not compute the LU factors of the matrix within the maximum number of allowed tries.

(2900) MSK_RES_ERR_INVALID_UTF8
An invalid UTF8 string is encountered.

(2901) MSK_RES_ERR_INVALID_WCHAR
An invalid `wchar` string is encountered.

(2950) MSK_RES_ERR_NO_DUAL_FOR_ITG_SOL
No dual information is available for the integer solution.

(3000) MSK_RES_ERR_INTERNAL
An internal error occurred. Please report this problem.

(3001) MSK_RES_ERR_API_ARRAY_TOO_SMALL
An input array was too short.

(3002) MSK_RES_ERR_API_CB_CONNECT
Failed to connect a callback object.

(3005) MSK_RES_ERR_API_FATAL_ERROR
An internal error occurred in the API. Please report this problem.

(3050) MSK_RES_ERR_SEN_FORMAT
Syntax error in sensitivity analysis file.

(3051) MSK_RES_ERR_SEN_UNDEF_NAME
An undefined name was encountered in the sensitivity analysis file.

(3052) MSK_RES_ERR_SEN_INDEX_RANGE
Index out of range in the sensitivity analysis file.

(3053) MSK_RES_ERR_SEN_BOUND_INVALID_UP
Analysis of upper bound requested for an index, where no upper bound exists.

(3054) MSK_RES_ERR_SEN_BOUND_INVALID_LO
Analysis of lower bound requested for an index, where no lower bound exists.

(3055) MSK_RES_ERR_SEN_INDEX_INVALID
Invalid range given in the sensitivity file.

(3056) MSK_RES_ERR_SEN_INVALID_REGEXP
Syntax error in regexp or regexp longer than 1024.

(3057) MSK_RES_ERR_SEN_SOLUTION_STATUS
No optimal solution found to the original problem given for sensitivity analysis.

(3058) MSK_RES_ERR_SEN_NUMERICAL
Numerical difficulties encountered performing the sensitivity analysis.

(3059) MSK_RES_ERR_CONCURRENT_OPTIMIZER
An unsupported optimizer was chosen for use with the concurrent optimizer.

(3100) MSK_RES_ERR_UNB_STEP_SIZE
A step size in an optimizer was unexpectedly unbounded. For instance, if the step-size becomes unbounded in phase 1 of the simplex algorithm then an error occurs. Normally this will happen only if the problem is badly formulated. Please contact MOSEK support if this error occurs.

(3101) MSK_RES_ERR_IDENTICAL_TASKS
Some tasks related to this function call were identical. Unique tasks were expected.

(3102) MSK_RES_ERR_AD_INVALID_CODELIST
The code list data was invalid.

(3103) MSK_RES_ERR_AD_INVALID_OPERATOR
The code list data was invalid. An unknown operator was used.

(3104) MSK_RES_ERR_AD_INVALID_OPERAND
The code list data was invalid. An unknown operand was used.

(3105) MSK_RES_ERR_AD_MISSING_OPERAND
The code list data was invalid. Missing operand for operator.

(3106) MSK_RES_ERR_AD_MISSING_RETURN
The code list data was invalid. Missing return operation in function.

(3200) MSK_RES_ERR_INVALID_BRANCH_DIRECTION
An invalid branching direction is specified.

(3201) MSK_RES_ERR_INVALID_BRANCH_PRIORITY
An invalid branching priority is specified. It should be nonnegative.

(3500) MSK_RES_ERR_INTERNAL_TEST_FAILED
An internal unit test function failed.

(3600) MSK_RES_ERR_XML_INVALID_PROBLEM_TYPE
The problem type is not supported by the XML format.

(3700) MSK_RES_ERR_INVALID_AMPL_STUB
Invalid AMPL stub.

(3800) MSK_RES_ERR_INT64_TO_INT32_CAST
An 32 bit integer could not cast to a 64 bit integer.

(3900) MSK_RES_ERR_SIZE_LICENSE_NUMCORES
The computer contains more cpu cores than the license allows for.

(3910) MSK_RES_ERR_INFEAS_UNDEFINED
The requested value is not defined for this solution type.

(3999) MSK_RES_ERR_API_INTERNAL
An internal fatal error occurred in an interface function.

(4000) MSK_RES_TRM_MAX_ITERATIONS
The optimizer terminated at the maximum number of iterations.

(4001) MSK_RES_TRM_MAX_TIME
The optimizer terminated at the maximum amount of time.

(4002) MSK_RES_TRM_OBJECTIVE_RANGE
The optimizer terminated on the bound of the objective range.

(4003) MSK_RES_TRM_MIO_NEAR_REL_GAP
The mixed-integer optimizer terminated because the near optimal relative gap tolerance was satisfied.

(4004) MSK_RES_TRM_MIO_NEAR_ABS_GAP
The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied.

(4005) MSK_RES_TRM_USER_BREAK
Not in use.

(4006) MSK_RES_TRM_STALL
The optimizer terminated due to slow progress. Normally there are three possible reasons for this: Either a bug in MOSEK, the problem is badly formulated, or, for nonlinear problems, the nonlinear call-back functions are incorrect.

The solution returned may or may not be of acceptable quality. Therefore, the solution status should be examined to determine the status of the solution.

In particular, if a linear optimization problem is solved with the interior-point optimizer with basis identification turned on, the returned solution may be of acceptable quality, even in the optimizer stalled.

(4007) MSK_RES_TRM_USER_CALLBACK
The optimizer terminated due to the return of the user-defined call-back function.

(4008) MSK_RES_TRM_MIO_NUM_RELAXS
The mixed-integer optimizer terminated as the maximum number of relaxations was reached.

(4009) MSK_RES_TRM_MIO_NUM_BRANCHES
The mixed-integer optimizer terminated as to the maximum number of branches was reached.

(4015) MSK_RES_TRM_NUM_MAX_NUM_INT_SOLUTIONS
The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.

(4020) MSK_RES_TRM_MAX_NUM_SETBACKS
The optimizer terminated as the maximum number of set-backs was reached. This indicates numerical problems and a possibly badly formulated problem.

(4025) MSK_RES_TRM_NUMERICAL_PROBLEM
The optimizer terminated due to numerical problems.

(4030) MSK_RES_TRM_INTERNAL
The optimizer terminated due to some internal reason. Please contact MOSEK support.

(4031) MSK_RES_TRM_INTERNAL_STOP
The optimizer terminated for internal reasons. Please contact MOSEK support.

Chapter 18

Constants

18.1 Constraint or variable access modes

(1) `MSK_ACC_CON`
Access data by rows (constraint oriented)

(0) `MSK_ACC_VAR`
Access data by columns (variable orinted)

18.2 Function opcode

(0) `MSK_ADOP_ADD`
Add two operands.

(3) `MSK_ADOP_DIV`
Divide two operands.

(5) `MSK_ADOP_EXP`
Exponential function of one oparand.

(6) `MSK_ADOP_LOG`
Logarithm function of one operand.

(2) `MSK_ADOP_MUL`
Multiply two operands.

(4) MSK_ADOP_POW
First operand to the power the second operand.

(7) MSK_ADOP_RET
Return one operand.

(1) MSK_ADOP_SUB
Subtract two operands.

18.3 Function operand type

(1) MSK_ADOPTYPE_CONSTANT
Operand refers to a constant.

(0) MSK_ADOPTYPE_NONE
Operand not used.

(3) MSK_ADOPTYPE_REFERENCE
Operand refers to the result of another operation.

(2) MSK_ADOPTYPE_VARIABLE
Operand refers to a variable.

18.4 Basis identification

(1) MSK_BI_ALWAYS
Basis identification is always performed even if the interior-point optimizer terminates abnormally.

(3) 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.

(0) MSK_BI_NEVER
Never do basis identification.

(2) MSK_BI_NO_ERROR
Basis identification is performed if the interior-point optimizer terminates without an error.

(4) MSK_BI_OTHER
Try another BI method.

18.5 Bound keys

(3) MSK_BK_FR
The constraint or variable is free.

(2) MSK_BK_FX
The constraint or variable is fixed.

(0) MSK_BK_LO
The constraint or variable has a finite lower bound and an infinite upper bound.

(4) MSK_BK_RA
The constraint or variable is ranged.

(1) MSK_BK_UP
The constraint or variable has an infinite lower bound and an finite upper bound.

18.6 Specifies the branching direction.

(2) MSK_BRANCH_DIR_DOWN
The mixed-integer optimizer always chooses the down branch first.

(0) MSK_BRANCH_DIR_FREE
The mixed-integer optimizer decides which branch to choose.

(1) MSK_BRANCH_DIR_UP
The mixed-integer optimizer always chooses the up branch first.

18.7 Progress call-back codes

(0) MSK_CALLBACK_BEGIN_BI
The basis identification procedure has been started.

(1) MSK_CALLBACK_BEGIN_CONCURRENT
Concurrent optimizer is started.

(2) MSK_CALLBACK_BEGIN_CONIC
The call-back function is called when the conic optimizer is started.

(3) MSK_CALLBACK_BEGIN_DUAL_BI
The call-back function is called from within the basis identification procedure when the dual phase is started.

(4) MSK_CALLBACK_BEGIN_DUAL_SENSITIVITY
Dual sensitivity analysis is started.

(5) MSK_CALLBACK_BEGIN_DUAL_SETUP_BI
The call-back function is called when the dual BI phase is started.

(6) MSK_CALLBACK_BEGIN_DUAL_SIMPLEX
The call-back function is called when the dual simplex optimizer started.

(7) MSK_CALLBACK_BEGIN_DUAL_SIMPLEX_BI
The call-back function is called from within the basis identification procedure when the dual simplex clean-up phase is started.

(8) MSK_CALLBACK_BEGIN_FULL_CONVEXITY_CHECK
Begin full convexity check.

(9) MSK_CALLBACK_BEGIN_INFEAS_ANA
The call-back function is called when the infeasibility analyzer is started.

(10) MSK_CALLBACK_BEGIN_INTPNT
The call-back function is called when the interior-point optimizer is started.

(11) MSK_CALLBACK_BEGIN_LICENSE_WAIT
Begin waiting for license.

(12) MSK_CALLBACK_BEGIN_MIO
The call-back function is called when the mixed-integer optimizer is started.

(13) MSK_CALLBACK_BEGIN_NETWORK_DUAL_SIMPLEX
The call-back function is called when the dual network simplex optimizer is started.

(14) MSK_CALLBACK_BEGIN_NETWORK_PRIMAL_SIMPLEX
The call-back function is called when the primal network simplex optimizer is started.

(15) MSK_CALLBACK_BEGIN_NETWORK_SIMPLEX
The call-back function is called when the simplex network optimizer is started.

(16) MSK_CALLBACK_BEGIN_NONCONVEX
The call-back function is called when the nonconvex optimizer is started.

(17) MSK_CALLBACK_BEGIN_OPTIMIZER
The call-back function is called when the optimizer is started.

(18) MSK_CALLBACK_BEGIN_PRESOLVE
The call-back function is called when the presolve is started.

(19) MSK_CALLBACK_BEGIN_PRIMAL_BI
The call-back function is called from within the basis identification procedure when the primal phase is started.

(20) MSK_CALLBACK_BEGIN_PRIMAL_DUAL_SIMPLEX
The call-back function is called when the primal-dual simplex optimizer is started.

(21) MSK_CALLBACK_BEGIN_PRIMAL_DUAL_SIMPLEX_BI
The call-back function is called from within the basis identification procedure when the primal-dual simplex clean-up phase is started.

(22) MSK_CALLBACK_BEGIN_PRIMAL_SENSITIVITY
Primal sensitivity analysis is started.

(23) MSK_CALLBACK_BEGIN_PRIMAL_SETUP_BI
The call-back function is called when the primal BI setup is started.

(24) MSK_CALLBACK_BEGIN_PRIMAL_SIMPLEX
The call-back function is called when the primal simplex optimizer is started.

(25) MSK_CALLBACK_BEGIN_PRIMAL_SIMPLEX_BI
The call-back function is called from within the basis identification procedure when the primal simplex clean-up phase is started.

(26) MSK_CALLBACK_BEGIN_QCQO_REFORMULATE
Begin QCQO reformulation.

(27) MSK_CALLBACK_BEGIN_READ
MOSEK has started reading a problem file.

(28) MSK_CALLBACK_BEGIN_SIMPLEX
The call-back function is called when the simplex optimizer is started.

(29) MSK_CALLBACK_BEGIN_SIMPLEX_BI
The call-back function is called from within the basis identification procedure when the simplex clean-up phase is started.

(30) MSK_CALLBACK_BEGIN_SIMPLEX_NETWORK_DETECT
The call-back function is called when the network detection procedure is started.

(31) MSK_CALLBACK_BEGIN_WRITE
MOSEK has started writing a problem file.

(32) MSK_CALLBACK_CONIC
The call-back function is called from within the conic optimizer after the information database has been updated.

(33) MSK_CALLBACK_DUAL_SIMPLEX
The call-back function is called from within the dual simplex optimizer.

(34) MSK_CALLBACK_END_BI
The call-back function is called when the basis identification procedure is terminated.

(35) MSK_CALLBACK_END_CONCURRENT
Concurrent optimizer is terminated.

(36) MSK_CALLBACK_END_CONIC
The call-back function is called when the conic optimizer is terminated.

(37) MSK_CALLBACK_END_DUAL_BI
The call-back function is called from within the basis identification procedure when the dual phase is terminated.

(38) MSK_CALLBACK_END_DUAL_SENSITIVITY
Dual sensitivity analysis is terminated.

(39) MSK_CALLBACK_END_DUAL_SETUP_BI
The call-back function is called when the dual BI phase is terminated.

(40) MSK_CALLBACK_END_DUAL_SIMPLEX
The call-back function is called when the dual simplex optimizer is terminated.

(41) MSK_CALLBACK_END_DUAL_SIMPLEX_BI
The call-back function is called from within the basis identification procedure when the dual clean-up phase is terminated.

(42) MSK_CALLBACK_END_FULL_CONVEXITY_CHECK
End full convexity check.

(43) MSK_CALLBACK_END_INFEAS_ANA
The call-back function is called when the infeasibility analyzer is terminated.

(44) MSK_CALLBACK_END_INTPNT
The call-back function is called when the interior-point optimizer is terminated.

(45) MSK_CALLBACK_END_LICENSE_WAIT
End waiting for license.

(46) MSK_CALLBACK_END_MIO
The call-back function is called when the mixed-integer optimizer is terminated.

(47) MSK_CALLBACK_END_NETWORK_DUAL_SIMPLEX
The call-back function is called when the dual network simplex optimizer is terminated.

(48) MSK_CALLBACK_END_NETWORK_PRIMAL_SIMPLEX
The call-back function is called when the primal network simplex optimizer is terminated.

(49) MSK_CALLBACK_END_NETWORK_SIMPLEX
The call-back function is called when the simplex network optimizer is terminated.

(50) MSK_CALLBACK_END_NONCONVEX
The call-back function is called when the nonconvex optimizer is terminated.

(51) MSK_CALLBACK_END_OPTIMIZER
The call-back function is called when the optimizer is terminated.

(52) MSK_CALLBACK_END_PRESOLVE
The call-back function is called when the presolve is completed.

(53) MSK_CALLBACK_END_PRIMAL_BI
The call-back function is called from within the basis identification procedure when the primal phase is terminated.

(54) MSK_CALLBACK_END_PRIMAL_DUAL_SIMPLEX
The call-back function is called when the primal-dual simplex optimizer is terminated.

(55) MSK_CALLBACK_END_PRIMAL_DUAL_SIMPLEX_BI
The call-back function is called from within the basis identification procedure when the primal-dual clean-up phase is terminated.

(56) MSK_CALLBACK_END_PRIMAL_SENSITIVITY
Primal sensitivity analysis is terminated.

(57) MSK_CALLBACK_END_PRIMAL_SETUP_BI
The call-back function is called when the primal BI setup is terminated.

(58) MSK_CALLBACK_END_PRIMAL_SIMPLEX
The call-back function is called when the primal simplex optimizer is terminated.

(59) MSK_CALLBACK_END_PRIMAL_SIMPLEX_BI
The call-back function is called from within the basis identification procedure when the primal clean-up phase is terminated.

(60) MSK_CALLBACK_END_QCQO_REFORMULATE
End QCQO reformulation.

(61) MSK_CALLBACK_END_READ
MOSEK has finished reading a problem file.

(62) MSK_CALLBACK_END_SIMPLEX
The call-back function is called when the simplex optimizer is terminated.

(63) MSK_CALLBACK_END_SIMPLEX_BI
The call-back function is called from within the basis identification procedure when the simplex clean-up phase is terminated.

(64) MSK_CALLBACK_END_SIMPLEX_NETWORK_DETECT
The call-back function is called when the network detection procedure is terminated.

(65) MSK_CALLBACK_END_WRITE
MOSEK has finished writing a problem file.

(66) MSK_CALLBACK_IM_BI
The call-back function is called from within the basis identification procedure at an intermediate point.

(67) MSK_CALLBACK_IM_CONIC
The call-back function is called at an intermediate stage within the conic optimizer where the information database has not been updated.

(68) MSK_CALLBACK_IM_DUAL_BI
The call-back function is called from within the basis identification procedure at an intermediate point in the dual phase.

(69) MSK_CALLBACK_IM_DUAL_SENSIVITY
The call-back function is called at an intermediate stage of the dual sensitivity analysis.

(70) MSK_CALLBACK_IM_DUAL_SIMPLEX
The call-back function is called at an intermediate point in the dual simplex optimizer.

(71) MSK_CALLBACK_IM_FULL_CONVEXITY_CHECK
The call-back function is called at an intermediate stage of the full convexity check.

(72) MSK_CALLBACK_IM_INTPNT
The call-back function is called at an intermediate stage within the interior-point optimizer where the information database has not been updated.

(73) MSK_CALLBACK_IM_LICENSE_WAIT
MOSEK is waiting for a license.

(74) MSK_CALLBACK_IM_LU
The call-back function is called from within the LU factorization procedure at an intermediate point.

(75) MSK_CALLBACK_IM_MIO
The call-back function is called at an intermediate point in the mixed-integer optimizer.

(76) MSK_CALLBACK_IM_MIO_DUAL_SIMPLEX
The call-back function is called at an intermediate point in the mixed-integer optimizer while running the dual simplex optimizer.

(77) MSK_CALLBACK_IM_MIO_INTPNT
The call-back function is called at an intermediate point in the mixed-integer optimizer while running the interior-point optimizer.

(78) MSK_CALLBACK_IM_MIO_PRESOLVE
The call-back function is called at an intermediate point in the mixed-integer optimizer while running the presolve.

(79) MSK_CALLBACK_IM_MIO_PRIMAL_SIMPLEX
The call-back function is called at an intermediate point in the mixed-integer optimizer while running the primal simplex optimizer.

(80) MSK_CALLBACK_IM_NETWORK_DUAL_SIMPLEX
The call-back function is called at an intermediate point in the dual network simplex optimizer.

(81) MSK_CALLBACK_IM_NETWORK_PRIMAL_SIMPLEX
The call-back function is called at an intermediate point in the primal network simplex optimizer.

(82) MSK_CALLBACK_IM_NONCONVEX
The call-back function is called at an intermediate stage within the nonconvex optimizer where the information database has not been updated.

(83) MSK_CALLBACK_IM_ORDER
The call-back function is called from within the matrix ordering procedure at an intermediate point.

(84) MSK_CALLBACK_IM_PRESOLVE
The call-back function is called from within the presolve procedure at an intermediate stage.

(85) MSK_CALLBACK_IM_PRIMAL_BI
The call-back function is called from within the basis identification procedure at an intermediate point in the primal phase.

(86) MSK_CALLBACK_IM_PRIMAL_DUAL_SIMPLEX
The call-back function is called at an intermediate point in the primal-dual simplex optimizer.

(87) MSK_CALLBACK_IM_PRIMAL_SENSIVITY
The call-back function is called at an intermediate stage of the primal sensitivity analysis.

(88) MSK_CALLBACK_IM_PRIMAL_SIMPLEX
The call-back function is called at an intermediate point in the primal simplex optimizer.

(89) MSK_CALLBACK_IM_QO_REFORMULATE
The call-back function is called at an intermediate stage of the QP to SOCP reformulation.

(90) MSK_CALLBACK_IM_SIMPLEX
The call-back function is called from within the simplex optimizer at an intermediate point.

(91) MSK_CALLBACK_IM_SIMPLEX_BI
The call-back function is called from within the basis identification procedure at an intermediate point in the simplex clean-up phase. The frequency of the call-backs is controlled by the MSK_IPAR_LOG_SIM_FREQ parameter.

(92) MSK_CALLBACK_INTPNT
The call-back function is called from within the interior-point optimizer after the information database has been updated.

(93) MSK_CALLBACK_NEW_INT_MIO
The call-back function is called after a new integer solution has been located by the mixed-integer optimizer.

(94) MSK_CALLBACK_NONCOVEX
The call-back function is called from within the nonconvex optimizer after the information database has been updated.

(95) MSK_CALLBACK_PRIMAL_SIMPLEX
The call-back function is called from within the primal simplex optimizer.

(96) MSK_CALLBACK_QCONE
The call-back function is called from within the Qcone optimizer.

(97) MSK_CALLBACK_READ_ADD_ANZ
A chunk of A non-zeos has been read from a problem file.

(98) MSK_CALLBACK_READ_ADD_CONES
A chunk of cones has been read from a problem file.

(99) MSK_CALLBACK_READ_ADD_CONS
A chunk of constraints has been read from a problem file.

(100) MSK_CALLBACK_READ_ADD_QNZ
A chunk of Q non-zeos has been read from a problem file.

(101) MSK_CALLBACK_READ_ADD_VARS
A chunk of variables has been read from a problem file.

(102) MSK_CALLBACK_READ_OPF
The call-back function is called from the OPF reader.

(103) MSK_CALLBACK_READ_OPF_SECTION
A chunk of Q non-zeos has been read from a problem file.

(104) MSK_CALLBACK_UPDATE_DUAL_BI
The call-back function is called from within the basis identification procedure at an intermediate point in the dual phase.

(105) MSK_CALLBACK_UPDATE_DUAL_SIMPLEX
The call-back function is called in the dual simplex optimizer.

(106) MSK_CALLBACK_UPDATE_DUAL_SIMPLEX_BI
The call-back function is called from within the basis identification procedure at an intermediate point in the dual simplex clean-up phase. The frequency of the call-backs is controlled by the MSK_IPAR_LOG_SIM_FREQ parameter.

(107) MSK_CALLBACK_UPDATE_NETWORK_DUAL_SIMPLEX
The call-back function is called in the dual network simplex optimizer.

(108) MSK_CALLBACK_UPDATE_NETWORK_PRIMAL_SIMPLEX
The call-back function is called in the primal network simplex optimizer.

(109) MSK_CALLBACK_UPDATE_NONCONVEX
The call-back function is called at an intermediate stage within the nonconvex optimizer where the information database has been updated.

(110) MSK_CALLBACK_UPDATE_PRESOLVE
The call-back function is called from within the presolve procedure.

(111) MSK_CALLBACK_UPDATE_PRIMAL_BI
The call-back function is called from within the basis identification procedure at an intermediate point in the primal phase.

(112) MSK_CALLBACK_UPDATE_PRIMAL_DUAL_SIMPLEX
The call-back function is called in the primal-dual simplex optimizer.

(113) MSK_CALLBACK_UPDATE_PRIMAL_DUAL_SIMPLEX_BI
The call-back function is called from within the basis identification procedure at an intermediate point in the primal-dual simplex clean-up phase. The frequency of the call-backs is controlled by the MSK_IPAR_LOG_SIM_FREQ parameter.

(114) MSK_CALLBACK_UPDATE_PRIMAL_SIMPLEX
The call-back function is called in the primal simplex optimizer.

(115) MSK_CALLBACK_UPDATE_PRIMAL_SIMPLEX_BI
The call-back function is called from within the basis identification procedure at an intermediate point in the primal simplex clean-up phase. The frequency of the call-backs is controlled by the MSK_IPAR_LOG_SIM_FREQ parameter.

(116) MSK_CALLBACK_WRITE_OPF
The call-back function is called from the OPF writer.

18.8 Types of convexity checks.

(2) MSK_CHECK_CONVEXITY_FULL
Perform a full convexity check.

(0) MSK_CHECK_CONVEXITY_NONE
No convexity check.

(1) MSK_CHECK_CONVEXITY_SIMPLE
Perform simple and fast convexity check.

18.9 Compression types

(1) MSK_COMPRESS_FREE
The type of compression used is chosen automatically.

(2) MSK_COMPRESS_GZIP
The type of compression used is gzip compatible.

(0) MSK_COMPRESS_NONE
No compression is used.

18.10 Cone types

(0) MSK_CT_QUAD
The cone is a quadratic cone.

(1) MSK_CT_RQUAD
The cone is a rotated quadratic cone.

18.11 CPU type

(4) MSK_CPU_AMD_ATHLON
An AMD Athlon.

(7) MSK_CPU_AMD_OPTERON
An AMD Opteron (64 bit).

(1) MSK_CPU_GENERIC
An generic CPU type for the platform

(5) MSK_CPU_HP_PARISC20
An HP PA RISC version 2.0 CPU.

(10) MSK_CPU_INTEL_CORE2
An Intel CORE2 cpu.

(6) MSK_CPU_INTEL_ITANIUM2
An Intel Itanium2.

(2) MSK_CPU_INTEL_P3
An Intel Pentium P3.

(3) MSK_CPU_INTEL_P4
An Intel Pentium P4 or Intel Xeon.

(9) MSK_CPU_INTEL_PM
An Intel PM cpu.

(8) MSK_CPU_POWERPC_G5
A G5 PowerPC CPU.

(0) MSK_CPU_UNKNOWN
An unknown CPU.

18.12 Data format types

(0) MSK_DATA_FORMAT_EXTENSION
The file extension is used to determine the data file format.

(6) MSK_DATA_FORMAT_FREE_MPS
The data data a free MPS formatted file.

(2) MSK_DATA_FORMAT_LP
The data file is LP formatted.

(3) MSK_DATA_FORMAT_MBT
The data file is a MOSEK binary task file.

(1) MSK_DATA_FORMAT_MPS
The data file is MPS formatted.

(4) MSK_DATA_FORMAT_OP
The data file is an optimization problem formatted file.

(5) MSK_DATA_FORMAT_XML
The data file is an XML formatted file.

18.13 Double information items

(0) MSK_DINF_BI_CLEAN_DUAL_TIME
Time spent within the dual clean-up optimizer of the basis identification procedure since its invocation.

(1) MSK_DINF_BI_CLEAN_PRIMAL_DUAL_TIME
Time spent within the primal-dual clean-up optimizer of the basis identification procedure since its invocation.

(2) MSK_DINF_BI_CLEAN_PRIMAL_TIME
Time spent within the primal clean-up optimizer of the basis identification procedure since its invocation.

(3) MSK_DINF_BI_CLEAN_TIME
Time spent within the clean-up phase of the basis identification procedure since its invocation.

(4) MSK_DINF_BI_DUAL_TIME
Time spent within the dual phase basis identification procedure since its invocation.

(5) MSK_DINF_BI_PRIMAL_TIME
Time spent within the primal phase of the basis identification procedure since its invocation.

(6) MSK_DINF_BI_TIME
Time spent within the basis identification procedure since its invocation.

(7) MSK_DINF_CONCURRENT_TIME
Time spent within the concurrent optimizer since its invocation.

(8) MSK_DINF_INTPNT_DUAL_FEAS
Dual feasibility measure reported by the interior-point and Qcone optimizer. (For the interior-point optimizer this measure does not directly related to the original problem because a homogeneous model is employed.)

(9) MSK_DINF_INTPNT_DUAL_OBJ
Dual objective value reported by the interior-point or Qcone optimizer.

(10) MSK_DINF_INTPNT_FACTOR_NUM_FLOPS
An estimate of the number of flops used in the factorization.

(11) MSK_DINF_INTPNT_KAP_DIV_TAU
This measure should converge to zero if the problem has a primal-dual optimal solution or to infinity if problem is (strictly) primal or dual infeasible. In case the measure is converging towards a positive but bounded constant the problem is usually ill-posed.

(12) MSK_DINF_INTPNT_ORDER_TIME
Order time (in seconds).

(13) MSK_DINF_INTPNT_PRIMAL_FEAS
Primal feasibility measure reported by the interior-point or Qcone optimizers. (For the interior-point optimizer this measure does not directly related to the original problem because a homogeneous model is employed).

(14) MSK_DINF_INTPNT_PRIMAL_OBJ
Primal objective value reported by the interior-point or Qcone optimizer.

(15) MSK_DINF_INTPNT_TIME
Time spent within the interior-point optimizer since its invocation.

(16) 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.

(17) MSK_DINF_MIO_HEURISTIC_TIME
Time spent in the optimizer while solving the relaxtions.

(18) 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.

(19) 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 stricly positive.

(20) 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 located i.e. check MSK_IINF_MIO_NUM_INT_SOLUTIONS.

(21) 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 paramater MSK_DPAR_MIO_REL_GAP_CONST. Otherwise it has the value -1.0.

(22) MSK_DINF_MIO_OPTIMIZER_TIME
Time spent in the optimizer while solving the relaxtions.

(23) MSK_DINF_MIO_ROOT_OPTIMIZER_TIME
Time spent in the optimizer while solving the root relaxation.

(24) MSK_DINF_MIO_ROOT_PRESOLVE_TIME
Time spent in while presolveing the root relaxation.

(25) MSK_DINF_MIO_TIME
Time spent in the mixed-integer optimizer.

(26) MSK_DINF_MIO_USER_OBJ_CUT
If the objective cut is used, then this information item has the value of the cut.

(27) MSK_DINF_OPTIMIZER_TIME
Total time spent in the optimizer since it was invoked.

(28) MSK_DINF_PRESOLVE_ELI_TIME
Total time spent in the eliminator since the presolve was invoked.

(29) MSK_DINF_PRESOLVE_LINDEP_TIME
Total time spent in the linear dependency checker since the presolve was invoked.

(30) MSK_DINF_PRESOLVE_TIME
Total time (in seconds) spent in the presolve since it was invoked.

(31) MSK_DINF_QCQO_REFORMULATE_TIME
Time spent with QP reformulation.

(32) MSK_DINF_RD_TIME
Time spent reading the data file.

(33) MSK_DINF_SIM_DUAL_TIME
Time spent in the dual simplex optimizer since invoking it.

(34) MSK_DINF_SIM_FEAS
Feasibility measure reported by the simplex optimizer.

(35) MSK_DINF_SIM_NETWORK_DUAL_TIME
Time spent in the dual network simplex optimizer since invoking it.

(36) MSK_DINF_SIM_NETWORK_PRIMAL_TIME
Time spent in the primal network simplex optimizer since invoking it.

(37) MSK_DINF_SIM_NETWORK_TIME
Time spent in the network simplex optimizer since invoking it.

(38) MSK_DINF_SIM_OBJ
Objective value reported by the simplex optimizer.

(39) MSK_DINF_SIM_PRIMAL_DUAL_TIME
Time spent in the primal-dual simplex optimizer optimizer since invoking it.

(40) MSK_DINF_SIM_PRIMAL_TIME
Time spent in the primal simplex optimizer since invoking it.

(41) MSK_DINF_SIM_TIME
Time spent in the simplex optimizer since invoking it.

(42) MSK_DINF_SOL_BAS_DUAL_OBJ
Dual objective value of the basic solution. Updated at the end of the optimization.

(43) MSK_DINF_SOL_BAS_MAX_DBI
Maximal dual bound infeasibility in the basic solution. Updated at the end of the optimization.

(44) MSK_DINF_SOL_BAS_MAX_DEQI
Maximal dual equality infeasibility in the basic solution. Updated at the end of the optimization.

(45) MSK_DINF_SOL_BAS_MAX_PBI
Maximal primal bound infeasibility in the basic solution. Updated at the end of the optimization.

(46) MSK_DINF_SOL_BAS_MAX_PEQI
Maximal primal equality infeasibility in the basic solution. Updated at the end of the optimization.

(47) MSK_DINF_SOL_BAS_MAX_PINTI
Maximal primal integer infeasibility in the basic solution. Updated at the end of the optimization.

(48) MSK_DINF_SOL_BAS_PRIMAL_OBJ
Primal objective value of the basic solution. Updated at the end of the optimization.

(49) MSK_DINF_SOL_INT_MAX_PBI
Maximal primal bound infeasibility in the integer solution. Updated at the end of the optimization.

(50) MSK_DINF_SOL_INT_MAX_PEQI
Maximal primal equality infeasibility in the basic solution. Updated at the end of the optimization.

(51) MSK_DINF_SOL_INT_MAX_PINTI
Maximal primal integer infeasibility in the integer solution. Updated at the end of the optimization.

(52) MSK_DINF_SOL_INT_PRIMAL_OBJ
Primal objective value of the integer solution. Updated at the end of the optimization.

(53) MSK_DINF_SOL_ITR_DUAL_OBJ
Dual objective value of the interior-point solution. Updated at the end of the optimization.

(54) MSK_DINF_SOL_ITR_MAX_DBI
Maximal dual bound infeasibility in the interior-point solution. Updated at the end of the optimization.

(55) MSK_DINF_SOL_ITR_MAX_DCNI
Maximal dual cone infeasibility in the interior-point solution. Updated at the end of the optimization.

(56) MSK_DINF_SOL_ITR_MAX_DEQI
Maximal dual equality infeasibility in the interior-point solution. Updated at the end of the optimization.

(57) MSK_DINF_SOL_ITR_MAX_PBI
Maximal primal bound infeasibility in the interior-point solution. Updated at the end of the optimization.

(58) MSK_DINF_SOL_ITR_MAX_PCNI
Maximal primal cone infeasibility in the interior-point solution. Updated at the end of the optimization.

(59) MSK_DINF_SOL_ITR_MAX_PEQI
Maximal primal equality infeasibility in the interior-point solution. Updated at the end of the optimization.

(60) MSK_DINF_SOL_ITR_MAX_PINTI
Maximal primal integer infeasibility in the interior-point solution. Updated at the end of the optimization.

(61) MSK_DINF_SOL_ITR_PRIMAL_OBJ
Primal objective value of the interior-point solution. Updated at the end of the optimization.

18.14 Feasibility repair types

(2) MSK_FEASREPAIR_OPTIMIZE_COMBINED
Minimize with original objective subject to minimal weighted violation of bounds.

(0) MSK_FEASREPAIR_OPTIMIZE_NONE
Do not optimize the feasibility repair problem.

(1) MSK_FEASREPAIR_OPTIMIZE_PENALTY
Minimize weighted sum of violations.

18.15 License feature

(2) MSK_FEATURE_PTOM
Mixed-integer extension.

(1) MSK_FEATURE_PTON
Nonlinear extension.

(3) MSK_FEATURE_PTOX
Non-convex extension.

(0) MSK_FEATURE_PTS
Base system.

18.16 Integer information items.

(0) MSK_IINF_ANA_PRO_NUM_CON
Number of constraints in the problem.

This value is set by MSK_analyzeproblem.

(1) MSK_IINF_ANA_PRO_NUM_CON_EQ
Number of equality constraints.

This value is set by MSK_analyzeproblem.

(2) MSK_IINF_ANA_PRO_NUM_CON_FR
Number of unbounded constraints.

This value is set by MSK_analyzeproblem.

(3) MSK_IINF_ANA_PRO_NUM_CON_LO
Number of constraints with a lower bound and an infinite upper bound.

This value is set by MSK_analyzeproblem.

(4) MSK_IINF_ANA_PRO_NUM_CON_RA
Number of constraints with finite lower and upper bounds.

This value is set by MSK_analyzeproblem.

(5) MSK_IINF_ANA_PRO_NUM_CON_UP
Number of constraints with an upper bound and an infinite lower bound.

This value is set by MSK_analyzeproblem.

(6) MSK_IINF_ANA_PRO_NUM_VAR
Number of variables in the problem.

This value is set by MSK_analyzeproblem.

(7) MSK_IINF_ANA_PRO_NUM_VAR_BIN
Number of binary (0-1) variables.

This value is set by MSK_analyzeproblem.

(8) MSK_IINF_ANA_PRO_NUM_VAR_CONT
Number of continuous variables.

This value is set by MSK_analyzeproblem.

(9) MSK_IINF_ANA_PRO_NUM_VAR_EQ
Number of fixed variables.

This value is set by MSK_analyzeproblem.

(10) MSK_IINF_ANA_PRO_NUM_VAR_FR
Number of free variables.

This value is set by MSK_analyzeproblem.

(11) MSK_IINF_ANA_PRO_NUM_VAR_INT
Number of general integer variables.

This value is set by MSK_analyzeproblem.

(12) MSK_IINF_ANA_PRO_NUM_VAR_LO
Number of variables with a lower bound and an infinite upper bound.

This value is set by MSK_analyzeproblem.

(13) MSK_IINF_ANA_PRO_NUM_VAR_RA
Number of variables with finite lower and upper bounds.

This value is set by MSK_analyzeproblem.

(14) MSK_IINF_ANA_PRO_NUM_VAR_UP
Number of variables with an upper bound and an infinite lower bound. This value is set by

This value is set by MSK_analyzeproblem.

(15) MSK_IINF_CACHE_SIZE_L1
L1 cache size used.

(16) MSK_IINF_CACHE_SIZE_L2
L2 cache size used.

(17) MSK_IINF_CONCURRENT_FASTEST_OPTIMIZER
The type of the optimizer that finished first in a concurrent optimization.

(18) MSK_IINF_CPU_TYPE
The type of cpu detected.

(19) MSK_IINF_INTPNT_FACTOR_NUM_OFFCOL
Number of columns in the constraint matrix (or Jacobian) that has an offending structure.

(20) MSK_IINF_INTPNT_ITER
Number of interior-point iterations since invoking the interior-point optimizer.

(21) MSK_IINF_INTPNT_NUM_THREADS
Number of threads that the interior-point optimizer is using.

(22) MSK_IINF_INTPNT_SOLVE_DUAL
Non-zero if the interior-point optimizer is solving the dual problem.

(23) MSK_IINF_MIO_CONSTRUCT_SOLUTION
If this item has the value 0, then MOSEK did not try to construct an initial integer feasible solution. If the item has a positive value, then MOSEK successfully constructed an initial integer feasible solution.

(24) MSK_IINF_MIO_INITIAL_SOLUTION
Is non-zero if an initial integer solution is specified.

(25) MSK_IINF_MIO_NUM_ACTIVE_NODES
Number of active nodes in the branch and bound tree.

(26) MSK_IINF_MIO_NUM_BRANCH
Number of branches performed during the optimization.

(27) MSK_IINF_MIO_NUM_CUTS
Number of cuts generated by the mixed-integer optimizer.

(28) MSK_IINF_MIO_NUM_INT_SOLUTIONS
Number of integer feasible solutions that has been found.

(29) MSK_IINF_MIO_NUM_RELAX
Number of relaxations solved during the optimization.

(30) MSK_IINF_MIO_NUMCON
Number of constraints in the problem solved be the mixed-integer optimizer.

(31) MSK_IINF_MIO_NUMINT
Number of integer variables in the problem solved be the mixed-integer optimizer.

(32) MSK_IINF_MIO_NUMVAR
Number of variables in the problem solved be the mixed-integer optimizer.

(33) MSK_IINF_MIO_TOTAL_NUM_BASIS_CUTS
Number of basis cuts.

(34) MSK_IINF_MIO_TOTAL_NUM_BRANCH
Number of branches performed during the optimization.

(35) MSK_IINF_MIO_TOTAL_NUM_CARDGUB_CUTS
Number of cardgub cuts.

(36) MSK_IINF_MIO_TOTAL_NUM_CLIQUE_CUTS
Number of clique cuts.

(37) MSK_IINF_MIO_TOTAL_NUM_COEF_REDC_CUTS
Number of coef. redc. cuts.

(38) MSK_IINF_MIO_TOTAL_NUM_CONTRA_CUTS
Number of contra cuts.

(39) MSK_IINF_MIO_TOTAL_NUM_CUTS
Total number of cuts generated by the mixed-integer optimizer.

(40) MSK_IINF_MIO_TOTAL_NUM_DISAGG_CUTS
Number of diasagg cuts.

(41) MSK_IINF_MIO_TOTAL_NUM_FLOW_COVER_CUTS
Number of flow cover cuts.

(42) MSK_IINF_MIO_TOTAL_NUM_GCD_CUTS
Number of gcd cuts.

(43) MSK_IINF_MIO_TOTAL_NUM_GOMORY_CUTS
Number of Gomory cuts.

(44) MSK_IINF_MIO_TOTAL_NUM_GUB_COVER_CUTS
Number of GUB cover cuts.

(45) MSK_IINF_MIO_TOTAL_NUM_KNAPSUR_COVER_CUTS
Number of knapsack cover cuts.

(46) MSK_IINF_MIO_TOTAL_NUM_LATTICE_CUTS
Number of lattice cuts.

(47) MSK_IINF_MIO_TOTAL_NUM_LIFT_CUTS
Number of lift cuts.

(48) MSK_IINF_MIO_TOTAL_NUM_OBJ_CUTS
Number of obj cuts.

(49) MSK_IINF_MIO_TOTAL_NUM_PLAN_LOC_CUTS
Number of loc cuts.

(50) MSK_IINF_MIO_TOTAL_NUM_RELAX
Number of relaxations solved during the optimization.

(51) MSK_IINF_MIO_USER_OBJ_CUT
If it is non-zero, then the objective cut is used.

(52) MSK_IINF_OPT_NUMCON
Number of constraints in the problem solved when the optimizer is called.

(53) MSK_IINF_OPT_NUMVAR
Number of variables in the problem solved when the optimizer is called

(54) MSK_IINF_OPTIMIZE_RESPONSE
The reponse code returned by optimize.

(55) MSK_IINF_RD_NUMCON
Number of constraints read.

(56) MSK_IINF_RD_NUMCONE
Number of conic constraints read.

(57) MSK_IINF_RD_NUMINTVAR
Number of integer-constrained variables read.

(58) MSK_IINF_RD_NUMQ
Number of nonempty Q matrices read.

(59) MSK_IINF_RD_NUMVAR
Number of variables read.

(60) MSK_IINF_RD_PROTYPE
Problem type.

(61) MSK_IINF_SIM_DUAL_DEG_ITER
The number of dual degenerate iterations.

(62) MSK_IINF_SIM_DUAL_HOTSTART
If 1 then the dual simplex algorithm is solving from an advanced basis.

(63) 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.

(64) MSK_IINF_SIM_DUAL_INF_ITER
The number of iterations taken with dual infeasibility.

(65) MSK_IINF_SIM_DUAL_ITER
Number of dual simplex iterations during the last optimization.

(66) MSK_IINF_SIM_NETWORK_DUAL_DEG_ITER
The number of dual network degenerate iterations.

(67) MSK_IINF_SIM_NETWORK_DUAL_HOTSTART
If 1 then the dual network simplex algorithm is solving from an advanced basis.

(68) MSK_IINF_SIM_NETWORK_DUAL_HOTSTART_LU
If 1 then a valid basis factorization of full rank was located and used by the dual network simplex algorithm.

(69) MSK_IINF_SIM_NETWORK_DUAL_INF_ITER
The number of iterations taken with dual infeasibility in the network optimizer.

(70) MSK_IINF_SIM_NETWORK_DUAL_ITER
Number of dual network simplex iterations during the last optimization.

(71) MSK_IINF_SIM_NETWORK_PRIMAL_DEG_ITER
The number of primal network degenerate iterations.

(72) MSK_IINF_SIM_NETWORK_PRIMAL_HOTSTART
If 1 then the primal network simplex algorithm is solving from an advanced basis.

(73) MSK_IINF_SIM_NETWORK_PRIMAL_HOTSTART_LU
If 1 then a valid basis factorization of full rank was located and used by the primal network simplex algorithm.

(74) MSK_IINF_SIM_NETWORK_PRIMAL_INF_ITER
The number of iterations taken with primal infeasibility in the network optimizer.

(75) MSK_IINF_SIM_NETWORK_PRIMAL_ITER
Number of primal network simplex iterations during the last optimization.

(76) MSK_IINF_SIM_NUMCON
Number of constraints in the problem solved by the simplex optimizer.

(77) MSK_IINF_SIM_NUMVAR
Number of variables in the problem solved by the simplex optimizer.

(78) MSK_IINF_SIM_PRIMAL_DEG_ITER
The number of primal degenerate iterations.

(79) MSK_IINF_SIM_PRIMAL_DUAL_DEG_ITER
The number of degenerate major iterations taken by the primal dual simplex algorithm.

(80) MSK_IINF_SIM_PRIMAL_DUAL_HOTSTART
If 1 then the primal dual simplex algorithm is solving from an advanced basis.

(81) MSK_IINF_SIM_PRIMAL_DUAL_HOTSTART_LU
If 1 then a valid basis factorization of full rank was located and used by the primal dual simplex algorithm.

(82) MSK_IINF_SIM_PRIMAL_DUAL_INF_ITER
The number of master iterations with dual infeasibility taken by the primal dual simplex algorithm.

(83) MSK_IINF_SIM_PRIMAL_DUAL_ITER
Number of primal dual simplex iterations during the last optimization.

(84) MSK_IINF_SIM_PRIMAL_HOTSTART
If 1 then the primal simplex algorithm is solving from an advanced basis.

(85) 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.

(86) MSK_IINF_SIM_PRIMAL_INF_ITER
The number of iterations taken with primal infeasibility.

(87) MSK_IINF_SIM_PRIMAL_ITER
Number of primal simplex iterations during the last optimization.

(88) MSK_IINF_SIM_SOLVE_DUAL
Is non-zero if dual problem is solved.

(89) MSK_IINF_SOL_BAS_PROSTA
Problem status of the basic solution. Updated after each optimization.

(90) MSK_IINF_SOL_BAS_SOLSTA
Solution status of the basic solution. Updated after each optimization.

(91) MSK_IINF_SOL_INT_PROSTA
Problem status of the integer solution. Updated after each optimization.

(92) MSK_IINF_SOL_INT_SOLSTA
Solution status of the integer solution. Updated after each optimization.

(93) MSK_IINF_SOL_ITR_PROSTA
Problem status of the interior-point solution. Updated after each optimization.

(94) MSK_IINF_SOL_ITR_SOLSTA
Solution status of the interior-point solution. Updated after each optimization.

(95) MSK_IINF_STO_NUM_A_CACHE_FLUSHES
Number of times the cache of A elements is flushed. A large number implies that `maxnumanz` is too small as well as an inefficient usage of MOSEK.

(96) 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.

(97) MSK_IINF_STO_NUM_A_TRANSPOSES
Number of times the A matrix is transposed. A large number implies that `maxnumanz` is too small or an inefficient usage of MOSEK. This will occur in particular if the code alternate between accessing rows and columns of A.

18.17 Information item types

(0) MSK_INF_DOU_TYPE
Is a double information type.

(1) MSK_INF_INT_TYPE
Is an integer.

(2) MSK_INF_LINT_TYPE
Is a long integer.

18.18 Input/output modes

(0) MSK_IOMODE_READ
The file is read-only.

(2) MSK_IOMODE_READWRITE
The file is to read and written.

(1) 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.

18.19 Language selection constants

(1) MSK_LANG_DAN
Danish language selection

(0) MSK_LANG_ENG
English language selection

18.20 Long integer information items.

(0) MSK_LIINF_BI_CLEAN_DUAL_DEG_ITER
Number of dual degenerate clean iterations performed in the basis identification.

(1) MSK_LIINF_BI_CLEAN_DUAL_ITER
Number of dual clean iterations performed in the basis identification.

(2) MSK_LIINF_BI_CLEAN_PRIMAL_DEG_ITER
Number of primal degenerate clean iterations performed in the basis identification.

(3) MSK_LIINF_BI_CLEAN_PRIMAL_DUAL_DEG_ITER
Number of primal-dual degenerate clean iterations performed in the basis identification.

(4) MSK_LIINF_BI_CLEAN_PRIMAL_DUAL_ITER
Number of primal-dual clean iterations performed in the basis identification.

(5) MSK_LIINF_BI_CLEAN_PRIMAL_DUAL_SUB_ITER
Number of primal-dual subproblem clean iterations performed in the basis identification.

(6) MSK_LIINF_BI_CLEAN_PRIMAL_ITER
Number of primal clean iterations performed in the basis identification.

(7) MSK_LIINF_BI_DUAL_ITER
Number of dual pivots performed in the basis identification.

(8) MSK_LIINF_BI_PRIMAL_ITER
Number of primal pivots performed in the basis identification.

(9) MSK_LIINF_INTPNT_FACTOR_NUM_NZ
Number of non-zeros in factorization.

(10) MSK_LIINF_MIO_INTPNT_ITER
Number of interior-point iterations performed by the mixed-integer optimizer.

(11) MSK_LIINF_MIO_SIMPLEX_ITER
Number of simplex iterations performed by the mixed-integer optimizer.

(12) MSK_LIINF_RD_NUMANZ
Number of non-zeros in A that is read.

(13) MSK_LIINF_RD_NUMQNZ
Number of Q non-zeros.

18.21 Mark

(0) MSK_MARK_LO
The lower bound is selected for sensitivity analysis.

(1) MSK_MARK_UP
The upper bound is selected for sensitivity analysis.

18.22 Continuous mixed-integer solution type

(2) 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.

(3) 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.

(0) MSK_MIO_CONT_SOL_NONE
No interior-point or basic solution are reported when the mixed-integer optimizer is used.

(1) 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.

18.23 Integer restrictions

(0) MSK_MIO_MODE_IGNORED
The integer constraints are ignored and the problem is solved as a continuous problem.

(2) MSK_MIO_MODE_LAZY
Integer restrictions should be satisfied if an optimizer is available for the problem.

(1) MSK_MIO_MODE_SATISFIED
Integer restrictions should be satisfied.

18.24 Mixed-integer node selection types

(2) MSK_MIO_NODE_SELECTION_BEST
The optimizer employs a best bound node selection strategy.

(1) MSK_MIO_NODE_SELECTION_FIRST
The optimizer employs a depth first node selection strategy.

(0) MSK_MIO_NODE_SELECTION_FREE
The optimizer decides the node selection strategy.

(4) MSK_MIO_NODE_SELECTION_HYBRID
The optimizer employs a hybrid strategy.

(5) MSK_MIO_NODE_SELECTION_PSEUDO
The optimizer employs selects the node based on a pseudo cost estimate.

(3) MSK_MIO_NODE_SELECTION_WORST
The optimizer employs a worst bound node selection strategy.

18.25 MPS file format type

(2) 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.

(1) MSK_MPS_FORMAT_RELAXED
It is assumed that the input file satisfies a slightly relaxed version of the MPS format.

(0) MSK_MPS_FORMAT_STRICT
It is assumed that the input file satisfies the MPS format strictly.

18.26 Message keys

(1100) MSK_MSG_MPS_SELECTED

(1000) MSK_MSG_READING_FILE

(1001) MSK_MSG_WRITING_FILE

18.27 Network detection method

(2) MSK_NETWORK_DETECT_ADVANCED
The network detection should use a more advanced heuristic.

(0) MSK_NETWORK_DETECT_FREE
The network detection is free.

(1) MSK_NETWORK_DETECT_SIMPLE
The network detection should use a very simple heuristic.

18.28 Objective sense types

(2) MSK_OBJECTIVE_SENSE_MAXIMIZE
The problem should be maximized.

(1) MSK_OBJECTIVE_SENSE_MINIMIZE
The problem should be minimized.

(0) MSK_OBJECTIVE_SENSE_UNDEFINED
The objective sense is undefined.

18.29 On/off

(0) MSK_OFF
Switch the option off.

(1) MSK_ON
Switch the option on.

18.30 Optimizer types

(10) MSK_OPTIMIZER_CONCURRENT
The optimizer for nonconvex nonlinear problems.

(2) MSK_OPTIMIZER_CONIC
The optimizer for problems having conic constraints.

(5) MSK_OPTIMIZER_DUAL_SIMPLEX
The dual simplex optimizer is used.

(0) MSK_OPTIMIZER_FREE
The optimizer is chosen automatically.

(7) MSK_OPTIMIZER_FREE_SIMPLEX
One of the simplex optimizers is used.

(1) MSK_OPTIMIZER_INTPNT
The interior-point optimizer is used.

(8) MSK_OPTIMIZER_MIXED_INT
The mixed-integer optimizer.

(9) MSK_OPTIMIZER_NONCONVEX
The optimizer for nonconvex nonlinear problems.

(6) MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX
The primal dual simplex optimizer is used.

(4) MSK_OPTIMIZER_PRIMAL_SIMPLEX
The primal simplex optimizer is used.

(3) MSK_OPTIMIZER_QCONE
For internal use only.

18.31 Ordering strategies

(1) MSK_ORDER_METHOD_APPMINLOC1
Approximate minimum local-fill-in ordering is used.

(2) MSK_ORDER_METHOD_APPMINLOC2
A variant of the approximate minimum local-fill-in ordering is used.

(0) MSK_ORDER_METHOD_FREE
The ordering method is chosen automatically.

(3) MSK_ORDER_METHOD_GRAPHPAR1
Graph partitioning based ordering.

(4) MSK_ORDER_METHOD_GRAPHPAR2
An alternative graph partitioning based ordering.

(5) MSK_ORDER_METHOD_NONE
No ordering is used.

18.32 Parameter type

(1) MSK_PAR_DOU_TYPE
Is a double parameter.

(2) MSK_PAR_INT_TYPE
Is an integer parameter.

(0) MSK_PAR_INVALID_TYPE
Not a valid parameter.

(3) MSK_PAR_STR_TYPE
Is a string parameter.

18.33 Presolve method.

(2) MSK_PRESOLVE_MODE_FREE
It is decided automatically whether to presolve before the problem is optimized.

(0) MSK_PRESOLVE_MODE_OFF
The problem is not presolved before it is optimized.

(1) MSK_PRESOLVE_MODE_ON
The problem is presolved before it is optimized.

18.34 Problem data items

(1) MSK_PI_CON
Item is a constraint.

(2) MSK_PI_CONE
Item is a cone.

(0) MSK_PI_VAR
Item is a variable.

18.35 Problem types

(4) MSK_PROBTYPE_CONIC
A conic optimization.

(3) MSK_PROBTYPE_GECO
General convex optimization.

(0) MSK_PROBTYPE_LO
The problem is a linear optimization problem.

(5) MSK_PROBTYPE_MIXED
General nonlinear constraints and conic constraints. This combination can not be solved by MOSEK.

(2) MSK_PROBTYPE_QCQO
The problem is a quadratically constrained optimization problem.

(1) MSK_PROBTYPE_QO
The problem is a quadratic optimization problem.

18.36 Problem status keys

(3) MSK_PRO_STA_DUAL_FEAS
The problem is dual feasible.

(5) MSK_PRO_STA_DUAL_INFEAS
The problem is dual infeasible.

(7) MSK_PRO_STA_ILL_POSED
The problem is ill-posed. For example, it may be primal and dual feasible but have a positive duality gap.

(10) MSK_PRO_STA_NEAR_DUAL_FEAS
The problem is at least nearly dual feasible.

(8) MSK_PRO_STA_NEAR_PRIM_AND_DUAL_FEAS
The problem is at least nearly primal and dual feasible.

(9) MSK_PRO_STA_NEAR_PRIM_FEAS
The problem is at least nearly primal feasible.

(1) MSK_PRO_STA_PRIM_AND_DUAL_FEAS
The problem is primal and dual feasible.

(6) MSK_PRO_STA_PRIM_AND_DUAL_INFEAS
The problem is primal and dual infeasible.

(2) MSK_PRO_STA_PRIM_FEAS
The problem is primal feasible.

(4) MSK_PRO_STA_PRIM_INFEAS
The problem is primal infeasible.

(11) MSK_PRO_STA_PRIM_INFEAS_OR_UNBOUNDED
The problem is either primal infeasible or unbounded. This may occur for mixed-integer problems.

(0) MSK_PRO_STA_UNKNOWN
Unknown problem status.

18.37 Interpretation of quadratic terms in MPS files

(0) MSK_Q_READ_ADD
All elements in a Q matrix are assumed to belong to the lower triangular part. Duplicate elements in a Q matrix are added together.

(1) MSK_Q_READ_DROP_LOWER
All elements in the strict lower triangular part of the Q matrices are dropped.

(2) MSK_Q_READ_DROP_UPPER
All elements in the strict upper triangular part of the Q matrices are dropped.

18.38 Response code type

(3) MSK_RESPONSE_ERR
The response code is an error.

(0) MSK_RESPONSE_OK
The response code is OK.

(2) MSK_RESPONSE_TRM
The response code is an optimizer termination status.

(4) MSK_RESPONSE_UNK
The response code does not belong to any class.

(1) MSK_RESPONSE_WRN
The response code is a warning.

18.39 Scaling type

(1) MSK_SCALING_METHOD_FREE
The optimizer chooses the scaling heuristic.

(0) MSK_SCALING_METHOD_POW2
Scales only with power of 2 leaving the mantissa untouched.

18.40 Scaling type

(3) MSK_SCALING_AGGRESSIVE
A very aggressive scaling is performed.

(0) MSK_SCALING_FREE
The optimizer chooses the scaling heuristic.

(2) MSK_SCALING_MODERATE
A conservative scaling is performed.

(1) MSK_SCALING_NONE
No scaling is performed.

18.41 Sensitivity types

(0) MSK_SENSITIVITY_TYPE_BASIS
Basis sensitivity analysis is performed.

(1) MSK_SENSITIVITY_TYPE_OPTIMAL_PARTITION
Optimal partition sensitivity analysis is performed.

18.42 Degeneracy strategies

(2) MSK_SIM_DEGEN_AGGRESSIVE
The simplex optimizer should use an aggressive degeneration strategy.

(1) MSK_SIM_DEGEN_FREE
The simplex optimizer chooses the degeneration strategy.

(4) MSK_SIM_DEGEN_MINIMUM
The simplex optimizer should use a minimum degeneration strategy.

(3) MSK_SIM_DEGEN_MODERATE
The simplex optimizer should use a moderate degeneration strategy.

(0) MSK_SIM_DEGEN_NONE
The simplex optimizer should use no degeneration strategy.

18.43 Exploit duplicate columns.

(2) MSK_SIM_EXPLOIT_DUPVEC_FREE
The simplex optimizer can choose freely.

(0) MSK_SIM_EXPLOIT_DUPVEC_OFF
Disallow the simplex optimizer to exploit duplicated columns.

(1) MSK_SIM_EXPLOIT_DUPVEC_ON
Allow the simplex optimizer to exploit duplicated columns.

18.44 Hot-start type employed by the simplex optimizer

(1) MSK_SIM_HOTSTART_FREE
The simplex optimize chooses the hot-start type.

(0) MSK_SIM_HOTSTART_NONE
The simplex optimizer performs a coldstart.

(2) MSK_SIM_HOTSTART_STATUS_KEYS
Only the status keys of the constraints and variables are used to choose the type of hot-start.

18.45 Problem reformulation.

(3) MSK_SIM_REFORMULATION_AGGRESSIVE
The simplex optimizer should use an aggressive reformulation strategy.

(2) MSK_SIM_REFORMULATION_FREE
The simplex optimizer can choose freely.

(0) MSK_SIM_REFORMULATION_OFF
Disallow the simplex optimizer to reformulate the problem.

(1) MSK_SIM_REFORMULATION_ON
Allow the simplex optimizer to reformulate the problem.

18.46 Simplex selection strategy

(2) MSK_SIM_SELECTION_ASE
The optimizer uses approximate steepest-edge pricing.

(3) MSK_SIM_SELECTION_DEVEX
The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).

(0) MSK_SIM_SELECTION_FREE
The optimizer chooses the pricing strategy.

(1) MSK_SIM_SELECTION_FULL
The optimizer uses full pricing.

(5) 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.

(4) MSK_SIM_SELECTION_SE
The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

18.47 Solution items

(3) MSK_SOL_ITEM_SLC
Lagrange multipliers for lower bounds on the constraints.

(5) MSK_SOL_ITEM_SLX
Lagrange multipliers for lower bounds on the variables.

(7) MSK_SOL_ITEM_SNX
Lagrange multipliers corresponding to the conic constraints on the variables.

(4) MSK_SOL_ITEM_SUC
Lagrange multipliers for upper bounds on the constraints.

(6) MSK_SOL_ITEM_SUX
Lagrange multipliers for upper bounds on the variables.

(0) MSK_SOL_ITEM_XC
Solution for the constraints.

(1) MSK_SOL_ITEM_XX
Variable solution.

(2) MSK_SOL_ITEM_Y
Lagrange multipliers for equations.

18.48 Solution status keys

(3) MSK_SOL_STA_DUAL_FEAS
The solution is dual feasible.

(6) MSK_SOL_STA_DUAL_INFEAS_CER
The solution is a certificate of dual infeasibility.

(14) MSK_SOL_STA_INTEGER_OPTIMAL
The primal solution is integer optimal.

(10) MSK_SOL_STA_NEAR_DUAL_FEAS
The solution is nearly dual feasible.

(13) MSK_SOL_STA_NEAR_DUAL_INFEAS_CER
The solution is almost a certificate of dual infeasibility.

(15) MSK_SOL_STA_NEAR_INTEGER_OPTIMAL
The primal solution is near integer optimal.

(8) MSK_SOL_STA_NEAR_OPTIMAL
The solution is nearly optimal.

(11) MSK_SOL_STA_NEAR_PRIM_AND_DUAL_FEAS
The solution is nearly both primal and dual feasible.

(9) MSK_SOL_STA_NEAR_PRIM_FEAS
The solution is nearly primal feasible.

(12) MSK_SOL_STA_NEAR_PRIM_INFEAS_CER
The solution is almost a certificate of primal infeasibility.

(1) MSK_SOL_STA_OPTIMAL
The solution is optimal.

(4) MSK_SOL_STA_PRIM_AND_DUAL_FEAS
The solution is both primal and dual feasible.

(2) MSK_SOL_STA_PRIM_FEAS
The solution is primal feasible.

(5) MSK_SOL_STA_PRIM_INFEAS_CER
The solution is a certificate of primal infeasibility.

(0) MSK_SOL_STA_UNKNOWN
Status of the solution is unknown.

18.49 Solution types

(1) MSK_SOL_BAS
The basic solution.

(2) MSK_SOL_ITG
The integer solution.

(0) MSK_SOL_ITR
The interior solution.

18.50 Solve primal or dual form

(2) MSK_SOLVE_DUAL
The optimizer should solve the dual problem.

(0) MSK_SOLVE_FREE
The optimizer is free to solve either the primal or the dual problem.

(1) MSK_SOLVE_PRIMAL
The optimizer should solve the primal problem.

18.51 Status keys

(1) MSK_SK_BAS
The constraint or variable is in the basis.

(5) MSK_SK_FIX
The constraint or variable is fixed.

(6) MSK_SK_INF
The constraint or variable is infeasible in the bounds.

(3) MSK_SK_LOW
The constraint or variable is at its lower bound.

(2) MSK_SK_SUPBAS
The constraint or variable is super basic.

(0) MSK_SK_UNK
The status for the constraint or variable is unknown.

(4) MSK_SK_UPR
The constraint or variable is at its upper bound.

18.52 Starting point types

(2) 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.

(0) MSK_STARTING_POINT_FREE
The starting point is chosen automatically.

(1) MSK_STARTING_POINT_GUESS
The optimizer guesses a starting point.

(3) MSK_STARTING_POINT_SATISFY_BOUNDS
The starting point is choosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

18.53 Stream types

(2) MSK_STREAM_ERR
Error stream. Error messages are written to this stream.

(0) 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.

(1) MSK_STREAM_MSG
Message stream. Log information relating to performance and progress of the optimization is written to this stream.

(3) MSK_STREAM_WRN
Warning stream. Warning messages are written to this stream.

18.54 Integer values

(20) MSK_LICENSE_BUFFER_LENGTH
The length of a license key buffer.

(1024) MSK_MAX_STR_LEN
Maximum string length allowed in MOSEK.

18.55 Variable types

(0) MSK_VAR_TYPE_CONT
Is a continuous variable.

(1) MSK_VAR_TYPE_INT
Is an integer variable.

18.56 XML writer output mode

(1) MSK_WRITE_XML_MODE_COL
Write in column order.

(0) `MSK_WRITE_XML_MODE_ROW`
Write in row order.

Appendix A

Troubleshooting

- *The application compiles, but when the first MOSEK function is called, an error "*`OMP abort: Initializing libguide40.lib, but found libguide.lib already initialized`*".*

  MOSEK used `libguide40.dll` (an Intel threading library). The error means that the application also links to some other library which is statically linked with `libguide.lib`, which may clash with `libguide40.dll`.

  If possible, relink the offending DLL with the dynamic version (`libguide40.lib` instead of `libguide.lib`), otherwise set the environment variable "`KMP_DUPLICATE_LIB_OK`" to "`TRUE`".

Appendix B

Problem analyzer examples

This appendix presents a few examples of the output produced by the problem analyzer described in Section 10.1. The first two problems are taken from the MIPLIB 2003 collection, http://miplib.zib.de/.

B.1 air04

```
Analyzing the problem

Constraints               Bounds                    Variables
 fixed   : all             ranged  : all             bin : all

--------------------------------------------------------------------------------

Objective, min cx
   range: min |c|: 31.0000     max |c|: 2258.00
 distrib:        |c|        vars
          [31, 100)         176
       [100, 1e+03)        8084
   [1e+03, 2.26e+03]        644

--------------------------------------------------------------------------------

Constraint matrix A has
       823 rows (constraints)
      8904 columns (variables)
     72965 (0.995703%) nonzero entries (coefficients)

Row nonzeros, A_i
   range: min A_i: 2 (0.0224618%)    max A_i: 368 (4.13297%)
 distrib:        A_i        rows       rows%        acc%
                   2           2        0.24        0.24
               [3, 7]          4        0.49        0.73
              [8, 15]         19        2.31        3.04
             [16, 31]         80        9.72       12.76
             [32, 63]        236       28.68       41.43
```

```
          [64, 127]         289       35.12       76.55
         [128, 255]         186       22.60       99.15
         [256, 368]           7        0.85      100.00

Column nonzeros, A|j
   range: min A|j: 2 (0.243013%)    max A|j: 15 (1.8226%)
 distrib:        A|j        cols       cols%        acc%
                   2         118        1.33        1.33
              [3, 7]        2853       32.04       33.37
             [8, 15]        5933       66.63      100.00

A nonzeros, A(ij)
   range: all |A(ij)| = 1.00000

--------------------------------------------------------------------------------

Constraint bounds, lb <= Ax <= ub
 distrib:        |b|             lbs             ubs
             [1, 10]             823             823

Variable bounds, lb <= x <= ub
 distrib:        |b|             lbs             ubs
                   0            8904
             [1, 10]                            8904

--------------------------------------------------------------------------------
```

B.2 arki001

```
Analyzing the problem

Constraints               Bounds                    Variables
 lower bd:        82       lower bd:        38       cont:       850
 upper bd:       946       fixed   :       353       bin :       415
 fixed   :        20       free    :         1       int :       123
                           ranged  :       996

--------------------------------------------------------------------------------

Objective, min cx
   range: all |c| in {0.00000, 1.00000}
 distrib:        |c|        vars
                   0        1387
                   1           1

--------------------------------------------------------------------------------

Constraint matrix A has
      1048 rows (constraints)
      1388 columns (variables)
     20439 (1.40511%) nonzero entries (coefficients)

Row nonzeros, A_i
   range: min A_i: 1 (0.0720461%)    max A_i: 1046 (75.3602%)
 distrib:        A_i        rows       rows%        acc%
                   1          29        2.77        2.77
```

```
                  2         476       45.42       48.19
             [3, 7]          49        4.68       52.86
            [8, 15]          56        5.34       58.21
           [16, 31]          64        6.11       64.31
           [32, 63]         373       35.59       99.90
       [1024, 1046]           1        0.10      100.00

Column nonzeros, A|j
   range: min A|j: 1 (0.0954198%)    max A|j: 29 (2.76718%)
 distrib:        A|j        cols       cols%        acc%
                   1         381       27.45       27.45
                   2          19        1.37       28.82
             [3, 7]          38        2.74       31.56
            [8, 15]         233       16.79       48.34
           [16, 29]         717       51.66      100.00

A nonzeros, A(ij)
   range: min |A(ij)|: 0.000200000     max |A(ij)|: 2.33067e+07
 distrib:      A(ij)      coeffs
     [0.0002, 0.001)         167
       [0.001, 0.01)        1049
         [0.01, 0.1)        4553
            [0.1, 1)        8840
             [1, 10)        3822
          [10, 100)          630
        [100, 1e+03)         267
      [1e+03, 1e+04)         699
      [1e+04, 1e+05)         291
      [1e+05, 1e+06)          83
      [1e+06, 1e+07)          19
   [1e+07, 2.33e+07]          19

--------------------------------------------------------------------------------

Constraint bounds, lb <= Ax <= ub
 distrib:        |b|             lbs             ubs
            [0.1, 1)                            386
             [1, 10)                             74
          [10, 100)             101             456
         [100, 1000)                             34
       [1000, 10000)                             15
     [100000, 1e+06]               1               1

Variable bounds, lb <= x <= ub
 distrib:        |b|             lbs             ubs
                   0             974             323
       [0.001, 0.01)                             19
            [0.1, 1)             370              57
             [1, 10)              41             704
          [10, 100]               2             246

--------------------------------------------------------------------------------
```

B.3 Problem with both linear and quadratic constraints

```
Analyzing the problem
```

```
Constraints               Bounds                    Variables
 lower bd:        40       upper bd:         1       cont: all
 upper bd:       121       fixed   :       204
 fixed   :      5480       free    :      5600
 ranged  :       161       ranged  :        40

--------------------------------------------------------------------------------

Objective, maximize cx
   range: all |c| in {0.00000, 15.4737}
 distrib:        |c|        vars
                   0        5844
             15.4737           1

--------------------------------------------------------------------------------

Constraint matrix A has
      5802 rows (constraints)
      5845 columns (variables)
      6480 (0.0191079%) nonzero entries (coefficients)

Row nonzeros, A_i
   range: min A_i: 0 (0%)    max A_i: 3 (0.0513259%)
 distrib:        A_i        rows       rows%        acc%
                   0          80        1.38        1.38
                   1        5003       86.23       87.61
                   2         680       11.72       99.33
                   3          39        0.67      100.00
0/80 empty rows have quadratic terms

Column nonzeros, A|j
   range: min A|j: 0 (0%)    max A|j: 15 (0.258532%)
 distrib:        A|j        cols       cols%        acc%
                   0         204        3.49        3.49
                   1        5521       94.46       97.95
                   2          40        0.68       98.63
              [3, 7]          40        0.68       99.32
             [8, 15]          40        0.68      100.00
0/204 empty columns correspond to variables used in conic
 and/or quadratic expressions only

A nonzeros, A(ij)
   range: min |A(ij)|: 2.02410e-05     max |A(ij)|: 35.8400
 distrib:      A(ij)      coeffs
  [2.02e-05, 0.0001)          40
     [0.0001, 0.001)         118
       [0.001, 0.01)         305
         [0.01, 0.1)         176
            [0.1, 1)          40
             [1, 10)        5721
          [10, 35.8]          80

--------------------------------------------------------------------------------

Constraint bounds, lb <= Ax <= ub
 distrib:        |b|             lbs             ubs
                   0            5481            5600
```

```
       [1000, 10000)                              1
     [10000, 100000)              2               1
      [1e+06, 1e+07)             78              40
      [1e+08, 1e+09]             120             120

Variable bounds, lb <= x <= ub
 distrib:        |b|             lbs             ubs
                   0             243             203
             [0.1, 1)              1               1
      [1e+06, 1e+07)                              40
      [1e+11, 1e+12]                               1

------------------------------------------------------------------------------

Quadratic constraints: 121

Gradient nonzeros, Qx
   range: min Qx: 1 (0.0171086%)    max Qx: 2720 (46.5355%)
 distrib:         Qx        cons       cons%        acc%
                   1          40       33.06       33.06
           [64, 127]          80       66.12       99.17
        [2048, 2720]           1        0.83      100.00

------------------------------------------------------------------------------
```

B.4 Problem with both linear and conic constraints

```
Analyzing the problem

Constraints               Bounds                    Variables
 upper bd:      3600       fixed   :      3601       cont: all
 fixed   :     21760       free    :     28802

------------------------------------------------------------------------------

Objective, minimize cx
   range: all |c| in {0.00000, 1.00000}
 distrib:        |c|        vars
                   0       32402
                   1           1

------------------------------------------------------------------------------

Constraint matrix A has
     25360 rows (constraints)
     32403 columns (variables)
     93339 (0.0113587%) nonzero entries (coefficients)

Row nonzeros, A_i
   range: min A_i: 1 (0.00308613%)    max A_i: 8 (0.0246891%)
 distrib:        A_i        rows       rows%        acc%
                   1        3600       14.20       14.20
                   2       10803       42.60       56.79
              [3, 7]        3995       15.75       72.55
                   8        6962       27.45      100.00
```

```
Column nonzeros, A|j
   range: min A|j: 0 (0%)    max A|j: 61 (0.240536%)
 distrib:        A|j        cols       cols%        acc%
                   0        3602       11.12       11.12
                   1       10800       33.33       44.45
                   2        7200       22.22       66.67
              [3, 7]        7279       22.46       89.13
             [8, 15]        3521       10.87      100.00
            [32, 61]           1        0.00      100.00
3600/3602 empty columns correspond to variables used in conic
 and/or quadratic constraints only

A nonzeros, A(ij)
   range: min |A(ij)|: 0.00833333     max |A(ij)|: 1.00000
 distrib:      A(ij)      coeffs
     [0.00833, 0.01)       57280
         [0.01, 0.1)          59
            [0.1, 1]       36000

--------------------------------------------------------------------------------

Constraint bounds, lb <= Ax <= ub
 distrib:        |b|             lbs             ubs
                   0           21760           21760
            [0.1, 1]                            3600

Variable bounds, lb <= x <= ub
 distrib:        |b|             lbs             ubs
             [1, 10]            3601            3601

--------------------------------------------------------------------------------


--------------------------------------------------------------------------------

Rotated quadratic cones: 3600
                 dim        RQCs
                   4        3600
```

Appendix C

The MPS file format

MOSEK supports the standard MPS format with some extensions. For a detailed description of the MPS format the book by Nazareth [20] is a good reference.

C.1 The MPS file format

The version of the MPS format supported by MOSEK allows specification of an optimization problem on the form

$$\begin{array}{rcccl} l^c & \leq & Ax + q(x) & \leq & u^c, \\ l^x & \leq & x & \leq & u^x, \\ & & x \in \mathcal{C}, & & \\ & & x_{\mathcal{J}} \text{ integer}, & & \end{array} \tag{C.1}$$

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.
- $q : \mathbb{R}^n \to \mathbb{R}$ is a vector of quadratic functions. Hence,

  $$q_i(x) = 1/2 x^T Q^i x$$

  where it is assumed that

  $$Q^i = (Q^i)^T. \tag{C.2}$$

  Please note the explicit $1/2$ in the quadratic term and that Q^i is required to be symmetric.

- $\mathcal{C}$ is a convex cone.
- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer-constrained variables.

An MPS file with one row and one column can be illustrated like this:

```
*        1         2         3         4         5         6
*234567890123456789012345678901234567890123456789012345678901234567890
NAME          [name]
OBJSENSE
    [objsense]
OBJNAME
    [objname]
ROWS
 ?  [cname1]
COLUMNS
    [vname1]  [cname1]    [value1]     [vname3]  [value2]
RHS
    [name]    [cname1]    [value1]     [cname2]  [value2]
RANGES
    [name]    [cname1]    [value1]     [cname2]  [value2]
QSECTION      [cname1]
    [vname1]  [vname2]    [value1]     [vname3]  [value2]
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 MOSEK specific extensions of the MPS format.

The standard MPS format is a fixed format, i.e. everything in the MPS file must be within certain fixed positions. MOSEK also supports a *free format*. See Section C.5 for details.

C.1.1 An example

A concrete example of a MPS file is presented below:

```
NAME          EXAMPLE
OBJSENSE
    MIN
ROWS
 N  obj
 L  c1
 L  c2
 L  c3
 L  c4
COLUMNS
    x1        obj       -10.0          c1        0.7
    x1        c2        0.5            c3        1.0
    x1        c4        0.1
    x2        obj       -9.0           c1        1.0
    x2        c2        0.8333333333   c3        0.66666667
    x2        c4        0.25
RHS
    rhs       c1        630.0          c2        600.0
    rhs       c3        708.0          c4        135.0
ENDATA
```

Subsequently each individual section in the MPS format is discussed.

C.1.2 `NAME`

In this section a name (`[name]`) is assigned to the problem.

C.1.3 `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.

C.1.4 OBJNAME (optional)

This is an optional section that can be used to specify the name of the row that is used as objective function. The OBJNAME section contains one line at most which has the form

```
objname
```

objname should be a valid row name.

C.1.5 ROWS

A record in the ROWS section has the form

```
?  [cname1]
```

where the requirements for the fields are as follows:

Field	Starting position	Maximum width	Required	Description
?	2	1	Yes	Constraint key
[cname1]	5	8	Yes	Constraint name

Hence, in this section each constraint is assigned an unique name denoted by [cname1]. Please note that [cname1] starts in position 5 and the field can be at most 8 characters wide. An initial key (?) must be present to specify the type of the constraint. The key can have the values E, G, L, or N whith ther following interpretation:

Constraint type	l_i^c	u_i^c
E	finite	l_i^c
G	finite	∞
L	$-\infty$	finite
N	$-\infty$	∞

In the MPS format an objective vector is not specified explicitly, but one of the constraints having the key N will be used as the objective vector c. In general, if multiple N type constraints are specified, then the first will be used as the objective vector c.

C.1.6 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	Maximum 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.

C.1.7 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	Maximum 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 ith constraint and v_1 denotes the value specified by [value1], then the interpretation of v_1 is:

Constraint type	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.

C.1.8 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	Maximum 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 anhd [cname1] is a valid constraint name. Assume that [cname1] is assigned to the ith 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			

C.1.9 QSECTION (optional)

Within the QSECTION the label [cname1] must be a constraint name previously specified in the ROWS section. The label [cname1] denotes the constraint to which the quadratic term belongs. A record in the QSECTION has the form

```
[vname1]   [vname2]     [value1]      [vname3]   [value2]
```

where the requirements for each field are:

Field	Starting position	Maximum 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 kth and jth 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

$$\begin{array}{lcll} \text{minimize} & -x_2 + 0.5(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & & \\ \text{subject to} & x_1 + x_2 + x_3 & \geq & 1, \\ & x \geq 0 & & \end{array}$$

has the following MPS file representation

```
NAME          qoexp
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        1
    x2        obj       -1
    x2        c1        1
    x3        c1        1
RHS
    rhs       c1        1
QSECTION      obj
    x1        x1        2
    x1        x3        -1
    x2        x2        0.2
    x3        x3        2
ENDATA
```

Regarding the QSECTIONs please note that:

- Only one QSECTION is allowed for each constraint.

- The `QSECTION`s can appear in an arbitrary order after the `COLUMNS` section.
- All variable names occurring in the `QSECTION` must already be specified in the `COLUMNS` section.
- All entries specified in a `QSECTION` are assumed to belong to the lower triangular part of the quadratic term of Q.

C.1.10 `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	Maximum 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	Variable name

Hence, a record in the `BOUNDS` section has the following interpretation: `[name]` is the name of the bound vector and `[vname1]` is the name of the variable which bounds are modified by the record. `??` and `[value1]` are used to modify the bound vectors according to the following table:

`??`	l_j^x	u_j^x	Made integer (added to $\mathcal{J}$)
`FR`	$-\infty$	∞	No
`FX`	v_1	v_1	No
`LO`	v_1	unchanged	No
`MI`	$-\infty$	unchanged	No
`PL`	unchanged	∞	No
`UP`	unchanged	v_1	No
`BV`	0	1	Yes
`LI`	$\lceil v_1 \rceil$	∞	Yes
`UI`	unchanged	$\lfloor v_1 \rfloor$	Yes

v_1 is the value specified by `[value1]`.

C.1.11 `CSECTION` (optional)

The purpose of the `CSECTION` is to specify the constraint

$$x \in \mathcal{C}.$$

in (C.1).

It is assumed that $\mathcal{C}$ satisfies the following requirements. Let

$$x^t \in \mathbb{R}^{n^t},\ t = 1, \ldots, k$$

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

$$x^1 = \begin{bmatrix} x_1 \\ x_4 \\ x_7 \end{bmatrix} \text{ and } x^2 = \begin{bmatrix} x_6 \\ x_5 \\ x_3 \\ x_2 \end{bmatrix}.$$

Next define

$$\mathcal{C} := \left\{ x \in \mathbb{R}^n :\ x^t \in \mathcal{C}_t,\ t = 1, \ldots, k \right\}$$

where $\mathcal{C}_t$ must have one of the following forms

- $\mathbb{R}$ set:
$$\mathcal{C}_t = \{x \in \mathbb{R}^{n^t}\}.$$
- Quadratic cone:
$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \tag{C.3}$$
- Rotated quadratic cone:
$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1 x_2 \geq \sum_{j=3}^{n^t} x_j^2,\ x_1, x_2 \geq 0 \right\}. \tag{C.4}$$

In general, only quadratic and rotated quadratic cones are specified in the MPS file whereas membership of the $\mathbb{R}$ set is not. If a variable is not a member of any other cone then it is assumed to be a member of an $\mathbb{R}$ cone.

Next, let us study an example. Assume that the quadratic cone

$$x_4 \geq \sqrt{x_5^2 + x_0^2} \tag{C.5}$$

and the rotated quadratic cone

$$2x_3 x_7 \geq x_1^2 + x_8^2,\ x_3, x_7 \geq 0, \tag{C.6}$$

should be specified in the MPS file. One `CSECTION` is required for each cone and they are specified as follows:

```
*        1         2         3         4         5         6
*23456789012345678901234567890123456789012345678901234567890
CSECTION       konea        0.0           QUAD
    x4
    x5
    x8
CSECTION       koneb        0.0           RQUAD
    x7
    x3
    x1
    x0
```

This first `CSECTION` specifies the cone (C.5) which is given the name `konea`. This is a quadratic cone which is specified by the keyword `QUAD` in the `CSECTION` header. The 0.0 value in the `CSECTION` header is not used by the `QUAD` cone.

The second `CSECTION` specifies the rotated quadratic cone (C.6). Please note the keyword `RQUAD` in the `CSECTION` which is used to specify that the cone is a rotated quadratic cone instead of a quadratic cone. The 0.0 value in the `CSECTION` header is not used by the `RQUAD` cone.

In general, a `CSECTION` header has the format

```
CSECTION       [kname1]     [value1]     [ktype]
```

where the requirement for each field are as follows:

Field	Starting position	Maximum width	Required	Description
`[kname1]`	5	8	Yes	Name of the cone
`[value1]`	15	12	No	Cone parameter
`[ktype]`	25		Yes	Type of the cone.

The possible cone type keys are:

Cone type key	Members	Interpretation.
`QUAD`	≥ 1	Quadratic cone i.e. (C.3).
`RQUAD`	≥ 2	Rotated quadratic cone i.e. (C.4).

Please note that a quadratic cone must have at least one member whereas a rotated quadratic cone must have at least two members. A record in the `CSECTION` has the format

```
    [vname1]
```

where the requirements for each field are

Field	Starting position	Maximum width	Required	Description
`[vname1]`	2	8	Yes	A valid variable name

The most important restriction with respect to the CSECTION is that a variable must occur in only one CSECTION.

C.1.12 ENDATA

This keyword denotes the end of the MPS file.

C.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 compability and we recommend that it is not used. This method requires that markers are placed in the COLUMNS section as in the example:

```
COLUMNS
    x1        obj       -10.0          c1        0.7
    x1        c2        0.5            c3        1.0
    x1        c4        0.1
* Start of integer-constrained variables.
    MARK000   'MARKER'                 'INTORG'
    x2        obj       -9.0           c1        1.0
    x2        c2        0.8333333333   c3        0.66666667
    x2        c4        0.25
    x3        obj       1.0            c6        2.0
    MARK001   'MARKER'                 'INTEND'
* End of integer-constrained variables.
```

Please note that special marker lines are used to indicate the start and the end of the integer variables. Furthermore be aware of the following

- IMPORTANT: All variables between the markers are assigned a default lower bound of 0 and a default upper bound of 1. **This may not be what is intended.** If it is not intended, the correct bounds should be defined in the BOUNDS section of the MPS formatted file.
- MOSEK ignores field 1, i.e. MARK0001 and MARK001, however, other optimization systems require them.
- Field 2, i.e. 'MARKER', must be specified including the single quotes. This implies that no row can be assigned the name 'MARKER'.
- Field 3 is ignored and should be left blank.
- Field 4, i.e. 'INTORG' and 'INTEND', must be specified.
- It is possible to specify several such integer marker sections within the COLUMNS section.

C.3 General limitations

- An MPS file should be an ASCII file.

C.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.

C.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, it also presents two main limitations:

- By default a line in the MPS file must not contain more than 1024 characters. However, by modifying the parameter `MSK_IPAR_READ_MPS_WIDTH` an arbitrary large line width will be accepted.
- A name must not contain any blanks.

To use the free MPS format instead of the default MPS format the MOSEK parameter `MSK_IPAR_READ_MPS_FORMAT` should be changed.

Appendix D

The LP file format

MOSEK supports the LP file format with some extensions i.e. MOSEK can read and write LP formatted files.

D.1 A warning

The LP format is not a well-defined standard and hence different optimization packages may interpretate a specific LP formatted file differently.

D.2 The LP file format

The LP file format can specify problems on the form

$$\begin{array}{lccccc} \text{minimize/maximize} & & & c^T x + \frac{1}{2} q^o(x) & & \\ \text{subject to} & l^c & \leq & Ax + \frac{1}{2} q(x) & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x_{\mathcal{J}} \text{ integer}, & & \end{array}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear term in the objective.
- $q^o :\in \mathbb{R}^n \to \mathbb{R}$ is the quadratic term in the objective where

$$q^o(x) = x^T Q^o x$$

  and it is assumed that

$$Q^o = (Q^o)^T. \tag{D.1}$$

- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \to \mathbb{R}$ is a vector of quadratic functions. Hence,

$$q_i(x) = x^T Q^i x$$

  where it is assumed that

$$Q^i = (Q^i)^T. \tag{D.2}$$

- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer constrained variables.

D.2.1 The 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.

D.2.1.1 The objective

The first section beginning with one of the keywords

```
max
maximum
maximize
min
minimum
minimize
```

defines the objective sense and the objective function, i.e.

$$c^T x + \frac{1}{2} x^T Q^o x.$$

The objective may be given a name by writing

```
myname:
```

before the expressions. If no name is given, then the objective is named `obj`.

The objective function contains linear and quadratic terms. The linear terms are written as in the example

```
4 x1 + x2 - 0.1 x3
```

and so forth. The quadratic terms are written in square brackets (`[ ]`) and are either squared or multiplied as in the examples

```
x1 ^ 2
```

and

```
x1 * x2
```

There may be zero or more pairs of brackets containing quadratic expressions.

An example of an objective section is:

```
minimize
  myobj: 4 x1 + x2 - 0.1 x3 + [ x1 ^ 2 + 2.1 x1 * x2 ]/2
```

Please note that the quadratic expressions are multiplied with $\frac{1}{2}$, so that the above expression means

$$\text{minimize } 4x_1 + x_2 - 0.1 \cdot x_3 + \frac{1}{2}(x_1^2 + 2.1 \cdot x_1 \cdot x_2)$$

If the same variable occurs more than once in the linear part, the coefficients are added, so that `4 x1 + 2 x1` is equivalent to `6 x1`. In the quadratic expressions `x1 * x2` is equivalent to `x2 * x1` and as in the linear part , if the same variables multiplied or squared occur several times their coefficients are added.

D.2.1.2 The constraints

The second section beginning with one of the keywords

```
subj to
subject to
s.t.
st
```

defines the linear constraint matrix (A) and the quadratic matrices (Q^i).

A constraint contains a name (optional), expressions adhering to the same rules as in the objective and a bound:

```
subject to
  con1: x1 + x2 + [ x3 ^ 2 ]/2 <= 5.1
```

The bound type (here `<=`) may be any of `<`, `<=`, `=`, `>`, `>=` (`<` and `<=` mean the same), and the bound may be any number.

In the standard LP format it is not possible to define more than one bound, but MOSEK supports defining ranged constraints by using double-colon (‘‘::’’) instead of a single-colon (“:”) after the constraint name, i.e.

$$-5 \leq x_1 + x_2 \leq 5 \tag{D.3}$$

may be written as

```
con:: -5 < x_1 + x_2 < 5
```

By default MOSEK writes ranged constraints this way.

If the files must adhere to the LP standard, ranged constraints must either be split into upper bounded and lower bounded constraints or be written as en equality with a slack variable. For example the expression (D.3) may be written as

$$x_1 + x_2 - sl_1 = 0, \ -5 \leq sl_1 \leq 5.$$

D.2.1.3 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
```

D.2.1.4 Variable types

The final two sections are optional and must begin with one of the keywords

```
bin
binaries
binary
```

and

```
gen
general
```

Under `general` all integer variables are listed, and under `binary` all binary (integer variables with bounds 0 and 1) are listed:

```
general
  x1 x2
binary
  x3 x4
```

Again, all variables listed in the binary or general sections must occur in either the objective or a constraint.

D.2.1.5 Terminating section

Finally, an LP formatted file must be terminated with the keyword

```
end
```

D.2.1.6 An example

A simple example of an LP file with two variables, four constraints and one integer variable is:

```
minimize
  -10 x1 -9 x2
subject to
  0.7 x1 +       x2 <= 630
  0.5 x1 + 0.833 x2 <= 600
      x1 + 0.667 x2 <= 708
  0.1 x1 + 0.025 x2 <= 135
bounds
  10 <= x1
  x1 <= +inf
  20 <= x2 <= 500
general
  x1
end
```

D.2.2 LP format peculiarities

D.2.2.1 Comments

Anything on a line after a "\" is ignored and is treated as a comment.

D.2.2.2 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.

It is strongly recommended not to use double quotes (") in names.

D.2.2.3 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.

D.2.2.4 MOSEK specific extensions to the LP format

Some optimization software packages employ a more strict definition of the LP format that the one used by MOSEK. The limitations imposed by the strict LP format are the following:

- Quadratic terms in the constraints are not allowed.
- Names can be only 16 characters long.
- Lines must not exceed 255 characters in length.

If an LP formatted file created by MOSEK should satisfies the strict definition, then the parameter

```
MSK_IPAR_WRITE_LP_STRICT_FORMAT
```

should be set; note, however, that some problems cannot be written correctly as a strict LP formatted file. For instance, all names are truncated to 16 characters and hence they may loose their uniqueness and change the problem.

To get around some of the inconveniences converting from other problem formats, MOSEK allows lines to contain 1024 characters and names may have any length (shorter than the 1024 characters).

Internally in MOSEK names may contain any (printable) character, many of which cannot be used in LP names. Setting the parameters

```
MSK_IPAR_READ_LP_QUOTED_NAMES
```

and

```
MSK_IPAR_WRITE_LP_QUOTED_NAMES
```

allows MOSEK to use quoted names. The first parameter tells MOSEK to remove quotes from quoted names e.g, `"x1"`, when reading LP formatted files. The second parameter tells MOSEK to put quotes around any semi-illegal name (names beginning with a number or a period) and fully illegal name (containing illegal characters). As double quote is a legal character in the LP format, quoting semi-illegal names makes them legal in the pure LP format as long as they are still shorter than 16 characters. Fully illegal names are still illegal in a pure LP file.

D.2.3 The strict LP format

The LP format is not a formal standard and different vendors have slightly different interpretations of the LP format. To make MOSEK's definition of the LP format more compatible whith the definitions of other vendors use the paramter setting

```
MSK_IPAR_WRITE_LP_STRICT_FORMAT MSK_ON
```

This setting may lead to truncation of some names and hence to an invalid LP file. The simple solution to this problem is to use the paramter setting

```
MSK_IPAR_WRITE_GENERIC_NAMES MSK_ON
```

which will cause all names to be renamed systematically in the output file.

D.2.4 Formatting of an LP file

A few parameters control the visual formatting of LP files written by MOSEK in order to make it easier to read the files. These parameters are

```
MSK_IPAR_WRITE_LP_LINE_WIDTH
MSK_IPAR_WRITE_LP_TERMS_PER_LINE
```

The first parameter sets the maximum number of characters on a single line. The default value is 80 corresponding roughly to the width of a standard text document.

The second parameter sets the maximum number of terms per line; a term means a sign, a coefficient, and a name (for example "`+ 42 elephants`"). The default value is 0, meaning that there is no maximum.

D.2.4.1 Speeding up file reading

If the input file should be read as fast as possible using the least amount of memory, then it is important to tell MOSEK how many non-zeros, variables and constraints the problem contains. These values can be set using the parameters

```
MSK_IPAR_READ_CON
```

```
MSK_IPAR_READ_VAR
MSK_IPAR_READ_ANZ
MSK_IPAR_READ_QNZ
```

D.2.4.2 Unnamed constraints

Reading and writing an LP file with MOSEK may change it superficially. If an LP file contains unnamed constraints or objective these are given their generic names when the file is read (however unnamed constraints in MOSEK are written without names).

Appendix E

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.

E.1 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 to; 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.

E.2 The file format

The format uses tags to structure data. A simple example with the basic sections may look like this:

```
[comment]
  This is a comment. You may write almost anything here...
[/comment]

# This is a single-line comment.
```

```
[objective min 'myobj']
  x + 3 y + x^2 + 3 y^2 + z + 1
[/objective]

[constraints]
  [con 'con01'] 4 <= x + y  [/con]
[/constraints]

[bounds]
  [b] -10 <= x,y <= 10  [/b]

  [cone quad] x,y,z [/cone]
[/bounds]
```

A scope is opened by a tag of the form `[tag]` and closed by a tag of the form `[/tag]`. An opening tag may accept a list of unnamed and named arguments, for examples

```
[tag value] tag with one unnamed argument [/tag]
[tag arg=value] tag with one named argument in quotes [/tag]
```

Unnamed arguments are identified by their order, while named arguments may appear in any order, but never before an unnamed argument. The `value` can be a quoted, single-quoted or double-quoted text string, i.e.

```
[tag 'value']     single-quoted value [/tag]
[tag arg='value'] single-quoted value [/tag]
[tag "value"]     double-quoted value [/tag]
[tag arg="value"] double-quoted value [/tag]
```

E.2.1 Sections

The recognized tags are

- `[comment]` A comment section. This can contain *almost* any text: Between single quotes (') or double quotes (") any text may appear. Outside quotes the markup characters (`[` and `]`) must be prefixed by backslashes. Both single and double quotes may appear alone or inside a pair of quotes if it is prefixed by a backslash.

- `[objective]` The objective function: This accepts one or two parameters, where the first one (in the above example '`min`') is either `min` or `max` (regardless of case) and defines the objective sense, and the second one (above '`myobj`'), if present, is the objective name. The section may contain linear and quadratic expressions.

  If several objectives are specified, all but the last are ignored.

- `[constraints]` This does not directly contain any data, but may contain the subsection '`con`' defining a linear constraint.

  `[con]` defines a single constraint; if an argument is present (`[con NAME]`) this is used as the name of the constraint, otherwise it is given a null-name. The section contains a constraint definition written as linear and quadratic expressions with a lower bound, an upper bound, with both or with an equality. Examples:

```
[constraints]
  [con 'con1'] 0 <= x + y        [/con]
  [con 'con2'] 0 >= x + y        [/con]
  [con 'con3'] 0 <= x + y <= 10  [/con]
  [con 'con4']      x + y  = 10  [/con]
[/constraints]
```

  Constraint names are unique. If a constraint is apecified which has the same name as a previously defined constraint, the new constraint replaces the existing one.

- `[bounds]` This does not directly contain any data, but may contain the subsections '`b`' (linear bounds on variables) and '`cone`' (quadratic cone).

  - `[b]`. Bound definition on one or several variables separated by comma ('`,`'). An upper or lower bound on a variable replaces any earlier defined bound on that variable. If only one bound (upper or lower) is given only this bound is replaced. This means that upper and lower bounds can be specified separately. So the OPF bound definition:

```
[b]  x,y >= -10  [/b]
[b]  x,y <= 10   [/b]
```

    results in the bound

$$-10 \leq x, y \leq 10. \tag{E.1}$$

  - `[cone]`. Currently, the supported cones are the *quadratic cone* and the *rotated quadratic cone* (see section 5.4). A conic constraint is defined as a set of variables which belongs to a single unique cone.

    A quadratic cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1^2 > \sum_{i=2}^{n} x_i^2.$$

    A rotated quadratic cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1 x_2 > \sum_{i=3}^{n} x_i^2.$$

  A `[bounds]`-section example:

```
[bounds]
  [b]  0 <= x,y <= 10  [/b] # ranged bound
  [b] 10 >= x,y >=  0  [/b] # ranged bound
  [b]  0 <= x,y <= inf [/b] # using inf
  [b]       x,y free   [/b] # free variables
  # Let (x,y,z,w) belong to the cone K
  [cone quad]  x,y,z,w  [/cone] # quadratic cone
  [cone rquad] x,y,z,w  [/cone] # rotated quadratic cone
[/bounds]
```

By default all variables are free.

- `[variables]` This defines an ordering of variables as they should appear in the problem. This is simply a space-separated list of variable names.
- `[integer]` This contains a space-separated list of variables and defines the constraint that the listed variables must be integer values.
- `[hints]` This may contain only non-essential data; for example estimates of the number of variables, constraints and non-zeros. Placed before all other sections containing data this may reduce the time spent reading the file.

  In the `hints` section, any subsection which is not recognized by MOSEK is simply ignored. In this section a hint in a subsection is defined as follows:

  ```
  [hint ITEM] value [/hint]
  ```

  where `ITEM` may be replaced by `numvar` (number of variables), `numcon` (number of linear/quadratic constraints), `numanz` (number if linear non-zeros in constraints) and `numqnz` (number of quadratic non-zeros in constraints).
- `[solutions]` This section can contain a number of full or partial solutions to a problem, each inside a `[solution]`-section. The syntax is

  ```
  [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 values for a single variable or constraint, each value written as

```
KEYWORD=value
```

where KEYWORD defines a solution item and value defines its 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 variable associated with its lower bound.
- su Defines the level of the 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 and lvl, and optionally sl, su and sn.

A [con] section should always contain sk and lvl, and optionally 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.

E.2.2 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]+)?
```

E.2.3 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_unqouted_name
another_name{123}
'single qouted name'
"double qouted name"
"name with \"qoute\" in it"
"name with []s in it"
```

E.3 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]
```

E.4 Writing OPF files from MOSEK

The function `MSK_writedata` can be used to produce an OPF file from a task.

To write an OPF file set the parameter `MSK_IPAR_WRITE_DATA_FORMAT` to `MSK_DATA_FORMAT_OP` as this ensures that `OPF` format is used. Then modify the following parameters to define what the file should contain:

- `MSK_IPAR_OPF_WRITE_HEADER`, include a small header with comments.
- `MSK_IPAR_OPF_WRITE_HINTS`, include hints about the size of the problem.
- `MSK_IPAR_OPF_WRITE_PROBLEM`, include the problem itself — objective, constraints and bounds.

- MSK_IPAR_OPF_WRITE_SOLUTIONS, include solutions if they are defined. If this is off, no solutions are included.
- MSK_IPAR_OPF_WRITE_SOL_BAS, include basic solution, if defined.
- MSK_IPAR_OPF_WRITE_SOL_ITG, include integer solution, if defined.
- MSK_IPAR_OPF_WRITE_SOL_ITR, include interior solution, if defined.
- MSK_IPAR_OPF_WRITE_PARAMETERS, include all parameter settings.

E.5 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

E.5.1 Linear example `lo1.opf`

Consider the example:

$$\begin{array}{lrcrcl} \text{minimize} & -10x_1 & & -9x_2, & & \\ \text{subject to} & 7/10x_1 & + & 1x_2 & \leq & 630, \\ & 1/2x_1 & + & 5/6x_2 & \leq & 600, \\ & 1x_1 & + & 2/3x_2 & \leq & 708, \\ & 1/10x_1 & + & 1/4x_2 & \leq & 135, \\ & x_1, & & x_2 & \geq & 0. \end{array} \tag{E.2}$$

In the OPF format the example is displayed as shown below:

```
[comment]
  Example lo1.mps converted to OPF.
[/comment]

[hints]
  # Give a hint about the size of the different elements in the problem.
  # These need only be estimates, but in this case they are exact.
  [hint NUMVAR] 2 [/hint]
  [hint NUMCON] 4 [/hint]
  [hint NUMANZ] 8 [/hint]
[/hints]

[variables]
  # All variables that will appear in the problem
  x1 x2
[/variables]

[objective minimize 'obj']
   - 10 x1 - 9 x2
[/objective]

[constraints]
  [con 'c1']  0.7 x1 +              x2 <= 630 [/con]
```

```
  [con 'c2']  0.5 x1 + 0.8333333333 x2 <= 600 [/con]
  [con 'c3']       x1 + 0.66666667   x2 <= 708 [/con]
  [con 'c4']  0.1 x1 + 0.25          x2 <= 135 [/con]
[/constraints]

[bounds]
  # By default all variables are free. The following line will
  # change this to all variables being nonnegative.
  [b] 0 <= * [/b]
[/bounds]
```

E.5.2 Quadratic example `qo1.opf`

An example of a quadratic optimization problem is

$$\begin{array}{lcc} \text{minimize} & & x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ \text{subject to} \quad 1 & \leq & x_1 + x_2 + x_3, \\ & & x \geq 0. \end{array} \tag{E.3}$$

This can be formulated in `opf` as shown below.

```
[comment]
  Example qo1.mps converted to OPF.
[/comment]

[hints]
  [hint NUMVAR] 3 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
[/hints]

[variables]
  x1 x2 x3
[/variables]

[objective minimize 'obj']
   # The quadratic terms are often multiplied by 1/2,
   # but this is not required.

   - x2 + 0.5 ( 2 x1 ^ 2 - 2 x3 * x1 + 0.2 x2 ^ 2 + 2 x3 ^ 2 )
[/objective]

[constraints]
  [con 'c1'] 1 <= x1 + x2 + x3 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]
```

E.5.3 Conic quadratic example `cqo1.opf`

Consider the example:

$$\begin{array}{lrcl} \text{minimize} & 1x_1 + 2x_2 & & \\ \text{subject to} & 2x_3 + 4x_4 & = & 5, \\ & x_5^2 & \leq & 2x_1x_3, \\ & x_6^2 & \leq & 2x_2x_4, \\ & x_5 & = & 1, \\ & x_6 & = & 1, \\ & x \geq 0. & & \end{array} \tag{E.4}$$

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.

```
[comment]
  Example cqo1.mps converted to OPF.
[/comment]

[hints]
  [hint NUMVAR] 6 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 2 [/hint]
[/hints]

[variables]
  x1 x2 x3 x4 x5 x6
[/variables]

[objective minimize 'obj']
   x1 + 2 x2
[/objective]

[constraints]
  [con 'c1']  2 x3 + 4 x4 = 5 [/con]
[/constraints]

[bounds]
  # We let all variables default to the positive orthant
  [b] 0 <= * [/b]
  # ... and change those that differ from the default.
  [b] x5,x6 = 1 [/b]

  # We define two rotated quadratic cones

  # k1: 2 x1 * x3 >= x5^2
  [cone rquad 'k1'] x1, x3, x5 [/cone]

  # k2: 2 x2 * x4 >= x6^2
  [cone rquad 'k2'] x2, x4, x6 [/cone]
[/bounds]
```

E.5.4 Mixed integer example `milo1.opf`

Consider the mixed integer problem:

$$\begin{array}{lrcl} \text{maximize} & x_0 + 0.64x_1 & & \\ \text{subject to} & 50x_0 + 31x_1 & \leq & 250, \\ & 3x_0 - 2x_1 & \geq & -4, \\ & x_0, x_1 \geq 0 & & \text{and integer} \end{array} \tag{E.5}$$

This can be implemented in `OPF` with:

```
[comment]
   Written by MOSEK version 5.0.0.7
   Date 20-11-06
   Time 14:42:24
[/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]
```

Appendix F

The XML (OSiL) format

MOSEK can write data in the standard `OSiL` xml format. For a definition of the `OSiL` format please see `http://www.optimizationservices.org/`. Only linear constraints (possibly with integer variables) are supported. By default output files with the extension `.xml` are written in the `OSiL` format.

The parameter `MSK_IPAR_WRITE_XML_MODE` controls if the linear coefficients in the A matrix are written in row or column order.

Appendix G

The ORD file format

An ORD formatted file specifies in which order the mixed integer optimizer branches on variables. The format of an ORD file is shown in Figure G.1. In the figure names in capitals are keywords of the ORD format, whereas names in brackets are custom names or values. The `??` is an optional key specifying the preferred branching direction. The possible keys are `DN` and `UP` which indicate that down or up is the preferred branching direction respectively. The branching direction key is optional and is left blank the mixed integer optimizer will decide whether to branch up or down.

```
*        1         2         3         4         5         6
*2345678901234567890123456789012345678901234567890123456789012345678890
NAME          [name]
 ?? [vname1]              [value1]
ENDATA
```

Figure G.1: The standard ORD format.

G.1 An example

A concrete example of a ORD file is presented below:

```
NAME          EXAMPLE
 DN x1                    2
 UP x2                    1
    x3                    10
ENDATA
```

This implies that the priorities 2, 1, and 10 are assigned to variable `x1`, `x2`, and `x3` respectively. The higher the priority value assigned to a variable the earlier the mixed integer optimizer will branch on that variable. The key `DN` implies that the mixed integer optimizer first will branch down on variable whereas the key `UP` implies that the mixed integer optimizer will first branch up on a variable.

If no branch direction is specified for a variable then the mixed integer optimizer will automatically choose the branching direction for that variable. Similarly, if no priority is assigned to a variable then it is automatically assigned the priority of 0.

Appendix H

The solution file format

MOSEK provides one or two solution files depending on the problem type and the optimizer used. If a problem is optimized using the interior-point optimizer and no basis identification is required, then a file named `probname.sol` is provided. `probname` is the name of the problem and `.sol` is the file extension. If the problem is optimized using the simplex optimizer or basis identification is performed, then a file named `probname.bas` is created presenting the optimal basis solution. Finally, if the problem contains integer constrained variables then a file named `probname.int` is created. It contains the integer solution.

H.1 The basic and interior solution files

In general both the interior-point and the basis solution files have the format:

```
NAME               : <problem name>
PROBLEM STATUS     : <status of the problem>
SOLUTION STATUS    : <status of the solution>
OBJECTIVE NAME     : <name of the objective function>
PRIMAL OBJECTIVE   : <primal objective value corresponding to the solution>
DUAL OBJECTIVE     : <dual objective value corresponding to the solution>
CONSTRAINTS
INDEX  NAME            AT ACTIVITY          LOWER LIMIT       UPPER LIMIT       DUAL LOWER        DUAL UPPER
?      <name>          ?? <a value>         <a value>         <a value>         <a value>         <a value>
VARIABLES
INDEX  NAME            AT ACTIVITY          LOWER LIMIT       UPPER LIMIT       DUAL LOWER        DUAL UPPER         CONIC DUAL
?      <name>          ?? <a value>         <a value>         <a value>         <a value>         <a value>          <a value>
```

In the example the fields `?` and `<>` will be filled with problem and solution specific information. As can be observed a solution report consists of three sections, i.e.

`HEADER` In this section, first the name of the problem is listed and afterwards the problem and solution statuses are shown. In this case the information shows that the problem is primal and dual feasible and the solution is optimal. Next the primal and dual objective values are displayed.

`CONSTRAINTS` Subsequently in the constraint section the following information is listed for each constraint:

`INDEX` A sequential index assigned to the constraint by MOSEK.

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.

Table H.1: Status keys.

NAME The name of the constraint assigned by the user.

AT The status of the constraint. In Table H.1 the possible values of the status keys and their interpretation are shown.

ACTIVITY Given the ith constraint on the form

$$l_i^c \leq \sum_{j=1}^{n} a_{ij} x_j \leq u_i^c, \tag{H.1}$$

then activity denote the quantity $\sum_{j=1}^{n} a_{ij} x_j^*$, where x^* is the value for the x solution.

LOWER LIMIT Is the quantity l_i^c (see (H.1)).

UPPER LIMIT Is the quantity u_i^c (see (H.1)).

DUAL LOWER Is the dual multiplier corresponding to the lower limit on the constraint.

DUAL UPPER Is the dual multiplier corresponding to the upper limit on the constraint.

VARIABLES The last section of the solution report lists information for the variables. This information has a similar interpretation as for the constraints. However, the column with the header [CONIC DUAL] is only included for problems having one or more conic constraints. This column shows the dual variables corresponding to the conic constraints.

H.2 The integer solution file

The integer solution is equivalent to the basic and interior solution files except that no dual information is included.

Bibliography

[1] Richard C. Grinold abd Ronald N. Kahn. *Active portfolio management*. McGraw-Hill, New York, 2 edition, 2000.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Optimization*, volume 1, pages 211–369. North Holland, Amsterdam, 1989.

[3] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Math. Programming*, 95(1):3–51, 2003.

[4] E. D. Andersen and K. D. Andersen. Presolving in linear programming. *Math. Programming*, 71(2):221–245, 1995.

[5] 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.

[6] E. D. Andersen, C. Roos, and T. Terlaky. On implementing a primal-dual interior-point method for conic quadratic optimization. *Math. Programming*, 95(2), February 2003.

[7] E. D. Andersen and Y. Ye. Combining interior-point and pivoting algorithms. *Management Sci.*, 42(12):1719–1731, December 1996.

[8] E. D. Andersen and Y. Ye. A computational study of the homogeneous algorithm for large-scale convex optimization. *Computational Optimization and Applications*, 10:243–269, 1998.

[9] E. D. Andersen and Y. Ye. On a homogeneous algorithm for the monotone complementarity problem. *Math. Programming*, 84(2):375–399, February 1999.

[10] Erling D. Andersen. The homogeneous and self-dual model and algorithm for linear optimization. Technical Report TR-1-2009, MOSEK ApS, 2009. http://www.mosek.com/fileadmin/reports/tech/homolo.pdf.

[11] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear programming: Theory and algorithms*. John Wiley and Sons, New York, 2 edition, 1993.

[12] C. Beightler and D. T. Phillips. *Applied geometric programming*. John Wiley and Sons, New York, 1976.

[13] A. Ben-Tal and A Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. MPS/SIAM Series on Optimization. SIAM, 2001.

[14] S.P. Boyd, S.J. Kim, L. Vandenberghe, and A. Hassibi. A Tutorial on Geometric Programming. Technical report, ISL, Electrical Engineering Department, Stanford University, Stanford, CA, 2004. Available at http://www.stanford.edu/~boyd/gp_tutorial.html.

[15] V. Chvátal. *Linear programming*. W.H. Freeman and Company, 1983.

[16] N. Gould and P. L. Toint. Preprocessing for quadratic programming. *Math. Programming*, 100(1):95–132, 2004.

[17] J. L. Kenningon and K. R. Lewis. Generalized networks: The theory of preprocessing and an emperical analysis. *INFORMS Journal on Computing*, 16(2):162–173, 2004.

[18] M. S. Lobo, L. Vanderberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra Appl.*, 284:193–228, November 1998.

[19] M. S. Lobo and M. Fazel, and S. Boyd. Portfolio optimization with linear and fixed transaction costs. Technical report, CDS, California Institute of Technology, 2005. To appear in Annals of Operations Research. http://www.cds.caltech.edu/~maryam/portfolio.html.

[20] J. L. Nazareth. *Computer Solution of Linear Programs*. Oxford University Press, New York, 1987.

[21] G. Pataki. Teaching integer programming formulations using the traveling salesman problem. *SIAM Rev.*, 45(1):116–123, 2003.

[22] C. Roos, T. Terlaky, and J. -Ph. Vial. *Theory and algorithms for linear optimization: an interior point approach*. John Wiley and Sons, New York, 1997.

[23] Bernd Scherer. *Portfolio construction and risk budgeting*. Risk Books, 2 edition, 2004.

[24] G. W. Stewart. *Matrix Algorithms. Volume 1: Basic decompositions*. SIAM, 1998.

[25] S. W. Wallace. Decision making under uncertainty: Is sensitivity of any use. *Oper. Res.*, 48(1):20–25, January 2000.

[26] H. P. Williams. *Model building in mathematical programming*. John Wiley and Sons, 3 edition, 1993.

[27] L. A. Wolsey. *Integer programming*. John Wiley and Sons, 1998.

Index