The MOSEK C API manual.
Version 5.0 (Revision 138).

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 incd

`mosek\5\license\index.html`

Chapter 1

Changes and new features in MOSEK

The section presents improvements and new features added to MOSEK in version 5.0.

1.1 File formats

- The `OSiL` XML format for linear problems is now supported as output-only format.
- The new Optimization Problem file Format (OPF) is now available. It incorporates linear, quadratic, and conic problems in a single format, as well as parameter settings and solutions.
- The `OBJNAME` section is now supported in the MPS format.

1.2 Optimizers

- The interior-point solver is about 20% faster on average for large linear problems, compared to MOSEK 4.0.
- The dual simplex solver is about 40% faster on average compared to MOSEK 4.0.
- For the primal simplex solver, handling of problems with long slim structure has been improved.
- For both simplex optimizers numerical stability, hot-start efficiency and degeneracy handling has been improved substantially.
- A simplex network flow optimizer is now available. In many cases the specialized simplex optimizer can solve a pure network flow optimization problem up to 10 times faster than the standard simplex optimizer.
- Presolve is now by default turned on for hot-start with the simplex optimizers.

- The mixed integer optimizer now includes the feasibility pump heuristic to find a good initial feasible solution.
- Full support for setting branching priorities on integer constrained variables.

1.3 API changes

- The function MSK_putobjsense has been introduced. This should be used to define objective sense instead of the parameter MSK_IPAR_OBJECTIVE_SENSE.

1.4 License system

- The Flexlm license software has been upgraded to version 11.4.
- Dongles are supported in 64 bit Windows.

1.5 Other changes

- The documentation has been improved. Each interface now have a complete dedicated manual, and many code examples have been added. The HTML version has been subject to heavy cosmetical changes.

1.6 Interfaces

- A complete Python interface is now available.
- The MATLAB interface supports the MATLAB versions R2006a, R2006b, and R2007a.
- The general convex interface has been disabled in the Java and .NET interfaces.
- The Java API provides an interface to the native `scopt` functionality.

1.7 Supported platforms

- Mac OSX 32 bit for x86 version has been added.
- Solaris 32 bit for x86 version has been added
- Solaris 64 bit for x86 version has been added.

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 languages such as C, C++, and Fortran using the MOSEK callable library.

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\5\help\index.html`

in the distribution.

3.2 Additional reading

In this manual it is assumed the reader is familiar with mathematics and in particular mathematical optimization. Some introduction to linear programming can be found in books such as “Linear programming” by Chvátal [15] or “Computer Solution of Linear Programs” by Nazareth [20]. For more theoretical aspects see for example “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 modelling 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\5\
```

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 5.0.0.2(alpha) (Build date: Nov 16 2006 10:24:36)
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 13:28:28.
Using FLEXlm version: 11.3.
Hostname: 'skalbjerg' Hostid: '"000c6e5cab33 005056c00001 005056c00008"'

Operating system variables
MOSEKLM_LICENSE_FILE     : C:\Program Files\mosek\5\licenses
PATH                     : c:\local\python24;c:\local\bin;C:\Program Files\mosek\5\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\5
```

and that the platform-specific files are located in

```
c:\Program Files\mosek\5\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\5\tools\examples\c
```

The directory contains several makefiles. You should use either `Makefile.win32x86` or `Makefile.win64x86`, depending on your () installation. For 32-bit Windows type

```
nmake /f Makefile.win32x86 all
```

and similarly for 64-bit Windows, type

```
nmake /f Makefile.win64x86 all
```

To only build a single example instead of all examples, replace "`all`" by the corresponding executable name. For example, to build `lo1.exe` on 32-bit Windows, type

```
nmake /f Makefile.win32x86 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\5\tools\platform\<platform>\h\mosek.h
  ```

- The MOSEK `lib` file located in

  ```
  c:\Program Files\mosek\5\platform\<platform>\dll
  ```

  The relevant `lib` file is

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

    `mosek64_5_0.lib`

  – on 32-bit Microsoft Windows

    `mosek5_0.lib`

- The MOSEK solver `dll` located in

  ```
  c:\Program Files\mosek\5\platform\<platform>\bin
  ```

  The relevant `dll` file is

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

    `mosek64_5_0.dll`

  – on 32-bit Microsoft Windows

    `mosek5_0.dll`

Finally, the distributed C examples are located in the directory

```
c:\Program Files\mosek\5\tools\examples\c
```

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

1. Change directory:

   ```
   c:
   cd "\Program Files\mosek\5\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\<platform>\h\mosek.h platform\win\dll\mosek5_0.lib
   ```

For Windows 64:

```
cl examples\c\lo1.c /out:lo1.exe /I platform\<platform>\h\mosek.h platform\win64x86\dll\mosek64_
```

For Windows Itanium:

```
cl examples\c\lo1.c /out:lo1.exe /I platform\<platform>\h\mosek.h platform\winia64\dll\mosek64_5
```

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\5\tools\platform\win\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\5\tools\platform\win\dll\mosek5_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/5/tools/h/mosek.h
```

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

```
mosek/5/tools/platform/<platform>/bin/libmosek64.so.5.0
```

for 64-bit architectures, and in

```
mosek/5/tools/platform/<platform>/bin/libmosek.so.5.0
```

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="-lpthread -lc -ldl -lm"
    ;;

  linux64x86)
    MSKCC="gcc -m64"
    MSKLINKFLAGS="-lpthread -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/5/tools/examples/c
```

The directory contains several makefiles. You should use either `Makefile.lnx32x86` or `Makefile.lnx64x86`, depending on your installation. For 32-bit Linux, type

```
gmake -f Makefile.lnx32x86 all
```

and similarly for 64-bit Windows, type

```
gmake -f Makefile.lnx64x86 all
```

To build one example instead of all examples, replace "all" by the corresponding executable name. For example, to build the lo1 executable on 32-bit Linux, type

```
gmake -f Makefile.lnx64x86 lo1
```

4.2.2.2 Example: Linking with GNU C under Linux

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

```
# 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/linux32x86/bin/:$LD_LIBRARY_PATH

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

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

# Run lo1 executable
./lo1
```

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.

```
/*
  Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.
```

```
  File:    simple.c

  Purpose: To demonstrate a very simple example using MOSEK by
           reading a problem file, solving the problem and
           writing the solution to a file.
*/

#include "mosek.h"

int main (int argc, char * argv[])
{
  MSKtask_t   task = NULL;
  MSKenv_t    env  = NULL;
  MSKrescodee res  = MSK_RES_OK;

  if (argc <= 1)
  {
    printf ("Missing argument. The syntax is:\n");
    printf (" simple inputfile [ solutionfile ]\n");
  }
  else
  {
    /* Create the mosek environment.
       The 'NULL' arguments here, are used to specify customized
       memory allocators and a memory debug file. These can
       safely be ignored for now. */

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

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

    /* Create a task object linked to the environment env.
       Initially we create it with 0 variables and 0 columns,
       since we do not know the size of the problem. */
    if ( res==MSK_RES_OK )
      res = MSK_maketask (env, 0,0, &task);

    /* We assume that a problem file was given as the first command
       line argument (received in 'argv'). */
    if ( res==MSK_RES_OK )
      res = MSK_readdata (task, argv[1]);

    /* Solve the problem */
    if ( res==MSK_RES_OK )
      MSK_optimize(task);

    /* Print a summary of the solution. */
    MSK_solutionsummary(task, MSK_STREAM_MSG);

    /* If an output file was specified, write a solution */
    if ( res==MSK_RES_OK && argc>2 )
    {
      /* We define the output format to be OPF, and tell MOSEK to
         leave out parameters and problem data from the output file. */
      MSK_putintparam (task,MSK_IPAR_WRITE_DATA_FORMAT,    MSK_DATA_FORMAT_OP);
```

```
      MSK_putintparam (task,MSK_IPAR_OPF_WRITE_SOLUTIONS,  MSK_ON);
      MSK_putintparam (task,MSK_IPAR_OPF_WRITE_HINTS,      MSK_OFF);
      MSK_putintparam (task,MSK_IPAR_OPF_WRITE_PARAMETERS, MSK_OFF);
      MSK_putintparam (task,MSK_IPAR_OPF_WRITE_PROBLEM,    MSK_OFF);
      MSK_writedata(task,argv[2]);
    }

    MSK_deletetask(&task);
    MSK_deleteenv(&env);
  }
  return res;
}
```

5.1.2.1 Writing a problem to a file

Use the MSK_writedata function to write a problem to a file. By default MOSEK will determine the output file format by the extension of the filename, for example to write an OPF file:

```
MSK_writedata(task,"problem.opf");
```

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/5/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 scalar constant,

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

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

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 Example: lo1

The following is an example of a linear optimization problem:

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

having the bounds

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

5.2.1.1 Source code

The data structures used in the following example will be explained in detail in 5.8.

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

```
mosek\5\tools\examp\
```

```
/*
   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.

   File:      lo1.c

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

 */

#include <stdio.h>

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

#define NUMCON 3   /* Number of constraints.             */
#define NUMVAR 4   /* Number of variables.               */
#define NUMANZ 9   /* 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;
  double       c[]    = {3.0, 1.0, 5.0, 1.0};
```

```
  MSKlidxt      ptrb[] = {0, 2, 5, 7};
  MSKlidxt      ptre[] = {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};

  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         };

  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;
  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 )
  {
    /* Direct the environment log stream to
       the 'printstr' function. */
    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");

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

    if ( r==MSK_RES_OK )
    {
      /* Direct the log task stream to
         the 'printstr' function. */
      MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);

      r = MSK_inputdata(task,
                        NUMCON,NUMVAR,
                        NUMCON,NUMVAR,
```

```
                         c,
                         0.0,
                         ptrb,
                         ptre,
                         asub,
                         aval,
                         bkc,
                         blc,
                         buc,
                         bkx,
                         blx,
                         bux);

      if ( r==MSK_RES_OK )
      {
        MSK_putobjsense(task,MSK_OBJECTIVE_SENSE_MAXIMIZE);

        MSK_echotask(task,
                     MSK_STREAM_MSG,
                     "Start optimizing\n");

        r = MSK_optimize(task);

        if ( r==MSK_RES_OK )
        {
          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);
          printf("Primal solution\n");
          for(j=0; j<NUMVAR; ++j)
            printf("x[%d]: %e\n",j,xx[j]);
        }
      }
    }

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

  printf("Return code: %d (0 means no error occured.)\n",r);

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

5.2.1.2 Example code comments

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

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

/* Check if return code is ok. */
```

```
if ( r==MSK_RES_OK )
{
  /* Direct the environment log stream to
     the 'printstr' function. */
  MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
}

/* Initialize the environment. */
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.

MOSEK optimization task: Next, an empty task object is created:

```
r = MSK_maketask(env,NUMCON,NUMVAR,&task);
```

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.

Inputting the problem data: When the task has been created, data can be loaded into it. This happens here:

```

r = MSK_inputdata(task,
                  NUMCON,NUMVAR,
                  NUMCON,NUMVAR,
                  c,
                  0.0,
                  ptrb,
                  ptre,
                  asub,
                  aval,
                  bkc,
                  blc,
                  buc,
                  bkx,
                  blx,
                  bux);
```

There are several different ways to set up an optimization problem; in this case we loaded the whole problem using a single function, `MSK_inputdata`.

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).

The `c` argument is a full vector defining the objective function.

The precise relation between the arguments and the mathematical expressions in (5.1)...(5.3) is as follows.

- The linear terms in the constraints:

$$a_{\mathtt{sub[t]},\mathtt{j}} = \mathtt{val}[\mathtt{t}], \quad \begin{array}{l} t = \mathtt{ptrb}[\mathtt{j}], \ldots, \mathtt{ptre}[\mathtt{j}] - 1, \\ j = 0, \ldots, \mathtt{numvar} - 1. \end{array} \tag{5.6}$$

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 5.1: Interpretation of the bound keys.

For an illustrated example of the meaning of `ptrb` and `ptre` see Section 5.8.3.2.

- The linear terms in the objective:

$$c_j = \texttt{c}[\texttt{j}], \;\; j = 0, \ldots, \texttt{numvar} - 1 \tag{5.7}$$

- The bounds for the constraints are specified using the `bkc`, `blc`, and `buc` variables. The components of the `bkc` integer array specify the type of the bounds according to Table 5.1. For instance `bkc[2]= MSK_BK_LO` means that $-\infty < l_2^c$ and $u_2^c = \infty$. Finally, the numerical values of the bounds are given by

$$l_k^c = \texttt{blc}[\texttt{k}], \;\; k = 0, \ldots, \texttt{numcon} - 1 \tag{5.8}$$

and

$$u_k^c = \texttt{buc}[\texttt{k}], \;\; k = 0, \ldots, \texttt{numcon} - 1. \tag{5.9}$$

- The bounds on the variables are specified using the `bkx`, `blx`, and `bux` variables. The components in the `bkx` integer array specifies the type of the bounds according to Table 5.1. The numerical values for the lower bounds on the variables are given by

$$l_j^x = \texttt{blx}[\texttt{j}], \;\; j = 0, \ldots, \texttt{numvar} - 1. \tag{5.10}$$

The numerical values for the upper bounds on the variables are given by

$$u_j^x = \texttt{bux}[\texttt{j}], \;\; j = 0, \ldots, \texttt{numvar} - 1. \tag{5.11}$$

Optimization: After set-up the task can be optimized.

```
r = MSK_optimize(task);
```

Outputting the solution: Finally, the primal solution is retrieved and printed.

```
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. In fact MOSEK may compute several solutions depending on the optimizer employed. In this example the *basic solution* is requested, specified by MSK_SOL_BAS. The MSK_SOL_ITEM_XX specifies that we want the variable values of the solution, and the following 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'').

5.2.2 An alternative implementation: lo2

In the previous example the problem data is loaded in one chunk. It is often more convenient to add one constraint or one variable at a time — this is possible using the following approach:

- Before a constraint or a variable can be used it has to be added with MSK_append or a similar function. By default the appended constraints will be empty and the bounds of the appended constraints are infinite. Variables are fixed at zero.
- The objective function is specified using MSK_putcfix and MSK_putcj.
- The lower and upper bounds on the constraints and variables are specified using MSK_putbound.
- The non-zero entries in A are added one column at a time using MSK_putavec.

```
/*
  Copyright: Copyright (c) 1998-2007 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" /* Include the MOSEK definition file. */

#define NUMCON 3   /* Number of constraints.             */
#define NUMVAR 4   /* Number of variables.               */
#define NUMANZ 9   /* 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;
  double       c[]    = {3.0, 1.0, 5.0, 1.0};
  MSKlidxt     ptrb[] = {0, 2, 5, 7};
```

```
MSKlidxt     ptre[] = {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};

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         };

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);

/* Check if return code is ok. */
if ( r==MSK_RES_OK )
{
  /* Directs the env log stream to the 'printstr' function. */
  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");

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

  /* Directs the log task stream to the 'printstr' function. */

  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);

  if (r == MSK_RES_OK)
    r = MSK_putmaxnumanz(task,NUMANZ);

  /* Append the constraints. */
  if (r == MSK_RES_OK)
    r = MSK_append(task,MSK_ACC_CON,NUMCON);

  /* Append the variables. */
  if (r == MSK_RES_OK)
    r = MSK_append(task,MSK_ACC_VAR,NUMVAR);

  /* Inpput C. */
  if (r == MSK_RES_OK)
    r = MSK_putcfix(task, 0.0);

  if (r == MSK_RES_OK)
    for(j=0; j<NUMVAR; ++j)
      r = MSK_putcj(task,j,c[j]);

  /* Put constraint bounds. */
  if (r == MSK_RES_OK)
    for(i=0; i<NUMCON; ++i)
      r = MSK_putbound(task,MSK_ACC_CON,i,bkc[i],blc[i],buc[i]);

  /* Put variable bounds. */
  if (r == MSK_RES_OK)
    for(j=0; j<NUMVAR; ++j)
      r = MSK_putbound(task,MSK_ACC_VAR,j,bkx[j],blx[j],bux[j]);

  /* Put A. */
  if (r == MSK_RES_OK)
    if ( NUMCON>0 )
      for(j=0; j<NUMVAR; ++j)
        r = MSK_putavec(task,
                        MSK_ACC_VAR,
                        j,
                        ptre[j]-ptrb[j],
                        asub+ptrb[j],
                        aval+ptrb[j]);

  if (r == MSK_RES_OK)
    r = MSK_putobjsense(task,
                        MSK_OBJECTIVE_SENSE_MAXIMIZE);

  if (r == MSK_RES_OK)
    r = MSK_optimize(task);

  if (r == MSK_RES_OK)
    MSK_getsolutionslice(task,
                         MSK_SOL_BAS,       /* Basic solution.       */
                         MSK_SOL_ITEM_XX,   /* Which part of solution.  */
                         0,                 /* Index of first variable. */
                         NUMVAR,            /* Index of last variable+1. */
                         xx);
```

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

  return r;
}
```

5.3 Quadratic optimization

It is possible to solve quadratic and quadratically constrained convex problems using MOSEK. This class of problems can be formulated as follows:

$$\begin{array}{lcccll} \text{minimize} & & \frac{1}{2}x^TQ^ox + c^Tx + c^f & & & \\ \text{subject to} \quad l_k^c & \leq & \frac{1}{2}x^TQ^kx + \sum\limits_{j=0}^{n-1} a_{k,j}x_j & \leq & u_k^c, & k = 0,\ldots,m-1, \\ l^x & \leq & x & \leq & u^x, & j = 0,\ldots,n-1. \end{array} \tag{5.12}$$

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

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

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

A very important restriction in MOSEK is that the problem should be convex. This implies that the matrix Q^o should be positive semi-definite and that the kth constraint must be of the form

$$l_k^c \leq \frac{1}{2}x^TQ^kx + \sum_{j=0}^{n-1} a_{k,j}x_j \tag{5.13}$$

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

$$\frac{1}{2}x^TQ^kx + \sum_{j=0}^{n-1} a_{k,j}x_j \leq u_k^c. \tag{5.14}$$

with a positive semi-definite Q^k. This implies that quadratic equalities are specifically *not* allowed.

5.3.1 Example: Quadratic objective

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

$$\begin{array}{lccl} \text{minimize} & & & x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ \text{subject to} & 1 & \leq & x_1 + x_2 + x_3 \\ & & & x \geq 0 \end{array} \tag{5.15}$$

This can be written equivalently as

$$\begin{array}{lccc} \text{minimize} & 1/2x^TQ^ox + c^Tx & & \\ \text{subject to} & Ax & \geq & b \\ & x & \geq & 0, \end{array} \tag{5.16}$$

where

$$Q^o = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 0.2 & 0 \\ -1 & 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \text{ and } b = 1. \tag{5.17}$$

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

5.3.1.1 Source code

```
/*
   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.

   File:      qo1.c

   Purpose: To demonstrate how to solve a 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 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[])
{

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

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

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

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

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

  MSKidxt         qsubi[NUMQNZ];
  MSKidxt         qsubj[NUMQNZ];
  double          qval[NUMQNZ];

  MSKidxt         j;
  double          xx[NUMVAR];

  MSKenv_t        env;
  MSKtask_t       task;
  MSKrescodee     r;

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

  /* Check whether the 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. */
  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 )
    {
      r = MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);

      if ( r==MSK_RES_OK )
      {
        r = MSK_inputdata(task,
                          NUMCON,NUMVAR,
                          NUMCON,NUMVAR,
                          c,0.0,
                          ptrb,
                          ptre,
                          asub,
                          aval,
                          bkc,
                          blc,
                          buc,
                          bkx,
                          blx,
                          bux);
      }

      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 )
        r = MSK_optimize(task);

      if ( r==MSK_RES_OK )
      {
        MSK_getsolutionslice(task,
                             MSK_SOL_ITR,
                             MSK_SOL_ITEM_XX,
                             0,
                             NUMVAR,
                             xx);

        printf("Primal solution\n");
        for(j=0; j<NUMVAR; ++j)
          printf("x[%d]: %e\n",j,xx[j]);
      }
    }

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

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

  return ( r );
} /* 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):

```
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;
```

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:

```
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 (5.13).

Consider the problem:

$$\begin{array}{lccl} \text{minimize} & & & x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ \text{subject to} & 1 & \leq & x_1 + x_2 + x_3 - x_1^2 - x_2^2 - 0.1x_3^2 + 0.2x_1x_3, \\ & & & x \geq 0. \end{array} \tag{5.18}$$

This is equivalent to

$$\begin{array}{lccc} \text{minimize} & 1/2x^TQ^ox + c^Tx & & \\ \text{subject to} & 1/2x^TQ^0x + Ax & \geq & b, \end{array} \tag{5.19}$$

where

$$Q^o = \begin{bmatrix} 2 & 0 & -1 \\ 0 & 0.2 & 0 \\ -1 & 0 & 2 \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \quad b = 1. \tag{5.20}$$

$$Q^0 = \begin{bmatrix} -2 & 0 & 0.2 \\ 0 & -2 & 0 \\ 0.2 & 0 & -0.2 \end{bmatrix}. \tag{5.21}$$

5.3.2.1 Source code

```
/*
   Copyright: Copyright (c) 1998-2007 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};

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

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

  MSKidxt      qsubi[NUMQNZ];
```

```
  MSKidxt       qsubj[NUMQNZ];
  double        qval[NUMQNZ];

  MSKidxt       j;
  double        xx[NUMVAR];
  MSKenv_t      env;
  MSKtask_t     task;

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

  /* Check whether the 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. */
  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 )
    {
      r = MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);

      if ( r==MSK_RES_OK )
      {
        r = MSK_inputdata(task,
                          NUMCON,NUMVAR,
                          NUMCON,NUMVAR,
                          c,0.0,
                          ptrb,
                          ptre,
                          asub,
                          aval,
                          bkc,
                          blc,
                          buc,
                          bkx,
                          blx,
                          bux);
      }

      if ( r==MSK_RES_OK )
      {
        /*
         * The lower triangular part of the Q^o
         * 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^o for the objective. */

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


    if ( r==MSK_RES_OK )
    {
       /*
       * The lower triangular part of the Q^0
       * matrix in the first constraint is specified.
       This corresponds to adding the term
       - x_1^2 - x_2^2 - 0.1 x_3^2 + 0.2 x_1 x_3
       */

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

      /* Put Q^0 in constraint with index 0. */

       r = MSK_putqconk(task,
                        0,
                        4,
                        qsubi,
                        qsubj,
                        qval);
    }

    if ( r==MSK_RES_OK )
      r = MSK_putobjsense(task, MSK_OBJECTIVE_SENSE_MINIMIZE);


    if ( r==MSK_RES_OK )
      r = MSK_optimize(task);

    if ( r==MSK_RES_OK )
    {
      MSK_getsolutionslice(task,
                           MSK_SOL_ITR,
                           MSK_SOL_ITEM_XX,
                           0,
                           NUMVAR,
                           xx);

      printf("Primal solution\n");
      for(j=0; j<NUMVAR; ++j)
        printf("x[%d]: %e\n",j,xx[j]);
    }
  }

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

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

```
  return ( r );
} /* 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

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x \in \mathcal{C} & & \end{array} \tag{5.22}$$

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

MOSEK supports two specific cones apart from the real numbers:

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

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 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

$$\begin{array}{lccc} \text{minimize} & x_4 + x_5 & & \\ \text{subject to} & x_0 + x_1 + x_2 + x_3 & = & 1, \\ & x_0, x_1, x_2, x_3 & \geq & 0, \\ & x_4 \geq \sqrt{x_0^2 + x_2^2}, & & \\ & x_5 \geq \sqrt{x_1^2 + x_3^2} & & \end{array} \tag{5.23}$$

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) 1998-2007 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};

MSKlidxt    ptrb[] = {0, 1, 2, 3,5, 5};
MSKlidxt    ptre[] = {1, 2, 3, 4, 5, 5};
double      aval[] = {1.0, 1.0, 1.0, 1.0};
MSKidxt     asub[] = {0, 0, 0, 0};

MSKidxt     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);

    MSK_echotask(task,
                 MSK_STREAM_MSG,
                 "Defining the problem data.\n");

    r = MSK_inputdata(task,
                      NUMCON,NUMVAR,
                      NUMCON,NUMVAR,
                      c,0.0,
```

```
                        ptrb,
                        ptre,
                        asub,
                        aval,
                        bkc,
                        blc,
                        buc,
                        bkx,
                        blx,
                        bux);

      if ( r==MSK_RES_OK )
      {
        /* Append the first cone. */

        csub[0] = 4;
        csub[1] = 0;
        csub[2] = 2;
        r = MSK_appendcone(task,
                           MSK_CT_QUAD,
                           0.0, /* For future use only, can be set to 0.0 */
                           3,
                           csub);
      }

      if ( r==MSK_RES_OK )
      {
        /* Append the second cone. */
        csub[0] = 5;
        csub[1] = 1;
        csub[2] = 3;

        r       =  MSK_appendcone(task,
                                  MSK_CT_QUAD,
                                  0.0,
                                  3,
                                  csub);
      }

      if ( r==MSK_RES_OK )
      {
        MSK_echotask(task,
                     MSK_STREAM_MSG,
                     "Start optimizing\n");

        r = MSK_optimize(task);

        if ( r==MSK_RES_OK )
        {
          MSK_getsolutionslice(task,
                               MSK_SOL_ITR,
                               MSK_SOL_ITEM_XX,
                               0,
                               NUMVAR,
                               xx);

          printf("Primal solution\n");
          for(j=0; j<NUMVAR; ++j)
```

```
            printf("x[%d]: %e\n",j,xx[j]);
        }
      }
    }
    /* Delete the task and the associated data. */
    MSK_deletetask(&task);
  }

  /* Delete the environment and the associated data. */
  MSK_deleteenv(&env);

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

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

5.4.1.2 Source code comments

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

```
r = MSK_appendcone(task,
                   MSK_CT_QUAD,
                   0.0, /* For future use only, can be set to 0.0 */
                   3,
                   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. c

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

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

$$\begin{array}{lccl} \text{maximize} & x_0 + 0.64x_1 & & \\ \text{subject to} & 50x_0 + 31x_1 & \leq & 250, \\ & 3x_0 - 2x_1 & \geq & -4, \\ & x_0, x_1 \geq 0 & & \text{and integer} \end{array} \tag{5.24}$$

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

5.5.1.1 Source code

The example (5.24) 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:

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

The complete source for the example is listed below.

```
/*
   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.

   File:      milo1.c

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

#include <stdio.h>

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

#define NUMCON 2   /* Number of constraints.             */
#define NUMVAR 2   /* 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;
  double       c[]   = {  1.0, 0.64 };
  MSKboundkeye bkc[] = { MSK_BK_UP,    MSK_BK_LO };
  double       blc[] = { -MSK_INFINITY,-4.0 };
  double       buc[] = { 250.0,        MSK_INFINITY };

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


  MSKlidxt     ptrb[] = { 0, 2 };
  MSKlidxt     ptre[] = { 2, 4 };
  MSKidxt      asub[] = { 0,    1,   0,    1 };
  double       aval[] = { 50.0, 3.0, 31.0, -2.0 };
  MSKidxt      j;

  double       xx[NUMVAR];
  MSKenv_t     env;
```

```
MSKtask_t    task;

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

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

/* 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);

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

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

  if ( r==MSK_RES_OK )
    r = MSK_inputdata(task,
                      NUMCON,NUMVAR,
                      NUMCON,NUMVAR,
                      c,0.0,
                      ptrb,
                      ptre,
                      asub,
                      aval,
                      bkc,
                      blc,
                      buc,
                      bkx,
                      blx,
                      bux);

  /* Specify integer variables. */
  for(j=0; j<NUMVAR && r == MSK_RES_OK; ++j)
    r = MSK_putvartype(task,j,MSK_VAR_TYPE_INT);

  if ( r==MSK_RES_OK )
    r =  MSK_putobjsense(task,
                         MSK_OBJECTIVE_SENSE_MAXIMIZE);

  if ( r==MSK_RES_OK )
    r = MSK_optimize(task);

  if ( r==MSK_RES_OK )
  {
    MSK_getsolutionslice(task,
                         /* Ask for integer solution */
                         MSK_SOL_ITG,
                         MSK_SOL_ITEM_XX,
                         0,
                         NUMVAR,
                         xx);
```

```
      printf("Primal solution\n");
      for(j=0; j<NUMVAR; ++j)
        printf("x[%d]: %e\n",j,xx[j]);
    }
  }

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

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

  return ( r );
} /* 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

$$\begin{array}{ll} \text{maximize} & 7x_0 + 10x_1 + x_2 + 5x_3 \\ \text{subject to} & x_0 + x_1 + x_2 + x_3 \leq 2.5 \\ & x_0, x_1, x_2 \text{ integer}, \quad x_0, x_1, x_2, x_3 \geq 0 \end{array} \tag{5.25}$$

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) 1998-2007 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  };
  MSKidxt      intsub[] = {0,1,2};
  MSKidxt      j;

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

  if ( r==MSK_RES_OK )
  {
    MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
  }

  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_inputdata(task,
                      NUMCON,NUMVAR,
                      NUMCON,NUMVAR,
                      c,
                      0.0,
                      ptrb,
                      ptre,
                      asub,
                      aval,
                      bkc,
                      blc,
                      buc,
                      bkx,
                      blx,
                      bux);

  if (r == MSK_RES_OK)
    MSK_putobjsense(task,MSK_OBJECTIVE_SENSE_MAXIMIZE);

  for(j=0; j<NUMINTVAR && r == MSK_RES_OK; ++j)
    r = MSK_putvartype(task,intsub[j],MSK_VAR_TYPE_INT);


  /* Construct an initial feasible solution from the
     values of the integer variables specified */

  if (r == MSK_RES_OK)
    r = MSK_putintparam(task,MSK_IPAR_MIO_CONSTRUCT_SOL,MSK_ON);

  /* Set status of all variables to unknown */
  if (r == MSK_RES_OK)
    r = MSK_makesolutionstatusunknown(task, MSK_SOL_ITG);

  /* Assign values 1,1,0 to integer variables */

  if (r == MSK_RES_OK)
    r = MSK_putsolutioni (
                          task,
                          MSK_ACC_VAR,
                          0,
                          MSK_SOL_ITG,
                          MSK_SK_SUPBAS,
                          0.0,
                          0.0,
                          0.0,
                          0.0);

  if (r == MSK_RES_OK)
    r = MSK_putsolutioni (
                          task,
                          MSK_ACC_VAR,
                          1,
                          MSK_SOL_ITG,
                          MSK_SK_SUPBAS,
                          2.0,
                          0.0,
                          0.0,
                          0.0);
```

```
  if (r == MSK_RES_OK)
    r = MSK_putsolutioni (
                          task,
                          MSK_ACC_VAR,
                          2,
                          MSK_SOL_ITG,
                          MSK_SK_SUPBAS,
                          0.0,
                          0.0,
                          0.0,
                          0.0);

  /* solve */

  if (r == MSK_RES_OK)
    r = MSK_optimize(task);

  /* Did mosek construct a feasible initial solution ? */

  {
    int isok;

    if (r == MSK_RES_OK)
      r = MSK_getintinf(task,MSK_IINF_MIO_CONSTRUCT_SOLUTION,&isok);

    if ( isok>0 && r == MSK_RES_OK)
      printf("MOSEK constructed a feasible initial soulution.\n");
  }
  /* Delete the task. */

  MSK_deletetask(&task);

  MSK_deleteenv(&env);

  printf("Return code: %d\n",r);
  if ( r!=MSK_RES_OK )
  {
    MSK_getcodedisc(r,buffer,NULL);
    printf("Description: %s\n",buffer);
  }

  return (r);
}
```

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:

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

and

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

The following code loads this problem into the optimization task.

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

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

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

MSKboundkeye bkc[]  = {MSK_BK_UP, MSK_BK_UP, MSK_BK_UP    };
double       blc[]  = {-MSK_INFINITY, -MSK_INFINITY, -MSK_INFINITY};
double       buc[]  = {100000, 50000, 60000};

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

double       xx[NUMVAR];
```

```
MSKenv_t      env;
MSKtask_t     task;
MSKintt       numvar,numcon;

/* 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 env log stream to the
     'printstr' function. */
  MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
}

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

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

  /* Directs the log task stream to the
     'printstr' function. */

  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 efficiency 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);

  if (r == MSK_RES_OK)
    r = MSK_putmaxnumanz(task,NUMANZ);

  /* Append the constraints. */
  if (r == MSK_RES_OK)
    r = MSK_append(task,MSK_ACC_CON,NUMCON);

  /* Append the variables. */
  if (r == MSK_RES_OK)
    r = MSK_append(task,MSK_ACC_VAR,NUMVAR);

  /* Put C. */
  if (r == MSK_RES_OK)
    r = MSK_putcfix(task, 0.0);

  if (r == MSK_RES_OK)
    for(j=0; j<NUMVAR; ++j)
      r = MSK_putcj(task,j,c[j]);
```

```
  /* Put constraint bounds. */
  if (r == MSK_RES_OK)
    for(i=0; i<NUMCON; ++i)
      r = MSK_putbound(task,MSK_ACC_CON,i,bkc[i],blc[i],buc[i]);

  /* Put variable bounds. */
  if (r == MSK_RES_OK)
    for(j=0; j<NUMVAR; ++j)
      r = MSK_putbound(task,MSK_ACC_VAR,j,bkx[j],blx[j],bux[j]);

  /* Put A. */
  if (r == MSK_RES_OK)
    if ( NUMCON>0 )
      for(j=0; j<NUMVAR; ++j)
        r = MSK_putavec(task,
                        MSK_ACC_VAR,
                        j,
                        ptre[j]-ptrb[j],
                        asub+ptrb[j],
                        aval+ptrb[j]);

  if (r == MSK_RES_OK)
    r = MSK_putobjsense(task,
                        MSK_OBJECTIVE_SENSE_MAXIMIZE);

  if (r == MSK_RES_OK)
    r = MSK_optimize(task);

  if (r == MSK_RES_OK)
    MSK_getsolutionslice(task,
                         MSK_SOL_BAS,        /* Basic solution.       */
                         MSK_SOL_ITEM_XX,    /* Which part of solution.  */
                         0,                  /* Index of first variable. */
                         NUMVAR,             /* Index of last variable+1 */
                         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.

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

The problem now has the form:

$$
\begin{array}{lccccccl}
\text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & & \\
\text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & \leq & 100000, \\
 & 3x_0 & + & 2x_1 & + & 3x_2 & \leq & 50000, \\
 & 2x_0 & + & 3x_1 & + & 2x_2 & \leq & 60000,
\end{array}
\tag{5.28}
$$

and

$$
x_0, x_1, x_2 \geq 0. \tag{5.29}
$$

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.

```
/* Append a new variable x_3 to the problem */
if (r == MSK_RES_OK)
  r = MSK_append(task,MSK_ACC_VAR,1);

/* Get index of new variable, this should be 3 */
if (r == MSK_RES_OK)
  r = MSK_getnumvar(task,&numvar);

/* Set bounds on new variable */
if (r == MSK_RES_OK)
  r = MSK_putbound(task,
                   MSK_ACC_VAR,
                   numvar-1,
                   MSK_BK_LO,
                   0,
                   +MSK_INFINITY);

/* 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:

$$\begin{array}{lccccccccc} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & + & 1.0x_3 & & \\ \text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & + & 4x_3 & \leq & 100000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & & & \leq & 50000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & + & 1x_3 & \leq & 60000, \end{array} \tag{5.30}$$

and

$$x_0, x_1, x_2, x_3 \geq 0. \tag{5.31}$$

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 (5.28) was found and then added a column was added to get (5.30). 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_optimize(task);
```

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

$$x_0 + 2x_1 + x_2 + x_3 \leq 30000 \tag{5.32}$$

to the problem which is done in the following code:

```
/* Append a new constraint */
if (r == MSK_RES_OK)
  r = MSK_append(task,MSK_ACC_CON,1);

/* Get index of new constraint, this should be 4 */
if (r == MSK_RES_OK)
  r = MSK_getnumcon(task,&numcon);

/* Set bounds on new constraint */
if (r == MSK_RES_OK)
  r = MSK_putbound(task,
```

```
                    MSK_ACC_CON,
                    numcon-1,
                    MSK_BK_UP,
                    -MSK_INFINITY,
                    30000);

/* Put new values in the A matrix */
if (r == MSK_RES_OK)
{
  MSKidxt arowsub[] = {0,   1,   2,   3  };
  double arowval[] =  {1.0, 2.0, 1.0, 1.0};

  r = MSK_putavec(task,
                  MSK_ACC_CON,
                  numcon-1, /* row index */
                  4, /* num nz in row*/
                  arowsub,
                  arowval);
}
```

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.

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:

$$q^o_{\texttt{qosubi[t]},\texttt{qosubj[t]}} = \texttt{qoval[t]},\ t = 0, \ldots, \texttt{numqonz} - 1. \tag{5.33}$$

- The linear terms in the objective:

$$c_j = \texttt{c[j]},\ j = 0, \ldots, \texttt{numvar} - 1 \tag{5.34}$$

- The fixed term in the objective:

$$c^f = \texttt{cfix}. \tag{5.35}$$

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

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 quadratic terms in the constraints:

$$q^{\texttt{qcsubk[t]}}_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} = \texttt{qcval}[\texttt{t}],\ t = 0, \ldots, \texttt{numqcnz} - 1. \tag{5.36}$$

- The linear terms in the constraints:

$$\begin{array}{ll} a_{\texttt{asub[t]},\texttt{j}} = \texttt{aval}[\texttt{t}], & t = \texttt{ptrb}[\texttt{j}], \ldots, \texttt{ptre}[\texttt{j}] - 1, \\ & j = 0, \ldots, \texttt{numvar} - 1. \end{array} \tag{5.37}$$

- 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

$$l^c_k = \texttt{blc}[\texttt{k}],\ k = 0, \ldots, \texttt{numcon} - 1 \tag{5.38}$$

  and

$$u^c_k = \texttt{buc}[\texttt{k}],\ k = 0, \ldots, \texttt{numcon} - 1. \tag{5.39}$$

- 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

$$l^x_j = \texttt{blx}[\texttt{j}],\ j = 0, \ldots, \texttt{numvar} - 1. \tag{5.40}$$

  The numerical values for the upper bounds on the variables are given by

$$u^x_j = \texttt{bux}[\texttt{j}],\ j = 0, \ldots, \texttt{numvar} - 1. \tag{5.41}$$

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

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

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

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

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

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

```
MSKidxt subi[] = {   1,   3,   5 };
MSKidxt subj[] = {   2,   3,   4 };
MSKrealt cof[] = { 1.1, 4.3, 0.2 };
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

$$a_{\texttt{asub}[k],j} = \texttt{aval}[k], \quad k = \texttt{ptrb}[j], \ldots, \texttt{ptre}[j] - 1. \tag{5.42}$$

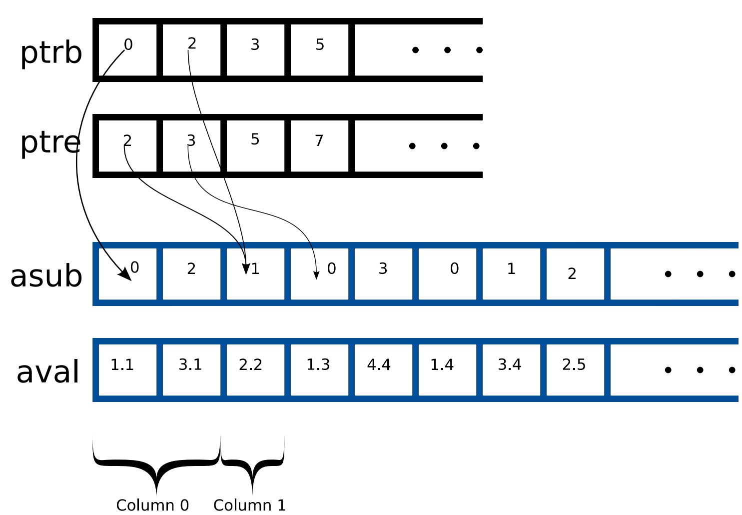

Figure 5.1: The matrix A (5.43) represented in column ordered sparse matrix format.

As an example consider the matrix

$$A = \begin{bmatrix} 1.1 & & 1.3 & 1.4 & \\ & 2.2 & & & 2.5 \\ 3.1 & & & 3.4 & \\ & & 4.4 & & \end{bmatrix}. \tag{5.43}$$

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 (5.43) is represented in column ordered sparse matrix format.

5.8.3.3 Row ordered sparse matrix

The matrix A (5.43) can also be represented in the row ordered sparse matrix format as:

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

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

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

   $$f^j : \; R \rightarrow R \text{ and } g_i^j : \; R \rightarrow R.$$

   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 7.5 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:

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

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\5\tools\examp\
```

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) 1998-2007 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. */
```

```
  numoprc  = NUMOPRC;

  oprc[0]  = MSK_OPR_POW;
  opric[0] = 0;
  oprjc[0] = 0;
  oprfc[0] = 1.0;
  oprgc[0] = 2.0;
  oprhc[0] = 0.0;

  oprc[1]  = MSK_OPR_POW;
  opric[1] = 0;
  oprjc[1] = 1;
  oprfc[1] = 1.0;
  oprgc[1] = 2.0;
  oprhc[1] = 0.0;

  /* Specify c */
  c[0] = 1.0; c[1] = 0.0; c[2] = 0.0;

  /* Specify a. */
  aptrb[0] = 0;   aptrb[1] = 1;   aptrb[2] = 2;
  aptre[0] = 1;   aptre[1] = 2;   aptre[2] = 3;
  asub[0]  = 1;   asub[1]  = 1;   asub[2]  = 1;
  aval[0]  = 1.0; aval[1]  = 2.0; aval[2]  = -1.0;

  /* Specify bounds for constraints. */
  bkc[0] = MSK_BK_UP;     bkc[1] = MSK_BK_FX;
  blc[0] = -MSK_INFINITY; blc[1] = 0.0;
  buc[0] = 1.0;           buc[1] = 0.0;

  /* Specify bounds for variables. */
  bkx[0] = MSK_BK_FR;      bkx[1] = MSK_BK_FR;     bkx[2] = MSK_BK_LO;
  blx[0] = -MSK_INFINITY;  blx[1] = -MSK_INFINITY; blx[2] = 0.0;
  bux[0] = MSK_INFINITY;   bux[1] = MSK_INFINITY;  bux[2] = MSK_INFINITY;

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

  /* Check whether the return code is ok. */
  if ( r==MSK_RES_OK )
  {
    /* Directs the log stream to the user
       specified procedure 'printstr'. */
    MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
  }

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

  if ( r==MSK_RES_OK )
  {
    /* Make the optimization task. */
    r = MSK_makeemptytask(env,&task);
    if ( r==MSK_RES_OK )
      MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
```

```
    if ( r==MSK_RES_OK )
    {
      r = MSK_inputdata(task,
                        numcon,numvar,
                        numcon,numvar,
                        c,0.0,
                        aptrb,aptre,
                        asub,aval,
                        bkc,blc,buc,
                        bkx,blx,bux);
    }

    if ( r== MSK_RES_OK )
    {
      /* Set-up of nonlinear expressions. */
      r = MSK_scbegin(task,
                      numopro,opro,oprjo,oprfo,oprgo,oprho,
                      numoprc,oprc,opric,oprjc,oprfc,oprgc,oprhc,
                      &sch);


      if ( r==MSK_RES_OK )
      {
        printf("Start optimizing\n");

        r = MSK_optimize(task);

        printf("Done optimizing\n");

        MSK_solutionsummary(task,MSK_STREAM_MSG);
      }

      /* The nonlinear expressions are no longer needed. */
      MSK_scend(task,&sch);
    }
    MSK_deletetask(&task);
  }
  MSK_deleteenv(&env);

  printf("Return code: %d\n",r);
  if ( r!=MSK_RES_OK )
  {
    MSK_getcodedisc(r,buffer,NULL);
    printf("Description: %s\n",buffer);
  }
} /* 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

```
#define MSK_OPR_ENT 0
```

```
#define MSK_OPR_EXP 1
#define MSK_OPR_LOG 2
#define MSK_OPR_POW 3
#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:

```
static int evalopr(int    opr,
                   double f,
                   double g,
                   double h,
                   double xj,
                   double *fxj,
                   double *grdfxj,
                   double *hesfxj)
/* Purpose: To evaluate an operator and its derivatives.
     fxj:    Is the function value
     grdfxj: Is the first derivative.
     hexfxj: Is the second derivative.
 */
{
  double rtemp;

  switch ( opr )
  {
    case MSK_OPR_ENT:
      if ( fxj )
      fxj[0] = f*xj*log(xj);

      if ( grdfxj )
      grdfxj[0] = f*(1.0+log(xj));

      if ( hesfxj )
      hesfxj[0] = f/xj;
      break;
    case MSK_OPR_EXP:
      if ( fxj || grdfxj || hesfxj )
      {
        rtemp = exp(g*xj+h);

        if ( fxj )
        fxj[0] = f*rtemp;

        if ( grdfxj )
        grdfxj[0] = f*g*rtemp;

        if ( hesfxj )
        hesfxj[0] = f*g*g*rtemp;
      }
      break;
    case MSK_OPR_LOG:
      rtemp = g*xj+h;
      if ( rtemp<=0.0 )
        return ( 1 );
```

```
      if ( fxj )
      fxj[0] = f*log(rtemp);

      if ( grdfxj )
      grdfxj[0] = (g*f)/(rtemp);

      if ( hesfxj )
      hesfxj[0] = -(f*g*g)/(rtemp*rtemp);
      break;
    case MSK_OPR_POW:
      if ( fxj )
      fxj[0] = f*pow(xj+h,g);

      if ( grdfxj )
      grdfxj[0] = f*g*pow(xj+h,g-1.0);

      if ( hesfxj )
        hesfxj[0] = f*g*(g-1.0)*pow(xj+h,g-2.0);
      break;
    case MSK_OPR_SQRT: /* handle operator f * sqrt(x+g) */
      if ( fxj )
        fxj[0] = f*sqrt(g*xj+h); /* The function value. */

      if ( grdfxj )
        grdfxj[0] = 0.5*f*g/sqrt(g*xj+h); /* The gradient. */


      if ( hesfxj )
        hesfxj[0] = -0.25*f*g*g*pow(g*xj+h,-1.5);
      break;
    default:
      printf("scopt.c: Unknown operator %d\n",opr);
      exit(0);
  }

  return ( 0 );
} /* evalopr */
```

6.2 Exponential optimization

6.2.1 The problem

An exponential optimization problem has the form

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

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 the function

$$e^x$$

is only well-defined 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:

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

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

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

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:

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

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.

```
/*
   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.

   File     : tstexpopt.c

   Purpose  : To demonstrate a simple interface for exponential optimization.
*/

#include <string.h>

#include "expopt.h"

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


int main (int argc, char **argv)
{
  int          r = MSK_RES_OK, numcon = 1, numvar = 3 , numter = 5;
```

```
int           subi[]   = {0,0,0,1,1};
int           subk[]   = {0,0,0,1,1,2,2,2,3,3,4,4};
double        c[]      = {40.0,20.0,40.0,0.333333,1.333333};
int           subj[]   = {0,1,2,0,2,0,1,2,0,1,1,2};
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};
int           numanz   = 12;
double        objval;
double        xx[3];
double        y[5];
MSKenv_t      env;
MSKprostae    prosta;
MSKsolstae    solsta;
MSKtask_t     expopttask;
expopthand_t expopthnd = NULL;
/* Pointer to data structure that holds nonlinear information */

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

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

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

if (r == MSK_RES_OK)
  MSK_makeemptytask(env,&expopttask);

if (r == MSK_RES_OK)
  r = MSK_linkfunctotaskstream(expopttask,MSK_STREAM_LOG,NULL,printcb);

if (r == MSK_RES_OK)
{
  /* Initialize expopttask with problem data */
  r =  MSK_expoptsetup(expopttask,
                       1, /* Solve the dual formulation */
                       numcon,
                       numvar,
                       numter,
                       subi,
                       c,
                       subk,
                       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

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

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 to use, and it it recommended to use the conic solver, the quadratic solver or the `scopt` interface whenever possible. Alternatively the 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

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

subject to the functional constraints

$$l^c_k \leq g_k(x) + \frac{1}{2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} q^k_{i,j} x_i x_j + \sum_{j=0}^{n-1} a_{k,j} x_j \leq u^c_k, \; k = 0,\ldots,m-1, \tag{6.8}$$

and the bounds

$$l^x_j \leq x_j \leq u^x_j, \; j = 0,\ldots,n-1. \tag{6.9}$$

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

$$u^c_k < \infty$$

and a concave function if

$$l^c_k > -\infty.$$

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

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

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

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

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 $v \in R^n$ 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:

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

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

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

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\dgo.mps examp\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/5/tools/examp/`

and listed below:

```
/*
   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.

   File:      dgopt.c

   Purpose:   To solve dual geometric optimization problems
              using the MOSEK API.

              On the command line type

              dgopt go.mps go.f

              where

                go.mps: Is an MPS file specifing
                        the linear part of the
                        problem.

                go.f  : Is an (ASCII) file specifying
                        the nonlinear part of the
                        problem.
 */

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

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


#define MAX_LINE_LENGTH 256
#define DEBUG           0
#define PRINT_GRDLAG    0
#define PRINT_HESVAL    0
#define OBJSCAL         1.0
#define DUMPHESSIAN     0

#if DEBUG
#include <assert.h>
#endif

typedef struct
        {
          /*
           * Data structure for storing
           * data about the nonlinear
           * function in the objective.
           */

          MSKtask_t   task;

          MSKintt     n;        /* Number of variables.  */
```

```
          MSKintt     t;        /* Number of terms in
                                   the objective of the primal problem. */
          MSKintt     *p;
          MSKlintt    numhesnz; /* Number of non-zeros in
                                   the Hessian.          */
        } nlhandt;

typedef nlhandt *nlhand_t;


static int MSKAPI printnldata(nlhand_t nlh)
{
  MSKidxt i;

  printf ("* Begin: dgo nl data debug. *\n");

  printf ("n = %d, t = %d\n",nlh->n,nlh->t);

  for (i=0; i<nlh->t + 1;++i)
    printf("p[%d] = %d\n",i,nlh->p[i]);


  printf ("* End: dgo nl data debug. *\n");

  return 0;
} /* printnldata */


int MSKAPI dgostruc(void    *nlhandle,
                    MSKintt  *numgrdobjnz,
                    MSKidxt  *grdobjsub,
                    MSKidxt  i,
                    int      *convali,
                    MSKidxt  *grdconinz,
                    MSKidxt  *grdconisub,
                    MSKintt  yo,
                    MSKintt  numycnz,
                    MSKidxt  *ycsub,
                    MSKlintt maxnumhesnz,
                    MSKlintt *numhesnz,
                    MSKidxt  *hessubi,
                    MSKidxt  *hessubj)
/* Purpose: Provide information to MOSEK about the
            problem structure and sparsity.
 */
{
  MSKidxt  j,k,l;
  nlhand_t nlh;

  nlh = (nlhand_t) nlhandle;

  MSK_checkmemtask(nlh->task,__FILE__,__LINE__);

  if ( numgrdobjnz )
  {
    /* All the variables appear nonlinearly
     * in the objective.
     */
```

```
    numgrdobjnz[0] = 0;

    for(k=0; k<1; ++k)
    {
      for(j=nlh->p[k]; j<nlh->p[k+1]; ++j)
      {
        if ( grdobjsub )
          grdobjsub[numgrdobjnz[0]] = j;

        ++ numgrdobjnz[0];
      }
    }

    for(k=1; k<nlh->t; ++k)
    {
      if ( nlh->p[k+1]-nlh->p[k]>1 )
      {
        for(j=nlh->p[k]; j<nlh->p[k+1]; ++j)
        {
          if ( grdobjsub )
            grdobjsub[numgrdobjnz[0]] = j;

          ++ numgrdobjnz[0];
        }
      }
    }
  }

  if ( convali )
    convali[0] = 0;    /* Zero because no nonlinear
                        * expression in the constraints.
                        */

  if ( grdconinz )
    grdconinz[0] = 0;  /* Zero because no nonlinear
                        * expression in the constraints.
                        */

  if ( numhesnz )
  {
    if ( yo )
      numhesnz[0] = nlh->numhesnz;
    else
      numhesnz[0] = 0;
  }

  if ( maxnumhesnz )
  {
    /* Should return information about the Hessian too. */

    if ( maxnumhesnz<numhesnz[0] )
    {
      /* Not enough space have been allocated for
       * storing the Hessian.
       */
```

```
      return ( 1 );
    }
    else
    {
      if ( yo )
      {
        if ( hessubi && hessubj )
        {
          /*
           * Compute and store the sparsity pattern of the
           * Hessian of the Lagrangian.
           */

          l = 0;
          for(j=nlh->p[0]; j<nlh->p[1]; ++j)
          {
            hessubi[l]  = j;
            hessubj[l]  = j;
                      ++ l;
          }

          for(k=1; k<nlh->t; ++k)
          {
            for(j=nlh->p[k]; j<nlh->p[k+1]; ++j)
            {
              for(i=j; i<nlh->p[k+1]; ++i)
              {
                if (nlh->p[k+1]-nlh->p[k]>1)
                {
                  hessubi[l]  = i;
                  hessubj[l]  = j;
                            ++ l;
                }
              }
            }
          }
        }
      }
    }
  }

  return ( 0 );
} /* dgostruc */

static int MSKAPI dgoeval(void      *nlhandle,
                          double    *xx,
                          double    yo,
                          double    *yc,
                          double    *objval,
                          MSKintt   *numgrdobjnz,
                          MSKidxt   *grdobjsub,
                          double    *grdobjval,
                          MSKintt   numi,
                          MSKidxt   *subi,
                          double    *conval,
                          MSKlidxt  *grdconptrb,
                          MSKlidxt  *grdconptre,
                          MSKidxt   *grdconsub,
```

```
                            double    *grdconval,
                            double    *grdlag,
                            MSKlintt  maxnumhesnz,
                            MSKlintt  *numhesnz,
                            MSKidxt   *hessubi,
                            MSKidxt   *hessubj,
                            double    *hesval)
/* Purpose: To evaluate the nonlinear function and return the
        requested information to MOSEK.
     */
{
  double   rtemp;
  int      evalok=1;
  MSKidxt  i,j,k,l,itemp;
  nlhand_t nlh;

  nlh = (nlhand_t) nlhandle;

  MSK_checkmemtask(nlh->task,__FILE__,__LINE__);

  if ( objval )
  {
    /* f(x) is computed and stored in objval[0]. */
    objval[0] = 0.0;

    for(k=0; k<1 && evalok; ++k)
    {
      for(j=nlh->p[k]; j<nlh->p[k+1] && evalok; ++j)
      {


        #if DEBUG

        printf("(%d) xx = %p, k = %d, j = %d, nlh = %p, p[0] = %d\n",
               __LINE__,xx,k,j,nlh,nlh->p[0]);
        if ( xx[j]<=0.0 )
          printf("Zero xx[%d]: %e",j,xx[j]);

        assert(xx[j] > 0.0 );
        #endif

        if ( xx[j]<=0 )
          evalok     = 0;
        else
          objval[0] -= xx[j]*log(xx[j]);
      }
    }

    for(k=1; k<nlh->t && evalok; ++k)
    {
      if ( nlh->p[k+1]-nlh->p[k]>1 )
      {
        for(j=nlh->p[k]; j<nlh->p[k+1]; ++j)
        {
#if DEBUG
          if ( xx[j]<=0.0 )
            printf("Zero xx[%d]: %e",j,xx[j]);
```

```
          assert(xx[j] > 0);
#endif
          objval[0] -= xx[j]*log(xx[j]);
        }
        rtemp = 0.0;

        for(j=nlh->p[k]; j<nlh->p[k+1]; ++j)
          rtemp += xx[j];

        if ( rtemp<=0.0 )
          return ( 1 );

#if DEBUG
        assert(rtemp > 0);
#endif

        objval[0] += rtemp*log(rtemp);
      }
    }

    objval[0] *= OBJSCAL;

#if DEBUG
    printf ("objval = %e\n",objval[0]);
#endif
  }


  if ( numgrdobjnz )
  {
    /* Compute and store the gradient of the f. */


    itemp = 0;

    for(k=0; k<1 && evalok; ++k)
    {
      for(j=nlh->p[k]; j<nlh->p[k+1]; ++j)
      {
        grdobjsub[itemp]  = j;
#if DEBUG
        assert(xx[j] > 0);
#endif
        grdobjval[itemp]  = -log(xx[j])-1.0;
                          ++ itemp;
      }
    }

    for(k=1; k<nlh->t && evalok; ++k)
    {
      if ( nlh->p[k+1]-nlh->p[k]>1 )
      {
        rtemp = 0.0;
        for(j=nlh->p[k]; j<nlh->p[k+1]; ++j)
          rtemp += xx[j];

        for(j=nlh->p[k]; j<nlh->p[k+1]; ++j)
        {
```

```
          grdobjsub[itemp]  = j;
#if DEBUG
          assert(xx[j] > 0);
#endif
          grdobjval[itemp]  = log(rtemp/xx[j]);
                           ++ itemp;
        }
      }
    }

    numgrdobjnz[0] = itemp;

    for(k=0; k<numgrdobjnz[0]; ++k)
      grdobjval[k] *= OBJSCAL;

  }

  if ( conval )
    for(k=0; k<numi; ++k)
      conval[k] = 0.0;

  if ( grdlag && evalok )
  {
    /* Compute and store the gradient of the Lagrangian.
     * Note that it is stored as a dense vector.
     */


    for(j=0; j<nlh->n; ++j)
      grdlag[j] = 0.0;

    for(k=0; k<1 && evalok; ++k)
    {
      for(j=nlh->p[k]; j<nlh->p[k+1] && evalok; ++j)
      {
        if (xx[j] <= 0.0 )
          evalok = 0;
        else
          grdlag[j]  = yo*(-log(xx[j])-1.0);
      }
    }

    for(k=1; k<nlh->t && evalok; ++k)
    {
      if ( nlh->p[k+1]-nlh->p[k]>1 )
      {
        rtemp = 0.0;
        for(j=nlh->p[k]; j<nlh->p[k+1]; ++j)
          rtemp += xx[j];

        for(j=nlh->p[k]; j<nlh->p[k+1] && evalok; ++j)
        {
          if ( xx[j]<=0.0 )
            evalok = 0.0;
          else
            grdlag[j] = yo*log(rtemp/xx[j]);
        }
      }
```

```
    }


    for(j=0; j<nlh->n; ++j)
      grdlag[j] *= OBJSCAL;

#if DEBUG && PRINT_GRDLAG
     for(j=0; j<nlh->n; ++j)
       printf("grdlag[%d] = %e\n",j,grdlag[j]);
#endif

  }

  if ( maxnumhesnz )
  {
    /* Compute and store the Hessian of the Lagrangian
     * which in this case is identical to the Hessian
     * of f times yo.
     */

    if ( yo==0.0 )
    {
      if ( numhesnz )
        numhesnz[0] = 0;
    }
    else
    {
      if ( numhesnz )
      {
        numhesnz[0] = nlh->numhesnz;

        if ( maxnumhesnz<nlh->numhesnz )
          return ( 1 );

        /* The diagonal element. */
        l = 0;
        for(j=nlh->p[0]; j<nlh->p[1]; ++j)
        {
          hessubi[l]  = j;
          hessubj[l]  = j;
          hesval[l]   =  -yo/xx[j];
                        ++ l;

        }

        for(k=1; k<nlh->t; ++k)
        {
          if ( nlh->p[k+1]-nlh->p[k]>1)
          {
            double invrtemp;

            rtemp = 0.0;
            for(j=nlh->p[k]; j<nlh->p[k+1]; ++j)
            {
              rtemp += xx[j];
            }

            invrtemp = 1.0/rtemp;
```

```
            /* The diagonal element. */
            for(j=nlh->p[k]; j<nlh->p[k+1]; ++j)
            {
              hessubi[l]  = j;
              hessubj[l]  = j;
              /* equivalent to hesval[l]   = yo*(invrtemp - 1.0/xx[j]); */
              hesval[l]   = yo*(xx[j]-rtemp)/(rtemp*xx[j]);
                         ++ l;

              /* The off diagonal elements. */
             for(i=j+1; i<nlh->p[k+1]; ++i)
             {
               hessubi[l]  = i;
               hessubj[l]  = j;
               hesval[l]   = yo*invrtemp;
                          ++ l;
             }
            }
          }
        }

        for(k=0; k<numhesnz[0]; ++k)
          hesval[k] *= OBJSCAL;


        #if DUMPHESSIAN
        {
          FILE *f;

          f = fopen("hessian.txt","wt");
          for(k=0; k<numhesnz[0]; ++k)
            fprintf(f,"%d %d %24.16e\n",hessubi[k],hessubj[k],hesval[k]);

          fclose(f);
        }
        #endif


#if DEBUG && PRINT_HESVAL
        for(k=0; k<numhesnz[0]; ++k)
          printf("hesval[%d] = %e\n",k,hesval[k]);
#endif

      }
    }
  }
  MSK_checkmemtask(nlh->task,__FILE__,__LINE__);

  return ( !evalok );
} /* dgoeval */

MSKrescodee MSK_dgoread(MSKtask_t task,
                        char      *nldatafile,
                        MSKintt   *numvar,   /* numterms in primal */
                        MSKintt   *numcon,   /* numvar in primal */
                        MSKintt   *t,        /* constraints in primal */
                        double    **v,       /* coefficients for terms */
```

```
                        MSKintt    **p         /* corresponds to number of
                                                  terms in each constraint in the
                                                  primal */
                       )
{
  MSKrescodee r=MSK_RES_OK;
  MSKenv_t    env;
  char        buf[MAX_LINE_LENGTH];
  FILE        *f;
  MSKintt     i;

  MSK_getenv(task,&env);
  v[0] = NULL; p[0] = NULL;

  f         = fopen(nldatafile,"rt");

  if (f)
  {
    fgets(buf,sizeof(buf),f);
    t[0] = (int) atol(buf);
  }
  else
  {
    printf("Could not open file '%s'\n",nldatafile);
    r = MSK_RES_ERR_FILE_OPEN;
  }

  if (r == MSK_RES_OK)
    r = MSK_getnumvar(task,numvar);

  if (r == MSK_RES_OK)
    r = MSK_getnumcon(task,numcon);


  if (r == MSK_RES_OK)
  {
    p[0] = (int*) MSK_calloctask(task,t[0],sizeof(int));
    if (p[0] == NULL)
      r = MSK_RES_ERR_SPACE;
  }


  if (r == MSK_RES_OK)
  {
    v[0] = (double*) MSK_calloctask(task,numvar[0],sizeof(double));
    if (v[0] == NULL)
      r = MSK_RES_ERR_SPACE;
  }

  if (r == MSK_RES_OK)
  {
    for(i=0; i<numvar[0]; ++i)
    {
       fgets(buf,sizeof(buf),f);
       v[0][i] = atof(buf);
    }

    for(i=0; i<t[0]; ++i)
```

```
    {
      fgets(buf,sizeof(buf),f);
      p[0][i] = (int) atol(buf);
    }
  }

  return ( r );
}

MSKrescodee
MSK_dgosetup(MSKtask_t task,
             MSKintt   numvar,
             MSKintt   numcon,
             MSKintt   t,
             double    *v,
             MSKintt   *p,
             nlhand_t  *nlh)
{

  MSKintt     j,k;
  MSKrescodee r=MSK_RES_OK;
  MSKenv_t    env;


  nlh[0] = NULL;

  MSK_getenv(task,&env);

    /* set up nonlinear part */

  if (r == MSK_RES_OK)
  {
    nlh[0] = (nlhand_t) MSK_calloctask(task,1,sizeof(nlhandt));
    if (nlh[0] == NULL)
      r = MSK_RES_ERR_SPACE;
  }


  nlh[0]->p = NULL;

  if ( r == MSK_RES_OK )
  {
    nlh[0]->n = numvar;


    if ( r==MSK_RES_OK )
    {
      nlh[0]->t = t;
      nlh[0]->task = task;


      if (r == MSK_RES_OK)
      {
        nlh[0]->p = MSK_calloctask(task,nlh[0]->t+1,sizeof(int));
        if (nlh[0]->p == NULL)
          r = MSK_RES_ERR_SPACE;
      }
```

```
      if ( r == MSK_RES_OK )
      {
        nlh[0]->p[0] = 0;
        for(k=0; k<nlh[0]->t; ++k)
        {
          nlh[0]->p[k+1] = nlh[0]->p[k]+p[k];
        }



        for(k=0; k<nlh[0]->t; ++k)
        {
          for(j=nlh[0]->p[k]; j<nlh[0]->p[k+1]; ++j)
          {
#if DEBUG
            assert(v[j] > 0);
#endif
            MSK_putcj(task,j,OBJSCAL*log(v[j]));
          }
        }


        if ( nlh[0]->p[nlh[0]->t]==nlh[0]->n )
        {
          /*
           * The problem is now defined
           * and the setup can proceed.
           * Next, the number of Hessian non-zeros
           * is computed.
           */

          nlh[0]->numhesnz = nlh[0]->p[1]-nlh[0]->p[0];
          for(k=1; k<nlh[0]->t; ++k)
          {
            if (( nlh[0]->p[k+1]-nlh[0]->p[k])>1 )
            {
              /* If only one term in primal constraint,
                 the corresponding value in H is zero.
               */
              nlh[0]->numhesnz += ((nlh[0]->p[k+1]-nlh[0]->p[k])
                                   * (1+nlh[0]->p[k+1]-nlh[0]->p[k]))/2;
            }
          }
          printf("Number of Hessian non-zeros: %d\n",nlh[0]->numhesnz);

          MSK_putnlfunc(task,nlh[0],dgostruc,dgoeval);
        }
        else
        {
          printf("Incorrect function definition.\n");
          printf("n gathered from the task file: %d\n",nlh[0]->n);
          printf("n computed based on p         : %d\n",nlh[0]->p[nlh[0]->t]);
          r = MSK_RES_ERR_UNKNOWN;
        }
      }
    }
  }
```

```
  if (r == MSK_RES_OK)
    r = MSK_putobjsense(task,MSK_OBJECTIVE_SENSE_MAXIMIZE);

  return ( r );
} /* dgosetup */

int MSK_freedgo(MSKtask_t task,
                nlhand_t  *nlh)
{

  if ( nlh )
  {
    /* Free allocated data. */

    MSK_freetask(task,nlh[0]->p);
    MSK_freetask(task,nlh[0]);
  }

  nlh[0] = NULL;

  return ( MSK_RES_OK );
}
```

```
/*
   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.

   File:      mskdgopt.c

   Purpose:
              To solve the dual geometric programming problem.
              Input consists of:

                1. An MPS file containing the linear part of the problem
                2. A file containing information about the nonlinear objective.

              E.g

                msdgopt dgo.mps dgo.f
*/



#include "dgopt.h"

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


int main (int argc,char ** argv)
{
  int         numvar,numcon,t,i;
  double      *v = NULL;
  int         *p = NULL;
  char        buffer[MSK_MAX_STR_LEN],symnam[MSK_MAX_STR_LEN];
  dgohand_t   nlh=NULL;
```

```
  MSKenv_t    env;
  MSKrescodee r = MSK_RES_OK;
  MSKtask_t   task;


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

  /* Check whether the return code is ok. */
  if ( r==MSK_RES_OK )
  {
    /* Directs the log stream to the user
       specified procedure 'printstr'. */
    MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
  }

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

  if ( r==MSK_RES_OK )
  {
    /* Make the optimization task. */
    r = MSK_makeemptytask(env,&task);

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

    if ( r==MSK_RES_OK && argc>3 )
    {
      /* Read parameter file if defined. */
      MSK_putstrparam(task,MSK_SPAR_PARAM_READ_FILE_NAME,argv[3]);
      r = MSK_readparamfile(task);
    }
  }

  if ( r==MSK_RES_OK )
    r = MSK_readdata(task,argv[1]);

  if (r == MSK_RES_OK)
    r =  MSK_dgoread( task,
                      argv[2],
                      &numvar,
                      &numcon,
                      &t,
                      &v,
                      &p
                      );


  if (r == MSK_RES_OK)
    r = MSK_dgosetup(task,
                     numvar,
                     numcon,
                     t,
                     v,
```

```
                     p,
                     &nlh);

  if (r == MSK_RES_OK)
    r = MSK_optimize(task);

  if (r == MSK_RES_OK)
  {
    MSK_putintparam(task,MSK_IPAR_WRITE_GENERIC_NAMES,MSK_ON);

    MSK_solutionsummary(task,MSK_STREAM_MSG);

    /*
     * The solution is written to the file dgopt.sol.
     */

    r = MSK_writesolution(task,MSK_SOL_ITR,"dgopt.sol");
  }

  MSK_freetask(task,v);
  MSK_freetask(task,p);

  if (nlh)
    MSK_freedgo(task,
                &nlh);

  MSK_deletetask(&task);

  MSK_deleteenv(&env);

  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 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 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

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

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

$$B\bar{x} = w \tag{6.13}$$

and

$$B^T\bar{x} = w \tag{6.14}$$

each has a unique solution for all w.

MOSEK provides functions for solving the linear systems (6.13) and (6.14) for an arbitrary w.

6.5.1 Identifying the basis

To use the solutions to (6.13) and (6.14) it is important to know how the basis matrix B is constructed.

Internally MOSEK employs the linear optimization problem

$$\begin{array}{lccccl} \text{maximize} & & & c^T x & & \\ \text{subject to} & & & Ax - x^c & = & 0 \\ & l^x & \leq & x & \leq & u^x, \\ & l^c & \leq & x^c & \leq & u^c. \end{array} \tag{6.15}$$

where

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

The basis matrix is constructed of m columns taken from

$$\left[\begin{array}{cc} A & -I \end{array}\right].$$

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

```
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_i^c. Moreover, the i'th column in B will be the i'th column of $-I$. On the other hand if

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

then the i'th basis variable is variable

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

and the i'th column of B is the column

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

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

6.5.2 An example

Consider the linear optimization problem:

$$\begin{array}{lccl} \text{minimize} & x_0 + x_1 & & \\ \text{subject to} & x_0 + 2x_1 & \leq & 2, \\ & x_0 + x_1 & \leq & 6, \\ & x_0, x_1 \geq 0. & & \end{array} \tag{6.16}$$

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

$$\left[\begin{array}{cc} 0 & 1 \\ -1 & 1 \end{array} \right]. \tag{6.17}$$

Please note the ordering of the columns in B.

The following program demonstrates the use of MSK_solvewithbasis.

```
/*
  Copyright: Copyright (c) 1998-2007 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.

               maximize  x0 + x1
               st.
                  x0 + 2.0 x1 -xc1       = 2
                  x0  +    x1       -xc2 = 6
                  x0 >= 0, x1>= 0,
                  xc1 <=  0 , xc2 <= 0


             problem data is read from basissolve.lp.

  Syntax:    solvebasis basissolve.lp

 */
#include "mosek.h"

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

int main(int argc,char **argv)
{
  MSKenv_t  env;
  MSKtask_t task;
  MSKintt NUMCON = 2;
  MSKintt NUMVAR = 2;

  double c[]    = {1.0, 1.0};
  MSKlidxt    ptrb[] = {0, 2};
  MSKlidxt    ptre[] = {2, 3};
  MSKlidxt    asub[] = {0, 1,
                        0, 1};
  double aval[] = {1.0, 1.0,
                   2.0, 1.0};
  MSKboundkeye bkc[]  = {MSK_BK_UP,
                       MSK_BK_UP};

  double blc[]  = {-MSK_INFINITY,
```

```
                  -MSK_INFINITY};
double buc[]  = {2.0,
                  6.0};

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

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


MSKrescodee       r = MSK_RES_OK;
MSKidxt       i,nz;
double    w1[] = {2.0,6.0};
double    w2[] = {1.0,0.0};
MSKidxt   sub[] = {0,1};
MSKidxt   *basis;

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

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

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

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

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

if ( r == MSK_RES_OK)
  r = MSK_inputdata(task, NUMCON,NUMVAR, NUMCON,NUMVAR, c, 0.0,
                    ptrb, ptre, asub, aval, bkc, blc, buc, bkx, blx, bux);

if (r == MSK_RES_OK)
  r = MSK_putobjsense(task,MSK_OBJECTIVE_SENSE_MAXIMIZE);



if (r == MSK_RES_OK)
  r = MSK_optimize(task);

if (r == MSK_RES_OK)
  basis = MSK_calloctask(task,NUMCON,sizeof(MSKidxt));

if (r == MSK_RES_OK)
  r = MSK_initbasissolve(task,basis);

/* List basis variables corresponding to columns of B */
for (i=0;i<NUMCON && r == MSK_RES_OK;++i)
{
  printf("basis[%d] = %d\n",i,basis[i]);
  if (basis[sub[i]] < NUMCON)
```

```
      printf ("Basis variable no %d is xc%d.\n",i, basis[i]);
    else
      printf ("Basis variable no %d is x%d.\n",i,basis[i] - NUMCON);
  }

  nz = 2;
  /* solve Bx = w1 */
  /* sub contains index of non-zeros in w1.
     On return w1 contains the solution x and sub
     the index of the non-zeros in x.
   */
  if (r == MSK_RES_OK)
    r = MSK_solvewithbasis(task,0,&nz,sub,w1);

  if (r == MSK_RES_OK)
  {
    printf("\nSolution to Bx = w1:\n\n");

    /* Print solution and b. */

    for (i=0;i<nz;++i)
    {
      if (basis[sub[i]] < NUMCON)
        printf ("xc%d = %e\n",basis[sub[i]] , w1[sub[i]] );
      else
        printf ("x%d = %e\n",basis[sub[i]] - NUMCON , w1[sub[i]] );
    }
  }
    /* Solve B^Tx = c */
  nz = 2;
  sub[0] = 0;
  sub[1] = 1;

  if (r == MSK_RES_OK)
    r = MSK_solvewithbasis(task,1,&nz,sub,w2);

  if (r == MSK_RES_OK)
  {
    printf("\nSolution to B^Tx = w2:\n\n");
    /* Print solution and y. */
    for (i=0;i<nz;++i)
    {
      if (basis[sub[i]] < NUMCON)
        printf ("xc%d = %e\n",basis[sub[i]] , w2[sub[i]] );
      else
        printf ("x%d = %e\n",basis[sub[i]] - NUMCON , w2[sub[i]] );
    }
  }

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

   return ( r );

}/* main */
```

In the example above the linear system is solved using the optimal basis for (6.16) 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:

$$B = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \tag{6.18}$$

It can be verified that

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

is a solution to

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

6.5.3 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

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{6.19}$$

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

```
/*
   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.

   File     :  solvelinear.c

   Purpose  :  To demonstrate the usage of MSK_solvewithbasis
               to solve the linear system:

                1.0  x1             = b1
               -1.0  x0  +  1.0  x1 = b2

               with two different right hand sides

               b = (1.0, -2.0)

               and

               b = (7.0, 0.0)
 */

#include "mosek.h"

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


MSKrescodee put_a(MSKtask_t task,
                   double *aval,
                   MSKidxt *asub,
                   MSKlidxt *ptrb,
                   MSKlidxt *ptre,
                   int numvar,
                  MSKidxt *basis
                  )

{
  MSKrescodee r = MSK_RES_OK;
  int i;
  MSKstakeye *skx = NULL , *skc = NULL;


  skx = (MSKstakeye *) calloc(numvar,sizeof(MSKstakeye));
  if (skx == NULL && numvar)
    r = MSK_RES_ERR_SPACE;

  skc = (MSKstakeye *) calloc(numvar,sizeof(MSKstakeye));
  if (skc == NULL && numvar)
    r = MSK_RES_ERR_SPACE;

  for (i=0;i<numvar && r == MSK_RES_OK;++i)
  {
    skx[i] = MSK_SK_BAS;
    skc[i] = MSK_SK_FIX;
  }
```

```
/* Create a coefficient matrix and right hand
   side with the data from the linear system */
if (r == MSK_RES_OK)
  r = MSK_append(task,MSK_ACC_VAR,numvar);

if (r == MSK_RES_OK)
  r = MSK_append(task,MSK_ACC_CON,numvar);

for (i=0;i<numvar && r == MSK_RES_OK;++i)
  r = MSK_putavec(task,MSK_ACC_VAR,i,ptre[i]-ptrb[i],asub+ptrb[i],aval+ptrb[i]);

for (i=0;i<numvar && r == MSK_RES_OK;++i)
  r = MSK_putbound(task,MSK_ACC_CON,i,MSK_BK_FX,0,0);

for (i=0;i<numvar && r == MSK_RES_OK;++i)
  r = MSK_putbound(task,MSK_ACC_VAR,i,MSK_BK_FR,-MSK_INFINITY,MSK_INFINITY);

/* Allocate space for the solution and set status to unknown */

if (r == MSK_RES_OK)
{
  r = MSK_makesolutionstatusunknown(task, MSK_SOL_BAS);
}

/* Define a basic solution by specifying
   status keys for variables & constraints. */
for (i=0; i<numvar && r==MSK_RES_OK;++i)
    r = MSK_putsolutioni (
                          task,
                          MSK_ACC_VAR,
                          i,
                          MSK_SOL_BAS,
                          skx[i],
                          0.0,
                          0.0,
                          0.0,
                          0.0);

for (i=0;i<numvar && r == MSK_RES_OK;++i)
    r = MSK_putsolutioni (
                          task,
                          MSK_ACC_CON,
                          i,
                          MSK_SOL_BAS,
                          skc[i],
                          0.0,
                          0.0,
                          0.0,
                          0.0);


if (r == MSK_RES_OK)
  r = MSK_initbasissolve(task,basis);

free (skx);
free (skc);
```

```
  return ( r );

}

#define NUMCON 2
#define NUMVAR 2


int main(int argc,char **argv)
{
  MSKenv_t  env;
  MSKtask_t task;
  MSKrescodee r = MSK_RES_OK;
  MSKintt   numvar = NUMCON;
  MSKintt   numcon = NUMVAR;   /* we must have numvar == numcon */
  int       i,nz;
  double    aval[] = {-1.0,1.0,1.0};
  MSKidxt   asub[] = {1,0,1};
  MSKlidxt  ptrb[] = {0,1};
  MSKlidxt  ptre[] = {1,3};

  MSKidxt   bsub[NUMCON];
  double    b[NUMCON];

  MSKidxt   *basis = NULL;

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

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

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

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

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

  basis = (MSKidxt *) calloc(numcon,sizeof(MSKidxt));
  if ( basis == NULL && numvar)
    r = MSK_RES_ERR_SPACE;


  /* Put A matrix and factor A.
     Call this function only once for a given task. */
  if (r == MSK_RES_OK)
    r = put_a( task,
               aval,
               asub,
               ptrb,
               ptre,
               numvar,
               basis
               );
```

```
  /* now solve rhs */
  b[0] = 1;
  b[1] = -2;
  bsub[0] = 0;
  bsub[1] = 1;
  nz = 2;

  if (r == MSK_RES_OK)
    r = MSK_solvewithbasis(task,0,&nz,bsub,b);

  if (r == MSK_RES_OK)
  {
    printf("\nSolution to Bx = b:\n\n");
    /* Print solution and show correspondents
       to original variables in the problem */
    for (i=0;i<nz;++i)
    {
      if (basis[bsub[i]] < numcon)
        printf("This should never happen\n");
      else
        printf ("x%d = %e\n",basis[bsub[i]] - numcon , b[bsub[i]] );
    }
  }

  b[0] = 7;
  bsub[0] = 0;
  nz = 1;

  if (r == MSK_RES_OK)
    r = MSK_solvewithbasis(task,0,&nz,bsub,b);

  if (r == MSK_RES_OK)
  {
    printf("\nSolution to Bx = b:\n\n");
    /* Print solution and show correspondents
       to original variables in the problem */
    for (i=0;i<nz;++i)
    {
      if (basis[bsub[i]] < numcon)
        printf("This should never happen\n");
      else
        printf ("x%d = %e\n",basis[bsub[i]] - numcon , b[bsub[i]] );
    }
  }

  free (basis);
  return r;
}
```

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 (6.19).

6.6 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.

6.6.1 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) 1998-2007 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,cputime=0.0,scputime=0.0,
         *maxtime=(double *) handle;

  switch ( caller )
  {
    case MSK_CALLBACK_BEGIN_INTPNT:
      printf("Starting interior-point optimizer\n");
      break;
    case MSK_CALLBACK_INTPNT:
      MSK_getintinf(task,
                    MSK_IINF_INTPNT_ITER,
                    &iter);
      MSK_getdouinf(task,
                    MSK_DINF_INTPNT_PRIMAL_OBJ,
                    &pobj);
      MSK_getdouinf(task,
                    MSK_DINF_INTPNT_DUAL_OBJ,
                    &dobj);
      MSK_getdouinf(task,
                    MSK_DINF_INTPNT_CPUTIME,
                    &scputime);
      MSK_getdouinf(task,
                    MSK_DINF_OPTIMIZER_CPUTIME,
                    &cputime);

      printf("Iterations: %-3d  Time: %6.2f(%.2f)  ",
             iter,cputime,scputime);
      printf("Primal obj.: %-18.6e  Dual obj.: %-18.6e\n",
              pobj,dobj);
      break;
    case MSK_CALLBACK_END_INTPNT:
      printf("Interior-point optimizer finished.\n");
      break;
    case MSK_CALLBACK_BEGIN_PRIMAL_SIMPLEX:
      printf("Primal simplex optimizer started.\n");
      break;
    case MSK_CALLBACK_UPDATE_PRIMAL_SIMPLEX:
      MSK_getintinf(task,
                    MSK_IINF_SIM_PRIMAL_ITER,
                    &iter);
      MSK_getdouinf(task,
                    MSK_DINF_SIM_OBJ,
                    &pobj);
      MSK_getdouinf(task,
```

```
                    MSK_DINF_SIM_CPUTIME,
                    &scputime);
      MSK_getdouinf(task,
                    MSK_DINF_OPTIMIZER_CPUTIME,
                    &cputime);

      printf("Iterations: %-3d  ",iter);
      printf("  Elapsed time: %6.2f(%.2f)\n",
             cputime,scputime);
      printf("Obj.: %-18.6e\n",pobj);
      break;
    case MSK_CALLBACK_END_PRIMAL_SIMPLEX:
      printf("Primal simplex optimizer finished.\n");
      break;
    case MSK_CALLBACK_BEGIN_DUAL_SIMPLEX:
      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_CPUTIME,
                    &scputime);
      MSK_getdouinf(task,
                    MSK_DINF_OPTIMIZER_CPUTIME,
                    &cputime);

      printf("Iterations: %-3d  ",iter);
      printf("  Elapsed time: %6.2f(%.2f)\n",
             cputime,scputime);
      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 ( cputime>=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 )
  {
    printf("Too few input arguments\n");
    exit(0);
  }

  /*
   * It is assumed that we are working in a
   * windows environment.
   */

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

  /* Check the return code. */
  if ( r==MSK_RES_OK )
    r = MSK_initenv(env);

  /* Check the return code. */
  if ( r==MSK_RES_OK )
  {
    /* Create an (empty) optimization task. */
    r = MSK_makeemptytask(env,&task);

    if ( r==MSK_RES_OK )
    {
      MSK_linkfunctotaskstream(task,MSK_STREAM_MSG,NULL, printtxt);
      MSK_linkfunctotaskstream(task,MSK_STREAM_ERR,NULL, printtxt);
    }

    /* Specifies that data should be read from the
       file argv[2].
     */

    if ( r==MSK_RES_OK )
      r = MSK_readdata(task,argv[2]);

    if ( r==MSK_RES_OK )
    {
      if ( 0==strcmp(argv[1],"psim") )
        MSK_putintparam(task,MSK_IPAR_OPTIMIZER,MSK_OPTIMIZER_PRIMAL_SIMPLEX);
      else  if ( 0==strcmp(argv[1],"dsim") )
        MSK_putintparam(task,MSK_IPAR_OPTIMIZER,MSK_OPTIMIZER_DUAL_SIMPLEX);
      else  if ( 0==strcmp(argv[1],"intpnt") )
```

```
        MSK_putintparam(task,MSK_IPAR_OPTIMIZER,MSK_OPTIMIZER_INTPNT);


      /* Tell mosek about the call-back function. */
      maxtime = 3600;
      MSK_putcallbackfunc(task,
                          usercallback,
                          (void *) &maxtime);

      /* Turn all MOSEK logging off. */
      MSK_putintparam(task,
                      MSK_IPAR_LOG,
                      0);

      r = MSK_optimize(task);

      MSK_solutionsummary(task,MSK_STREAM_MSG);
    }


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

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

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

6.7 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) 1998-2007 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);
  }

  return ( MSK_RES_OK );

} /* handlerespone */


int main(int argc, char *argv[])
{
  MSKenv_t    env;
  MSKrescodee r;
  MSKtask_t   task;

  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 )
    {
      /*
       * Input a custom warning and error handler function.
       */

      MSK_putresponsefunc(task,handleresponse,NULL);

      /* User defined code goes here */
      /* This will provoke an error */

      if (r == MSK_RES_OK)
        r = MSK_putaij(task,10,10,1.0);

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

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

```
  if (r == MSK_RES_ERR_INDEX_IS_TOO_LARGE)
    return ( MSK_RES_OK);
  else
    return (-1);
} /* 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
```

6.8 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.

6.8.1 A source code example

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

```
/*
   Copyright: Copyright (c) 1998-2007 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 )
    {
      /*
         The wchar_t string "input" specifying a file name
         is converted to a UTF8 string that can be inputted
         to MOSEK.
        */

      r = MSK_wchartoutf8(sizeof(output),&len,&conv,output,input);

      if ( r==MSK_RES_OK )
      {
        /* output is now an UTF8 encoded string. */
        r = MSK_readdata(task,output);
      }

      if ( r==MSK_RES_OK )
      {
        r = MSK_optimize(task);
        MSK_solutionsummary(task,MSK_STREAM_MSG);
      }
    }
    MSK_deletetask(&task);
  }
  MSK_deleteenv(&env);

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

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

6.8.2 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.
- $x \in R^n$ is a vector of decision variables.
- $c \in R^n$ is the linear part of the objective function.
- $A \in R^{m \times n}$ is the constraint matrix.

- $l^c \in R^m$ is the lower limit[1] on the activity for the constraints.
- $u^c \in R^m$ is the upper limit on the activity for the constraints.
- $l^x \in R^n$ is the lower limit on the activity for the variables.
- $u^x \in R^n$ 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

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

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. For example

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

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 primal-dual* solution.

[1] We will use the words “bound” and “limit” interchangeably.

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 such that the duality gap is zero, or, equivalently, that the *complementarity conditions*

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

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 primal certificate of the infeasibility: The dual solution reported is a certificate of infeasibility, and the primal solution is undefined.

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

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

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 whose 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 dual certificate of the 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

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

where

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

and

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

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

We note that the dual of (7.4) is a problem whose 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 Linear network flow problems

Network flow problems are a special class of linear optimization problems which has many applications. The class of network flow problems can be specified as follows. Let $G = (\mathcal{N}, \mathcal{A})$ be a directed network. Associated with every arc $(i, j) \in \mathcal{A}$ is a cost c_{ij} and a capacity $[l^x_{ij}, u^x_{ij}]$. Moreover, associated with each node $i \in \mathcal{N}$ in the network is a lower limit l^c_i and an upper limit u^c_i on the demand(supply) of the node. Now the minimum cost of a network flow problem can be stated as follows

$$
\begin{array}{lccccl}
\text{minimize} & & \displaystyle\sum_{(i,j)\in\mathcal{A}} c_{ij} x_{ij} & & & \\
\text{subject to} & l^c_i \leq & \displaystyle\sum_{\{j:(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j:(j,i)\in\mathcal{A}\}} x_{ji} & \leq & u^c_i & \forall i \in \mathcal{N}, \\
& l^x_{ij} \leq & x_{ij} & \leq & u^x_{ij} & \forall (i,j) \in \mathcal{A}.
\end{array} \tag{7.5}
$$

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.

It is well-known that problems with network flow structure can be solved efficiently with a specialized version of the simplex method. MOSEK includes a highly tuned network simplex implementation, see Section 8.3.1 for further details on how to invoke the network optimizer.

7.3 Quadratic and quadratically constrained optimization

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

$$
\begin{array}{lccccl}
\text{minimize} & & \frac{1}{2} x^T Q^o x + c^T x + c^f & & & \\
\text{subject to} & l^c_k \leq & \frac{1}{2} x^T Q^k x + \displaystyle\sum_{j=0}^{n-1} a_{k,i} x_j & \leq & u^c_k, & k = 0, \ldots, m-1, \\
& l^x \leq & x & \leq & u^x, & j = 0, \ldots, n-1,
\end{array} \tag{7.6}
$$

where the convexity requirement implies that

- Q^o is a symmetric positive semi-definite matrix.
- If $l^c_k = -\infty$, then Q^k is a symmetric positive semi-definite matrix.
- If $u^c_k = \infty$, 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.

7.3.1 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 7.4.4.1 for further details.

7.3.2 Reformulating as a separable quadratic problem

The simplest quadratic optimization problem is

$$\begin{array}{lccl} \text{minimize} & 1/2x^TQx + c^Tx & & \\ \text{subject to} & Ax & = & b, \\ & x \geq 0. & & \end{array} \tag{7.7}$$

The problem (7.7) is said to be a separable problem if Q is a diagonal matrix or, in other words, that 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

$$Q = F^T F. \tag{7.8}$$

In many practical applications of quadratic optimization F is known explicitly; for example if Q is a covariance matrix, F would be the set of observations producing it.

Using (7.8), the problem (7.7) can be reformulated as

$$\begin{array}{lccl} \text{minimize} & 1/2y^TIy + c^Tx & & \\ \text{subject to} & Ax & = & b, \\ & Fx - y & = & 0, \\ & x \geq 0. & & \end{array} \tag{7.9}$$

The problem (7.9) is also a quadratic optimization problem and has more constraints and variables than (7.7). 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 dense matrix.

The idea presented above is applicable to quadratic constraints too. Now consider the constraint

$$1/2x^T(F^TF)x \leq b \tag{7.10}$$

where F is a matrix and b is a scalar. (7.10) can be reformulated as

$$\begin{array}{rcl} 1/2y^T I y & \leq & b, \\ Fx - y & = & 0. \end{array}$$

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, i.e. D is for instance a diagonal or a block diagonal matrix. If this is the case, then it may be worthwhile performing the reformulation

$$\begin{array}{rcl} 1/2((x^T D x) + y^T I y) & \leq & b, \\ Fx - y & = & 0. \end{array}$$

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.

7.4 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

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x \in \mathcal{C}. & & \end{array} \tag{7.11}$$

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 $x_t \in R^{n_t}$. Please note that the n-dimensional Euclidean space 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

- R set:

$$\mathcal{C}_t = \{x \in R^{n^t}\}.$$

- Quadratic cone:

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

- Rotated quadratic cone:

$$\mathcal{C}_t = \left\{ x \in R^{n^t} : 2x_1 x_2 \geq \sum_{j=3}^{n^t} x_j^2,\ x_1, x_2 \geq 0 \right\}.$$

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 7.4.4.

7.4.1 Duality for conic optimization

The dual problem corresponding to the conic optimization problem (7.11) is given by

$$\begin{array}{lccc} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x + s_n^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, \\ & s_n^x \in \mathcal{C}^* & & \end{array} \tag{7.12}$$

where the dual cone $\mathcal{C}^*$ is a product of 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:

- R set:

$$\mathcal{C}_t = \left\{ x \in R^{n^t} \right\} \quad \Leftrightarrow \quad \mathcal{C}_t^* := \left\{ s \in R^{n^t} :\ s = 0 \right\}.$$

- Quadratic cone:

$$\mathcal{C}_t := \left\{ x \in R^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\} \quad \Leftrightarrow \quad \mathcal{C}_t^* = \mathcal{C}_t.$$

- Rotated quadratic cone:

$$\mathcal{C}_t := \left\{ x \in R^{n^t} : 2x_1 x_2 \geq \sum_{j=3}^{n^t} x_j^2,\ x_1, x_2 \geq 0 \right\}. \quad \Leftrightarrow \quad \mathcal{C}_t^* = \mathcal{C}_t.$$

7.4.2 The dual of the dual

The dual problem corresponding to the dual problem is the primal problem.

7.4.3 Infeasibility

In case MOSEK finds a problem to be infeasible it will report a certificate of the infeasibility. This works exactly as for linear problems (see sections 7.1.1.3 and 7.1.1.4).

7.4.4 Examples

This section contains several examples of inequalities and problems that can be cast as conic optimization problems.

7.4.4.1 Quadratic objective and constraints

From Section 7.3.2 we know that any convex quadratic problem can be stated on the form

$$\begin{array}{lrcl} \text{minimize} & 0.5\left\|Fx\right\|^2 + c^T x, & & \\ \text{subject to} & 0.5\left\|Gx\right\|^2 + a^T x & \leq & b, \end{array} \tag{7.13}$$

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 (7.13) can be reformulated as

$$\begin{array}{lrcl} \text{minimize} & 0.5\left\|t\right\|^2 + c^T x, & & \\ \text{subject to} & 0.5\left\|z\right\|^2 + a^T x & \leq & b, \\ & Fx - t & = & 0, \\ & Gx - z & = & 0 \end{array} \tag{7.14}$$

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.

$$\begin{array}{lrcll} \text{minimize} & v + c^T x, & & & \\ \text{subject to} & p + a^T x & = & b, & \\ & Fx - t & = & 0, & \\ & Gx - z & = & 0, & \\ & w & = & 1, & \\ & q & = & 1, & \\ & \left\|t\right\|^2 & \leq & 2vw, & v, w \geq 0, \\ & \left\|z\right\|^2 & \leq & 2pq, & p, q \geq 0. \end{array} \tag{7.15}$$

In this case we can model the last two inequalities using rotated quadratic cones.

If we assume that F is a non-singular matrix — for instance a diagonal matrix — then

$$x = F^{-1}t.$$

and hence we can eliminate x from the problem to obtain:

$$\begin{array}{lrcll} \text{minimize} & v + c^T F^{-1} t, & & & \\ \text{subject to} & p + a^T F^{-1} t & = & b, & \\ & V F^{-1} t - z & = & 0, & \\ & w & = & 1, & \\ & q & = & 1, & \\ & \|t\|^2 & \leq & 2vw, & v, w \geq 0, \\ & \|z\|^2 & \leq & 2pq, & p, q \geq 0. \end{array} \tag{7.16}$$

In most cases MOSEK will perform this reduction automatically during the presolve phase before the optimization is performed.

7.4.4.2 Minimizing a sum of norms

The next example is the problem of minimizing a sum of norms i.e. the problem

$$\begin{array}{lccc} \text{minimize} & \sum_{i=1}^{k} \left\| x^i \right\| & & \\ \text{subject to} & Ax & = & b, \end{array} \tag{7.17}$$

where

$$x := \left[\begin{array}{c} x^1 \\ \vdots \\ x^k \end{array} \right].$$

This problem is equivalent to

$$\begin{array}{lcccl} \text{minimize} & \sum_{i=1}^{k} z_i & & & \\ \text{subject to} & Ax & = & b, & \\ & \left\| x^i \right\| & \leq & z_i, & i = 1, \ldots, k, \end{array} \tag{7.18}$$

which in turn is equivalent to

$$\begin{array}{lccl} \text{minimize} & \sum_{i=1}^{k} z_i & & \\ \text{subject to} & Ax & = & b, \\ & (z_i, x^i) \in \mathcal{C}_i, & & i = 1, \ldots, k \end{array} \tag{7.19}$$

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 (7.19) is

$$\begin{array}{lccll} \text{maximize} & b^T y & & & \\ \text{subject to} & A^T y + s & = & c, & \\ & t_i & = & 1, & i = 1, \ldots, k, \\ & (t_i, s^i) \in \mathcal{C}_i, & & & i = 1, \ldots, k \end{array} \tag{7.20}$$

where

$$s := \begin{bmatrix} s^1 \\ \vdots \\ s^k \end{bmatrix}.$$

This problem is equivalent to

$$\begin{array}{lcccl} \text{maximize} & b^T y & & & \\ \text{subject to} & A^T y + s & = & c, & \\ & \left\| s^i \right\|_2^2 & \leq & 1, & i = 1, \ldots, k. \end{array} \tag{7.21}$$

Please note that the dual problem can be reduced to an “ordinary” convex quadratically constrained optimization problem in this case 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 (7.20) rather than the primal problem (7.19).

7.4.4.3 Modelling polynomial terms using conic optimization

Generally an arbitrary polynomial term of the form

$$f x^g$$

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.

For example, consider the problem

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & \sum_{j=1}^{n} \frac{f_j}{x_j} \quad \leq \quad b, \\ & x \geq 0, \end{array}$$

where it is assumed that $f_j > 0$ and $b > 0$. This problem is equivalent to

$$\begin{array}{lllll} \text{minimize} & c^T x & & & \\ \text{subject to} & \sum_{j=1}^{n} z_j & = & b, & \\ & v_j & = & \sqrt{2}, & j = 1, \ldots, n, \\ & v_j^2 & \leq & 2 z_j x_j, & j = 1, \ldots, n, \\ & x, z \geq 0, & & & \end{array} \tag{7.22}$$

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 (7.22) 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

$$\begin{array}{rcl} t^2 & \leq & 2xz, \\ z & = & 0.5, \\ x, z, & \geq & 0. \end{array} \tag{7.23}$$

Occasionally when modelling the *market impact* term in portfolio optimization, the polynomial term $x^{\frac{3}{2}}$ occurs. Therefore, consider the set defined by the inequalities

$$\begin{array}{rcl} x^{1.5} & \leq & t, \\ 0 & \leq & x. \end{array} \tag{7.24}$$

We will exploit that $x^{1.5} = x^2/\sqrt{x}$. First define the set

$$\begin{array}{rcl} x^2 & \leq & 2st, \\ s, t & \geq & 0. \end{array} \tag{7.25}$$

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

$$\begin{array}{rcl} x^2 & \leq & 2st, \\ w^2 & \leq & 2vr, \\ x & = & v, \\ s & = & w, \\ r & = & \frac{1}{8}, \\ s,t,v,r & \geq & 0, \end{array} \tag{7.26}$$

then

$$\begin{array}{rcl} s^2 & = & w^2 \\ & \leq & 2vr \\ & \leq & \frac{v}{4} \\ & = & \frac{x}{4}. \end{array}$$

Moreover,

$$\begin{array}{rcl} x^2 & \leq & 2st, \\ & \leq & 2\sqrt{\frac{x}{4}}t \end{array}$$

leading to the conclusion

$$x^{1.5} \leq t.$$

(7.26) is a conic reformulation which is equivalent to (7.24). Please note that the $x \geq 0$ constraint does not appear explicitly in (7.25) and (7.26), but implicitly since $x = v \geq 0$.

Finally, it should be mentioned that any polynomial term of form x^g where g is a positive rational number can be represented using conic quadratic constraints [3, pp. 12-13]

7.4.4.4 Further reading

If you want to know more about what can be modelled as a conic optimization problem we recommend the references [18, 13, 3].

7.4.5 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.

7.4.5.1 Non-attainment in the primal problem

Consider the example

$$\begin{array}{lrcl} \text{minimize} & z & & \\ \text{subject to} & 2yz & \geq & x^2, \\ & x & = & \sqrt{2}, \\ & y, z & \geq & 0. \end{array} \tag{7.27}$$

which corresponds to the problem

$$\begin{array}{lrcl} \text{minimize} & \frac{1}{y} & & \\ \text{subject to} & y & \geq & 0. \end{array} \tag{7.28}$$

Clearly, the optimal objective value is zero but it is never attained because implicitly we assume that the optimal y should be finite.

7.4.5.2 Non-attainment in the dual problem

Next, consider the example

$$\begin{array}{lrcl} \text{minimize} & x_4 & & \\ \text{subject to} & x_3 + x_4 & = & 1, \\ & x_1 & = & 0, \\ & x_2 & = & 1, \\ & 2x_1x_2 & \geq & x_3^2, \\ & x_1x_2 & \geq & 0. \end{array} \tag{7.29}$$

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 (7.29) is

$$\begin{array}{lrcl} \text{maximize} & y_1 + y_3 & & \\ \text{subject to} & y_2 + s_1 & = & 0, \\ & y_3 + s_2 & = & 0, \\ & y_1 + s_3 & = & 0, \\ & y_1 & = & 1, \\ & 2s_1s_2 & \geq & s_3^2, \\ & s_1s_2 & \geq & 0. \end{array} \tag{7.30}$$

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, there no primal and dual bounded optimal solution that has zero duality gap exists. Of course it is possible to find a primal and dual feasible solution such that the duality gap is close to zero, however, s_1^* will be very large (unless a large duality gap is allowed). This is likely to make the problem (7.29) hard to solve.

An inspection of problem (7.29) 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 (7.29) or eliminate x_1 and x_3 from the problem it becomes easy to solve.

7.5 Nonlinear convex optimization

MOSEK is capable of solving smooth convex nonlinear optimization problems of the form

$$\begin{array}{lccccc} \text{minimize} & & & f(x) + c^T x & & \\ \text{subject to} & & & g(x) + Ax - x^c & = & 0, \\ & l^c & \leq & x^c & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \tag{7.31}$$

where

- m is the number of constraints.
- n is the number of decision variables.
- $x \in R^n$ is a vector of decision variables.
- $x^c \in R^m$ is a vector of constraints or slack variables.
- $c \in R^n$ is the linear part objective function.
- $A \in R^{m \times n}$ is the constraint matrix.
- $l^c \in R^m$ is the lower limit[2] on the activity for the constraints.
- $u^c \in R^m$ is the upper limit on the activity for the constraints.
- $l^x \in R^n$ is the lower limit on the activity for the variables.
- $u^x \in R^n$ is the upper limit on the activity for the variables.
- $f : R^n \rightarrow R$ is a nonlinear function.

[2] We will use the words “bound” and “limit” interchangeably.

- $g : R^n \rightarrow R^m$ 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 (twice differentiable) in all $x \in [l^x; u^x]$. Moreover, $f(x)$ must be a convex function and $g_i(x)$ must satisfy

$$\begin{array}{rcll} l_i^c = -\infty & \Rightarrow & g_i(x) & \text{is convex}, \\ u_i^c = \infty & \Rightarrow & g_i(x) & \text{is concave}, \\ -\infty < l_i^c \leq u_i^c < \infty & \Rightarrow & g_i(x) = 0. & \end{array}$$

7.5.1 Duality

So far, we have not discussed what happens when MOSEK is used to solve a primal or dual infeasible problem. In the subsequent 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

$$\begin{array}{rcl} L(x^c, x, y, s_l^c, s_u^c, s_l^x, s_u^x) & := & f(x) + c^T x + c^f \\ & & -y^T (Ax + g(x) - x^c) \\ & & -(s_l^c)^T (x^c - l^c) - (s_u^c)^T (u^c - x^c) \\ & & -(s_l^x)^T (x - l^x) - (s_u^x)^T (u^x - x). \end{array}$$

and the dual problem is given by

$$\begin{array}{lccc} \text{maximize} & L(x^c, x, y, s_l^c, s_u^c, s_l^x, s_u^x) & & \\ \text{subject to} & \nabla_{(x^c,x)} L(x^c, x, y, s_l^c, s_u^c, s_l^x, s_u^x) & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & \end{array}$$

which is equivalent to

$$\begin{array}{lccc} \text{maximize} & f(x) - y^T g(x) - x^T (\nabla f(x)^T - \nabla g(x)^T y) & & \\ & +((l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & -\nabla f(x)^T + A^T y + \nabla g(x)^T y + s_l^x - s_u^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & \end{array} \tag{7.32}$$

7.6 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 leading to the introduction of 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 are either 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 might be a good idea to introduce more constraints and variables if it makes the model separable. Also a big but sparse problem might 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].

7.6.1 Avoid nearly infeasible models

Consider the linear optimization problem

$$\begin{array}{lrcl} \text{minimize} & & & \\ \text{subject to} & x + y & \leq & 10^{-10} + \alpha, \\ & 1.0e4x + 2.0e4y & \geq & 10^{-6}, \\ & x, y \geq 0. & & \end{array} \tag{7.33}$$

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.

7.7 Examples continued

7.7.1 The absolute value

Assume we have a constraint for the form

$$|f^T x + g| \leq b \tag{7.34}$$

where $x \in R^n$ is a vector of variables, and $f \in R^n$ and $g, b \in R$ are constants.

It is easy to verify that the constraint (7.34) is equivalent to

$$-b \leq f^T x + g - t \leq b \tag{7.35}$$

which is a set of ordinary linear inequality constraints.

Please note that equalities involving and absolute value such as

$$|x| = 1$$

cannot be formulated as a linear or even a convex optimization problem. It requires integer optimization.

7.7.2 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 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.

7.7.2.1 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

$$\begin{array}{lrcl} \text{minimize} & V[r^T(w^0+x)] & & \\ \text{subject to} & E[r^T(w^0+x)] & = & t, \\ & e^Tx & = & 0, \end{array} \tag{7.36}$$

where $e := (1,\ldots,1)^T$. Using the definitions above this may be formulated as a quadratic optimization problem:

$$\begin{array}{lrcl} \text{minimize} & (w^0+x)^T\Sigma(w^0+x) & & \\ \text{subject to} & \bar{r}^T(w^0+x) & = & t, \\ & e^Tx & = & 0, \end{array} \tag{7.37}$$

7.7.2.2 Conic quadratic reformulation.

An equivalent conic quadratic reformulation is given by:

$$\begin{array}{lrcl} \text{minimize} & f & & \\ \text{subject to} & \Sigma^{\frac{1}{2}}(w^0+x)-g & = & 0, \\ & \bar{r}^T(w^0+x) & = & t, \\ & e^Tx & = & 0, \\ & f \geq \|g\|. & & \end{array} \tag{7.38}$$

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.

7.7.2.3 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 modelled as follows

$$\text{commission} + \frac{\text{bid}}{\text{ask}} - \text{spread} + \theta\sqrt{\frac{\text{trade volume}}{\text{daily volume}}}, \tag{7.39}$$

and that these are important to incorporate 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:

$$\begin{array}{lccc} \text{minimize} & f & & \\ \text{subject to} & \Sigma^{\frac{1}{2}}(w^0+x)-g & = & 0, \\ & \bar{r}^T(w^0+x) & = & t, \\ & e^Tx+e^Ty & = & 0, \\ & |x_j|(m_j|x_j|^{\frac{1}{2}}) & \leq & y_j, \\ & f \geq \|g\|. & & \end{array} \tag{7.40}$$

Now, defining the variable transformation

$$y_j = m_j\bar{y}_j$$

we obtain

$$\begin{array}{lccc} \text{minimize} & f & & \\ \text{subject to} & \Sigma^{\frac{1}{2}}(w^0+x)-g & = & 0, \\ & \bar{r}^T(w^0+x) & = & t, \\ & e^Tx+m^T\bar{y} & = & 0, \\ & |x_j|^{3/2} & \leq & \bar{y}_j, \\ & f \geq \|g\|. & & \end{array} \tag{7.41}$$

As shown in Section 7.4.4.3 the set

$$|x_j|^{3/2} \leq \bar{y}_j$$

can be modelled by

$$
\begin{array}{rcl}
x_j & \leq & z_j, \\
-x_j & \leq & z_j, \\
z_j^2 & \leq & 2s_j\bar{y}_j, \\
u_j^2 & \leq & 2v_jq_j, \\
z_j & = & v_j, \\
s_j & = & u_j, \\
q_j & = & \frac{1}{8}, \\
q_j, s_j, \bar{y}_j, v_j, q_j & \geq & 0.
\end{array} \tag{7.42}
$$

7.7.2.4 Further reading

For further reading please see the reader to [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: Solving the actual optimization.

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 normally 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 the presolve is known to be unable to reduce the size of a problem significantly, then turning off the presolve is beneficial. 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 usually

- the eliminator, and
- the linear dependency check.

Therefore, in some cases it is worthwhile to disable one or both of these.

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. By implied free variable is meant that the constraint $y \geq 0$ is redundant and hence y can be treated as a free variable.

For large scale problems the eliminator usually removes many constraints and variables. However, in some cases few or no eliminations can be performed and moreover, the eliminator may consume a lot of memory and time. If this is the case it is worthwhile to disable the eliminator by setting the parameter `MSK_IPAR_PRESOLVE_ELIMINATOR_USE` to `MSK_OFF`.

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, i.e. one of the constraints is redundant. 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, and therefore it is strongly recommended to build models without linear dependencies. In case the linear dependencies are removed at the modelling 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

It is well-known that all linear, conic, and convex optimization problems have an associated dual problem. Moreover, even if the dual instead of the primal problem is solved, it is possible to recover the solution to the original primal problem.

In general, it is very hard to say whether it is easier to solve the primal or the dual problem but MOSEK has some heuristics for deciding which of the two problems to solve. Which form of the problem (primal or dual) that is solved is displayed in the MOSEK log. Please note that the dualizer is transparent, and all solution values returned by the optimizer refer to the original primal problem.

The dualizer can be controlled 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.

Finally, please 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 might 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

For linear optimization problems two different types of optimizers are available. The default for linear problems is an interior-point optimizer, however, as an alternative the simplex optimizer can be employed.

The curious reader can consult [24] for a discussion about interior-point and simplex algorithms.

8.2.2 The interior-point optimizer

The MOSEK interior-point optimizer is an implementation of the homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see [10].

8.2.2.1 Basis identification

It is well-known that 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 is 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.

Parameter name	Purpose
MSK_DPAR_INTPNT_TOL_PFEAS	Controls primal feasibility.
MSK_DPAR_INTPNT_TOL_DFEAS	Controls dual feasibility.
MSK_DPAR_INTPNT_TOL_REL_GAP	Controls relative gap.
MSK_DPAR_INTPNT_TOL_INFEAS	Controls when the problem is declared primal or dual infeasible.
MSK_DPAR_INTPNT_TOL_MU_RED	Controls when the complementarity is reduced enough.

Table 8.1: Parameters employed in termination criterion.

8.2.2.2 Interior-point termination criterion

The parameters in Table 8.1 control when the interior-point optimizer terminates.

8.2.3 The simplex based optimizer

An alternative to the interior-point optimizer is the simplex optimizer. The simplex optimizer employs a different approach than the interior-point optimizer when solving a problem. Contrary to the interior-point optimizer the simplex optimizer can exploit a guess for the optimal solution to reduce solution time. Depending on the problem it may be faster or slower to exploit a guess for the optimal solution. 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 given the 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

MOSEK is designed to minimize numerical difficulties, however, in rate cases the optimizer may have a hard time solving a problem. MOSEK counts a numerical unexpected behavior inside the optimizer as a “set-back”. The user can define how many set-backs the optimizer accepts, and 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.

What counts as a set-back? It is hard to say without getting very technical but obvious cases are repeated singularities when factorizing the basis matrix, repeated loss of feasibility, degeneracy problems (no progress in objective) or 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. If a lot of set-backs still occur, then 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 simply impossible, however, in recent years the dual optimizer has experienced 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 specified in Section 7.2 can in most cases be solved significantly faster with a specialized version of the simplex method [2], rather than with the general solvers.

MOSEK includes a network simplex solver, which usually solves network problems 10 to 100 times faster than the standard simplex optimizers implemented by MOSEK.

To use the network simplex optimizer, do the following

- Input the network flow problem as an ordinary linear optimization problem.
- Set
  - the MSK_IPAR_SIM_NETWORK_DETECT parameter to 0, and
  - the MSK_IPAR_OPTIMIZER parameter to MSK_OPTIMIZER_FREE_SIMPLEX.
- Optimize the problem.

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 above is applied, MOSEK will try to exploit this structure to speed up the optimization.

This is done by heuristically detecting the largest network embedded in the problem, solving this using the network simplex optimizer, and using this solution to hot-start a normal simplex optimizer.

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.

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.

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 only 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 Interior-point termination criteria

The parameters controlling when the general convex interior-point optimizer terminates are shown in Table 8.3.

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 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.

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

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.

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.

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.

```
/*
   Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.

   File:      concurrent1.c

   Purpose:   To demonstrate how to solve a problem
              with the concurrent optimizer.
```

```
 */

#include <stdio.h>

#include "mosek.h"

static void MSKAPI printstr(void *handle,
                            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) 1998-2007 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;
}
```

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

$$\begin{array}{lllllll} z^* & = & \text{minimize} & & c^T x & & \\ & & \text{subject to} & l^c \leq & Ax & \leq & u^c, \\ & & & l^x \leq & Ax & \leq & u^x, \\ & & & & x_j \in \mathcal{Z}, & & \forall j \in \mathcal{J}, \end{array} \tag{9.1}$$

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

$$\begin{array}{rclccccc} \underline{z} & = & \text{minimize} & & & c^T x & & \\ & & \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & & & l^x & \leq & Ax & \leq & u^x \end{array} \tag{9.2}$$

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

$$\min(|x_j| - \lfloor x_j \rfloor, \lceil x_j \rceil - |x_j|) \le \max(\delta_1, \delta_2 |x_j|) \ \forall j \in \mathcal{J}$$

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

$$\bar{z} - \underline{z} \le \max(\delta_3, \delta_4 \max(1, |\bar{z}|))$$

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.

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].

Chapter 10

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.

10.1 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

$$2300 = 1100 + 200 + 500 + 500 \tag{10.1}$$

exceeds the total supply

$$2200 = 200 + 1000 + 1000 \tag{10.2}$$

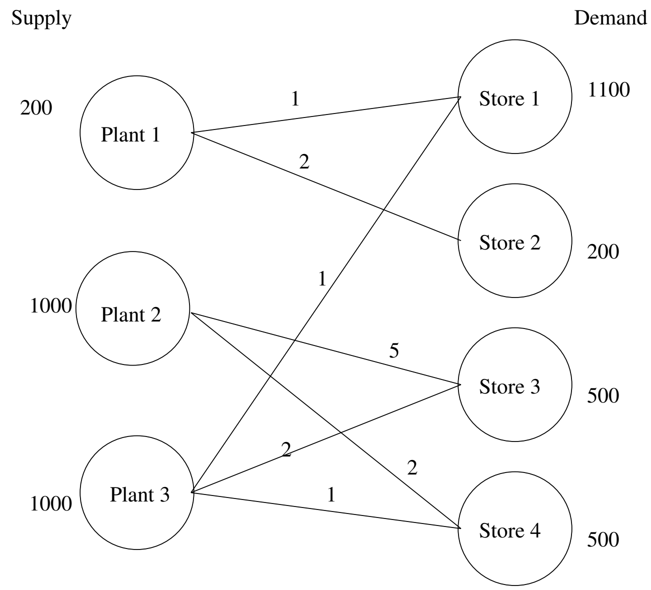

Figure 10.1: Supply, demand and cost of transportation.

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:

$$\begin{array}{llcccccccccccccll}
\text{minimize} & x_{11} & + & 2x_{12} & + & 5x_{23} & + & 2x_{24} & + & x_{31} & + & 2x_{33} & + & x_{34} & & \\
\text{subject to} & x_{11} & + & x_{12} & & & & & & & & & & & \leq & 200, \\
& & & & & x_{23} & + & x_{24} & & & & & & & \leq & 1000, \\
& & & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\
& x_{11} & & & & & & & + & x_{31} & & & & & = & 1100, \\
& & & x_{12} & & & & & & & & & & & = & 200, \\
& & & & & x_{23} & + & & & & & x_{33} & & & = & 500, \\
& & & & & & & x_{24} & + & & & & & x_{34} & = & 500, \\
& x_{ij} \geq 0. & & & & & & & & & & & & & &
\end{array} \tag{10.3}$$

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.

10.1.1 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 10.1.3) 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.

10.1.2 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.

10.1.2.1 A cautious note

The problem

$$\begin{array}{lll} \text{minimize} & 0 & \\ \text{subject to} & 0 \leq x_1, & \\ & x_j \leq x_{j+1}, & j = 1, \ldots, n-1, \\ & x_n \leq -1 & \end{array} \tag{10.6}$$

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.

10.1.3 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.

10.1.3.1 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 executing the command

```
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. Please note that this optimization 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**. However, since the reduced problem is usually smaller, it should be easier to locate the cause of the infeasibility in this rather than in the original problem (10.3).

10.1.3.2 Example: Dual infeasibility

The example problem

```
minimize -  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):

```
he 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 might 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 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

$$\begin{array}{lccccl} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x \end{array} \tag{10.8}$$

where the corresponding dual problem is

$$\begin{array}{lccc} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & \end{array} \tag{10.9}$$

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

$$l_j^x = -\infty \ \Rightarrow \ (s_l^x)_j = 0 \tag{10.10}$$

10.2.1 Certificat of primal infeasibility

A certificate of primal infeasibility is *any* solution to the homogenized dual problem

$$\begin{array}{lccc} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & 0, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & \end{array} \tag{10.11}$$

with a positive objective value. That is, $(s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*})$ is a certificat 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

$$\begin{array}{lcc} A^T y + s_l^{x*} - s_u^{x*} & = & 0, \\ -y + s_l^{c*} - s_u^{c*} & = & 0, \\ s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*} \geq 0. & & \end{array} \tag{10.13}$$

The well-known Farkas Lemma tells us that (10.8) is infeasible if and only if a certificat 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.

10.2.2 Certificat 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

$$\bar{l}_i^c := \left\{ \begin{array}{ll} 0, & l_i^c > -\infty, \\ -\infty, & \text{otherwise}, \end{array} \right. \qquad \bar{u}_i^c := \left\{ \begin{array}{ll} 0, & u_i^c < \infty, \\ \infty, & \text{otherwise}, \end{array} \right. \tag{10.17}$$

and

$$\bar{l}_i^x := \left\{ \begin{array}{ll} 0, & l_i^x > -\infty, \\ -\infty, & \text{otherwise}, \end{array} \right. \quad \text{and } \bar{u}_i^x := \left\{ \begin{array}{ll} 0, & u_i^x < \infty, \\ \infty, & \text{otherwise}. \end{array} \right. \tag{10.18}$$

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 certificat of dual infeasibility exists.

Observe 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 10.1.1 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

$$\begin{array}{lcccl} \text{minimize} & & c^T x + c^f & & \\ \text{subject to} & l^c \leq & Ax & \leq & u^c, \\ & l^x \leq & x & \leq & u^x, \end{array} \tag{11.1}$$

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:

- $w_l^c \in R^m$ (associated with l^c),
- $w_u^c \in R^m$ (associated with u^c),
- $w_l^x \in R^n$ (associated with l^x),
- $w_u^x \in R^n$ (associated with u^x),

Now, the problem

$$\begin{array}{lccccc} \text{minimize} & & & p & & \\ \text{subject to} & l^c & \leq & Ax + v_l^c - v_u^c & \leq & u^c, \\ & l^x & \leq & x + v_l^x - v_u^x & \leq & u^x, \\ & & & (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq & 0, \\ & & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0 & & \end{array} \tag{11.4}$$

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

$$\begin{array}{lccccc} \text{minimize} & & & c^T x & & \\ \text{subject to} & l^c & \leq & Ax + v_l^c - v_u^c & \leq & u^c, \\ & l^x & \leq & x + v_l^x - v_u^x & \leq & u^x, \\ & & & (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq & 0, \\ & & & p & = & p^*, \\ & & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0 & & \end{array} \tag{11.6}$$

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

$$
\begin{array}{lccccc}
\text{minimize} & -10x_1 & & -9x_2, & & \\
\text{subject to} & 7/10x_1 & + & 1x_2 & \leq & 630, \\
 & 1/2x_1 & + & 5/6x_2 & \leq & 600, \\
 & 1x_1 & + & 2/3x_2 & \leq & 708, \\
 & 1/10x_1 & + & 1/4x_2 & \leq & 135, \\
 & x_1, & & x_2 & \geq & 0. \\
 & & x_2 \geq 650 & & &
\end{array}
\tag{11.7}
$$

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:

```
/*
  Copyright: Copyright (c) 1998-2007 MOSEK ApS, Denmark. All rights reserved.

  File:      feasrepairex1.c

  Purpose:   To demonstrate how to use the MSK_relaxprimal function to
             locate the cause of an infeasibility.

  Syntax: On command line
          feasrepairex1 feasrepair.lp
```

```
          feasrepair.lp is located in mosek\<version>\tools\examples.
*/


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

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

  MSKenv_t  env;
  MSKintt   i;
  MSKtask_t task  = NULL, task_relaxprimal = NULL;
  double wlc[4] = {1.0,1.0,1.0,1.0};
  double wuc[4] = {1.0,1.0,1.0,1.0};
  double wlx[2] = {1.0,1.0};
  double wux[2] = {1.0,1.0};
  double sum_violation;
  MSKrescodee r = MSK_RES_OK;
  char buf[80];
  char      buffer[MSK_MAX_STR_LEN],symnam[MSK_MAX_STR_LEN];

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

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

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

  /* Read file from current dir */
  if ( r == MSK_RES_OK )
    r = MSK_readdata(task,argv[1]);

  /* Set type of relaxation */

  if (r == MSK_RES_OK)
    r = MSK_putintparam(task,MSK_IPAR_FEASREPAIR_OPTIMIZE,MSK_FEASREPAIR_OPTIMIZE_PENALTY);

  /* Call relaxprimal, minimizing sum of violations */

  if (r == MSK_RES_OK)
    r = MSK_relaxprimal(task,
                        &task_relaxprimal,
                        wlc,
                        wuc,
                        wlx,
                        wux);

  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. For instance, 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

$$\begin{array}{rclccccc} z(l^c, u^c, l^x, u^x, c) & = & \text{minimize} & & & c^T x & & \\ & & \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & & & & & l^x \leq x \leq u^x, & & \end{array} \tag{12.1}$$

and we want to know how the optimal objective value changes as l^c_i is perturbed. To answer this question we define the perturbed problem for l^c_i as follows

$$\begin{array}{rclcccc} f_{l^c_i}(\beta) & = & \text{minimize} & & & c^T x & \\ & & \text{subject to} & l^c + \beta e_i & \leq & Ax & \leq u^c, \\ & & & & & l^x \leq x \leq u^x, & \end{array} \tag{12.2}$$

where e_i is the ith column of the identity matrix. The function

$$f_{l^c_i}(\beta) \tag{12.3}$$

shows the optimal objective value as a function of β. Please note that a change in β corresponds to a perturbation in l^c_i and hence (12.3) shows the optimal objective value as a function of l^c_i.

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.

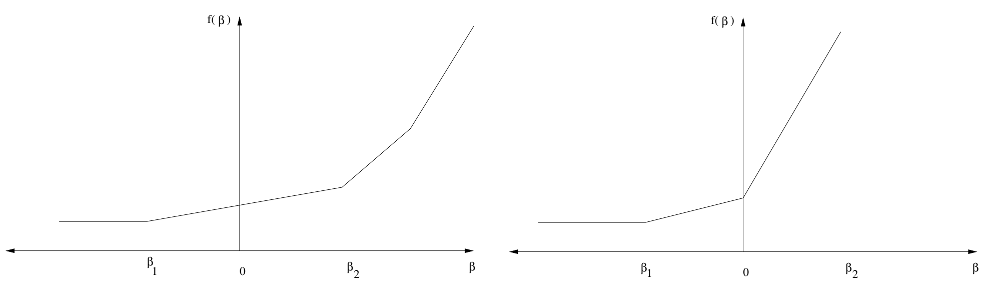

Figure 12.1: The optimal value function $f_{l^c_i}(\beta)$. Left: $\beta = 0$ is in the interior of linearity interval. Right: $\beta = 0$ is a breakpoint.

Clearly, if the function $f_{l^c_i}(\beta)$ does not change much when β is changed, then we can conclude that the optimal objective value is insensitive to changes in l^c_i. Therefore, we are interested in the rate of change in $f_{l^c_i}(\beta)$ for small changes in β — specifically the gradient

$$f'_{l^c_i}(0), \tag{12.4}$$

which is called the *shadow price* related to l^c_i. 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:

$$\begin{array}{lclccc} \sigma_1 & = & \text{minimize} & e_i^T s_l^c & & \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x & = & c, \\ & & & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) & = & z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0 & & \end{array} \tag{12.10}$$

and

$$\begin{array}{lclccc} \sigma_2 & = & \text{maximize} & e_i^T s_l^c & & \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x & = & c, \\ & & & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) & = & z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0. & & \end{array} \tag{12.11}$$

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

$$\begin{array}{lcllcccc} \beta_1 & = & \text{minimize} & & & \beta & & \\ & & \text{subject to} & l^c + \beta e_i & \leq & Ax & \leq & u^c, \\ & & & & & c^T x - \sigma_1 \beta & = & z^*, \\ & & & & & l^x \leq x \leq u^x, & & \end{array} \tag{12.13}$$

and

$$\begin{array}{lcllcccc} \beta_2 & = & \text{maximize} & & & \beta & & \\ & & \text{subject to} & l^c + \beta e_i & \leq & Ax & \leq & u^c, \\ & & & & & c^T x - \sigma_2 \beta & = & z^*, \\ & & & & & l^x \leq x \leq u^x. & & \end{array} \tag{12.14}$$

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:

$$\begin{array}{lcllcccc} \sigma_1 & = & \text{minimize} & & & e_j^T x & & \\ & & \text{subject to} & l^c + \beta e_i & \leq & Ax & \leq & u^c, \\ & & & & & c^T x & = & z^*, \\ & & & & & l^x \leq x \leq & u^x & \end{array} \tag{12.15}$$

and

$$\begin{array}{rclcccc} \sigma_2 & = & \text{maximize} & & e_j^T x & & \\ & & \text{subject to} & l^c + \beta e_i \leq & Ax & \leq & u^c, \\ & & & & c^T x & = & z^*, \\ & & & & l^x \leq x \leq & u^x. & \end{array} \tag{12.16}$$

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:

$$\begin{array}{rclccc} \beta_1 & = & \text{minimize} & \beta & & \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x & = & c + \beta e_j, \\ & & & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) - \sigma_1 \beta & \leq & z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0 & & \end{array} \tag{12.18}$$

and

$$\begin{array}{rclccc} \beta_2 & = & \text{maximize} & \beta & & \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x & = & c + \beta e_j, \\ & & & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) - \sigma_2 \beta & \leq & z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0. & & \end{array} \tag{12.19}$$

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

$$\begin{array}{ccccccccccccccc} x_{11} & + & x_{12} & & & & & & & & & & & \leq & 400, \\ & & & & x_{23} & + & x_{24} & & & & & & & \leq & 1200, \\ & & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\ x_{11} & & & & & & & + & x_{31} & & & & & = & 800, \\ & & x_{12} & & & & & & & & & & & = & 100, \\ & & & & x_{23} & + & & & & & x_{33} & & & = & 500, \\ & & & & & & x_{24} & + & & & & & x_{34} & = & 500, \\ x_{11}, & & x_{12}, & & x_{23}, & & x_{24}, & & x_{31}, & & x_{33}, & & x_{34} & \geq & 0. \end{array} \tag{12.21}$$

The basis type and the optimal partition type sensitivity results for the transportation problem are shown in Table 12.1 and 12.2 respectively.

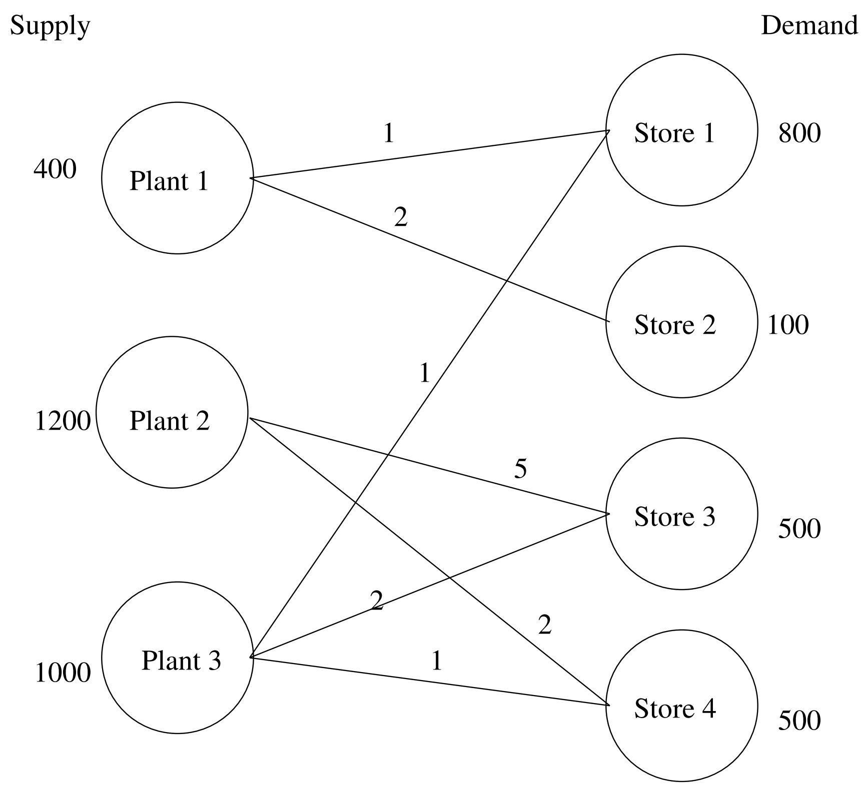

Figure 12.2: Supply, demand and cost of transportation.

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/5/tools/examp/capi/sensitivity.c`

```
/*
  Copyright: Copyright (c) 1998-2007 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. */
```

```
    r = MSK_maketask(env,NUMCON,NUMVAR,&task);

    if ( r==MSK_RES_OK )
    {
      /* Directs the log task stream to the user
         specified procedure 'printstr'. */

      MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);

      MSK_echotask(task,
                   MSK_STREAM_MSG,
                   "Defining the problem data.\n");

      /* Append the constraints. */
      if ( r==MSK_RES_OK )
        r = MSK_append(task,MSK_ACC_CON,NUMCON);

      /* Append the variables. */
      if ( r==MSK_RES_OK )
        r = MSK_append(task,MSK_ACC_VAR,NUMVAR);

      /* Put C. */
      if ( r==MSK_RES_OK )
        r = MSK_putcfix(task,0.0);

      for(j=0; j<NUMVAR && r==MSK_RES_OK; ++j)
      {
        r = MSK_putcj(task,j,c[j]);
        printf("%f\n",c[j]);
      }

      /* Put constraint bounds. */
      for(i=0; i<NUMCON && r==MSK_RES_OK; ++i)
        r = MSK_putbound(task,MSK_ACC_CON,i,bkc[i],blc[i],buc[i]);

      /* Put variable bounds. */
      for(j=0; j<NUMVAR && r==MSK_RES_OK; ++j)
        r = MSK_putbound(task,MSK_ACC_VAR,
                         j,bkx[j],blx[j],bux[j]);

      /* Put A. */
      if ( NUMCON>0 )
      {
        for(j=0; j<NUMVAR && r==MSK_RES_OK; ++j)
          r = MSK_putavec(task,MSK_ACC_VAR,
                          j,ptre[j]-ptrb[j],sub+ptrb[j],val+ptrb[j]);
      }

      if ( r==MSK_RES_OK )
      {
        MSK_putobjsense(task,MSK_OBJECTIVE_SENSE_MINIMIZE);

        MSK_echotask(task,
                     MSK_STREAM_MSG,
                     "Start optimizing\n");

        r = MSK_optimize(task);
```

```
    }
  }

  if (r == MSK_RES_OK)
  {
    /* Analyze upper bound on c1 and the equality constraint on c4 */
    MSKidxt subi[]  = {0,3};
    MSKmarke marki[] = {MSK_MARK_UP,MSK_MARK_UP};

    /* Analyze lower bound on the variables x12 and x31 */
    MSKidxt subj[]  = {1,4};
    MSKmarke markj[] = {MSK_MARK_LO,MSK_MARK_LO};

    MSKrealt leftpricei[2];
    MSKrealt rightpricei[2];
    MSKrealt leftrangei[2];
    MSKrealt rightrangei[2];
    MSKrealt leftpricej[2];
    MSKrealt rightpricej[2];
    MSKrealt leftrangej[2];
    MSKrealt rightrangej[2];

    r = MSK_primalsensitivity( task,
                               2,
                               subi,
                               marki,
                               2,
                               subj,
                               markj,
                               leftpricei,
                               rightpricei,
                               leftrangei,
                               rightrangei,
                               leftpricej,
                               rightpricej,
                               leftrangej,
                               rightrangej);

    printf("Results from sensitivity analysis on bounds:\n");

    printf("For constraints:\n");
    for (i=0;i<2;++i)
      printf("leftprice = %e, rightprice = %e,leftrange = %e, rightrange =%e\n",
             leftpricei[i], rightpricei[i], leftrangei[i], rightrangei[i]);

    printf("For variables:\n");
    for (i=0;i<2;++i)
      printf("leftprice = %e, rightprice = %e,leftrange = %e, rightrange =%e\n",
             leftpricej[i], rightpricej[i], leftrangej[i], rightrangej[i]);
  }

  if (r == MSK_RES_OK)
  {

    MSKidxt subj[]  = {2,5};

    MSKrealt leftprice[2];
    MSKrealt rightprice[2];
```

```
      MSKrealt leftrange[2];
      MSKrealt rightrange[2];

      r = MSK_dualsensitivity(task,
                              2,
                              subj,
                              leftprice,
                              rightprice,
                              leftrange,
                              rightrange
                              );

      printf("Results from sensitivity analysis on objective coefficients:\n");

      for (i=0;i<2;++i)
        printf("leftprice = %e, rightprice = %e,leftrange = %e, rightrange =%e\n",
               leftprice[i], rightprice[i], leftrange[i], rightrange[i]);
    }

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

  printf("Return code: %d (0 means no error occured.)\n",r);

  return ( r );
} /* 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|UL [cname1]
 U|L|UL [cname2]-[cname3]
BOUNDS VARIABLES
 U|L|UL [vname1]
 U|L|UL [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

$$\begin{array}{ll} \text{minimize} & \sum\limits_{i,j} c_{ij} x_{ij} \\ \text{subject to} & \sum\limits_{i} x_{ij} = 1 \qquad \forall j, \\ & \sum\limits_{j} x_{ij} = 1 \qquad \forall i, \\ & 0 \leq x_{ij} \leq 1, \quad x_{ij} \text{ integer}. \end{array} \tag{13.1}$$

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

$$\begin{aligned} u_1 &= 1, && \\ 2 \leq u_i &\leq n && \forall i \neq 1, \\ u_i - u_j + 1 &\leq (n-1)(1 - x_{ij}) && \forall i \neq 1, j \neq 1. \end{aligned} \tag{13.2}$$

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) 1998-2007 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
```

```
#define MAXADDPERROUND 1000

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


/* conversion from n x n tsp city matrix indices to array index */
#define IJ(i,j) (n*(i)+(j))

/* mallocs and returns costmatrix, returns number of cities in ncities */
int* readtspfromfile(char* filename, int* ncities)
{
    FILE *tspfile;
    char sbuf[21];
    tspfile = fopen(filename,"r");
    if (!tspfile) return NULL;
    do
    {
        if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
    } while (strncmp(sbuf,"DIMENSION",9) != 0);
    if (1 != fscanf(tspfile,"%d ",ncities)) return NULL;
    do
    {
        if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
    } while (strncmp(sbuf,"EDGE_WEIGHT_TYPE",16) != 0);
    if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
    if (strcmp(sbuf,"EXPLICIT") == 0)
    {
        do
        {
            if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
        } while (strncmp(sbuf,"EDGE_WEIGHT_FORMAT",18) != 0);
        if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
        if (strcmp(sbuf,"FULL_MATRIX") == 0)
        {
            int* cost;
            int ij, n2;
            do
            {
                if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
            } while (strncmp(sbuf,"EDGE_WEIGHT_SECTION",19) != 0);
            n2 = *ncities;
            n2 *= n2;
            cost = (int*) malloc(n2*sizeof(int));
            assert(cost);
            for (ij = 0; ij<n2; ij++)
            {
                if (1 != fscanf(tspfile,"%d ",&cost[ij]))
                {
                    free(cost);
                    return NULL;
                }
            }
            return cost;
        }
        else if (strcmp(sbuf,"LOWER_DIAG_ROW") == 0)
```

```
        {
            int* cost;
            int i, j, n;
            do
            {
                if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
            } while (strncmp(sbuf,"EDGE_WEIGHT_SECTION",19) != 0);
            n = *ncities;
            cost = (int*) malloc(n*n*sizeof(int));
            assert(cost);
            for (i=0; i<n; i++) for (j=0; j<=i; j++)
            {
                int c;
                if (1 != fscanf(tspfile,"%d ",&c))
                {
                    free(cost);
                    return NULL;
                }
                cost[IJ(i,j)] = c;
                cost[IJ(j,i)] = c;
            }
            return cost;
        }
        else
        {
            printf("Format not supported\n");
            return NULL;
        }
    }
    else if (strcmp(sbuf,"EUC_2D") == 0)
    {
        int* cost;
        double *xcoord, *ycoord;
        int i, j, n;
        do
        {
            if (1 != fscanf(tspfile,"%20s ",sbuf)) return NULL;
        } while (strncmp(sbuf,"NODE_COORD_SECTION",18) != 0);
        n = *ncities;
        xcoord = (double*) malloc(n*sizeof(double));
        ycoord = (double*) malloc(n*sizeof(double));
        cost = (int*) malloc(n*n*sizeof(int));
        assert(xcoord); assert(ycoord); assert(cost);
        for (i = 0; i<n; i++)
        {
            int dummy;
            if (3 != fscanf(tspfile,"%d %lf %lf ",&dummy,&xcoord[i],&ycoord[i]))
            {
                free(cost);
                return NULL;
            }
        }
        for (i = 0; i<n; i++) for (j=0; j<n; j++)
        {
            double xd = xcoord[i] - xcoord[j];
            double yd = ycoord[i] - ycoord[j];
            cost[IJ(i,j)] = (int) (0.5 + sqrt(xd*xd + yd*yd));
        }
```

```
        return cost;
    }
    else
    {
        printf("E_W_Type not supported\n");
        return NULL;
    }
} /* readtspfromfile */

/* add the x_ij variables */
void add_vars(MSKtask_t task, int n)
{
    MSKrescodee r;
    int ij;
    int n2 = n*n;
    r = MSK_append(task,MSK_ACC_VAR,n2); assert(r==MSK_RES_OK);
    for(ij=0; ij<n2; ++ij)
    {
        r = MSK_putbound(task,MSK_ACC_VAR,ij,MSK_BK_RA,0,1);
        assert(r==MSK_RES_OK);
        r = MSK_putvartype(task,ij,MSK_VAR_TYPE_INT); assert(r==MSK_RES_OK);
    }
    for (ij=0; ij<n; ij++)
    {
        r = MSK_putbound(task,MSK_ACC_VAR,IJ(ij,ij),MSK_BK_FX,0,0);
        assert(r==MSK_RES_OK);
    }
} /* add_vars */

/* adds the tsp objective function and frees cost */
void add_objective_function(MSKtask_t task, int n, int* cost)
{
    MSKrescodee r;
    int ij;
    int n2 = n*n;
    r = MSK_putcfix(task,0.0); assert(r==MSK_RES_OK);
    for(ij=0; ij<n2; ++ij)
    {
        r = MSK_putcj(task,ij,cost[ij]); assert(r==MSK_RES_OK);
    }
    free(cost);
} /* add_objective_function */

/* adds the tsp assignment constraints */
void add_assignment_constraints(MSKtask_t task, int n)
{
    MSKrescodee r;
    int i, j;
    double* aval;
    int *asub;
    aval = (double*) malloc(n*sizeof(double)); assert(aval);
    asub = (int*) malloc(n*sizeof(int)); assert(asub);
    for (i=0; i<n; i++) aval[i] = 1;
    r = MSK_append(task,MSK_ACC_CON,n*2); assert(r==MSK_RES_OK);
    /* Constraint 0--(n-1) is \sum_j x_{ij} = 1 */
    for (i=0; i<n; i++)
    {
        r = MSK_putbound(task,MSK_ACC_CON,i,MSK_BK_FX,1,1);
```

```
        assert(r==MSK_RES_OK);
        for (j=0; j<n; j++)
            asub[j] = IJ(i,j);
        r = MSK_putavec(task,MSK_ACC_CON,i,n,asub,aval); assert(r==MSK_RES_OK);
    }
    /* Constraint n--(2n-1) is \sum_i x_{ij} = 1 */
    for (j=0; j<n; j++)
    {
        r = MSK_putbound(task,MSK_ACC_CON,j+n,MSK_BK_FX,1,1);
        assert(r==MSK_RES_OK);
        for (i=0; i<n; i++)
            asub[i] = IJ(i,j);
        r = MSK_putavec(task,MSK_ACC_CON,j+n,n,asub,aval);
        assert(r==MSK_RES_OK);
    }
    free(aval);
    free(asub);
} /* add_assignment_constraints */

/* adds the Miller-Tucker-Zemlin arc constraints */
void add_MTZ_arc_constraints(MSKtask_t task, int n)
{
    MSKrescodee r;
    int varidx, conidx, i, j;
    r = MSK_getnumvar(task,&varidx); assert(r==MSK_RES_OK);
    r = MSK_getnumcon(task,&conidx); assert(r==MSK_RES_OK);
    /* add the vars u_k for k=1..(n-1) getting index
     * from varidx to varidx+n-2 */
    r = MSK_append(task,MSK_ACC_VAR,n-1); assert(r==MSK_RES_OK);
    for(i=varidx; i<varidx+n-1; ++i)
    {
        /* set bound: 2 <= u_k <= n, k=1..(n-1) */
        r = MSK_putbound(task,MSK_ACC_VAR,i,MSK_BK_RA,2,n);
        assert(r==MSK_RES_OK);
    }
    /* add the (n-1)^2 constraints:
     * u_i - u_j + 1 <= (n - 1)(1 - x_ij) or equivalently
     * u_i - u_j + (n - 1)x_ij <= n - 2, for i,j != 0 */
    r = MSK_append(task,MSK_ACC_CON,(n-1)*(n-1)); assert(r==MSK_RES_OK);
    for (i=1; i<n; i++) for (j=1; j<n; j++)
    {
        double aval[3];
        int asub[3];
        aval[0] = 1; aval[1] = -1; aval[2] = n-1;
        asub[0] = varidx + i - 1; /* u_i */
        asub[1] = varidx + j - 1; /* u_j */
        asub[2] = IJ(i,j);        /* x_ij */
        r = MSK_putbound(task,MSK_ACC_CON,conidx,MSK_BK_UP,-MSK_INFINITY,n-2);
        assert(r==MSK_RES_OK);
        r = MSK_putavec(task,MSK_ACC_CON,conidx,3,asub,aval);
        assert(r==MSK_RES_OK);
        conidx++;
    }
} /* add_MTZ_arc_constraints */

/* construct the list of cities in the chosen subtours */
int* subtourstolist(MSKtask_t task, int n, int nextnode[],
        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;
```

```
    xx = (double*) malloc(n2*sizeof(double));
    nextnode = (int*) malloc(n*sizeof(int));
    assert(xx);
    assert(nextnode);
    r = MSK_getsolutionslice(task,MSK_SOL_ITG,MSK_SOL_ITEM_XX,0,n2,xx);
    assert(r==MSK_RES_OK);
    /* convert matrix representation of graph (xx) to
     * adjacency(-list) (nextnode) */
    for (i=0; i<n; i++) for (j=0; j<n; j++)
    {
        if (xx[IJ(i,j)]>0.5) /* i.e. x_ij = 1 */
            nextnode[i] = j;
    }
    free(xx); xx = NULL;
    visited = (int*) calloc(n,sizeof(int)); /* visited is initialized to 0 */
    subtour = (int*) malloc(n*sizeof(int));
    assert(visited);
    assert(subtour);
    /* identify subtours; keep count in nsubt, save starting
     * pointers in subtour[0..(nsubt-1)] */
    for (i=0; i<n; i++) if (!visited[i]) /* find an unvisited node;
                                          * this starts a new subtour */
    {
        subtour[nsubt] = i;
        nsubt++;
        j = i;
        do
        {
            assert(!visited[j]);
            visited[j] = 1;
            j = nextnode[j];
        } while (j!=i);
    }
    free(visited); visited = NULL;
    *nsubtours = nsubt;
    *ncuts = 0;
    chosen = (int*) malloc(nsubt*sizeof(int)); /* list of chosen subtours */
    for (k=1; k<=nsubt; k++) /* choose k of nsubt subtours */
    {
        int nchosen = 1;
        chosen[0] = nsubt - 1;
        while (*ncuts < maxcuts)
        {
            if (nchosen == k)
            {
                int *citylist;
                int size;
                citylist = subtourstolist(task,n,nextnode,subtour,
                                          chosen,k,&size);
                if (size <= n/2) /* add only subtour constraints
                                  * of size n/2 or less */
                {
                    addcut(task,n,citylist,size);
                    (*ncuts)++;
                }
                free(citylist);
                j=0;
                while (j<k && chosen[k - 1 - j] == j) j++;
```

```
                if (k==j) break; /* all k-size subsets done */
                nchosen = k - j;
                chosen[nchosen - 1]--;
            }
            else /* 0 < nchosen < k */
            {
                chosen[nchosen] = chosen[nchosen - 1] - 1;
                nchosen++;
            }
        }
    }
    free(nextnode);
    free(subtour);
    free(chosen);
} /* addcuts */

int main(int argc, char *argv[])
{
    int            *cost;     /* tsp cost matrix */
    int            n;         /* number of cities */
    MSKenv_t       env;       /* Mosek environment */
    MSKtask_t      task;      /* Mosek task */
    MSKrescodee    r;         /* Mosek return code */
    double         ObjVal;    /* Value of the objective function */
    int            maxrounds; /* number of cutting rounds */
    int            maxcuts;   /* maximum number of cuts added per round */
    int k;
    int nsubtours, ncuts;
    double t;
    double cuttime = 0;

    if (argc < 2)
    {
        printf("Usage: ./tsp filename.tsp [rounds] [maxcuts]\n\n"
               "rounds is the maximum number of cutting rounds (default = %d)\n"
               "maxcuts is the maximum number of cuts added per round "
               "(default = %d)\n",
                MAXCUTROUNDS,MAXADDPERROUND);
        return 1;
    }
    maxrounds = MAXCUTROUNDS;
    if (argc >= 3) maxrounds = atoi(argv[2]);
    maxcuts = MAXADDPERROUND;
    if (argc >= 4) maxcuts = atoi(argv[3]);

    cost = readtspfromfile(argv[1],&n);
    if (!cost)
    {
        printf("Bad tsp file\n");
        return 1;
    }

    r = MSK_makeenv(&env,NULL,NULL,NULL,NULL); assert(r==MSK_RES_OK);
    MSK_linkfunctoenvstream(env,MSK_STREAM_LOG,NULL,printstr);
    r = MSK_initenv(env);                      assert(r==MSK_RES_OK);
    r = MSK_makeemptytask(env,&task);          assert(r==MSK_RES_OK);
    MSK_linkfunctotaskstream(task,MSK_STREAM_LOG,NULL,printstr);
```

```
    add_vars(task,n);
    add_objective_function(task,n,cost);
    add_assignment_constraints(task,n);

    nsubtours = 2;
    for (k=0; k<maxrounds; k++)
    {
        r = MSK_optimize(task);                          assert(r==MSK_RES_OK);
        r = MSK_getprimalobj(task,MSK_SOL_ITG,&ObjVal); assert(r==MSK_RES_OK);
        MSK_getdouinf(task,MSK_DINF_OPTIMIZER_CPUTIME,&t);
        cuttime += t;
        addcuts(task,n,maxcuts,&nsubtours,&ncuts);
        printf("\n"
               "Round: %d\n"
               "ObjValue: %e\n"
               "Number of subtours: %d\n"
               "Number of cuts added: %d\n\n",k+1,ObjVal,nsubtours,ncuts);
        if (nsubtours == 1) break; /* problem solved! */
    }

    t = 0;
    if (nsubtours > 1)
    {
        printf("Adding MTZ arc constraints\n\n");
        add_MTZ_arc_constraints(task,n);
        r = MSK_optimize(task);                          assert(r==MSK_RES_OK);
        r = MSK_getprimalobj(task,MSK_SOL_ITG,&ObjVal); assert(r==MSK_RES_OK);
        MSK_getdouinf(task,MSK_DINF_OPTIMIZER_CPUTIME,&t);
    }

    printf("\n"
           "Done solving.\n"
           "Time spent cutting: %.2f\n"
           "Total time spent: %.2f\n"
           "ObjValue: %e\n",cuttime,cuttime+t,ObjVal);

    MSK_deletetask(&task);
    MSK_deleteenv(&env);
    return 0;
} /* main */
```

13.2 Geometric (posynomial) optimization

13.2.1 The problem

A *geometric optimization* problem can be stated as follows

$$\begin{array}{lll} \text{minimize} & \sum_{k \in J_0} c_k \prod_{j=0}^{n-1} t_j^{a_{kj}} & \\ \text{subject to} & \sum_{k \in J_i} c_k \prod_{j=0}^{n-1} t_j^{a_{kj}} \quad \leq \quad 1, \quad i = 1, \ldots, m, & \\ & t > 0, & \end{array} \tag{13.4}$$

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

$$\begin{array}{lll} \text{minimize} & \sum\limits_{k \in J_0} c_k e^{\sum\limits_{j=0}^{n-1} a_{kj} x_j} & \\ \text{subject to} & \sum\limits_{k \in J_i} c_k e^{\sum\limits_{j=0}^{n-1} a_{kj} x_j} & \leq \quad 1, \quad i = 1, \ldots, m, \end{array} \tag{13.6}$$

which is a convex optimization problem that can be solved using MOSEK. We will call

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

for 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

$$\begin{array}{llll} \text{minimize} & \sum\limits_{t \in J_0} e^{v_t} & & \\ \text{subject to} & \sum\limits_{t \in J_i} e^{v_t} & \leq \quad 1, & i = 1, \ldots, m, \\ & \sum\limits_{j=0}^{n-1} a_{tj} x_j - v_t & = \quad -\log(c_t), & t = 0, \ldots, T, \end{array} \tag{13.7}$$

which is a separable convex optimization problem.

One warning about this approach is that the function

$$e^x$$

is only well-defined for small values of x in absolute value. Indeed 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 those applications here but rather we refer the reader to [14] and the references therein.

13.2.3 Modelling tricks

A lot of tricks that can be used modelling 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

$$\begin{array}{lcll} xyz^{-1} & & & \leq & 1, \\ \frac{1}{xyz^{-1}} & = & x^{-1}y^{-1}z & \leq & 1. \end{array}$$

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:

$$\begin{array}{lccc} \text{minimize} & x^2 y & & \\ \text{subject to} & xy & \leq & 1, \\ & x, y > 0. & & \end{array}$$

Clearly, the optimal objective value is 0, but because of the constraint the constraint $x, y > 0$ 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

$$\begin{array}{lccc} \text{minimize} & e^{2t_x} e^{t_y} & & \\ \text{subject to} & e^{t_x} e^{t_y} & \leq & 1, \\ & t_x, t_y \in R. & & \end{array}$$

We now see that the optimal solution implies that $t_x = -\infty$ or $t_y = -\infty$, which is unattainable.

It should now be clear what the issue is: 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:

$$\begin{array}{lccc} \text{minimize} & x^{-2} & & \\ \text{subject to} & x^{-1} & \leq & 1, \\ & x > 0, & & \end{array}$$

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 $x \rightarrow \infty$.

13.2.5 An example

Consider the example

$$\begin{array}{lccc} \text{minimize} & x^{-1} y & & \\ \text{subject to} & x^2 y^{-\frac{1}{2}} + 3 y^{\frac{1}{2}} z^{-1} & \leq & 1, \\ & x y^{-1} & = & z^2, \\ & -x & \leq & -\frac{1}{10}, \\ & x & \leq & 3, \\ & x, y, z > 0, & & \end{array}$$

which is not a geometric optimization problem. However, using the obvious transformations we obtain the problem

$$
\begin{array}{lrcl}
\text{minimize} & x^{-1}y & & \\
\text{subject to} & x^2y^{-\frac{1}{2}} + 3y^{\frac{1}{2}}z^{-1} & \leq & 1, \\
& xy^{-1}z^{-2} & \leq & 1, \\
& x^{-1}yz^2 & \leq & 1, \\
& \frac{1}{10}x^{-1} & \leq & 1, \\
& \frac{1}{3}x & \leq & 1, \\
& x, y, z > 0, & &
\end{array}
\tag{13.8}
$$

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

API developer guidelines

The purpose of this chapter is to present some guidelines for developing an application which uses the MOSEK API.

14.1 Turn on logging

While developing a new application it is beneficial 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 implies that all messages sent to a task stream are also written to a file. As an example consider the code fragment

```
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
```

14.2 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 numbers and outputs a warning if any are found.

14.3 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.

14.4 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.

14.5 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.

14.6 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 [1] or valgrind [2] 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.

14.7 Check the problem status and solution status

If a problem is primal or dual infeasible and MOSEK detects this, it is **not** reported as an error. Therefore, it is important to check the problem status and solution status after the optimization optimization ended using the `MSK_getsolutionstatus` function or the `MSK_getsolutioninf`. function.

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.

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.

[1] Purify is a commercial product available from IBM, which runs on Windows and various UNIXes.

[2] Valgrind is open source product available for Linux on X86 and other architectures.

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.

- `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 $2^{3}2$ 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.

- `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.

- `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 MSKAPI 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 18.5 for the possible values of this argument.

- `MSKctrlcfunc`

Description: Definition of a user-defined `ctrl-c` function. When the function is called from MOSEK, a non-zero return value is interpreted as if `ctrl-c` had been pressed. If a `ctrl-c` function is attached to an environment, MOSEK will not otherwise check for `ctrl-c` events.

Syntax: `MSKintt MSKAPI MSKctrlcfunc (MSKuserhandle_t usrptr)`

Arguments: `usrptr` (input/output)
A pointer to a user-defined structure.

- `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 MSKAPI 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 MSKAPI MSKfreefunc (
        MSKuserhandle_t usrptr,
        MSKuserhandle_t buffer);
```

Arguments: `usrptr` (input)
A pointer to a user-defined structure.

`buffer` (input)
A pointer to the buffer which should be freed.

- `MSKmallocfunc`

Description: A user-defined memory allocation function.

Syntax:
```
void * MSKAPI MSKmallocfunc (
    MSKuserhandle_t usrptr,
    MSKCONST size_t size);
```

Arguments: `usrptr` (input)
A pointer to a user-defined structure.

`size` (input)
The number of chars 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 MSKAPI MSKnlgetspfunc (
    MSKuserhandle_t nlhandle,
    MSKintt * numgrdobjnz,
    MSKidxt * grdobjsub,
    MSKidxt i,
    MSKbooleant * convali,
    MSKintt * grdconinz,
    MSKidxt * grdconisub,
    MSKintt yo,
    MSKintt numycnz,
    MSKCONST MSKidxt * ycsub,
    MSKlintt maxnumhesnz,
    MSKlintt * 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

$$L := \mathtt{yo} * f(\mathtt{xx}) - \sum_{i=0}^{\mathtt{numi}-1} \mathtt{yc}_{\mathtt{subi[k]}} g_{\mathtt{subi[k]}}(\mathtt{xx}). \tag{15.3}$$

Syntax:
```
MSKintt MSKAPI 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 MSKlidxt * grdconptrb,
    MSKCONST MSKlidxt * grdconptre,
    MSKidxt * grdconsub,
    MSKrealt * grdconval,
    MSKrealt * grdlag,
    MSKlintt maxnumhesnz,
    MSKlintt * 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 `numi`.

`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

$$\texttt{grdobjval}[\texttt{k}] = \frac{\partial f}{\partial x_{\texttt{grdobjsub[k]}}}(\texttt{xx})$$

for $k = 0, \ldots, \texttt{numgrdobjnz} - 1$ is used.

`numi` (input)
: Number of elements in `subi`.

`subi` (input)
: $\texttt{subi}[0, ..., \texttt{numi} - 1]$ contain the indexes of the constraints that has to be evaluated. The length is `numi`.

`conval` (output)
: $g(\texttt{xx})$ for the required constraint functions i.e.

$$\texttt{conval}[\texttt{k}] = g_{\texttt{subi[k]}}(\texttt{xx})$$

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` (output)
: 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

$$\begin{array}{l} \texttt{grdconval}[\texttt{k}] = \frac{\partial g_{\texttt{subi[i]}}(xx)}{\partial xx_{\texttt{grdconsub[k]}}}, \quad \text{for} \\ \quad k = \texttt{grdconptrb}[i], \ldots, \texttt{grdconptre}[i] - 1, \\ \quad i = 0, \ldots, \texttt{numi} - 1. \end{array}$$

`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 MSKAPI 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 MSKAPI 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 Reading and writing data files.

Reading and writing data files.

MSK_readbranchpriorities (page 334)
Reads branching priority data from a file.

MSK_readdata (page 334)
Reads problem data from a file.

MSK_readparamfile (page 335)
Reads a parameter file.

MSK_readsolution (page 335)
Reads a solution from a file.

MSK_readsummary (page 335)
Prints information about last file read.

MSK_writebranchpriorities (page 345)
Writes branching priority data to a file.

MSK_writeparamfile (page 346)
Writes all the parameters to a parameter file.

MSK_writesolution (page 346)
Write a solution to a file.

15.2.2 Solutions.

Obtain or define a solution.

MSK_deletesolution (page 270)
Undefines a solution and frees the memory it uses.

MSK_getdualobj (page 281)
Obtains the dual objective value.

15.2.3 Ccall-backs (put/get).

Manipulating call-backs.

15.2.4 Memory allocation and deallocation.

Memory allocation and deallocation.

15.2.5 Changing problem specification.

Input or change problem specification

MSK_putvartypelist (page 333)
Sets the variable type for one or more variables.

15.2.6 Delete problem elements (variables,constraints,cones).

Functionality for deleting problem elements such as variables, constraints or cones.

MSK_remove (page 338)
The function removes a number of constraints or variables.

MSK_removecone (page 338)
Removes a conic constraint from the problem.

15.2.7 Add problem elements (variables,constraints,cones).

Functionality for adding problem elements such as variables, constraints or cones.

MSK_append (page 263)
Appends a number of variables or constraints to the optimization task.

MSK_appendcone (page 264)
Appends a new cone constraint to the problem.

15.2.8 Inspect problem specification.

Functionality for inspecting the problem specification (A,Q, bounds, objective e.t.c).

MSK_getaij (page 272)
Obtains a single coefficient in A.

MSK_getaslice (page 273)
Obtains a sequence of rows or columns from A.

MSK_getaslicetrip (page 275)
Obtains a sequence of rows or columns from A in triplet format.

MSK_getavec (page 275)
Obtains one row or column of A.

MSK_getavecnumnz (page 276)
Obtains the number of non-zero elements in one row or column of A.

MSK_getbound (page 276)
Obtains bound information for one constraint or variable.

MSK_getboundslice (page 277)
Obtains bounds information for a sequence of variables or constraints.

MSK_getc (page 277)
Obtains all objective coefficients c.

MSK_getcfix (page 278)
Obtains the fixed term in the objective.

MSK_getcone (page 278)
Obtains a conic constraint.

MSK_getconeinfo (page 279)
Obtains information about a conic constraint.

MSK_getcslice (page 280)
Obtains a part of c.

MSK_getnumanz (page 289)
Obtains the number of non-zeros in A.

MSK_getnumcon (page 289)
Obtains the number of constraints.

MSK_getnumcone (page 290)
Obtains the number of cones.

MSK_getnumconemem (page 290)
Obtains the number of members in a cone.

MSK_getnumintvar (page 290)
Obtains the number of integer constrained variables.

MSK_getnumqconnz (page 291)
Obtains the number of non-zero quadratic terms in a constraint.

MSK_getnumqobjnz (page 291)
Obtains the number of non-zero quadratic terms in the objective.

MSK_getnumvar (page 291)
Obtains the number of variables.

MSK_getobjsense (page 292)
Gets the objective sense.

MSK_getprobtype (page 293)
Obtains the problem type.

MSK_getqconk (page 293)
Obtains all the quadratic terms in a constraint.

MSK_getqobj (page 294)
Obtains all the quadratic terms in the objective.

MSK_getqobjij (page 295)
Obtains one coefficient from the quadratic term of the objective

15.2.9 Conic constraints.

Functionality related to conic terms in the problem.

15.2.10 Bounds.

Functionality related to changing or inspecting bounds on variables or constraints.

15.2.11 Task initialization and deletion.

Task initialization and deletion.

MSK_deletetask (page 270)
Deletes an optimization task.

MSK_makeemptytask (page 247)
Creates a new and empty optimization task.

MSK_maketask (page 248)
Creates a new optimization task.

15.2.12 Error handling.

Error handling.

MSK_exceptiontask (page 271)
Echo a response code to a task stream.

MSK_getcodedisc (page 244)
Obtains a short description of a response code.

MSK_getresponseclass (page 244)
Obtain the class of a response code.

MSK_putresponsefunc (page 330)
Inputs a user-defined error call-back function.

15.2.13 Output stream functions.

Output stream functions.

MSK_echoenv (page 242)
Sends a message to a given environment stream.

MSK_echointro (page 242)
Prints an intro to message stream.

MSK_echotask (page 271)
Prints a format string to a task stream.

MSK_exceptiontask (page 271)
Echo a response code to a task stream.

MSK_linkfiletoenvstream (page 246)
Directs all output from a stream to a file.

MSK_linkfiletotaskstream (page 308)
Directs all output from a task stream to a file.

MSK_linkfunctoenvstream (page 246)
Connects a user-defined function to a stream.

MSK_linkfunctotaskstream (page 308)
Connects a user-defined function to a task stream.

MSK_printdata (page 313)
Prints a part of the problem data to a stream.

MSK_printparam (page 314)
Prints the current parameter settings.

MSK_readsummary (page 335)
Prints information about last file read.

MSK_solutionsummary (page 341)
Prints a short summary of a solution.

MSK_unlinkfuncfromenvstream (page 252)
Disconnects a user-defined function from a stream.

MSK_unlinkfuncfromtaskstream (page 344)
Disconnects a user-defined function from a task stream.

15.2.14 Objective function.

Change or inspect objective function.

MSK_getc (page 277)
Obtains all objective coefficients c.

MSK_getcfix (page 278)
Obtains the fixed term in the objective.

MSK_getcslice (page 280)
Obtains a part of c.

MSK_getdualobj (page 281)
Obtains the dual objective value.

MSK_getnumqobjnz (page 291)
Obtains the number of non-zero quadratic terms in the objective.

MSK_getobjname (page 292)
Obtains the name assigned to the objective function.

MSK_getobjsense (page 292)
Gets the objective sense.

MSK_getprimalobj (page 293)
Obtains the primal objective value.

MSK_getqobj (page 294)
Obtains all the quadratic terms in the objective.

MSK_getqobjij (page 295)
Obtains one coefficient from the quadratic term of the objective

MSK_putcfix (page 320)
Replaces the fixed term in the objective.

MSK_putcj (page 320)
Modifies one linear coefficient in the objective.

MSK_putclist (page 320)
Modifies a part of c.

MSK_putobjsense (page 326)
Sets the objective sense.

MSK_putqobj (page 328)
Replaces all quadratic terms in the objective.

MSK_putqobjij (page 329)
Replaces one of the coefficients in the quadratic term in the objective.

15.2.15 Inspect statistics from the optimizer.

Inspect statistics from the optimizer.

MSK_appendstat (page 265)
Appends a record the statistics file.

MSK_getdouinf (page 280)
Obtains a double information item.

MSK_getinfindex (page 282)
Obtains the index of a named information item.

MSK_getinfmax (page 282)
Obtains the maximum index of an information of a given type `inftype` plus 1.

MSK_getinfname (page 282)
Obtains the name of an information item.

MSK_getintinf (page 283)
Obtains an integer information item.

MSK_getnadouinf (page 286)
Obtains a double information item.

MSK_getnaintinf (page 286)
Obtains an integer information item.

MSK_getnaintparam (page 287)
Obtains an integer parameter.

MSK_startstat (page 342)
Starts the statistics file.

MSK_stopstat (page 342)
Stops the statistics file.

15.2.16 Parameters (set/get).

Setting and inspecting solver parameters.

MSK_getdouparam (page 280)
Obtains a double parameter.

MSK_getintparam (page 283)
Obtains an integer parameter.

MSK_getnadouparam (page 286)
Obtains a double parameter.

MSK_getnastrparam (page 288)
Obtains a string parameter.

MSK_getnastrparamal (page 288)
Obtains the value of a string parameter.

MSK_getnumparam (page 290)
Obtains the number of parameters of a given type.

MSK_getparammax (page 292)
Obtains the maximum index of a parameter of a given type plus 1.

MSK_getparamname (page 293)
Obtains the name of a parameter.

MSK_getstrparam (page 302)
Obtains the value of a string parameter.

MSK_getstrparamal (page 302)
Obtains the value a string parameter.

MSK_getsymbcondim (page 245)
Obtains dimensional information for the defined symbolic constants.

MSK_iparvaltosymnam (page 245)
Obtains the symbolic name corresponding to a value that can be assigned to an integer parameter.

15.2.17 Naming.

Functionality related to naming.

15.2.18 Preallocating space for problem data.

Functionality related to preallocating space for problem data.

MSK_putmaxnumvar (page 324)
Sets the number of preallocated variables in the optimization task.

15.2.19 Integer variables.

Functionality related to integer variables.

MSK_getnumintvar (page 290)
Obtains the number of integer constrained variables.

MSK_getvarbranchdir (page 304)
Obtains the branching direction for a variable.

MSK_getvarbranchorder (page 304)
Obtains the branching priority for a variable.

MSK_getvarbranchpri (page 304)
Obtains the branching priority for a variable.

MSK_getvartype (page 305)
Gets the variable type of one variable.

MSK_getvartypelist (page 305)
Obtains the variable type for one or more variables.

MSK_putvarbranchorder (page 333)
Assigns a branching priority and direction to a variable.

MSK_putvartype (page 333)
Sets the variable type of one variable.

MSK_putvartypelist (page 333)
Sets the variable type for one or more variables.

15.2.20 Quadratic terms.

Functionality related to quadratic terms.

MSK_getqconk (page 293)
Obtains all the quadratic terms in a constraint.

MSK_getqobj (page 294)
Obtains all the quadratic terms in the objective.

MSK_getqobjij (page 295)
Obtains one coefficient from the quadratic term of the objective

MSK_putqcon (page 327)
Replaces all quadratic terms in constraints.

15.2.21 Diagnosing infeasibility.

Functions for diagnosing infeasibility.

15.2.22 Optimization.

Functions for optimization.

15.2.23 Sensitivity analysis.

Functions for sensitivity analysis.

15.2.24 Testing data validity.

Functions for testing data validity.

MSK_checkconvexity (page 267)
Checks if a quadratic optimization problem is convex.

15.2.25 Solving with the basis.

Functions for solving linear systems with the basis matrix.

MSK_initbasissolve (page 306)
Prepare a task for use with the `MSK_solvewithbasis` function.

MSK_solvewithbasis (page 341)
Solve a linear equation system involving a basis matrix.

15.2.26 Initialization of environment.

Creation and initialization of environment.

MSK_deleteenv (page 242)
Deletes the MOSEK environment.

MSK_initenv (page 245)
Initialize a MOSEK environment.

MSK_makeenv (page 247)
Creates a new MOSEK environment.

MSK_putlicensedefaults (page 250)
Set defaults used by the license manager.

15.2.27 Change A.

Change elements in the coefficient (A) matrix.

MSK_appendcons (page 264)
Appends one or more constraints and specifies bounds and A coefficients.

MSK_appendvars (page 265)
Appends one or more variables and specifies bounds on variables, c coefficients and A coefficients.

MSK_commitchanges (page 269)
Commits all cached problem changes.

MSK_putaij (page 314)
Changes a single value in the linear coefficient matrix.

MSK_putaijlist (page 315)
Changes one or more coefficients in A.

MSK_putavec (page 315)
Replaces all elements in one row or column of A.

MSK_putaveclist (page 316)
Replaces all elements in one or more rows or columns in A by new values.

15.3 Mosek Env

Description:

A Mosek Environment

15.3.1 Methods

- MSK_callocdbgenv

Syntax:

```
void * MSKAPI MSK_callocdbgenv (
    MSKenv_t env,
    MSKCONST size_t number,
    MSKCONST size_t size,
    MSKCONST char * file,
    MSKCONST unsigned line);
```

Arguments:

`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_callocdbgtask

Syntax:

```
void * MSKAPI MSK_callocdbgtask (
    MSKtask_t task,
    MSKCONST size_t number,
    MSKCONST size_t size,
    MSKCONST char * file,
    MSKCONST unsigned line);
```

Arguments:

`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_callocenv`

Syntax:

```
void * MSKAPI MSK_callocenv (
    MSKenv_t env,
    MSKCONST size_t number,
    MSKCONST size_t size);
```

Arguments:

`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_checkmemenv`

Syntax:

```
MSKrescodee MSKAPI MSK_checkmemenv (
    MSKenv_t env,
    MSKCONST char * file,
    MSKintt line);
```

Arguments:

`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_checkversion`

Syntax:

```
MSKrescodee MSKAPI MSK_checkversion (
    MSKenv_t env,
    MSKintt major,
    MSKintt minor,
    MSKintt build,
    MSKintt revision);
```

Arguments:

`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 MSKAPI MSK_deleteenv (MSKenv_t * env)
```

Arguments:

`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 MSKAPI MSK_echoenv (
    MSKenv_t env,
    MSKstreamtypee whichstream,
    MSKCONST char * format,
    ...);
```

Arguments:

`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 MSKAPI MSK_echointro (
    MSKenv_t env,
    MSKintt longver);
```

Arguments:

`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 MSKAPI MSK_freedbgenv (
    MSKenv_t env,
    MSKCONST void * buffer,
    MSKCONST char * file,
    MSKCONST unsigned line);
```

Arguments:

`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 MSKAPI MSK_freeenv (
    MSKenv_t env,
    MSKCONST void * buffer);
```

Arguments:

`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 MSKAPI MSK_getbuildinfo (
    char * buildstate,
    char * builddate,
    char * buildtool);
```

Arguments:

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_getcodedisc

Syntax:

```
MSKrescodee MSKAPI MSK_getcodedisc (
    MSKrescodee code,
    char * symname,
    char * str);
```

Arguments:

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 MSKAPI MSK_getglbdllname (
    MSKenv_t env,
    MSKCONST size_t sizedllname,
    char * dllname);
```

Arguments:

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 MSKAPI MSK_getresponseclass (
    MSKrescodee r,
    MSKrescodetypee * rc);
```

Arguments:

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 MSKAPI MSK_getsymbcondim (
    MSKenv_t env,
    MSKintt * num,
    size_t * maxlen);
```

  Arguments:

  `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 MSKAPI MSK_getversion (
    MSKintt * major,
    MSKintt * minor,
    MSKintt * build,
    MSKintt * revision);
```

  Arguments:

  `major` **(output)** Major version number. Modified only if a non-null pointer.

  `minor` **(output)** Minor version number. Modified only if a non-null pointer.

  `build` **(output)** Build number. Modified only if a non-null pointer.

  `revision` **(output)** Revision number. Modified only if a non-null pointer.

  Description: Obtains MOSEK version information.

- MSK_initenv

  Syntax:

```
MSKrescodee MSKAPI MSK_initenv (MSKenv_t env)
```

  Arguments:

  `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 MSKAPI MSK_iparvaltosymnam (
    MSKenv_t env,
    MSKiparame whichparam,
    MSKintt whichvalue,
    char * symbolicname);
```

Arguments:

`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 MSKAPI MSK_isinfinity (MSKrealt value)
```

Arguments:

`value`

Description: Return true if `value` considered infinity by MOSEK

- `MSK_linkfiletoenvstream`

Syntax:

```
MSKrescodee MSKAPI MSK_linkfiletoenvstream (
    MSKenv_t env,
    MSKstreamtypee whichstream,
    MSKCONST char * filename,
    MSKintt append);
```

Arguments:

`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 MSKAPI MSK_linkfunctoenvstream (
    MSKenv_t env,
    MSKstreamtypee whichstream,
    MSKuserhandle_t handle,
    MSKstreamfunc func);
```

Arguments:

`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 MSKAPI MSK_makeemptytask (
    MSKenv_t env,
    MSKtask_t * task);
```

Arguments:

`env` **(input)** The MOSEK environment.

`task` **(output)** An optimization task.

Description: Creates a new optimization task.

- MSK_makeenv

Syntax:

```
MSKrescodee MSKAPI MSK_makeenv (
    MSKenv_t * env,
    MSKuserhandle_t usrptr,
    MSKmallocfunc usrmalloc,
    MSKfreefunc usrfree,
    MSKCONST char * dbgfile);
```

Arguments:

`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 MSKAPI MSK_maketask (
    MSKenv_t env,
    MSKintt maxnumcon,
    MSKintt maxnumvar,
    MSKtask_t * task);
```

Arguments:

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 MSKAPI MSK_putcpudefaults (
    MSKenv_t env,
    int cputype,
    MSKintt sizel1,
    MSKintt sizel2);
```

Arguments:

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_putctrlcfunc

Syntax:

```
MSKrescodee MSKAPI MSK_putctrlcfunc (
    MSKenv_t env,
    MSKctrlcfunc ctrlcfunc,
    MSKuserhandle_t handle);
```

Arguments:

`env` **(input)** The MOSEK environment.

`ctrlcfunc` **(input)** A user-defined ctrl-c function.

`handle` **(input)** A pointer to some user-defined data structure (or a NULL pointer). This pointer is passed to `ctrlcfunc` whenever it is called.

Description: The function is used to input a user-defined `ctrl-c` function which is called occasionally during the optimization process. If the `ctrl-c` function returns a non-zero value, then MOSEK terminates the optimization process and returns with the return code `MSK_RES_TRM_USER_BREAK`.

Please note that the function is only called if the parameter `MSK_IPAR_CHECK_CTRL_C` is set to `MSK_ON`.

- `MSK_putdllpath`

Syntax:

```
MSKrescodee MSKAPI MSK_putdllpath (
    MSKenv_t env,
    MSKCONST char * dllpath);
```

Arguments:

`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 MSKAPI MSK_putexitfunc (
    MSKenv_t env,
    MSKexitfunc exitfunc,
    MSKuserhandle_t handle);
```

Arguments:

`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 MSKAPI MSK_putkeepdlls (
    MSKenv_t env,
    MSKintt keepdlls);
```

Arguments:

`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 MSKAPI MSK_putlicensedefaults (
    MSKenv_t env,
    MSKCONST char * licensefile,
    MSKCONST MSKintt * licensebuf,
    MSKintt licwait,
    MSKintt licdebug);
```

Arguments:

`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 used as the default value for

MSK_IPAR_LICENSE_PAUSE_TIME

.

`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 MSKAPI MSK_replacefileext (
    char * filename,
    MSKCONST char * newextension);
```

Arguments:

`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 * MSKAPI MSK_strdupdbgenv (
    MSKenv_t env,
    MSKCONST char * str,
    MSKCONST char * file,
    MSKCONST unsigned line);
```

Arguments:

`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 * MSKAPI MSK_strdupenv (
    MSKenv_t env,
    MSKCONST char * str);
```

Arguments:

`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 MSKAPI MSK_symnamtovalue (
    MSKCONST char * name,
    char * value);
```

Arguments:

`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 MSKAPI MSK_unlinkfuncfromenvstream (
    MSKenv_t env,
    MSKstreamtypee whichstream);
```

Arguments:

`env` **(input)** The MOSEK environment.

`whichstream` **(input)** Index of the stream.

Description: Disconnects a user-defined function from a stream.

- MSK_utf8towchar

Syntax:

```
MSKrescodee MSKAPI MSK_utf8towchar (
    MSKCONST size_t outputlen,
    size_t * len,
    size_t * conv,
    MSKwchart * output,
    MSKCONST char * input);
```

Arguments:

`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 chars 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 MSKAPI MSK_wchartoutf8 (
    MSKCONST size_t outputlen,
    size_t * len,
    size_t * conv,
    char * output,
    MSKCONST MSKwchart * input);
```

Arguments:

`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 chars 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_append

Syntax:

```
MSKrescodee MSKAPI MSK_append (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKintt num);
```

Arguments:

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 MSKAPI MSK_appendcone (
    MSKtask_t task,
    MSKconetypee conetype,
    MSKrealt conepar,
    MSKintt nummem,
    MSKCONST MSKidxt * submem);
```

Arguments:

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

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

to the problem where $\mathcal{C}$ is a convex cone. x^t is a subset of the variables which will be specified by the argument submem. 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_appendcons

Syntax:

```
MSKrescodee MSKAPI MSK_appendcons (
    MSKtask_t task,
    MSKintt num,
    MSKCONST MSKlidxt * aptrb,
    MSKCONST MSKlidxt * aptre,
    MSKCONST MSKidxt * asub,
    MSKCONST MSKrealt * aval,
```

```
    MSKCONST MSKboundkeye * bkc,
    MSKCONST MSKrealt * blc,
    MSKCONST MSKrealt * buc);
```

Arguments:

`task` **(input)** An optimization task.

`num` **(input)** Number of constraints to be appended.

`aptrb` **(input)** See (15.5).

`aptre` **(input)** See (15.5).

`asub` **(input)** Variable subscripts of the new A coefficients. See (15.5).

`aval` **(input)** A coefficients of the new constraints. See (15.5).

`bkc` **(input)** Bound keys for constraints to be appended. See (15.4).

`blc` **(input)** Lower bounds on constraints to be appended. See (15.4).

`buc` **(input)** Upper bounds on constraints to be appended. See (15.4).

Description: The function appends one or more constraints to the optimization task. The bounds and A are modified as follows

$$\begin{array}{rcl} l^c_{\mathtt{numcon+k}} & = & \mathtt{blc}[\mathtt{k}], \quad \mathtt{k} = 0,\ldots,\mathtt{num}-1, \\ u^c_{\mathtt{numcon+k}} & = & \mathtt{buc}[\mathtt{k}], \quad \mathtt{k} = 0,\ldots,\mathtt{num}-1, \end{array} \tag{15.4}$$

and

$$a_{\mathtt{numcon+k},\mathtt{asub}[\mathtt{l}]} = \mathtt{aval}[\mathtt{l}], \quad k = 0,\ldots,\mathtt{num}-1, \quad l = \mathtt{aptrb}[\mathtt{k}],\ldots,\mathtt{aptre}[\mathtt{k}]-1. \tag{15.5}$$

See also:

`MSK_putmaxnumcon` Sets the number of preallocated constraints in the optimization task.

- `MSK_appendstat`

Syntax:

```
MSKrescodee MSKAPI MSK_appendstat (MSKtask_t task)
```

Arguments:

`task` **(input)** An optimization task.

Description: Appends a record to the statistics file.

- `MSK_appendvars`

Syntax:

```
MSKrescodee MSKAPI MSK_appendvars (
    MSKtask_t task,
    MSKintt num,
    MSKCONST MSKrealt * cval,
    MSKCONST MSKlidxt * aptrb,
    MSKCONST MSKlidxt * aptre,
    MSKCONST MSKidxt * asub,
```

```
    MSKCONST MSKrealt * aval,
    MSKCONST MSKboundkeye * bkx,
    MSKCONST MSKrealt * blx,
    MSKCONST MSKrealt * bux);
```

Arguments:

`task` **(input)** An optimization task.

`num` **(input)** Number of variables to be appended.

`cval` **(input)** Values of c for the variables to be appended. See (15.6).

`aptrb` **(input)** See (15.7).

`aptre` **(input)** See (15.7).

`asub` **(input)** Constraint subscripts of the A coefficients to be added. See (15.7).

`aval` **(input)** The A coefficients corresponding to the appended variables. See (15.7).

`bkx` **(input)** Bound keys on variables to be appended. See (15.6).

`blx` **(input)** Lower bounds on variables to be appended. See (15.6).

`bux` **(input)** Upper bounds on variables to be appended. See (15.6).

Description: The function appends one or more variables to the optimization problem. Moreover, the function initializes c, A and the bounds corresponding to the appended variables as follows

$$\begin{array}{rcll} c_{\mathtt{numvar}+\mathtt{k}} & = & \mathtt{cval}[\mathtt{k}], & \mathtt{k} = 0,\ldots,\mathtt{num}-1, \\ l^x_{\mathtt{numvar}+\mathtt{k}} & = & \mathtt{blx}[\mathtt{k}], & \mathtt{k} = 0,\ldots,\mathtt{num}-1, \\ u^x_{\mathtt{numvar}+\mathtt{k}} & = & \mathtt{bux}[\mathtt{k}], & \mathtt{k} = 0,\ldots,\mathtt{num}-1, \end{array} \tag{15.6}$$

and

$$a_{\mathtt{asub}[\mathtt{l}],\mathtt{numvar}+\mathtt{k}} \quad = \quad \mathtt{aval}[\mathtt{l}], \quad k = 0,\ldots,\mathtt{num}-1, \quad l = \mathtt{aptrb}[\mathtt{k}],\ldots,\mathtt{aptre}[\mathtt{k}]-1 \tag{15.7}$$

where `numvar` is the number variables before the new variables are appended.

See also:

`MSK_putmaxnumvar` Sets the number of preallocated variables in the optimization task.

- `MSK_bktostr`

Syntax:

```
MSKrescodee MSKAPI MSK_bktostr (
    MSKtask_t task,
    MSKboundkeye bk,
    char * str);
```

Arguments:

`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 MSKAPI MSK_callbackcodetostr (
    MSKcallbackcodee code,
    char * callbackcodestr);
```

Arguments:

`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_calloctask

Syntax:

```
void * MSKAPI MSK_calloctask (
    MSKtask_t task,
    MSKCONST size_t number,
    MSKCONST size_t size);
```

Arguments:

`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 MSKAPI MSK_checkconvexity (MSKtask_t task)
```

Arguments:

`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 `MSK_RES_ERR_NONCONVEX` if the problem is not convex.

See also:

`MSK_IPAR_CHECK_CONVEXITY`

- MSK_checkdata

Syntax:

```
MSKrescodee MSKAPI MSK_checkdata (MSKtask_t task)
```

Arguments:

`task` **(input)** An optimization task.

Description: Checks the data of the optimization task.

- `MSK_checkmemtask`

Syntax:

```
MSKrescodee MSKAPI MSK_checkmemtask (
    MSKtask_t task,
    MSKCONST char * file,
    MSKintt line);
```

Arguments:

`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 MSKAPI MSK_chgbound (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKintt lower,
    MSKintt finite,
    MSKrealt value);
```

Arguments:

`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 MSKAPI MSK_clonetask (
    MSKtask_t task,
    MSKtask_t * clonedtask);
```

Arguments:

`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 MSKAPI MSK_commitchanges (MSKtask_t task)
```

Arguments:

`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 MSKAPI MSK_conetypetostr (
    MSKtask_t task,
    MSKconetypee conetype,
    char * str);
```

Arguments:

`task` **(input)** An optimization task.

`conetype` **(input)** Specifies the type of the cone.

`str` **(output)** String corresponding to the cone type code `codetype`.

Description: Obtains the cone string identifier corresponding to a cone type.

- `MSK_deletesolution`

Syntax:

```
MSKrescodee MSKAPI MSK_deletesolution (
    MSKtask_t task,
    MSKsoltypee whichsol);
```

Arguments:

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

Description: Undefines a solution and frees the memory it uses.

- `MSK_deletetask`

Syntax:

```
MSKrescodee MSKAPI MSK_deletetask (MSKtask_t * task)
```

Arguments:

`task` **(input/output)** An optimization task.

Description: Deletes a task.

- `MSK_dualsensitivity`

Syntax:

```
MSKrescodee MSKAPI MSK_dualsensitivity (
    MSKtask_t task,
    MSKlintt numj,
    MSKCONST MSKidxt * subj,
    MSKrealt * leftpricej,
    MSKrealt * rightpricej,
    MSKrealt * leftrangej,
    MSKrealt * rightrangej);
```

Arguments:

`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 MSKAPI MSK_echotask (
    MSKtask_t task,
    MSKstreamtypee whichstream,
    MSKCONST char * format,
    ...);
```

Arguments:

`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 MSKAPI MSK_exceptiontask (
    MSKtask_t task,
    MSKrescodee code,
    ...);
```

Arguments:

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 MSKAPI MSK_freedbgtask (
    MSKtask_t task,
    MSKCONST void * buffer,
    MSKCONST char * file,
    MSKCONST unsigned line);
```

Arguments:

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 MSKAPI MSK_freetask (
    MSKtask_t task,
    MSKCONST void * buffer);
```

Arguments:

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 MSKAPI MSK_getaij (
    MSKtask_t task,
    MSKidxt i,
    MSKidxt j,
    MSKrealt * aij);
```

Arguments:

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 MSKAPI MSK_getapiecenumnz (
    MSKtask_t task,
    MSKidxt firsti,
    MSKidxt lasti,
    MSKidxt firstj,
    MSKidxt lastj,
    MSKlintt * numnz);
```

Arguments:

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 A.

MSK_getaslicenumnz Obtains the number of non-zeros in a row or column slice of A.

- MSK_getaslice

Syntax:

```
MSKrescodee MSKAPI MSK_getaslice (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt first,
    MSKidxt last,
```

```
    MSKlintt maxnumnz,
    MSKlintt * surp,
    MSKlidxt * ptrb,
    MSKlidxt * ptre,
    MSKidxt * sub,
    MSKrealt * val);
```

Arguments:

`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 variable in the sequence.

`last` **(input)** Index of the last row or variable plus one 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 numerical elements.

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 A.

- `MSK_getaslicenumnz`

Syntax:

```
MSKrescodee MSKAPI MSK_getaslicenumnz (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt first,
    MSKidxt last,
    MSKlintt * numnz);
```

Arguments:

`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 variable in the sequence.

`last` **(input)** Index of the last row or variable plus one in the sequence **plus one**.

`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_getaslicetrip

Syntax:

```
MSKrescodee MSKAPI MSK_getaslicetrip (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt first,
    MSKidxt last,
    MSKlintt maxnumnz,
    MSKlintt * surp,
    MSKidxt * subi,
    MSKidxt * subj,
    MSKrealt * val);
```

Arguments:

`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 variable in the sequence.

`last` **(input)** Index of the last row or variable 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

$$p^1 = \mathtt{maxnumnz} - \mathtt{surp}[0]$$

when the function is called and p^2 by

$$p^2 = \mathtt{maxnumnz} - \mathtt{surp}[0],$$

where `surp[0]` is the value upon termination. Using this notation then

$$\mathtt{val}[k] = a_{\mathtt{subi[k]},\mathtt{subj[k]}}, \quad k = p^1, \ldots, p^2 - 1.$$

See also:

`MSK_getaslicenumnz` Obtains the number of non-zeros in a row or column slice of A.

- MSK_getavec

Syntax:

```
MSKrescodee MSKAPI MSK_getavec (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKintt * nzi,
    MSKidxt * subi,
    MSKrealt * vali);
```

Arguments:

`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 MSKAPI MSK_getavecnumnz (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKintt * nzj);
```

Arguments:

`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 MSKAPI MSK_getbound (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKboundkeye * bk,
    MSKrealt * bl,
    MSKrealt * bu);
```

Arguments:

`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 MSKAPI MSK_getboundslice (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt first,
    MSKidxt last,
    MSKboundkeye * bk,
    MSKrealt * bl,
    MSKrealt * bu);
```

Arguments:

`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 MSKAPI MSK_getc (
    MSKtask_t task,
    MSKrealt * c);
```

Arguments:

`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 MSKAPI MSK_getcallbackfunc (
    MSKtask_t task,
    MSKcallbackfunc * func,
    MSKuserhandle_t * handle);
```

Arguments:

`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 MSKAPI MSK_getcfix (
    MSKtask_t task,
    MSKrealt * cfix);
```

Arguments:

`task` **(input)** An optimization task.

`cfix` **(output)** Fixed term in the objective.

Description: Obtains the fixed term in the objective.

- `MSK_getcone`

Syntax:

```
MSKrescodee MSKAPI MSK_getcone (
    MSKtask_t task,
    MSKidxt k,
    MSKconetypee * conetype,
```

```
        MSKrealt * conepar,
        MSKintt * nummem,
        MSKidxt * submem);
```

Arguments:

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 MSKAPI MSK_getconeinfo (
    MSKtask_t task,
    MSKidxt k,
    MSKconetypee * conetype,
    MSKrealt * conepar,
    MSKintt * nummem);
```

Arguments:

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 MSKAPI MSK_getconname (
    MSKtask_t task,
    MSKidxt i,
    MSKCONST size_t maxlen,
    char * name);
```

Arguments:

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 of any objective, constraint, variable or cone name.

- `MSK_getcslice`

Syntax:

```
MSKrescodee MSKAPI MSK_getcslice (
    MSKtask_t task,
    MSKidxt first,
    MSKidxt last,
    MSKrealt * c);
```

Arguments:

`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_getdouinf`

Syntax:

```
MSKrescodee MSKAPI MSK_getdouinf (
    MSKtask_t task,
    MSKdinfiteme whichdinf,
    MSKrealt * dvalue);
```

Arguments:

`task` **(input)** An optimization task.

`whichdinf` **(input)** A `double` information item. See section 18.11 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 MSKAPI MSK_getdouparam (
    MSKtask_t task,
    MSKdparame param,
    MSKrealt * parvalue);
```

Arguments:

`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 MSKAPI MSK_getdualobj (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * dualobj);
```

Arguments:

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 MSKAPI MSK_getenv (
    MSKtask_t task,
    MSKenv_t * env);
```

Arguments:

task **(input)** An optimization task.

env **(output)** The MOSEK environment.

Description: Obtains the environment used to create the task.

- MSK_getinfeasiblesubproblem

Syntax:

```
MSKrescodee MSKAPI MSK_getinfeasiblesubproblem (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKtask_t * inftask);
```

Arguments:

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.

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 MSKAPI MSK_getinfindex (
    MSKtask_t task,
    MSKinftypee inftype,
    MSKCONST char * infname,
    MSKintt * infindex);
```

Arguments:

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 MSKAPI MSK_getinfmax (
    MSKtask_t task,
    MSKinftypee inftype,
    MSKintt * infmax);
```

Arguments:

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 MSKAPI MSK_getinfname (
    MSKtask_t task,
    MSKinftypee inftype,
    MSKintt whichinf,
    char * infname);
```

Arguments:

task **(input)** An optimization task.

`inftype` **(input)** Type of the information item.

`whichinf` **(input)** An information item. See section 18.11 and section 18.14 for the possible values.

`infname` **(output)** Name of the information item.

Description: Obtains the name of an information item.

- `MSK_getintinf`

Syntax:

```
MSKrescodee MSKAPI MSK_getintinf (
    MSKtask_t task,
    MSKiinfiteme whichiinf,
    MSKintt * ivalue);
```

Arguments:

`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 MSKAPI MSK_getintparam (
    MSKtask_t task,
    MSKiparame param,
    MSKintt * parvalue);
```

Arguments:

`task` **(input)** An optimization task.

`param` **(input)** Which parameter.

`parvalue` **(output)** Parameter value.

Description: Obtains the value of an integer parameter.

- `MSK_getlasterror`

Syntax:

```
MSKrescodee MSKAPI MSK_getlasterror (
    MSKtask_t task,
    MSKrescodee * lastrescode,
    MSKCONST size_t maxlen,
    size_t * lastmsglen,
    char * lastmsg);
```

Arguments:

`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:

- MSK_getmaxnamelen

Syntax:

```
MSKrescodee MSKAPI MSK_getmaxnamelen (
    MSKtask_t task,
    size_t * maxlen);
```

Arguments:

task **(input)** An optimization task.

maxlen **(output)** The maximum length of any name.

Description: Obtains the maximum length of any objective, constraint, variable or cone name.

- MSK_getmaxnumanz

Syntax:

```
MSKrescodee MSKAPI MSK_getmaxnumanz (
    MSKtask_t task,
    MSKlintt * maxnumanz);
```

Arguments:

task **(input)** An optimization task.

maxnumanz **(output)** Number of preallocated non-zero elements in A.

Description: Obtains number of preallocated non-zeros for A. When this number of non-zeros is reached MOSEK will automatically allocate more space for A.

- MSK_getmaxnumcon

Syntax:

```
MSKrescodee MSKAPI MSK_getmaxnumcon (
    MSKtask_t task,
    MSKintt * maxnumcon);
```

Arguments:

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 MSKAPI MSK_getmaxnumcone (
    MSKtask_t task,
    MSKintt * maxnumcone);
```

Arguments:

`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 MSKAPI MSK_getmaxnumqnz (
    MSKtask_t task,
    MSKintt * maxnumqnz);
```

Arguments:

`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 MSKAPI MSK_getmaxnumvar (
    MSKtask_t task,
    MSKintt * maxnumvar);
```

Arguments:

`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 MSKAPI MSK_getmemusagetask (
    MSKtask_t task,
    size_t * meminuse,
    size_t * maxmemuse);
```

Arguments:

`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 MSKAPI MSK_getnadouinf (
    MSKtask_t task,
    MSKCONST char * whichdinf,
    MSKrealt * dvalue);
```

Arguments:

`task` **(input)** An optimization task.

`whichdinf` **(input)** A `double` information item. See section 18.11 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 MSKAPI MSK_getnadouparam (
    MSKtask_t task,
    MSKCONST char * paramname,
    MSKrealt * parvalue);
```

Arguments:

`task` **(input)** An optimization task.

`paramname` **(input)** Name of a MOSEK parameter.

`parvalue` **(output)** Parameter value.

Description: Obtains the value of a named double parameter.

- `MSK_getnaintinf`

Syntax:

```
MSKrescodee MSKAPI MSK_getnaintinf (
    MSKtask_t task,
    MSKCONST char * infitemname,
    MSKintt * ivalue);
```

Arguments:

`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 MSKAPI MSK_getnaintparam (
    MSKtask_t task,
    MSKCONST char * paramname,
    MSKintt * parvalue);
```

Arguments:

`task` **(input)** An optimization task.

`paramname` **(input)** Name of a MOSEK parameter.

`parvalue` **(output)** Parameter value.

Description: Obtains the value of a named integer parameter.

- MSK_getname

Syntax:

```
MSKrescodee MSKAPI MSK_getname (
    MSKtask_t task,
    MSKproblemiteme whichitem,
    MSKidxt i,
    MSKCONST size_t maxlen,
    size_t * len,
    char * name);
```

Arguments:

`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`.

`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 of any objective, constraint, variable or cone name.

- MSK_getnameindex

Syntax:

```
MSKrescodee MSKAPI MSK_getnameindex (
    MSKtask_t task,
    MSKproblemiteme whichitem,
    MSKCONST char * name,
    MSKintt * asgn,
    MSKidxt * index);
```

Arguments:

`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_getnastrparam`

Syntax:

```
MSKrescodee MSKAPI MSK_getnastrparam (
    MSKtask_t task,
    MSKCONST char * paramname,
    MSKCONST size_t maxlen,
    size_t * len,
    char * parvalue);
```

Arguments:

`task` **(input)** An optimization task.

`paramname` **(input)** Name of a MOSEK 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 MSKAPI MSK_getnastrparamal (
    MSKtask_t task,
    MSKCONST char * paramname,
    MSKCONST size_t numaddchr,
    MSKstring_t * value);
```

Arguments:

`task` **(input)** An optimization task.

`paramname` **(input)** Name of a MOSEK parameter.

`numaddchr` **(input)** Number of additional chars 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 MSKAPI MSK_getnlfunc (
    MSKtask_t task,
    MSKuserhandle_t * nlhandle,
    MSKnlgetspfunc * nlgetsp,
    MSKnlgetvafunc * nlgetva);
```

Arguments:

`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 MSKAPI MSK_getnumanz (
    MSKtask_t task,
    MSKlintt * numanz);
```

Arguments:

`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 MSKAPI MSK_getnumcon (
    MSKtask_t task,
    MSKintt * numcon);
```

Arguments:

`task` **(input)** An optimization task.

`numcon` **(output)** Number of constraints.

Description: Obtains the number of constraints.

- `MSK_getnumcone`

Syntax:

```
MSKrescodee MSKAPI MSK_getnumcone (
    MSKtask_t task,
    MSKintt * numcone);
```

Arguments:

`task` **(input)** An optimization task.

`numcone` **(output)** Number conic constraints.

Description: Obtains the number of cones.

- `MSK_getnumconemem`

Syntax:

```
MSKrescodee MSKAPI MSK_getnumconemem (
    MSKtask_t task,
    MSKidxt k,
    MSKintt * nummem);
```

Arguments:

`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 MSKAPI MSK_getnumintvar (
    MSKtask_t task,
    MSKintt * numintvar);
```

Arguments:

`task` **(input)** An optimization task.

`numintvar` **(output)** Number of integer variables.

Description: Obtains the number of integer constrained variables.

- `MSK_getnumparam`

Syntax:

```
MSKrescodee MSKAPI MSK_getnumparam (
    MSKtask_t task,
    MSKparametertypee partype,
    MSKintt * numparam);
```

Arguments:

`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_getnumqconnz

Syntax:

```
MSKrescodee MSKAPI MSK_getnumqconnz (
    MSKtask_t task,
    MSKidxt i,
    MSKlintt * numqcnz);
```

Arguments:

`task` **(input)** An optimization task.

`i` **(input)** Index of the constraint for which the quadratic terms should be obtained.

`numqcnz` **(output)** Number of quadratic terms. See (5.36).

Description: Obtains the number of non-zero quadratic terms in a constraint.

- MSK_getnumqobjnz

Syntax:

```
MSKrescodee MSKAPI MSK_getnumqobjnz (
    MSKtask_t task,
    MSKlintt * numqonz);
```

Arguments:

`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 MSKAPI MSK_getnumvar (
    MSKtask_t task,
    MSKintt * numvar);
```

Arguments:

`task` **(input)** An optimization task.

`numvar` **(output)** Number of variables.

Description: Obtains the number of variables.

- `MSK_getobjname`

Syntax:

```
MSKrescodee MSKAPI MSK_getobjname (
    MSKtask_t task,
    MSKCONST size_t maxlen,
    size_t * len,
    char * objname);
```

Arguments:

`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 MSKAPI MSK_getobjsense (
    MSKtask_t task,
    MSKobjsensee * sense);
```

Arguments:

`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 MSKAPI MSK_getparammax (
    MSKtask_t task,
    MSKparametertypee partype,
    MSKCONST MSKintt * parammax);
```

Arguments:

`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 MSKAPI MSK_getparamname (
    MSKtask_t task,
    MSKparametertypee partype,
    MSKintt param,
    char * parname);
```

Arguments:

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_getprimalobj

Syntax:

```
MSKrescodee MSKAPI MSK_getprimalobj (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * primalobj);
```

Arguments:

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 MSKAPI MSK_getprobtype (
    MSKtask_t task,
    MSKproblemtypee * probtype);
```

Arguments:

task **(input)** An optimization task.

probtype **(output)** The problem type.

Description: Obtains the problem type.

- MSK_getqconk

Syntax:

```
MSKrescodee MSKAPI MSK_getqconk (
    MSKtask_t task,
    MSKidxt k,
    MSKlintt maxnumqcnz,
    MSKlintt * qcsurp,
    MSKlintt * numqcnz,
    MSKidxt * qcsubi,
    MSKidxt * qcsubj,
    MSKrealt * qcval);
```

Arguments:

`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 (5.36).

`qcsubi` **(output)** i subscripts for q_{ij}^k. See (5.36).

`qcsubj` **(output)** j subscripts for q_{ij}^k. See (5.36).

`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 MSKAPI MSK_getqobj (
    MSKtask_t task,
    MSKlintt maxnumqonz,
    MSKlintt * qosurp,
    MSKlintt * numqonz,
    MSKidxt * qosubi,
    MSKidxt * qosubj,
    MSKrealt * qoval);
```

Arguments:

`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 MSKAPI MSK_getqobjij (
    MSKtask_t task,
    MSKidxt i,
    MSKidxt j,
    MSKrealt * qoij);
```

Arguments:

`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 MSKAPI MSK_getreducedcosts (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKidxt first,
    MSKidxt last,
    MSKrealt * redcosts);
```

Arguments:

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

`first` **(input)** See formula (15.8) for the definition.

`last` **(input)** See formula (15.8) 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.

$$\texttt{redcosts}[j - \texttt{first} + 0] = (s_l^x)_j - (s_u^x)_j, \ j = \texttt{first}, \ldots, \texttt{last} - 1. \tag{15.8}$$

- MSK_getsolution

Syntax:

```
MSKrescodee MSKAPI 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);
```

Arguments:

`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 (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

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x. \end{array} \tag{15.9}$$

and the corresponding dual problem is

$$
\begin{array}{lccc}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\
 & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\
\text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\
 & -y + s_l^c - s_u^c & = & 0, \\
 & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & &
\end{array} \tag{15.10}
$$

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_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 MSKAPI MSK_getsolutioni (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKsoltypee whichsol,
    MSKstakeye * sk,
    MSKrealt * x,
    MSKrealt * sl,
    MSKrealt * su,
    MSKrealt * sn);
```

Arguments:

`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.

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 MSKAPI 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);
```

Arguments:

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 (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 MSKAPI 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);
```

Arguments:

task **(input)** An optimization task.
whichsol **(input)** Selects a solution.
prosta **(output)** Problem status.
solsta **(output)** Solution status.
primalobj **(output)** Objective value corresponding to the primal solution.
maxpbi **(output)** Maximum primal bound infeasibility.
maxpcni **(output)** Maximum infeasibility in the primal conic constraints.
maxpeqi **(output)** Maximum infeasibility in the primal equality constraints.
maxinti **(output)** Maximum infeasibility in integer constraints.
dualobj **(output)** Objective value corresponding to the dual solution.
maxdbi **(output)** Maximum dual bound infeasibility.

`maxdcni` **(output)** Maximum infeasibility in the dual conic constraints.

`maxdeqi` **(output)** Maximum infeasibility in the dual equality constraints.

Description: Obtains information about a solution.

- `MSK_getsolutionslice`

Syntax:

```
MSKrescodee MSKAPI MSK_getsolutionslice (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKsoliteme solitem,
    MSKidxt first,
    MSKidxt last,
    MSKrealt * values);
```

Arguments:

`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.h. 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

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x. \end{array} \tag{15.11}$$

and the corresponding dual problem is

$$\begin{array}{lccc} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & \end{array} \tag{15.12}$$

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:

$$
\begin{array}{lccl}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\
 & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\
\text{subject to} & A^T y + s_l^x - s_u^x + s_n^x & = & c, \\
 & -y + s_l^c - s_u^c & = & 0, \\
 & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, \\
 & s_n^x \in \mathcal{C}^* & &
\end{array}
\tag{15.13}
$$

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_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 MSKAPI MSK_getsolutionstatus (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKprostae * prosta,
    MSKsolstae * solsta);
```

Arguments:

`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 MSKAPI MSK_getsolutionstatuskeyslice (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKsoltypee whichsol,
    MSKidxt first,
    MSKidxt last,
    MSKstakeye * sk);
```

Arguments:

`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 MSKAPI MSK_getstrparam (
    MSKtask_t task,
    MSKsparame param,
    MSKCONST size_t maxlen,
    size_t * len,
    char * parvalue);
```

Arguments:

`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 MSKAPI MSK_getstrparamal (
    MSKtask_t task,
    MSKsparame param,
    MSKCONST size_t numaddchr,
    MSKstring_t * value);
```

Arguments:

`task` **(input)** An optimization task.

`param` **(input)** Which parameter.

`numaddchr` **(input)** Number of additional chars 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 MSKAPI MSK_getsymbcon (
    MSKtask_t task,
    MSKidxt i,
    MSKCONST size_t maxlen,
    char * name,
    MSKintt * value);
```

Arguments:

`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 MSKAPI MSK_gettaskname (
    MSKtask_t task,
    MSKCONST size_t maxlen,
    size_t * len,
    char * taskname);
```

Arguments:

`task` **(input)** An optimization task.

`maxlen` **(input)** Length of the array `taskname`.

`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 MSKAPI MSK_getvarbranchdir (
    MSKtask_t task,
    MSKidxt j,
    MSKbranchdire * direction);
```

Arguments:

`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 MSKAPI MSK_getvarbranchorder (
    MSKtask_t task,
    MSKidxt j,
    MSKintt * priority,
    MSKbranchdire * direction);
```

Arguments:

`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 variable j.

Description: Obtains the branching priority and direction for a given variable j.

- `MSK_getvarbranchpri`

Syntax:

```
MSKrescodee MSKAPI MSK_getvarbranchpri (
    MSKtask_t task,
    MSKidxt j,
    MSKintt * priority);
```

Arguments:

`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 MSKAPI MSK_getvarname (
    MSKtask_t task,
    MSKidxt i,
    MSKCONST size_t maxlen,
    char * name);
```

Arguments:

`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 of any objective, constraint, variable or cone name.

- MSK_getvartype

Syntax:

```
MSKrescodee MSKAPI MSK_getvartype (
    MSKtask_t task,
    MSKidxt j,
    MSKvariabletypee * vartype);
```

Arguments:

`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 MSKAPI MSK_getvartypelist (
    MSKtask_t task,
    MSKintt num,
    MSKCONST MSKidxt * subj,
    MSKvariabletypee * vartype);
```

Arguments:

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

Arguments:

`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.

- MSK_inputdata

Syntax:

```
MSKrescodee MSKAPI MSK_inputdata (
    MSKtask_t task,
    MSKintt maxnumcon,
    MSKintt maxnumvar,
    MSKintt numcon,
    MSKintt numvar,
    MSKCONST MSKrealt * c,
    MSKrealt cfix,
    MSKCONST MSKlidxt * aptrb,
    MSKCONST MSKlidxt * 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);
```

Arguments:

- `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 (5.37) and section 5.8.3.
- `aptre` **(input)** Pointers to the last element + 1 in the rows or the columns of A. See (5.37) and section 5.8.3
- `asub` **(input)** Coefficient subscripts. See (5.37) and section 5.8.3.
- `aval` **(input)** Coefficient values. See (5.37) 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 MSKAPI MSK_isdouparname (
    MSKtask_t task,
    MSKCONST char * parname,
    MSKdparame * param);
```

Arguments:

- `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 MSKAPI MSK_isintparname (
    MSKtask_t task,
    MSKCONST char * parname,
    MSKiparame * param);
```

Arguments:

`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 MSKAPI MSK_isstrparname (
    MSKtask_t task,
    MSKCONST char * parname,
    MSKsparame * param);
```

Arguments:

`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 MSKAPI MSK_linkfiletotaskstream (
    MSKtask_t task,
    MSKstreamtypee whichstream,
    MSKCONST char * filename,
    MSKintt append);
```

Arguments:

`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 MSKAPI MSK_linkfunctotaskstream (
    MSKtask_t task,
    MSKstreamtypee whichstream,
    MSKuserhandle_t handle,
    MSKstreamfunc func);
```

Arguments:

`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 MSKAPI MSK_makesolutionstatusunknown (
    MSKtask_t task,
    MSKsoltypee whichsol);
```

Arguments:

`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_optimize`

Syntax:

```
MSKrescodee MSKAPI MSK_optimize (MSKtask_t task)
```

Arguments:

`task` **(input)** An optimization task.

Description: Calls the optimizer. Depending on the problem type and the selected solver this will call one of the solvers in MOSEK.

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 MSKAPI MSK_optimizeconcurrent (
    MSKtask_t task,
    MSKCONST MSKtask_t * taskarray,
    MSKintt num);
```

Arguments:

`task` **(input)** An optimization task.

`taskarray` **(input)** An array of `num` tasks.

`num` **(input)** Length of `taskarrays`

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_optimizetrm`

Syntax:

```
MSKrescodee MSKAPI MSK_optimizetrm (
    MSKtask_t task,
    MSKrescodee * trmcode);
```

Arguments:

`task` **(input)** An optimization task.

`trmcode` **(output)** Is either `MSK_RES_OK` or a termination response code.

Description: 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`.

- MSK_primalsensitivity

Syntax:

```
MSKrescodee MSKAPI MSK_primalsensitivity (
    MSKtask_t task,
    MSKlintt numi,
    MSKCONST MSKidxt * subi,
    MSKCONST MSKmarke * marki,
    MSKlintt numj,
    MSKCONST MSKidxt * subj,
    MSKCONST MSKmarke * markj,
    MSKrealt * leftpricei,
    MSKrealt * rightpricei,
    MSKrealt * leftrangei,
    MSKrealt * rightrangei,
    MSKrealt * leftpricej,
    MSKrealt * rightpricej,
    MSKrealt * leftrangej,
    MSKrealt * rightrangej);
```

Arguments:

`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:

$$\begin{array}{llll} \text{minimize} & x_1 + x_2 & & \\ \text{subject to} -1 \leq & x_1 - x_2 & \leq & 1, \\ & x_1 & = & 0, \\ & x_1 \geq 0, x_2 \geq 0 & & \end{array} \tag{15.14}$$

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:

$$x_1 - x_2 \leq 1 \tag{15.15}$$

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 MSKAPI 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);
```

Arguments:

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, then c is printed.
qo **(input)** If non-zero, then Q^o is printed.
a **(input)** If non-zero, then A is printed.
qc **(input)** If non-zero, then Q^k is printed for the relevant constraints.
bc **(input)** If non-zero, then constraints bounds are printed.
bx **(input)** If non-zero, then variable bounds are printed.
vartype **(input)** If non-zero, then variable types are printed.
cones **(input)** If non-zero, then 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 MSKAPI MSK_printparam (MSKtask_t task)
```

Arguments:

task **(input)** An optimization task.

Description: Prints the current parameter settings to the message stream.

- MSK_probtypetostr

Syntax:

```
MSKrescodee MSKAPI MSK_probtypetostr (
    MSKtask_t task,
    MSKproblemtypee probtype,
    char * str);
```

Arguments:

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 MSKAPI MSK_prostatostr (
    MSKtask_t task,
    MSKprostae prosta,
    char * str);
```

Arguments:

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 MSKAPI MSK_putaij (
    MSKtask_t task,
    MSKidxt i,
    MSKidxt j,
    MSKrealt aij);
```

Arguments:

`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

$$a_{\mathtt{ij}} = \mathtt{aij}.$$

See also:

`MSK_putavec` Replaces all elements in one row or column of A.

`MSK_putaijlist` Changes one or more coefficients in A.

Comments:

- `MSK_putaijlist`

Syntax:

```
MSKrescodee MSKAPI MSK_putaijlist (
    MSKtask_t task,
    MSKintt num,
    MSKCONST MSKidxt * subi,
    MSKCONST MSKidxt * subj,
    MSKCONST MSKrealt * valij);
```

Arguments:

`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

$$a_{\mathtt{subi[k]},\mathtt{subj[k]}} = \mathtt{valij[k]}, \quad k = 0, \ldots, \mathtt{num} - 1.$$

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 A.

`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.

Comments:

- `MSK_putavec`

Syntax:

```
MSKrescodee MSKAPI MSK_putavec (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKlintt nzi,
    MSKCONST MSKidxt * asub,
    MSKCONST MSKrealt * aval);
```

Arguments:

`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 5.8.3.2 and **??**.

See also:

`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 MSKAPI MSK_putaveclist (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKlintt num,
    MSKCONST MSKidxt * sub,
```

```
    MSKCONST MSKlidxt * ptrb,
    MSKCONST MSKlidxt * ptre,
    MSKCONST MSKidxt * asub,
    MSKCONST MSKrealt * aval);
```

Arguments:

`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 5.8.3.2 and section **??**.

`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 5.8.3.2 and section **??**.

`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 (5.37) 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 A.

`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_putbound`

Syntax:

```
MSKrescodee MSKAPI MSK_putbound (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKboundkeye bk,
    MSKrealt bl,
    MSKrealt bu);
```

Arguments:

`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_chgbound` Changes the bounds for one constraint or variable.

`MSK_putboundlist` Changes the bounds of constraints or variables.

- `MSK_putboundlist`

Syntax:

```
MSKrescodee MSKAPI MSK_putboundlist (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKlintt num,
    MSKCONST MSKidxt * sub,
    MSKCONST MSKboundkeye * bk,
    MSKCONST MSKrealt * bl,
    MSKCONST MSKrealt * bu);
```

Arguments:

`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)** If `con` is non-zero (zero), then constraint (variable) `sub[t]` is assigned the key `bk[t]`.

`bl` **(input)** If `con` is non-zero (zero), then constraint (variable) `sub[t]` is assigned the lower bound `bl[t]`.

`bu` **(input)** If `con` is non-zero (zero), then constraint (variable) `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 MSKAPI MSK_putboundslice (
    MSKtask_t task,
    MSKaccmodee con,
    MSKidxt first,
    MSKidxt last,
    MSKCONST MSKboundkeye * bk,
    MSKCONST MSKrealt * bl,
    MSKCONST MSKrealt * bu);
```

Arguments:

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 MSKAPI MSK_putcallbackfunc (
    MSKtask_t task,
    MSKcallbackfunc func,
    MSKuserhandle_t handle);
```

Arguments:

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 callback 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 MSKAPI MSK_putcfix (
    MSKtask_t task,
    MSKrealt cfix);
```

Arguments:

`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 MSKAPI MSK_putcj (
    MSKtask_t task,
    MSKidxt j,
    MSKrealt cj);
```

Arguments:

`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.

$$c_{\mathtt{j}} = \mathtt{cj}.$$

- MSK_putclist

Syntax:

```
MSKrescodee MSKAPI MSK_putclist (
    MSKtask_t task,
    MSKintt num,
    MSKCONST MSKidxt * subj,
    MSKCONST MSKrealt * val);
```

Arguments:

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

$$c_{\texttt{subj[t]}} = \texttt{val}[\texttt{t}], \quad t = 0, \ldots, \texttt{num} - 1.$$

If a variable index is specified multiple times in subj only the last entry is used.

- MSK_putcone

Syntax:

```
MSKrescodee MSKAPI MSK_putcone (
    MSKtask_t task,
    MSKidxt k,
    MSKconetypee conetype,
    MSKrealt conepar,
    MSKintt nummem,
    MSKCONST MSKidxt * submem);
```

Arguments:

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 MSKAPI MSK_putdouparam (
    MSKtask_t task,
    MSKdparame param,
    MSKrealt parvalue);
```

Arguments:

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 MSKAPI MSK_putintparam (
    MSKtask_t task,
    MSKiparame param,
    MSKintt parvalue);
```

Arguments:

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 MSKAPI MSK_putmaxnumanz (
    MSKtask_t task,
    MSKlintt maxnumanz);
```

Arguments:

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_IPAR_MAXNUMANZ_DOUBLE_TRH

MSK_IINF_STO_NUM_A_REALLOC

- MSK_putmaxnumcon

Syntax:

```
MSKrescodee MSKAPI MSK_putmaxnumcon (
    MSKtask_t task,
    MSKintt maxnumcon);
```

Arguments:

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 MSKAPI MSK_putmaxnumcone (
    MSKtask_t task,
    MSKintt maxnumcone);
```

Arguments:

`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 MSKAPI MSK_putmaxnumqnz (
    MSKtask_t task,
    MSKlintt maxnumqnz);
```

Arguments:

`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 A 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 A.

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 MSKAPI MSK_putmaxnumvar (
    MSKtask_t task,
    MSKintt maxnumvar);
```

Arguments:

`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 MSKAPI MSK_putnadouparam (
    MSKtask_t task,
    MSKCONST char * paramname,
    MSKrealt parvalue);
```

Arguments:

`task` **(input)** An optimization task.

`paramname` **(input)** Name of a MOSEK parameter.

`parvalue` **(input)** Parameter value.

Description: Sets the value of a named double parameter.

- MSK_putnaintparam

Syntax:

```
MSKrescodee MSKAPI MSK_putnaintparam (
    MSKtask_t task,
    MSKCONST char * paramname,
    MSKintt parvalue);
```

Arguments:

`task` **(input)** An optimization task.

`paramname` **(input)** Name of a MOSEK parameter.

`parvalue` **(input)** Parameter value.

Description: Sets the value of a named integer parameter.

- MSK_putname

Syntax:

```
MSKrescodee MSKAPI MSK_putname (
    MSKtask_t task,
    MSKproblemiteme whichitem,
    MSKidxt i,
    MSKCONST char * name);
```

Arguments:

`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 MSKAPI MSK_putnastrparam (
    MSKtask_t task,
    MSKCONST char * paramname,
    MSKCONST char * parvalue);
```

Arguments:

`task` **(input)** An optimization task.

`paramname` **(input)** Name of a MOSEK parameter.

`parvalue` **(input)** Parameter value.

Description: Sets the value of a named string parameter.

- `MSK_putnlfunc`

Syntax:

```
MSKrescodee MSKAPI MSK_putnlfunc (
    MSKtask_t task,
    MSKuserhandle_t nlhandle,
    MSKnlgetspfunc nlgetsp,
    MSKnlgetvafunc nlgetva);
```

Arguments:

`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 MSKAPI MSK_putobjname (
    MSKtask_t task,
    MSKCONST char * objname);
```

Arguments:

`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 MSKAPI MSK_putobjsense (
    MSKtask_t task,
    MSKobjsensee sense);
```

Arguments:

`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 MSKAPI MSK_putparam (
    MSKtask_t task,
    MSKCONST char * parname,
    MSKCONST char * parvalue);
```

Arguments:

`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 MSKAPI MSK_putqcon (
    MSKtask_t task,
    MSKlintt numqcnz,
    MSKCONST MSKidxt * qcsubk,
    MSKCONST MSKidxt * qcsubi,
    MSKCONST MSKidxt * qcsubj,
    MSKCONST MSKrealt * qcval);
```

Arguments:

task (input) An optimization task.

numqcnz (input) Number of quadratic terms. See (5.36).

qcsubk (input) k subscripts for q_{ij}^k. See (5.36).

qcsubi (input) i subscripts for q_{ij}^k. See (5.36).

qcsubj (input) j subscripts for q_{ij}^k. See (5.36).

qcval (input) Numerical value for q_{ij}^k.

Description: Replaces all quadratic entries in the constraints. Consider constraints on the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{\texttt{numvar}-1} \sum_{j=0}^{\texttt{numvar}-1} q_{ij}^k x_i x_j + \sum_{j=0}^{\texttt{numvar}-1} a_{kj} x_j \leq u_k^c, \;\; k = 0, \ldots, m-1. \tag{15.16}$$

The function assigns values to q such that:

$$q_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}}^{\texttt{qcsubk[t]}} = \texttt{qcval[t]}, \;\; t = 0, \ldots, \texttt{numqcnz} - 1. \tag{15.17}$$

and

$$q_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}}^{\texttt{qcsubk[t]}} = \texttt{qcval[t]}, \;\; t = 0, \ldots, \texttt{numqcnz} - 1. \tag{15.18}$$

Values not assigned are set to zero.

See also:

MSK_putqconk Replaces all quadratic terms in a single constraint.

MSK_putmaxnumqnz Changes the size of the preallocated storage for Q.

- MSK_putqconk

Syntax:

```
MSKrescodee MSKAPI MSK_putqconk (
    MSKtask_t task,
    MSKidxt k,
    MSKlintt numqcnz,
    MSKCONST MSKidxt * qcsubi,
    MSKCONST MSKintt * qcsubj,
    MSKCONST MSKrealt * qcval);
```

Arguments:

`task` **(input)** An optimization task.

`k` **(input)** The constraint in which new the Q elements are inserted.

`numqcnz` **(input)** Number of quadratic terms. See (5.36).

`qcsubi` **(input)** i subscripts for q_{ij}^k. See (5.36).

`qcsubj` **(input)** j subscripts for q_{ij}^k. See (5.36).

`qcval` **(input)** Numerical value for q_{ij}^k.

Description: Replaces all the quadratic entries in one constraint k of the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{\texttt{numvar}-1} \sum_{j=0}^{\texttt{numvar}-1} q_{ij}^k x_i x_j + \sum_{j=0}^{\texttt{numvar}-1} a_{kj} x_j \leq u_k^c. \tag{15.19}$$

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 $\texttt{numqonz} - 1$
3. $\quad q^k_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} = q^k_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} + \texttt{qcval}[\texttt{t}]$
3. $\quad q^k_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} = q^k_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} + \texttt{qcval}[\texttt{t}]$

Please note that:

- For large problems it is essential for the efficiency that the function MSK_putmaxnumqnz is employed to specify an appropriate `maxnumqnz`.
- Only the lower triangular part should be specified because Q^k is symmetric. Specifying values for q_{ij}^k where $i < j$ will result in an error.
- Only non-zero elements should be specified.
- The order in which the non-zero elements are specified is insignificant.
- Duplicate elements are added together. Hence, it is recommended not to specify the same element multiple times in `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 Q.

- MSK_putqobj

Syntax:

```
MSKrescodee MSKAPI MSK_putqobj (
    MSKtask_t task,
    MSKlintt numqonz,
    MSKCONST MSKidxt * qosubi,
    MSKCONST MSKidxt * qosubj,
    MSKCONST MSKrealt * qoval);
```

Arguments:

`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

$$\frac{1}{2} \sum_{i=0}^{\texttt{numvar}-1} \sum_{j=0}^{\texttt{numvar}-1} q^o_{ij} x_i x_j + \sum_{j=0}^{\texttt{numvar}-1} c_j x_j + c^f. \tag{15.20}$$

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 elements are *not* added to together; only the last value for each entry is used.

For a code example see section 5.3.1.

- `MSK_putqobjij`

Syntax:

```
MSKrescodee MSKAPI MSK_putqobjij (
    MSKtask_t task,
    MSKidxt i,
    MSKidxt j,
    MSKrealt qoij);
```

Arguments:

`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 of the coefficients in the quadratic term in the objective. The function performs the assignment

$$q^o_{\mathtt{ij}} = \mathtt{qoij}.$$

- `MSK_putresponsefunc`

Syntax:

```
MSKrescodee MSKAPI MSK_putresponsefunc (
    MSKtask_t task,
    MSKresponsefunc responsefunc,
    MSKuserhandle_t handle);
```

Arguments:

`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 MSKAPI 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);
```

Arguments:

`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 (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 MSKAPI MSK_putsolutioni (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKsoltypee whichsol,
    MSKstakeye sk,
    MSKrealt x,
    MSKrealt sl,
    MSKrealt su,
    MSKrealt sn);
```

Arguments:

task **(input)** An optimization task.

accmode **(input)** If non-zero, then 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 function and variable, call MSK_putsolutioni to set the solution information.

See also:

MSK_makesolutionstatusunknown Sets the solution status to unknown.

- MSK_putsolutionyi

Syntax:

```
MSKrescodee MSKAPI MSK_putsolutionyi (
    MSKtask_t task,
    MSKidxt i,
    MSKsoltypee whichsol,
    MSKrealt y);
```

Arguments:

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 MSKAPI MSK_putstrparam (
    MSKtask_t task,
    MSKsparame param,
    MSKCONST char * parvalue);
```

Arguments:

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 MSKAPI MSK_puttaskname (
    MSKtask_t task,
    MSKCONST char * taskname);
```

Arguments:

task **(input)** An optimization task.

`taskname` **(input)** Name assigned to the task.

Description: Assigns the name `taskname` to the task.

- `MSK_putvarbranchorder`

Syntax:

```
MSKrescodee MSKAPI MSK_putvarbranchorder (
    MSKtask_t task,
    MSKidxt j,
    MSKintt priority,
    int direction);
```

Arguments:

`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 MSKAPI MSK_putvartype (
    MSKtask_t task,
    MSKidxt j,
    MSKvariabletypee vartype);
```

Arguments:

`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.

- `MSK_putvartypelist`

Syntax:

```
MSKrescodee MSKAPI MSK_putvartypelist (
    MSKtask_t task,
    MSKintt num,
    MSKCONST MSKidxt * subj,
    MSKCONST MSKvariabletypee * vartype);
```

Arguments:

`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 18.48 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.

- `MSK_readbranchpriorities`

Syntax:

```
MSKrescodee MSKAPI MSK_readbranchpriorities (
    MSKtask_t task,
    MSKCONST char * filename);
```

Arguments:

`task` **(input)** An optimization task.

`filename` **(input)** Data is written to 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 MSKAPI MSK_readdata (
    MSKtask_t task,
    MSKCONST char * filename);
```

Arguments:

`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" is interpreted as an OPF file and a compressed OPF file respectively.
- ".mps" and ".mps.gz" is interpreted as an MPS file and a compressed MPS file respectively.
- ".mbt" is interpreted as an MBT file (MOSEK binary task file).

See also:

MSK_writedata Writes problem data to a file.

MSK_IPAR_READ_DATA_FORMAT

- MSK_readparamfile

Syntax:

```
MSKrescodee MSKAPI MSK_readparamfile (MSKtask_t task)
```

Arguments:

`task` **(input)** An optimization task.

Description: Reads a parameter file.

- MSK_readsolution

Syntax:

```
MSKrescodee MSKAPI MSK_readsolution (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKCONST char * filename);
```

Arguments:

`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 MSKAPI MSK_readsummary (
    MSKtask_t task,
    MSKstreamtypee whichstream);
```

Arguments:

`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 MSKAPI MSK_relaxprimal (
    MSKtask_t task,
    MSKtask_t * relaxedtask,
    MSKrealt * wlc,
    MSKrealt * wuc,
    MSKrealt * wlx,
    MSKrealt * wux);
```

Arguments:

`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

$$\begin{array}{lccccl} \text{minimize} & & & c^T x & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \tag{15.21}$$

the function forms a new task `relaxedtask` having the form

$$\begin{array}{lccccl} \text{minimize} & & & p & & \\ \text{subject to} & l^c & \leq & Ax + v_l^c - v_u^c & \leq & u^c, \\ & l^x & \leq & x + v_l^x - v_u^x & \leq & u^x, \\ & & & (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq & 0, \\ & & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0. & & \end{array} \tag{15.22}$$

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 (15.22) 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 (15.22) 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 (15.22), then a natural thing to do is to solve the optimization problem

$$\begin{array}{lllll} \text{minimize} & & & c^T x & & \\ \text{subject to} & l^c & \leq & Ax + v_l^c - v_u^c & \leq & u^c, \\ & l^x & \leq & x + v_l^x - v_u^x & \leq & u^x, \\ & & & (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq & 0, \\ & & & p & = & p^*, \\ & & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0, & & \end{array} \tag{15.23}$$

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 (15.22) or the problem (15.23).The parameter can take one of the following values.

MSK_FEASREPAIR_OPTIMIZE_NONE : The returned task `relaxedtask` contains problem (15.22) and is not optimized.

MSK_FEASREPAIR_OPTIMIZE_PENALTY : The returned task `relaxedtask` contains problem (15.22) and is optimized.

MSK_FEASREPAIR_OPTIMIZE_COMBINED : The returned task `relaxedtask` contains problem (15.23) 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 MSKAPI MSK_remove (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKintt num,
    MSKCONST MSKintt * sub);
```

Arguments:

`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 MSKAPI MSK_removecone (
    MSKtask_t task,
    MSKidxt k);
```

Arguments:

`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 MSKAPI MSK_resizetask (
    MSKtask_t task,
    MSKintt maxnumcon,
    MSKintt maxnumvar,
    MSKintt maxnumcone,
    MSKlintt maxnumanz,
    MSKlintt maxnumqnz);
```

Arguments:

`task` **(input)** `task` that should be resized.

`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 Q.

- `MSK_sensitivityreport`

Syntax:

```
MSKrescodee MSKAPI MSK_sensitivityreport (
    MSKtask_t task,
    MSKstreamtypee whichstream);
```

Arguments:

`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 MSKAPI MSK_setdefaults (MSKtask_t task)
```

Arguments:

task **(input)** An optimization task.

Description: Resets all the parameters to their default values.

- MSK_sktostr

Syntax:

```
MSKrescodee MSKAPI MSK_sktostr (
    MSKtask_t task,
    MSKintt sk,
    char * str);
```

Arguments:

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 MSKAPI MSK_solstatostr (
    MSKtask_t task,
    MSKsolstae solsta,
    char * str);
```

Arguments:

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 MSKAPI MSK_solutiondef (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKintt * isdef);
```

Arguments:

`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 MSKAPI MSK_solutionsummary (
    MSKtask_t task,
    MSKstreamtypee whichstream);
```

Arguments:

`task` **(input)** An optimization task.

`whichstream` **(input)** Index of the stream.

Description: Prints a short summary of the current solution.

- `MSK_solvewithbasis`

Syntax:

```
MSKrescodee MSKAPI MSK_solvewithbasis (
    MSKtask_t task,
    MSKintt transp,
    MSKintt * numnz,
    MSKidxt * sub,
    MSKrealt * val);
```

Arguments:

`task` **(input)** An optimization task.

`transp` **(input)** If this argument is non-zero, then (15.25) is solved. Otherwise the system (15.24) 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 `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

$$B\bar{x} = b \tag{15.24}$$

or the system

$$B^T\bar{x} = b \tag{15.25}$$

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 use with the `MSK_solvewithbasis` function.

- `MSK_startstat`

Syntax:

```
MSKrescodee MSKAPI MSK_startstat (MSKtask_t task)
```

Arguments:

`task` **(input)** An optimization task.

Description: Starts the statistics file.

- `MSK_stopstat`

Syntax:

```
MSKrescodee MSKAPI MSK_stopstat (MSKtask_t task)
```

Arguments:

`task` **(input)** An optimization task.

Description: Stops the statistics file.

- `MSK_strdupdbgtask`

Syntax:

```
char * MSKAPI MSK_strdupdbgtask (
    MSKtask_t task,
    MSKCONST char * str,
    MSKCONST char * file,
    MSKCONST unsigned line);
```

Arguments:

`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 * MSKAPI MSK_strduptask (
    MSKtask_t task,
    MSKCONST char * str);
```

Arguments:

`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 MSKAPI MSK_strtoconetype (
    MSKtask_t task,
    MSKCONST char * str,
    MSKconetypee * conetype);
```

Arguments:

`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 MSKAPI MSK_strtosk (
    MSKtask_t task,
    MSKCONST char * str,
    MSKintt * sk);
```

Arguments:

`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 MSKAPI MSK_undefsolution (
    MSKtask_t task,
    MSKsoltypee whichsol);
```

Arguments:

`task` **(input)** An optimization task.

`whichsol` **(input)** Selects a solution.

Description: Undefines a solution. Purges all information regarding `whichsol`.

- `MSK_unlinkfuncfromtaskstream`

Syntax:

```
MSKrescodee MSKAPI MSK_unlinkfuncfromtaskstream (
    MSKtask_t task,
    MSKstreamtypee whichstream);
```

Arguments:

`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 MSKAPI MSK_whichparam (
    MSKtask_t task,
    MSKCONST char * parname,
    MSKparametertypee * partype,
    MSKintt * param);
```

Arguments:

`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 MSKAPI MSK_writebranchpriorities (
    MSKtask_t task,
    MSKCONST char * filename);
```

Arguments:

`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 MSKAPI MSK_writedata (
    MSKtask_t task,
    MSKCONST char * filename);
```

Arguments:

`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 C.

MPS : A text based column oriented format. File extension `.mps`. See Appendix B.

OPF : A text based row oriented format. File extension `.opf`. Supports more problem types than MPS and LP. See Appendix D.

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.

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 MSKAPI MSK_writeparamfile (
    MSKtask_t task,
    MSKCONST char * filename);
```

Arguments:

`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 MSKAPI MSK_writesolution (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKCONST char * filename);
```

Arguments:

`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.

- MSK_IPAR_LOG 417
  Controls the amount of log information.
- MSK_IPAR_LOG_BI 417
  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 417
  Controls the logging frequency.
- MSK_IPAR_LOG_CONCURRENT 418
  Controls amount of output printed by the concurrent optimizer.
- MSK_IPAR_LOG_CUT_SECOND_OPT 418
  Controls the reduction in the log levels for the second and any subsequent optimizations.
- MSK_IPAR_LOG_FACTOR 418
  If turned on, then the factor log lines are added to the log.
- MSK_IPAR_LOG_FEASREPAIR 419
  Controls the amount of output printed when performing feasibility repair.
- MSK_IPAR_LOG_FILE 419
  If turned on, then some log info is printed when a file is written or read.

16.1.2 Basis identification parameters.

16.1.3 The Interior-point method parameters.

Parameters defining the behavior of the interior-point method for linear, conic and convex problems.

16.1.4 Simplex optimizer parameters.

Parameters defining the behavior of the simplex optimizer for linear problems.

16.1.5 Primal simplex optimizer parameters.

Parameters defining the behavior of the primal simplex optimizer for linear problems.

16.1.6 Dual simplex optimizer parameters.

Parameters defining the behavior of the dual simplex optimizer for linear problems.

16.1.7 Network simplex optimizer parameters.

Parameters defining the behavior of the network simplex optimizer for linear problems.

- MSK_IPAR_SIM_REFACTOR_FREQ 450
  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 406
  Specify the level of convexity check on quadratic problems
- MSK_DPAR_INTPNT_NL_MERIT_BAL 381
  Controls if the complementarity and infeasibility is converging to zero at about equal rates.
- MSK_DPAR_INTPNT_NL_TOL_DFEAS 382
  Dual feasibility tolerance used when a nonlinear model is solved.
- MSK_DPAR_INTPNT_NL_TOL_MU_RED 382
  Relative complementarity gap tolerance.
- MSK_DPAR_INTPNT_NL_TOL_NEAR_REL 382
  Nonlinear solver optimality tolerance parameter.
- MSK_DPAR_INTPNT_NL_TOL_PFEAS 383
  Primal feasibility tolerance used when a nonlinear model is solved.
- MSK_DPAR_INTPNT_NL_TOL_REL_GAP 383
  Relative gap termination tolerance for nonlinear problems.
- MSK_DPAR_INTPNT_NL_TOL_REL_STEP 383
  Relative step size to the boundary for general nonlinear optimization problems.
- MSK_DPAR_INTPNT_TOL_INFEAS 384
  Nonlinear solver infeasibility tolerance parameter.

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 380
  Dual feasibility tolerance used by the conic interior-point optimizer.
- MSK_DPAR_INTPNT_CO_TOL_INFEAS 380
  Infeasibility tolerance for the conic solver.
- MSK_DPAR_INTPNT_CO_TOL_MU_RED 380
  Optimality tolerance for the conic solver.
- MSK_DPAR_INTPNT_CO_TOL_NEAR_REL 381
  Optimality tolerance for the conic solver.

16.1.10 The mixed integer optimization parameters.

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.

16.1.14 Non-convex solver parameters.

16.1.15 Feasibility repair parameters.

16.1.16 Optimization system parameters.

Parameters defining the overall solver system environment. This includes system and platform related information and behavior.

16.1.17 Output information parameters.

16.1.18 Extra information about the optimization problem.

16.1.19 Overall solver parameters.

16.1.20 Behavior of the optimization task.

Parameters defining the behavior of an optimization task when loading data.

16.1.21 Data input/output parameters.

Parameters defining the behavior of data readers and writers.

- MSK_IPAR_READ_CONE 439
  Controls the expected number of conic constraints.
- MSK_IPAR_READ_DATA_COMPRESSED 440
  Controls the input file decompression.
- MSK_IPAR_READ_DATA_FORMAT 440
  Format of the data file to be read.
- MSK_IPAR_READ_KEEP_FREE_CON 440
  Controls whether the free constraints are included in the problem.
- MSK_IPAR_READ_LP_DROP_NEW_VARS_IN_BOU 441
  Controls how the LP files are interpreted.
- MSK_IPAR_READ_LP_QUOTED_NAMES 441
  If a name is in quotes when reading an LP file, the quotes will be removed.
- MSK_SPAR_READ_MPS_BOU_NAME 466
  Name of the BOUNDS vector used. An empty name means that the first BOUNDS vector is used.
- MSK_IPAR_READ_MPS_FORMAT 441
  Controls how strictly the MPS file reader interprets the MPS format.
- MSK_IPAR_READ_MPS_KEEP_INT 442
  Controls if integer constraints are read.
- MSK_SPAR_READ_MPS_OBJ_NAME 466
  Objective name in the MPS file.
- MSK_IPAR_READ_MPS_OBJ_SENSE 442
  Controls the MPS format extensions.
- MSK_IPAR_READ_MPS_QUOTED_NAMES 442
  Controls the MPS format extensions.
- MSK_SPAR_READ_MPS_RAN_NAME 466
  Name of the RANGE vector used. An empty name means that the first RANGE vector is used.
- MSK_IPAR_READ_MPS_RELAX 442
  Controls the meaning of integer constraints.
- MSK_SPAR_READ_MPS_RHS_NAME 466
  Name of the RHS used. An empty name means that the first RHS vector is used.
- MSK_IPAR_READ_MPS_WIDTH 443
  Controls the maximal number of chars allowed in one line of the MPS file.
- MSK_IPAR_READ_Q_MODE 443
  Controls how the Q matrices are read from the MPS file.

16.1.22 Solution input/output parameters.

Parameters defining the behavior of solution reader and writer.

16.1.23 Infeasibility report parameters.

16.1.24 License manager parameters.

- MSK_IPAR_LICENSE_ALLOW_OVERUSE 415
  Controls if license overuse is allowed when caching licenses
- MSK_IPAR_LICENSE_CACHE_TIME 415
  Controls the license manager client behavior.
- MSK_IPAR_LICENSE_CHECK_TIME 415
  Controls the license manager client behavior.
- MSK_IPAR_LICENSE_DEBUG 416
  Controls the license manager client debugging behavior.
- MSK_IPAR_LICENSE_PAUSE_TIME 416
  Controls license manager client behavior.
- MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS 416
  Controls license manager client behavior.
- MSK_IPAR_LICENSE_WAIT 416
  Controls if MOSEK should queue for a license if none is available.

16.1.25 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 406
  Specify the level of convexity check on quadratic problems
- MSK_IPAR_CHECK_TASK_DATA 406
  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 378
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_AIJ_LARGE 378
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_BOUND_INF 378
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_BOUND_WRN 378
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_C_HUGE 379
  Data tolerance threshold.

16.2 Double parameters

- MSK_DPAR_DATA_TOL_CJ_LARGE ... 379
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_QIJ ... 379
  Data tolerance threshold.
- MSK_DPAR_DATA_TOL_X ... 379
  Data tolerance threshold.

- `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

- bi_lu_tol_rel_piv

  Corresponding constant:
  MSK_DPAR_BI_LU_TOL_REL_PIV

  Description:
  Relative pivot tolerance used in the LU factorization in the basis identification procedure.

  Possible Values:
  Any number between 1.0e-6 and 0.999999.

  Default value:
  0.01

- 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

- `data_tol_aij`

Corresponding constant:
`MSK_DPAR_DATA_TOL_AIJ`

Description:
Absolute zero tolerance for elements in A.

Possible Values:
Any number between 1.0e-16 and 1.0e-6.

Default value:
1.0e-12

- `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

- `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

- `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

- `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 might 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 might 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 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 a feasible solution having an objective value 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

- 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 criterias are never checked.

Possible Values:
Any number between 0.0 and +inf.

Default value:
0.0

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 criterias 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-5

See also:

MSK_DPAR_MIO_DISABLE_TERM_TIME Certain termination criterias 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_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_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-8

- 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 0.0 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

- `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 a feasible solution having and objective value 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

- `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

- MSK_IPAR_WRITE_XML_MODE
  Controls if linear coefficients should be written by row or column when writing in the XML file format.

- 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

- 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_CONIC Another cone optimizer.

  MSK_OPTIMIZER_NONCONVEX The optimizer for nonconvex nonlinear problems.

  MSK_OPTIMIZER_QCONE The Qcone optimizer is used.

  MSK_OPTIMIZER_PRIMAL_SIMPLEX The primal simplex optimizer is used.

  MSK_OPTIMIZER_FREE_SIMPLEX Either the primal or the dual simplex optimizer 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_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.

  Default value:
  MSK_CHECK_CONVEXITY_SIMPLE

- `check_ctrl_c`

  Corresponding constant:
  `MSK_IPAR_CHECK_CTRL_C`

  Description:
  Specifies whether MOSEK should check for `<ctrl>+<c>` key presses. In case it has, then control is returned to the user program.

  In case a user-defined ctrl-c function is defined then that is used to check for ctrl-c. Otherwise the system procedure `signal` is used.

  Possible Values:

  `MSK_ON` Switch the option on.

  `MSK_OFF` Switch the option off.

  Default value:
  MSK_OFF

- `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_OFF

- `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

- `flush_stream_freq`

Corresponding constant:
`MSK_IPAR_FLUSH_STREAM_FREQ`

Description:
Controls how frequent the message and log streams are flushed. A value of 0 means that it is never flushed. Otherwise a larger value results in less frequent flushes.

Possible Values:
Any number between 0 and +inf.

Default value:
24

- `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.

Possible Values:
Any integer greater than 1.

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_CONSTANT` The starting point is set to a constant. This is more reliable than a non-constant starting point.

`MSK_STARTING_POINT_FREE` The starting point is chosen automatically.

Default value:
MSK_STARTING_POINT_FREE

- `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:
Controls the amount of time a license is cached in the MOSEK environment for reuse. Checking out a license from the license server has a small overhead. Therefore, if a large number of optimizations is performed within a small amount of time, it is efficient to cache the license in the MOSEK environment for later use. This way a number of license check outs from the license server is avoided.

If a license has not been used in the given amount of time,MOSEK will automatically check in the license. To disable license caching set the value to 0.

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 micro seconds between each check of whether a license as 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_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:
250

- `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

- maxnumanz_double_trh

  Corresponding constant:
  MSK_IPAR_MAXNUMANZ_DOUBLE_TRH

  Description:
  Whenever MOSEK runs out of storage for the A matrix, it will double the value for **maxnumanz** until **maxnumnza** reaches the value of this parameter. When this threshold is reached it will use a slower increase.

  Possible Values:
  Any number between -inf and +inf.

Default value:
-1

- `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 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 MIP solution exists, then MOSEK generates an initial solution to the MIP by fixing all integer values and solving for the continuous variables.

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 MIP 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_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:

  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 criterias 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 criterias 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 criterias 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_CONIC Another cone optimizer.

MSK_OPTIMIZER_NONCONVEX The optimizer for nonconvex nonlinear problems.

MSK_OPTIMIZER_QCONE The Qcone optimizer is used.

MSK_OPTIMIZER_PRIMAL_SIMPLEX The primal simplex optimizer is used.

MSK_OPTIMIZER_FREE_SIMPLEX Either the primal or the dual simplex optimizer 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_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_CONIC Another cone optimizer.

MSK_OPTIMIZER_NONCONVEX The optimizer for nonconvex nonlinear problems.

MSK_OPTIMIZER_QCONE The Qcone optimizer is used.

MSK_OPTIMIZER_PRIMAL_SIMPLEX The primal simplex optimizer is used.

MSK_OPTIMIZER_FREE_SIMPLEX Either the primal or the dual simplex optimizer 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:
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_CONIC Another cone optimizer.

MSK_OPTIMIZER_NONCONVEX The optimizer for nonconvex nonlinear problems.

MSK_OPTIMIZER_QCONE The Qcone optimizer is used.

MSK_OPTIMIZER_PRIMAL_SIMPLEX The primal simplex optimizer is used.

MSK_OPTIMIZER_FREE_SIMPLEX Either the primal or the dual simplex optimizer 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_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

- 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_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 the relaxation of the MIP will be 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 chars 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_CONIC Another cone optimizer.

MSK_OPTIMIZER_NONCONVEX The optimizer for nonconvex nonlinear problems.

MSK_OPTIMIZER_QCONE The Qcone optimizer is used.

MSK_OPTIMIZER_PRIMAL_SIMPLEX The primal simplex optimizer is used.

MSK_OPTIMIZER_FREE_SIMPLEX Either the primal or the dual simplex optimizer 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_degen

Corresponding constant:
MSK_IPAR_SIM_DEGEN

Description:
Controls how aggressive degeneration is approached.

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_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_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_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 setbacks are allowed within a simplex optimizer. A setback 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_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_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 the problem is scaled 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_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

- `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 which format the data file has when a task is written to that file using `MSK_writedata`.

Possible Values:

`MSK_DATA_FORMAT_XML` The data file is an XML 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 540
No error occurred.

(50) MSK_RES_WRN_OPEN_PARAM_FILE 548
The parameter file could not be opened.

(51) MSK_RES_WRN_LARGE_BOUND 544
A very large bound in absolute value has been specified.

(52) MSK_RES_WRN_LARGE_LO_BOUND 545
A large but finite lower bound in absolute value has been specified.

(53) MSK_RES_WRN_LARGE_UP_BOUND 545
A large but finite upper bound in absolute value has been specified.

(57) MSK_RES_WRN_LARGE_CJ 544
A numerically large value is specified for one c_j.

(62) MSK_RES_WRN_LARGE_AIJ 544
A numerically large value is specified for one $a_{i,j}$.

(63) MSK_RES_WRN_ZERO_AIJ 549
One or more zero elements are specified in A.

(65) MSK_RES_WRN_NAME_MAX_LEN 547
A name is longer than the buffer that is supposed to hold it.

(66) MSK_RES_WRN_SPAR_MAX_LEN 548
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 547
An RHS vector is split into several nonadjacent parts in an MPS file.

(71) MSK_RES_WRN_MPS_SPLIT_RAN_VECTOR 546
A RANGE vector is split into several nonadjacent parts in an MPS file.

(72) MSK_RES_WRN_MPS_SPLIT_BOU_VECTOR 546
A BOUNDS vector is split into several nonadjacent parts in an MPS file.

(80) MSK_RES_WRN_LP_OLD_QUAD_FORMAT 546
Missing '/2' after quadratic expressions in bound or objective.

(85) MSK_RES_WRN_LP_DROP_VARIABLE 545
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 547
Non-zero elements specified in the upper triangle of a matrix were ignored.

(201) MSK_RES_WRN_DROPPED_NZ_QOBJ 543
One or more non-zero elements were dropped in the Q matrix in the objective.

(250) MSK_RES_WRN_IGNORE_INTEGER 544
Ignored integer constraints.

(251) MSK_RES_WRN_NO_GLOBAL_OPTIMIZER 547
No global optimizer is available.

(270) MSK_RES_WRN_MIO_INFEASIBLE_FINAL 546
The final mixed integer problem with all the integer variables fixed at their optimal values is infeasible.

(280) MSK_RES_WRN_FIXED_BOUND_VALUES 543
A fixed constraint/variable has been specified using the bound keys but the numerical bounds are different. The variable is fixed at the lower bound.

(300) MSK_RES_WRN_SOL_FILTER 548
Invalid solution filter is specified.

(350) MSK_RES_WRN_UNDEF_SOL_FILE_NAME 549
Undefined name occurred in a solution.

(400) MSK_RES_WRN_TOO_FEW_BASIS_VARS 548
An incomplete basis has been specified. Too few basis variables are specified.

(405) MSK_RES_WRN_TOO_MANY_BASIS_VARS 549
A basis with too many variables has been specified.

(500) MSK_RES_WRN_LICENSE_EXPIRE 545
The license expires.

(501) MSK_RES_WRN_LICENSE_SERVER 545
The license server is not responding.

(502) MSK_RES_WRN_EMPTY_NAME 543
A variable or constraint name is empty. The output file may be invalid.

(1011) MSK_RES_ERR_SIZE_LICENSE_VAR 533
The problem has too many variables to be solved with the available license.

(1012) MSK_RES_ERR_SIZE_LICENSE_INTVAR 533
The problem contains too many integer variables to be solved with the available license.

(1013) MSK_RES_ERR_OPTIMIZER_LICENSE 527
The optimizer required is not licensed.

(1014) MSK_RES_ERR_FLEXLM 496
The FLEXlm license manager reported an error.

(1015) MSK_RES_ERR_LICENSE_SERVER 510
The license server is not responding.

(1016) MSK_RES_ERR_LICENSE_MAX 510
Maximum number of licenses is reached.

(1017) MSK_RES_ERR_LICENSE_MOSEKLM_DAEMON 510
The MOSEKLM license manager daemon is not up and running.

(1018) MSK_RES_ERR_LICENSE_FEATURE 509
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 529
A requested license feature is not available for the required platform.

(1020) MSK_RES_ERR_LICENSE_CANNOT_ALLOCATE 509
The license system cannot allocate the memory required.

(1021) MSK_RES_ERR_LICENSE_CANNOT_CONNECT 509
MOSEK cannot connect to the license server. Most likely the license server is not up and running.

(1025) MSK_RES_ERR_LICENSE_INVALID_HOSTID 510
The host ID specified in the license file does not match the host ID of the computer.

(1026) MSK_RES_ERR_LICENSE_SERVER_VERSION 510
The version specified in the checkout request is greater than the highest version number the daemon supports.

(1030) MSK_RES_ERR_OPEN_DL 526
A dynamic link library could not be opened.

(1035) MSK_RES_ERR_OLDER_DLL 526
The dynamic link library is older than the specified version.

(1036) MSK_RES_ERR_NEWER_DLL 522
The dynamic link library is newer than the specified version.

(1040) MSK_RES_ERR_LINK_FILE_DLL 511
A file cannot be linked to a stream in the DLL version.

(1062) MSK_RES_ERR_INVALID_STREAM 507
An invalid stream is referenced.

(1063) MSK_RES_ERR_NO_INIT_ENV 523
`env` is not initialized.

(1064) MSK_RES_ERR_INVALID_TASK 507
The `task` is invalid.

(1065) MSK_RES_ERR_NULL_POINTER 524
An argument to a function is unexpectedly a NULL pointer.

(1070) MSK_RES_ERR_NULL_NAME 524
An all blank name has been specified.

(1071) MSK_RES_ERR_DUP_NAME 494
An error occurred while reading an MPS file..

(1075) MSK_RES_ERR_INVALID_OBJ_NAME 507
An invalid objective name is specified.

(1080) MSK_RES_ERR_SPACE_LEAKING 534
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 535
No available information about the space usage.

(1090) MSK_RES_ERR_READ_FORMAT 530
The specified format cannot be read.

(1100) MSK_RES_ERR_MPS_FILE 516
An error occurred while reading an MPS file.

(1101) MSK_RES_ERR_MPS_INV_FIELD 516
A field in the MPS file is invalid. Probably it is too wide.

(1102) MSK_RES_ERR_MPS_INV_MARKER 516
An invalid marker has been specified in the MPS file.

(1103) MSK_RES_ERR_MPS_NULL_CON_NAME 518
An empty constraint name is used in an MPS file.

(1104) MSK_RES_ERR_MPS_NULL_VAR_NAME 518
An empty variable name is used in an MPS file.

(1105) MSK_RES_ERR_MPS_UNDEF_CON_NAME 519
An undefined constraint name occurred in an MPS file.

(1106) MSK_RES_ERR_MPS_UNDEF_VAR_NAME 520
An undefined variable name occurred in an MPS file.

(1107) MSK_RES_ERR_MPS_INV_CON_KEY 516
An invalid constraint key occurred in an MPS file.

(1108) MSK_RES_ERR_MPS_INV_BOUND_KEY 516
An invalid bound key occurred in an MPS file.

(1109) MSK_RES_ERR_MPS_INV_SEC_NAME 517
An invalid section name occurred in an MPS file.

(1110) MSK_RES_ERR_MPS_NO_OBJECTIVE 518
No objective is defined in an MPS file.

(1111) MSK_RES_ERR_MPS_SPLITTED_VAR 519
A variable is split in an MPS data file.

(1112) MSK_RES_ERR_MPS_MUL_CON_NAME 517
A constraint name was specified multiple times in the ROWS section.

(1113) MSK_RES_ERR_MPS_MUL_QSEC 518
Multiple QSECTIONs are specified for a constraint in the MPS data file.

(1114) MSK_RES_ERR_MPS_MUL_QOBJ 518
The Q term in the objective is specified multiple times in the MPS data file.

(1115) MSK_RES_ERR_MPS_INV_SEC_ORDER 517
The sections in the MPS data file are not in the correct order.

(1116) MSK_RES_ERR_MPS_MUL_CSEC 518
Multiple CSECTIONs are given the same name.

(1117) MSK_RES_ERR_MPS_CONE_TYPE 515
Invalid cone type specified in a CSECTION.

(1118) MSK_RES_ERR_MPS_CONE_OVERLAP 515
A variable is specified to be a member of several cones.

(1119) MSK_RES_ERR_MPS_CONE_REPEAT 515
A variable is repeated within the CSECTION.

(1122) MSK_RES_ERR_MPS_INVALID_OBJSENSE 517
An invalid objective sense is specified.

(1125) MSK_RES_ERR_MPS_TAB_IN_FIELD2 519
A tab char occurred in field 2.

(1126) MSK_RES_ERR_MPS_TAB_IN_FIELD3 519
A tab char occurred in field 3.

(1127) MSK_RES_ERR_MPS_TAB_IN_FIELD5 519
A tab char occurred in field 5.

(1128) MSK_RES_ERR_MPS_INVALID_OBJ_NAME 517
An invalid objective name is specified.

(1130) MSK_RES_ERR_ORD_INVALID_BRANCH_DIR 527
An invalid branch direction key is specified.

(1131) MSK_RES_ERR_ORD_INVALID 527
Invalid content in branch ordering file.

(1150) MSK_RES_ERR_LP_INCOMPATIBLE 512
The problem cannot be written to an LP formatted file.

(1151) MSK_RES_ERR_LP_EMPTY 511
The problem cannot be written to an LP formatted file.

(1152) MSK_RES_ERR_LP_DUP_SLACK_NAME 511
The name of the slack variable added to a ranged constraint already exists.

(1153) MSK_RES_ERR_WRITE_MPS_INVALID_NAME 539
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 512
A variable name is invalid when used in an LP formatted file.

(1155) MSK_RES_ERR_LP_FREE_CONSTRAINT 512
Free constraints cannot be written in LP file format.

(1156) MSK_RES_ERR_WRITE_OPF_INVALID_VAR_NAME 539
Empty variable names cannot be written to OPF files.

(1157) MSK_RES_ERR_LP_FILE_FORMAT 511
Syntax error in an LP file.

(1158) MSK_RES_ERR_WRITE_LP_FORMAT 538
Problem cannot be written as an LP file.

(1160) MSK_RES_ERR_LP_FORMAT 511
Syntax error in an LP file.

(1161) MSK_RES_ERR_WRITE_LP_NON_UNIQUE_NAME 539
An auto-generated name is not unique.

(1162) MSK_RES_ERR_READ_LP_NONEXISTING_NAME 531
A variable never occurred in objective or constraints.

(1163) MSK_RES_ERR_LP_WRITE_CONIC_PROBLEM 512
The problem contains cones that cannot be written to an LP formatted file.

(1164) MSK_RES_ERR_LP_WRITE_GECO_PROBLEM 512
The problem contains general convex terms that cannot be written to an LP formatted file.

(1165) MSK_RES_ERR_NAME_MAX_LEN 520
A name is longer than the buffer that is supposed to hold it.

(1168) MSK_RES_ERR_OPF_FORMAT 526
Syntax error in an OPF file

(1170) MSK_RES_ERR_INVALID_NAME_IN_SOL_FILE 506
An invalid name occurred in a solution file.

(1197) MSK_RES_ERR_ARGUMENT_LENNEQ 490
Incorrect length of arguments.

(1198) MSK_RES_ERR_ARGUMENT_TYPE 491
Incorrect argument type.

(1199) MSK_RES_ERR_NR_ARGUMENTS 524
Incorrect number of function arguments.

(1200) MSK_RES_ERR_IN_ARGUMENT 497
A function argument is incorrect.

(1201) MSK_RES_ERR_ARGUMENT_DIMENSION 490
A function argument is of incorrect dimension.

(1203) MSK_RES_ERR_INDEX_IS_TOO_SMALL 498
An index in an argument is too small.

(1204) MSK_RES_ERR_INDEX_IS_TOO_LARGE 498
An index in an argument is too large.

(1205) MSK_RES_ERR_PARAM_NAME 528
The parameter name is not correct.

(1206) MSK_RES_ERR_PARAM_NAME_DOU 528
The parameter name is not correct for a double parameter.

(1207) MSK_RES_ERR_PARAM_NAME_INT 528
The parameter name is not correct for an integer parameter.

(1208) MSK_RES_ERR_PARAM_NAME_STR 528
The parameter name is not correct for a string parameter.

(1210) MSK_RES_ERR_PARAM_INDEX 527
Parameter index is out of range.

(1215) MSK_RES_ERR_PARAM_IS_TOO_LARGE 527
The parameter value is too large.

(1216) MSK_RES_ERR_PARAM_IS_TOO_SMALL 527
The parameter value is too small.

(1217) MSK_RES_ERR_PARAM_VALUE_STR 529
The parameter value string is incorrect.

(1218) MSK_RES_ERR_PARAM_TYPE 528
The parameter type is invalid.

(1219) MSK_RES_ERR_INF_DOU_INDEX 498
A double information index is out of range for the specified type.

(1220) MSK_RES_ERR_INF_INT_INDEX 499
An integer information index is out of range for the specified type.

(1221) MSK_RES_ERR_INDEX_ARR_IS_TOO_SMALL 498
An index in an array argument is too small.

(1222) MSK_RES_ERR_INDEX_ARR_IS_TOO_LARGE 497
An index in an array argument is too large.

(1230) MSK_RES_ERR_INF_DOU_NAME 498
A double information name is invalid.

(1231) MSK_RES_ERR_INF_INT_NAME 499
A integer information name is invalid.

(1232) MSK_RES_ERR_INF_TYPE 499
The information type is invalid.

(1235) MSK_RES_ERR_INDEX 497
An index is out of range.

(1236) MSK_RES_ERR_WHICHSOL 538
The solution defined by compwhichsol does not exists.

(1237) MSK_RES_ERR_SOLITEM 534
The solution item number `solitem` is invalid. Please note MSK_SOL_ITEM_SNX is invalid for the basic solution.

(1238) MSK_RES_ERR_WHICHITEM_NOT_ALLOWED 538
`whichitem` is unacceptable.

(1240) MSK_RES_ERR_MAXNUMCON 513
The maximum number of constraints specified is smaller than the number of constraints in the task.

(1241) MSK_RES_ERR_MAXNUMVAR 514
The maximum number of variables specified is smaller than the number of variables in the task.

(1242) MSK_RES_ERR_MAXNUMANZ 513
The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A.

(1243) MSK_RES_ERR_MAXNUMQNZ 514
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 525
Maximum number of constraints limit is exceeded.

(1251) MSK_RES_ERR_NUMVARLIM 525
Maximum number of variables limit is exceeded.

(1270) MSK_RES_ERR_INV_SK 504
Invalid status key code.

(1271) MSK_RES_ERR_INV_CONE_TYPE_STR 501
Invalid cone type string encountered.

(1272) MSK_RES_ERR_INV_CONE_TYPE 500
Invalid cone type code is encountered.

(1274) MSK_RES_ERR_INV_SKN 504
Invalid value in `skn`.

(1280) MSK_RES_ERR_INV_NAME_ITEM 501
An invalid name item code is used.

(1281) MSK_RES_ERR_PRO_ITEM 529
An invalid problem is used.

(1283) MSK_RES_ERR_INVALID_FORMAT_TYPE 506
Invalid format type.

(1285) MSK_RES_ERR_FIRSTI 496
Invalid `firsti`.

(1286) MSK_RES_ERR_LASTI 508
Invalid `lasti`.

(1287) MSK_RES_ERR_FIRSTJ 496
Invalid `firstj`.

(1288) MSK_RES_ERR_LASTJ 508
Invalid `lastj`.

(1290) MSK_RES_ERR_NONLINEAR_EQUALITY 523
The model contains a nonlinear equality which defines a nonconvex set.

(1291) MSK_RES_ERR_NONCONVEX 523
The optimization problem is nonconvex.

(1292) MSK_RES_ERR_NONLINEAR_RANGED 524
The model contains a nonlinear ranged constraint which by definition defines a nonconvex set.

(1293) MSK_RES_ERR_CON_Q_NOT_PSD 492
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 492
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 525
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 525
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 491
An invalid permutation array is specified.

(1300) MSK_RES_ERR_CONE_INDEX 493
An index of a non-existing cone has been specified.

(1301) MSK_RES_ERR_CONE_SIZE 493
A cone with too few members is specified.

(1302) MSK_RES_ERR_CONE_OVERLAP 493
A new cone which variables overlap with an existing cone has been specified.

(1303) MSK_RES_ERR_CONE_REP_VAR 493
A variable is included multiple times in the cone.

(1304) MSK_RES_ERR_MAXNUMCONE 513
The value specified for `maxnumcone` is too small.

(1305) MSK_RES_ERR_CONE_TYPE 493
Invalid cone type specified.

(1306) MSK_RES_ERR_CONE_TYPE_STR 493
Invalid cone type specified.

(1310) MSK_RES_ERR_REMOVE_CONE_VARIABLE 531
A variable cannot be removed because it will make a cone invalid.

(1350) MSK_RES_ERR_SOL_FILE_NUMBER 534
An invalid number is specified in a solution file.

(1375) MSK_RES_ERR_HUGE_C 497
A huge value in absolute size is specified for one c_j.

(1400) MSK_RES_ERR_INFINITE_BOUND 499
A finite bound value is too large in absolute value.

(1401) MSK_RES_ERR_INV_QOBJ_SUBI 503
Invalid value in `qosubi`.

(1402) MSK_RES_ERR_INV_QOBJ_SUBJ 503
Invalid value in `qosubj`.

(1403) MSK_RES_ERR_INV_QOBJ_VAL 503
Invalid value in `qoval`.

(1404) MSK_RES_ERR_INV_QCON_SUBK 503
Invalid value in `qcsubk`.

(1405) MSK_RES_ERR_INV_QCON_SUBI 502
Invalid value in qcsubi.

(1406) MSK_RES_ERR_INV_QCON_SUBJ 502
Invalid value in qcsubj.

(1407) MSK_RES_ERR_INV_QCON_VAL 503
Invalid value in qcval.

(1408) MSK_RES_ERR_QCON_SUBI_TOO_SMALL 530
Invalid value in qcsubi.

(1409) MSK_RES_ERR_QCON_SUBI_TOO_LARGE 530
Invalid value in qcsubi.

(1415) MSK_RES_ERR_QOBJ_UPPER_TRIANGLE 530
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 530
An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.

(1430) MSK_RES_ERR_USER_FUNC_RET 537
An user function reported an error.

(1431) MSK_RES_ERR_USER_FUNC_RET_DATA 537
An user function returned invalid data.

(1432) MSK_RES_ERR_USER_NLO_FUNC 538
The user-defined nonlinear function reported an error.

(1433) MSK_RES_ERR_USER_NLO_EVAL 537
The user-defined nonlinear function reported an error.

(1440) MSK_RES_ERR_USER_NLO_EVAL_HESSUBI 537
The user-defined nonlinear function reported an invalid subscript in the Hessian.

(1441) MSK_RES_ERR_USER_NLO_EVAL_HESSUBJ 538
The user-defined nonlinear function reported an invalid subscript in the Hessian.

(1445) MSK_RES_ERR_INVALID_OBJECTIVE_SENSE 507
An invalid objective sense is specified.

(1446) MSK_RES_ERR_UNDEFINED_OBJECTIVE_SENSE 536
The objective sense has not been specified before the optimization.

(1449) MSK_RES_ERR_Y_IS_UNDEFINED 539
The solution item y is undefined.

(1450) MSK_RES_ERR_NAN_IN_DOUBLE_DATA 521
An invalid floating point value was used in some double data.

(1461) MSK_RES_ERR_NAN_IN_BLC 520
l^c contains an invalid floating point value, i.e. a NaN.

(1462) MSK_RES_ERR_NAN_IN_BUC 521
u^c contains an invalid floating point value, i.e. a NaN.

(1470) MSK_RES_ERR_NAN_IN_C 521
c contains an invalid floating point value, i.e. a NaN.

(1471) MSK_RES_ERR_NAN_IN_BLX 520
l^x contains an invalid floating point value, i.e. a NaN.

(1472) MSK_RES_ERR_NAN_IN_BUX 521
u^x contains an invalid floating point value, i.e. a NaN.

(1500) MSK_RES_ERR_INV_PROBLEM 502
Invalid problem type. Probably a nonconvex problem has been specified.

(1501) MSK_RES_ERR_MIXED_PROBLEM 515
The problem contains both conic and nonlinear constraints.

(1550) MSK_RES_ERR_INV_OPTIMIZER 502
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 514
No optimizer is available for the current class of integer optimization problems.

(1552) MSK_RES_ERR_NO_OPTIMIZER_VAR_TYPE 523
No optimizer is available for this class of optimization problems.

(1553) MSK_RES_ERR_MIO_NOT_LOADED 514
The mixed-integer optimizer is not loaded.

(1580) MSK_RES_ERR_POSTSOLVE 529
An error occurred during the postsolve. Please contact MOSEK support.

(1600) MSK_RES_ERR_NO_BASIS_SOL 522
No basic solution is defined.

(1610) MSK_RES_ERR_BASIS_FACTOR 491
The factorization of the basis is invalid.

(1615) MSK_RES_ERR_BASIS_SINGULAR 491
The basis is singular and hence cannot be factored.

(1650) MSK_RES_ERR_FACTOR 494
An error occurred while factorizing a matrix.

(1700) MSK_RES_ERR_FEASREPAIR_CANNOT_RELAX 494
An optimization problem cannot be relaxed. This is the case e.g. for general nonlinear optimization problems.

(1701) MSK_RES_ERR_FEASREPAIR_SOLVING_RELAXED 495
The relaxed problem could not be solved to optimality. Please consult the log file for further details.

(1702) MSK_RES_ERR_FEASREPAIR_INCONSISTENT_BOUND 495
The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.

(1800) MSK_RES_ERR_INVALID_COMPRESSION 505
Invalid compression type.

(1801) MSK_RES_ERR_INVALID_IOMODE 506
Invalid io mode.

(2000) MSK_RES_ERR_NO_PRIMAL_INFEAS_CER 523
A certificate of primal infeasibility is not available.

(2001) MSK_RES_ERR_NO_DUAL_INFEAS_CER 522
A certificate of infeasibility is not available.

(2500) MSK_RES_ERR_NO_SOLUTION_IN_CALLBACK 523
The required solution is not available.

(2501) MSK_RES_ERR_INV_MARKI 501
Invalid value in marki.

(2502) MSK_RES_ERR_INV_MARKJ 501
Invalid value in markj.

(2503) MSK_RES_ERR_INV_NUMI 501
Invalid numi.

(2504) MSK_RES_ERR_INV_NUMJ 502
Invalid numj.

(2505) MSK_RES_ERR_CANNOT_CLONE_NL 491
A task with a nonlinear function call-back cannot be cloned.

(2506) MSK_RES_ERR_CANNOT_HANDLE_NL 492
A function cannot handle a task with nonlinear function call-backs.

(2520) MSK_RES_ERR_INVALID_ACCMODE 505
An invalid access mode is specified.

(2550) MSK_RES_ERR_MBT_INCOMPATIBLE 514
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 513
Could not compute the LU factors of the matrix within the maximum number of allowed tries.

(3059) MSK_RES_ERR_CONCURRENT_OPTIMIZER 492
An unsupported optimizer was chosen for use with the concurrent optimizer.

(3100) MSK_RES_ERR_UNB_STEP_SIZE 536
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 497
Some tasks related to this function call were identical. Unique tasks were expected.

(3200) MSK_RES_ERR_INVALID_BRANCH_DIRECTION 505
An invalid branching direction is specified.

(3201) MSK_RES_ERR_INVALID_BRANCH_PRIORITY 505
An invalid branching priority is specified. It should be nonnegative.

(3500) MSK_RES_ERR_INTERNAL_TEST_FAILED 499
An internal unit test function failed.

(3600) MSK_RES_ERR_XML_INVALID_PROBLEM_TYPE 539
The problem type is not supported by the XML format.

(3700) MSK_RES_ERR_INVALID_AMPL_STUB 505
Invalid AMPL stub.

(3999) MSK_RES_ERR_API_INTERNAL 490

(4000) MSK_RES_TRM_MAX_ITERATIONS 540
The optimizer terminated at the maximum number of iterations.

(4001) MSK_RES_TRM_MAX_TIME 541
The optimizer terminated at the maximum amount of time.

(4002) MSK_RES_TRM_OBJECTIVE_RANGE 542
The optimizer terminated on the bound of the objective range.

(4003) MSK_RES_TRM_MIO_NEAR_REL_GAP 541
The mixed-integer optimizer terminated because the near optimal relative gap tolerance was satisfied.

(4004) MSK_RES_TRM_MIO_NEAR_ABS_GAP 541
The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied.

(4005) MSK_RES_TRM_USER_BREAK 543
The optimizer terminated due to a user break.

(4006) MSK_RES_TRM_STALL 542
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, in case of nonlinear problems, the nonlinear call-back functions are incorrect.

Please contact MOSEK support if this happens.

(4007) MSK_RES_TRM_USER_CALLBACK 543
The optimizer terminated due to the return of the user-defined call-back function.

(4008) MSK_RES_TRM_MIO_NUM_RELAXS 541
The mixed-integer optimizer terminated as the maximum number of relaxations was reached.

(4009) MSK_RES_TRM_MIO_NUM_BRANCHES 541
The mixed-integer optimizer terminated as to the maximum number of branches was reached.

(4015) MSK_RES_TRM_NUM_MAX_NUM_INT_SOLUTIONS 542
The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.

(4020) MSK_RES_TRM_MAX_NUM_SETBACKS 540
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 542
The optimizer terminated due to numerical problems.

(4030) MSK_RES_TRM_INTERNAL 540
The optimizer terminated due to some internal reason. Please contact MOSEK support.

(4031) MSK_RES_TRM_INTERNAL_STOP 540
The optimizer terminated for internal reasons. Please contact MOSEK support.

- err_api_array_too_small

  Corresponding constant:
  MSK_RES_ERR_API_ARRAY_TOO_SMALL

  Description:
  An input array was too short.

  Response message string:
  “The input array ’0’ is too short in call to ’1’.”

- err_api_callback

  Corresponding constant:
  MSK_RES_ERR_API_CALLBACK

  Response message string:
  “”

- err_api_cb_connect

  Corresponding constant:
  MSK_RES_ERR_API_CB_CONNECT

  Response message string:
  “”

- err_api_fatal_error

  Corresponding constant:
  MSK_RES_ERR_API_FATAL_ERROR

  Description:
  An internal error occurred in the API. Please report this problem.

  Response message string:
  “An internal error occurred in the %s API: %s”

- err_api_internal

  Corresponding constant:
  MSK_RES_ERR_API_INTERNAL

  Response message string:
  “An internal fatal error occurred in an interface function”

- err_api_nl_data

  Corresponding constant:
  MSK_RES_ERR_API_NL_DATA

  Response message string:
  “”

- err_argument_dimension

  Corresponding constant:
  MSK_RES_ERR_ARGUMENT_DIMENSION

  Description:
  A function argument is of incorrect dimension.

  Response message string:
  “The argument ’%s’ is of incorrect dimension.”

- err_argument_lenneq

  Corresponding constant:
  MSK_RES_ERR_ARGUMENT_LENNEQ

  Description:
  Incorrect length of arguments.

  Response message string:
  “Incorrect argument length. The arguments %s are expected to be of equal length. The length of the arguments was %s.”

- err_argument_perm_array

  Corresponding constant:
  MSK_RES_ERR_ARGUMENT_PERM_ARRAY

  Description:
  An invalid permutation array is specified.

  Response message string:
  "An invalid permutation array named '%s' is supplied. Position %d has the invalid value %d."

- err_argument_type

  Corresponding constant:
  MSK_RES_ERR_ARGUMENT_TYPE

  Description:
  Incorrect argument type.

  Response message string:
  "Incorrect type in %s argument number: '%d'."

- err_basis

  Corresponding constant:
  MSK_RES_ERR_BASIS

  Description:
  An invalid basis is specified. Either too many or too few basis variables are specified.

  Response message string:
  "%d number of basis variables are specified. %d are expected."

- err_basis_factor

  Corresponding constant:
  MSK_RES_ERR_BASIS_FACTOR

  Description:
  The factorization of the basis is invalid.

  Response message string:
  "The factorization of the basis is invalid."

- err_basis_singular

  Corresponding constant:
  MSK_RES_ERR_BASIS_SINGULAR

  Description:
  The basis is singular and hence cannot be factored.

  Response message string:
  "The basis is singular."

- err_cannot_clone_nl

Corresponding constant:
MSK_RES_ERR_CANNOT_CLONE_NL

Description:
A task with a nonlinear function call-back cannot be cloned.

Response message string:
"A task with a nonlinear function call-back cannot be cloned."

- err_cannot_handle_nl

Corresponding constant:
MSK_RES_ERR_CANNOT_HANDLE_NL

Description:
A function cannot handle a task with nonlinear function call-backs.

Response message string:
"A function cannot handle a task with nonlinear function call-backs."

- err_con_q_not_nsd

Corresponding constant:
MSK_RES_ERR_CON_Q_NOT_NSD

Description:
The quadratic constraint matrix is not negative semi-definite as expected for a constraint with finite lower bound. This results in a nonconvex problem.

Response message string:
"The quadratic constraint matrix in constraint '%s'(%d) is not negative semi-definite as expected for a constraint with finite lower bound."

- err_con_q_not_psd

Corresponding constant:
MSK_RES_ERR_CON_Q_NOT_PSD

Description:
The quadratic constraint matrix is not positive semi-definite as expected for a constraint with finite upper bound. This results in a nonconvex problem.

Response message string:
"The quadratic constraint matrix in constraint '%s'(%d) is not positive semi-definite as expected for a constraint with finite upper bound."

- err_concurrent_optimizer

Corresponding constant:
MSK_RES_ERR_CONCURRENT_OPTIMIZER

Description:
An unsupported optimizer was chosen for use with the concurrent optimizer.

Response message string:
"Optimizer '%s' is unsupported in the concurrent optimizer."

- err_cone_index

  Corresponding constant:
  MSK_RES_ERR_CONE_INDEX

  Description:
  An index of a non-existing cone has been specified.

  Response message string:
  "No cone has index '%d'."

- err_cone_overlap

  Corresponding constant:
  MSK_RES_ERR_CONE_OVERLAP

  Description:
  A new cone which variables overlap with an existing cone has been specified.

  Response message string:
  "Variable '%s' (%d) is a member of cone '%s' (%d) and cone '%s' (%d)."

- err_cone_rep_var

  Corresponding constant:
  MSK_RES_ERR_CONE_REP_VAR

  Description:
  A variable is included multiple times in the cone.

  Response message string:
  "Variable '%s' (%d) are included multiple times in cone '%s' (%d)."

- err_cone_size

  Corresponding constant:
  MSK_RES_ERR_CONE_SIZE

  Description:
  A cone with too few members is specified.

  Response message string:
  "A cone with too few member is specified. At least %d members are required for cones of type %s."

- err_cone_type

  Corresponding constant:
  MSK_RES_ERR_CONE_TYPE

  Description:
  Invalid cone type specified.

  Response message string:
  "%d is an invalid cone type specified."

- err_cone_type_str

Corresponding constant:
MSK_RES_ERR_CONE_TYPE_STR

Description:
Invalid cone type specified.

Response message string:
“%d is an invalid cone type specified.”

- err_data_file_ext

Corresponding constant:
MSK_RES_ERR_DATA_FILE_EXT

Description:
The data file format cannot be determined from the file name.

Response message string:
“The data file format cannot be determined from the file name '%s'”

- err_dup_name

Corresponding constant:
MSK_RES_ERR_DUP_NAME

Description:
An error occurred while reading an MPS file..

Response message string:
“Name '%s' is already assigned for an item %s.”

- err_end_of_file

Corresponding constant:
MSK_RES_ERR_END_OF_FILE

Description:
End of file reached.

Response message string:
“End of file reached.”

- err_factor

Corresponding constant:
MSK_RES_ERR_FACTOR

Description:
An error occurred while factorizing a matrix.

Response message string:
“An error occurred while factorizing a matrix.”

- err_feasrepair_cannot_relax

Corresponding constant:
MSK_RES_ERR_FEASREPAIR_CANNOT_RELAX

Description:
An optimization problem cannot be relaxed. This is the case e.g. for general nonlinear optimization problems.

Response message string:
"An optimization problem cannot be relaxed."

- err_feasrepair_inconsistent_bound

  Corresponding constant:
  MSK_RES_ERR_FEASREPAIR_INCONSISTENT_BOUND

  Description:
  The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.

  Response message string:
  "The %s '%s' with index '%d' has lower bound larger than upper bound."

- err_feasrepair_solving_relaxed

  Corresponding constant:
  MSK_RES_ERR_FEASREPAIR_SOLVING_RELAXED

  Description:
  The relaxed problem could not be solved to optimality. Please consult the log file for further details.

  Response message string:
  "The relaxed problem could not be solved to optimality."

- err_file_license

  Corresponding constant:
  MSK_RES_ERR_FILE_LICENSE

  Description:
  Invalid license file.

  Response message string:
  "Invalid license file."

- err_file_open

  Corresponding constant:
  MSK_RES_ERR_FILE_OPEN

  Description:
  Error while opening a file.

  Response message string:
  "An error occurred while opening file '%s'."

- err_file_read

  Corresponding constant:
  MSK_RES_ERR_FILE_READ

Description:
File read error.

Response message string:
"`An error occurred while reading file '%s'.`"

- `err_file_write`

  Corresponding constant:
  `MSK_RES_ERR_FILE_WRITE`

  Description:
  File write error.

  Response message string:
  "`An error occurred while writing to file '%s'.`"

- `err_first`

  Corresponding constant:
  `MSK_RES_ERR_FIRST`

  Description:
  Invalid `first`.

  Response message string:
  "`Invalid first.`"

- `err_firsti`

  Corresponding constant:
  `MSK_RES_ERR_FIRSTI`

  Description:
  Invalid `firsti`.

  Response message string:
  "`'%d' is an invalid value for firsti.`"

- `err_firstj`

  Corresponding constant:
  `MSK_RES_ERR_FIRSTJ`

  Description:
  Invalid `firstj`.

  Response message string:
  "`'%d' is an invalid value for firstj.`"

- `err_flexlm`

  Corresponding constant:
  `MSK_RES_ERR_FLEXLM`

  Description:
  The FLEXlm license manager reported an error.

Response message string:
"The FLEXlm license manager reported '%s'."

- err_huge_c

  Corresponding constant:
  MSK_RES_ERR_HUGE_C

  Description:
  A huge value in absolute size is specified for one c_j.

  Response message string:
  "A large value of %8.1e has been specified in cx for variable '%s' (%d)."

- err_identical_tasks

  Corresponding constant:
  MSK_RES_ERR_IDENTICAL_TASKS

  Description:
  Some tasks related to this function call were identical. Unique tasks were expected.

  Response message string:
  "Some tasks related to this function call were identical. Unique tasks were expected."

- err_in_argument

  Corresponding constant:
  MSK_RES_ERR_IN_ARGUMENT

  Description:
  A function argument is incorrect.

  Response message string:
  "The argument '%s' is invalid."

- err_index

  Corresponding constant:
  MSK_RES_ERR_INDEX

  Description:
  An index is out of range.

  Response message string:
  "An index is out of range."

- err_index_arr_is_too_large

  Corresponding constant:
  MSK_RES_ERR_INDEX_ARR_IS_TOO_LARGE

  Description:
  An index in an array argument is too large.

Response message string:
"The index value %d occurring in argument '%s[%d]' is too large(<%d)."

- err_index_arr_is_too_small

Corresponding constant:
MSK_RES_ERR_INDEX_ARR_IS_TOO_SMALL

Description:
An index in an array argument is too small.

Response message string:
"The index value %d occurring in argument '%s[%d]' is too small(>=%d)."

- err_index_is_too_large

Corresponding constant:
MSK_RES_ERR_INDEX_IS_TOO_LARGE

Description:
An index in an argument is too large.

Response message string:
"The index value %d occurring in argument '%s' is too large."

- err_index_is_too_small

Corresponding constant:
MSK_RES_ERR_INDEX_IS_TOO_SMALL

Description:
An index in an argument is too small.

Response message string:
"The index value %d occurring in argument '%s' is too small."

- err_inf_dou_index

Corresponding constant:
MSK_RES_ERR_INF_DOU_INDEX

Description:
A double information index is out of range for the specified type.

Response message string:
"The double information index %d is out of range."

- err_inf_dou_name

Corresponding constant:
MSK_RES_ERR_INF_DOU_NAME

Description:
A double information name is invalid.

Response message string:
"The double information name '%s' is invalid."

- err_inf_int_index

  Corresponding constant:
  MSK_RES_ERR_INF_INT_INDEX

  Description:
  An integer information index is out of range for the specified type.

  Response message string:
  "The integer information index %d is out of range."

- err_inf_int_name

  Corresponding constant:
  MSK_RES_ERR_INF_INT_NAME

  Description:
  A integer information name is invalid.

  Response message string:
  "The integer information name '%s' is invalid."

- err_inf_type

  Corresponding constant:
  MSK_RES_ERR_INF_TYPE

  Description:
  The information type is invalid.

  Response message string:
  "The information type %d is invalid."

- err_infinite_bound

  Corresponding constant:
  MSK_RES_ERR_INFINITE_BOUND

  Description:
  A finite bound value is too large in absolute value.

  Response message string:
  "A finite bound value is too large in absolute value."

- err_internal

  Corresponding constant:
  MSK_RES_ERR_INTERNAL

  Description:
  An internal error occurred. Please report this problem.

  Response message string:
  "An internal error occurred '%s'."

- err_internal_test_failed

Corresponding constant:
MSK_RES_ERR_INTERNAL_TEST_FAILED

Description:
An internal unit test function failed.

Response message string:
"Internal unit test function failed."

- err_inv_aptre

  Corresponding constant:
  MSK_RES_ERR_INV_APTRE

  Description:
  aptre[j] is strictly smaller than aptrb[j] for some j.

  Response message string:
  "aptre is strictly smaller than aptrb at position %d."

- err_inv_bk

  Corresponding constant:
  MSK_RES_ERR_INV_BK

  Description:
  Invalid bound key.

  Response message string:
  "%d is an invalid bound key."

- err_inv_bkc

  Corresponding constant:
  MSK_RES_ERR_INV_BKC

  Description:
  Invalid bound key is specified for a constraint.

  Response message string:
  "An invalid bound key for a constraint value of %d in argument '%s' has been specified."

- err_inv_bkx

  Corresponding constant:
  MSK_RES_ERR_INV_BKX

  Description:
  An invalid bound key is specified for a variable.

  Response message string:
  "An invalid bound key for variable of value of %d in argument '%s' has been specified."

- err_inv_cone_type

Corresponding constant:
MSK_RES_ERR_INV_CONE_TYPE

Description:
Invalid cone type code is encountered.

Response message string:
"'%d' is an invalid cone type code."

- err_inv_cone_type_str

Corresponding constant:
MSK_RES_ERR_INV_CONE_TYPE_STR

Description:
Invalid cone type string encountered.

Response message string:
"'%s' is an invalid cone type string."

- err_inv_marki

Corresponding constant:
MSK_RES_ERR_INV_MARKI

Description:
Invalid value in marki.

Response message string:
"Invalid value in marki[%d]."

- err_inv_markj

Corresponding constant:
MSK_RES_ERR_INV_MARKJ

Description:
Invalid value in markj.

Response message string:
"Invalid value in markj[%d]."

- err_inv_name_item

Corresponding constant:
MSK_RES_ERR_INV_NAME_ITEM

Description:
An invalid name item code is used.

Response message string:
"'%d' is an invalid name item code."

- err_inv_numi

Corresponding constant:
MSK_RES_ERR_INV_NUMI

Description:
Invalid numi.

Response message string:
"`Invalid numi.`"

- `err_inv_numj`

Corresponding constant:
`MSK_RES_ERR_INV_NUMJ`

Description:
Invalid numj.

Response message string:
"`Invalid numj.`"

- `err_inv_optimizer`

Corresponding constant:
`MSK_RES_ERR_INV_OPTIMIZER`

Description:
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.

Response message string:
"`An invalid optimizer (%d) has been chosen for the problem.`"

- `err_inv_problem`

Corresponding constant:
`MSK_RES_ERR_INV_PROBLEM`

Description:
Invalid problem type. Probably a nonconvex problem has been specified.

Response message string:
"`Invalid problem type.`"

- `err_inv_qcon_subi`

Corresponding constant:
`MSK_RES_ERR_INV_QCON_SUBI`

Description:
Invalid value in `qcsubi`.

Response message string:
"`Invalid value %d at qcsubi[%d].`"

- `err_inv_qcon_subj`

Corresponding constant:
`MSK_RES_ERR_INV_QCON_SUBJ`

Description:
Invalid value in qcsubj.

Response message string:
"Invalid value %d at qcsubj[%d]."

- err_inv_qcon_subk

Corresponding constant:
MSK_RES_ERR_INV_QCON_SUBK

Description:
Invalid value in qcsubk.

Response message string:
"Invalid value %d at qcsubk[%d]."

- err_inv_qcon_val

Corresponding constant:
MSK_RES_ERR_INV_QCON_VAL

Description:
Invalid value in qcval.

Response message string:
"Invalid value %e at qcval[%d]."

- err_inv_qobj_subi

Corresponding constant:
MSK_RES_ERR_INV_QOBJ_SUBI

Description:
Invalid value in qosubi.

Response message string:
"Invalid value %d at qosubi[%d]."

- err_inv_qobj_subj

Corresponding constant:
MSK_RES_ERR_INV_QOBJ_SUBJ

Description:
Invalid value in qosubj.

Response message string:
"Invalid value %d at qosubj[%d]."

- err_inv_qobj_val

Corresponding constant:
MSK_RES_ERR_INV_QOBJ_VAL

Description:
Invalid value in qoval.

Response message string:
"Invalid value %e at qoval[%d]."

- err_inv_sk

  Corresponding constant:
  MSK_RES_ERR_INV_SK

  Description:
  Invalid status key code.

  Response message string:
  "'%d' is an invalid status key code."

- err_inv_sk_str

  Corresponding constant:
  MSK_RES_ERR_INV_SK_STR

  Description:
  Invalid status key string encountered.

  Response message string:
  "'%s' is an invalid status key string."

- err_inv_skc

  Corresponding constant:
  MSK_RES_ERR_INV_SKC

  Description:
  Invalid value in skc.

  Response message string:
  "Invalid value at skc[%d]."

- err_inv_skn

  Corresponding constant:
  MSK_RES_ERR_INV_SKN

  Description:
  Invalid value in skn.

  Response message string:
  "Invalid value at skn[%d]."

- err_inv_skx

  Corresponding constant:
  MSK_RES_ERR_INV_SKX

  Description:
  Invalid value in skx.

  Response message string:
  "Invalid value at skx[%d]."

- err_inv_var_type

  Corresponding constant:
  MSK_RES_ERR_INV_VAR_TYPE

  Description:
  An invalid variable type is specified for a variable.

  Response message string:
  "An invalid type %d is specified for variable '%s' (%d) in argument '%s'."

- err_invalid_accmode

  Corresponding constant:
  MSK_RES_ERR_INVALID_ACCMODE

  Description:
  An invalid access mode is specified.

  Response message string:
  "%d is an invalid access mode is specified."

- err_invalid_ampl_stub

  Corresponding constant:
  MSK_RES_ERR_INVALID_AMPL_STUB

  Description:
  Invalid AMPL stub.

  Response message string:
  "Invalid AMPL stub."

- err_invalid_branch_direction

  Corresponding constant:
  MSK_RES_ERR_INVALID_BRANCH_DIRECTION

  Description:
  An invalid branching direction is specified.

  Response message string:
  "%d is an invalid branching direction."

- err_invalid_branch_priority

  Corresponding constant:
  MSK_RES_ERR_INVALID_BRANCH_PRIORITY

  Description:
  An invalid branching priority is specified. It should be nonnegative.

  Response message string:
  "%d invalid branching priority is specified."

- err_invalid_compression

Corresponding constant:
MSK_RES_ERR_INVALID_COMPRESSION

Description:
Invalid compression type.

Response message string:
"%d is an invalid compression type."

- err_invalid_file_name

Corresponding constant:
MSK_RES_ERR_INVALID_FILE_NAME

Description:
An invalid file name has been specified.

Response message string:
"'%s' is invalid file name."

- err_invalid_format_type

Corresponding constant:
MSK_RES_ERR_INVALID_FORMAT_TYPE

Description:
Invalid format type.

Response message string:
"%d is an invalid format type.."

- err_invalid_iomode

Corresponding constant:
MSK_RES_ERR_INVALID_IOMODE

Description:
Invalid io mode.

Response message string:
"%d is an io mode."

- err_invalid_mbt_file

Corresponding constant:
MSK_RES_ERR_INVALID_MBT_FILE

Description:
A MOSEK binary task file is invalid.

Response message string:
"The binary task file is invalid."

- err_invalid_name_in_sol_file

Corresponding constant:
MSK_RES_ERR_INVALID_NAME_IN_SOL_FILE

Description:
An invalid name occurred in a solution file.

Response message string:
"The name '%s' is an invalid name."

- err_invalid_obj_name

Corresponding constant:
MSK_RES_ERR_INVALID_OBJ_NAME

Description:
An invalid objective name is specified.

Response message string:
"'%s' is an invalid objective name is specified."

- err_invalid_objective_sense

Corresponding constant:
MSK_RES_ERR_INVALID_OBJECTIVE_SENSE

Description:
An invalid objective sense is specified.

Response message string:
"%s is an invalid objective sense code."

- err_invalid_sol_file_name

Corresponding constant:
MSK_RES_ERR_INVALID_SOL_FILE_NAME

Description:
An invalid file name has been specified.

Response message string:
"'%s' is invalid solution file name."

- err_invalid_stream

Corresponding constant:
MSK_RES_ERR_INVALID_STREAM

Description:
An invalid stream is referenced.

Response message string:
"%d is an invalid stream."

- err_invalid_task

Corresponding constant:
MSK_RES_ERR_INVALID_TASK

Description:
The task is invalid.

Response message string:
"The task is invalid."

- err_invalid_utf8

  Corresponding constant:
  MSK_RES_ERR_INVALID_UTF8

  Description:
  An invalid UTF8 string is encountered.

  Response message string:
  "An invalid UTF8 string is encountered."

- err_invalid_wchar

  Corresponding constant:
  MSK_RES_ERR_INVALID_WCHAR

  Description:
  An invalid wchar string is encountered.

  Response message string:
  "An invalid wchar string is encountered."

- err_last

  Corresponding constant:
  MSK_RES_ERR_LAST

  Description:
  Invalid last.

  Response message string:
  "Invalid last."

- err_lasti

  Corresponding constant:
  MSK_RES_ERR_LASTI

  Description:
  Invalid lasti.

  Response message string:
  "'%d' is an invalid value for lasti."

- err_lastj

  Corresponding constant:
  MSK_RES_ERR_LASTJ

  Description:
  Invalid lastj.

  Response message string:
  "'%d' is an invalid value for lastj."

- err_license

  Corresponding constant:
  MSK_RES_ERR_LICENSE

  Description:
  Invalid license.

  Response message string:
  "Invalid license."

- err_license_cannot_allocate

  Corresponding constant:
  MSK_RES_ERR_LICENSE_CANNOT_ALLOCATE

  Description:
  The license system cannot allocate the memory required.

  Response message string:
  "The license system cannot allocate the memory required."

- err_license_cannot_connect

  Corresponding constant:
  MSK_RES_ERR_LICENSE_CANNOT_CONNECT

  Description:
  MOSEK cannot connect to the license server. Most likely the license server is not up and running.

  Response message string:
  "MOSEK cannot connect to the license server."

- err_license_expired

  Corresponding constant:
  MSK_RES_ERR_LICENSE_EXPIRED

  Description:
  The license has expired.

  Response message string:
  "The license has expired."

- err_license_feature

  Corresponding constant:
  MSK_RES_ERR_LICENSE_FEATURE

  Description:
  A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.

  Response message string:
  "The feature '%s' is not in license file. Consult the license manager error message."

- `err_license_invalid_hostid`

  Corresponding constant:
  `MSK_RES_ERR_LICENSE_INVALID_HOSTID`

  Description:
  The host ID specified in the license file does not match the host ID of the computer.

  Response message string:
  "`The host ID specified in the license file does not match the host ID of the computer.`"

- `err_license_max`

  Corresponding constant:
  `MSK_RES_ERR_LICENSE_MAX`

  Description:
  Maximum number of licenses is reached.

  Response message string:
  "`Maximum number of licenses is reached for feature '%s'.`"

- `err_license_moseklm_daemon`

  Corresponding constant:
  `MSK_RES_ERR_LICENSE_MOSEKLM_DAEMON`

  Description:
  The MOSEKLM license manager daemon is not up and running.

  Response message string:
  "`The MOSEKLM license manager daemon is not up and running.`"

- `err_license_server`

  Corresponding constant:
  `MSK_RES_ERR_LICENSE_SERVER`

  Description:
  The license server is not responding.

  Response message string:
  "`The license server is not responding.`"

- `err_license_server_version`

  Corresponding constant:
  `MSK_RES_ERR_LICENSE_SERVER_VERSION`

  Description:
  The version specified in the checkout request is greater than the highest version number the daemon supports.

  Response message string:
  "`License server system does not support this version '%d' of feature '%s'.`"

- `err_license_version`

  Corresponding constant:
  MSK_RES_ERR_LICENSE_VERSION

  Description:
  The license is valid for another version of MOSEK.

  Response message string:
  "Version '%s' of feature '%s' is not supported by the license file."

- `err_link_file_dll`

  Corresponding constant:
  MSK_RES_ERR_LINK_FILE_DLL

  Description:
  A file cannot be linked to a stream in the DLL version.

  Response message string:
  "A file cannot be linked to a stream in the DLL version."

- `err_lp_dup_slack_name`

  Corresponding constant:
  MSK_RES_ERR_LP_DUP_SLACK_NAME

  Description:
  The name of the slack variable added to a ranged constraint already exists.

  Response message string:
  "Slack variable name '%s' in constraint '%s' id defined already."

- `err_lp_empty`

  Corresponding constant:
  MSK_RES_ERR_LP_EMPTY

  Description:
  The problem cannot be written to an LP formatted file.

  Response message string:
  "A problem with no variables or constraints cannot be written to a LP formatted file."

- `err_lp_file_format`

  Corresponding constant:
  MSK_RES_ERR_LP_FILE_FORMAT

  Description:
  Syntax error in an LP file.

  Response message string:
  "Syntax error in an LP file at (%d:%d)."

- `err_lp_format`

Corresponding constant:
MSK_RES_ERR_LP_FORMAT

Description:
Syntax error in an LP file.

Response message string:
"Syntax error in an LP file at line number: %d."

- err_lp_free_constraint

  Corresponding constant:
  MSK_RES_ERR_LP_FREE_CONSTRAINT

  Description:
  Free constraints cannot be written in LP file format.

  Response message string:
  "Free constraints cannot be written in LP file format."

- err_lp_incompatible

  Corresponding constant:
  MSK_RES_ERR_LP_INCOMPATIBLE

  Description:
  The problem cannot be written to an LP formatted file.

  Response message string:
  "A problem of type '%s' cannot be written to a LP formatted file."

- err_lp_invalid_var_name

  Corresponding constant:
  MSK_RES_ERR_LP_INVALID_VAR_NAME

  Description:
  A variable name is invalid when used in an LP formatted file.

  Response message string:
  "The variable name '%s' cannot be written to an LP formatted file."

- err_lp_write_conic_problem

  Corresponding constant:
  MSK_RES_ERR_LP_WRITE_CONIC_PROBLEM

  Description:
  The problem contains cones that cannot be written to an LP formatted file.

  Response message string:
  "A problem of type '%s' contains cones that cannot be written to an LP formatted file."

- err_lp_write_geco_problem

Corresponding constant:
MSK_RES_ERR_LP_WRITE_GECO_PROBLEM

Description:
The problem contains general convex terms that cannot be written to an LP formatted file.

Response message string:
"A problem of type '%s' contains general convex terms that cannot be written to an LP formatted file."

- err_lu_max_num_tries

Corresponding constant:
MSK_RES_ERR_LU_MAX_NUM_TRIES

Description:
Could not compute the LU factors of the matrix within the maximum number of allowed tries.

Response message string:
"Could not compute the LU factors of the matrix within the maximum number of allowed tries."

- err_maxnumanz

Corresponding constant:
MSK_RES_ERR_MAXNUMANZ

Description:
The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A.

Response message string:
"Too small maximum number of non-zeros in A specified."

- err_maxnumcon

Corresponding constant:
MSK_RES_ERR_MAXNUMCON

Description:
The maximum number of constraints specified is smaller than the number of constraints in the task.

Response message string:
"Maximum number of constraints of '%d' is smaller than the number of constraints '%d'."

- err_maxnumcone

Corresponding constant:
MSK_RES_ERR_MAXNUMCONE

Description:
The value specified for `maxnumcone` is too small.

Response message string:
"The value %d specified for maxnumcone is too small."

- err_maxnumqnz

Corresponding constant:
MSK_RES_ERR_MAXNUMQNZ

Description:
The maximum number of non-zeros specified for the Q matrices is smaller than the number of non-zeros in the current Q matrices.

Response message string:
"Too small maximum number of non-zeros for the Q matrices is specified."

- err_maxnumvar

Corresponding constant:
MSK_RES_ERR_MAXNUMVAR

Description:
The maximum number of variables specified is smaller than the number of variables in the task.

Response message string:
"Too small maximum number of variables %d is specified. Currently, the number of variables is %d."

- err_mbt_incompatible

Corresponding constant:
MSK_RES_ERR_MBT_INCOMPATIBLE

Description:
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.

Response message string:
"The MBT file is incompatible withe this platform."

- err_mio_no_optimizer

Corresponding constant:
MSK_RES_ERR_MIO_NO_OPTIMIZER

Description:
No optimizer is available for the current class of integer optimization problems.

Response message string:
"No integer optimizer is available for the optimization problem."

- err_mio_not_loaded

Corresponding constant:
MSK_RES_ERR_MIO_NOT_LOADED

Description:
The mixed-integer optimizer is not loaded.

Response message string:
"The mixed-integer optimizer is not loaded."

- err_missing_license_file

  Corresponding constant:
  MSK_RES_ERR_MISSING_LICENSE_FILE

  Description:
  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.

  Response message string:
  "A license file is missing. Set MOSEKLM_LICENSE_FILE to point to your license file."

- err_mixed_problem

  Corresponding constant:
  MSK_RES_ERR_MIXED_PROBLEM

  Description:
  The problem contains both conic and nonlinear constraints.

  Response message string:
  "The problem contains both conic and nonlinear constraints."

- err_mps_cone_overlap

  Corresponding constant:
  MSK_RES_ERR_MPS_CONE_OVERLAP

  Description:
  A variable is specified to be a member of several cones.

  Response message string:
  "Variable '%s' is specified to be a member of CSECTION '%s' and CSECTION '%s'."

- err_mps_cone_repeat

  Corresponding constant:
  MSK_RES_ERR_MPS_CONE_REPEAT

  Description:
  A variable is repeated within the CSECTION.

  Response message string:
  "Variable '%s' is repeated in CSECTION '%s'."

- err_mps_cone_type

  Corresponding constant:
  MSK_RES_ERR_MPS_CONE_TYPE

Description:
Invalid cone type specified in a CSECTION.

Response message string:
"'%s' is an invalid cone type in a CSECTION."

- err_mps_file

Corresponding constant:
MSK_RES_ERR_MPS_FILE

Description:
An error occurred while reading an MPS file.

Response message string:
"An error occurred while reading an MPS file."

- err_mps_inv_bound_key

Corresponding constant:
MSK_RES_ERR_MPS_INV_BOUND_KEY

Description:
An invalid bound key occurred in an MPS file.

Response message string:
"'%s' is an invalid bound key."

- err_mps_inv_con_key

Corresponding constant:
MSK_RES_ERR_MPS_INV_CON_KEY

Description:
An invalid constraint key occurred in an MPS file.

Response message string:
"'%s' is an invalid constraint key for constraint '%s'."

- err_mps_inv_field

Corresponding constant:
MSK_RES_ERR_MPS_INV_FIELD

Description:
A field in the MPS file is invalid. Probably it is too wide.

Response message string:
"Field number %d is invalid."

- err_mps_inv_marker

Corresponding constant:
MSK_RES_ERR_MPS_INV_MARKER

Description:
An invalid marker has been specified in the MPS file.

Response message string:
"An invalid marker has been specified in the MPS file."

- err_mps_inv_sec_name

Corresponding constant:
MSK_RES_ERR_MPS_INV_SEC_NAME

Description:
An invalid section name occurred in an MPS file.

Response message string:
"An invalid section name was used."

- err_mps_inv_sec_order

Corresponding constant:
MSK_RES_ERR_MPS_INV_SEC_ORDER

Description:
The sections in the MPS data file are not in the correct order.

Response message string:
"Section '%s' was not expected."

- err_mps_invalid_obj_name

Corresponding constant:
MSK_RES_ERR_MPS_INVALID_OBJ_NAME

Description:
An invalid objective name is specified.

Response message string:
"'%s' is an invalid objective name is specified."

- err_mps_invalid_objsense

Corresponding constant:
MSK_RES_ERR_MPS_INVALID_OBJSENSE

Description:
An invalid objective sense is specified.

Response message string:
"'%s' is an invalid objective sense."

- err_mps_mul_con_name

Corresponding constant:
MSK_RES_ERR_MPS_MUL_CON_NAME

Description:
A constraint name was specified multiple times in the ROWS section.

Response message string:
"The constraint name '%s' was specified multiple times in the ROWS section."

- err_mps_mul_csec

  Corresponding constant:
  MSK_RES_ERR_MPS_MUL_CSEC

  Description:
  Multiple CSECTIONs are given the same name.

  Response message string:
  "Multiple CSECTIONs are given the name '%s'."

- err_mps_mul_qobj

  Corresponding constant:
  MSK_RES_ERR_MPS_MUL_QOBJ

  Description:
  The Q term in the objective is specified multiple times in the MPS data file.

  Response message string:
  "The Q term in the objective is specified multiple times."

- err_mps_mul_qsec

  Corresponding constant:
  MSK_RES_ERR_MPS_MUL_QSEC

  Description:
  Multiple QSECTIONs are specified for a constraint in the MPS data file.

  Response message string:
  "Multiple QSECTIONs are specified for a constraint '%s'."

- err_mps_no_objective

  Corresponding constant:
  MSK_RES_ERR_MPS_NO_OBJECTIVE

  Description:
  No objective is defined in an MPS file.

  Response message string:
  "No objective was defined."

- err_mps_null_con_name

  Corresponding constant:
  MSK_RES_ERR_MPS_NULL_CON_NAME

  Description:
  An empty constraint name is used in an MPS file.

  Response message string:
  "An empty constraint name is used in an MPS file."

- err_mps_null_var_name

Corresponding constant:
MSK_RES_ERR_MPS_NULL_VAR_NAME

Description:
An empty variable name is used in an MPS file.

Response message string:
"An empty variable name is used in an MPS file."

- err_mps_splitted_var

Corresponding constant:
MSK_RES_ERR_MPS_SPLITTED_VAR

Description:
A variable is split in an MPS data file.

Response message string:
"The variable '%s' was split."

- err_mps_tab_in_field2

Corresponding constant:
MSK_RES_ERR_MPS_TAB_IN_FIELD2

Description:
A tab char occurred in field 2.

Response message string:
"A tab char occurred in field 2."

- err_mps_tab_in_field3

Corresponding constant:
MSK_RES_ERR_MPS_TAB_IN_FIELD3

Description:
A tab char occurred in field 3.

Response message string:
"A tab char occurred in field 3."

- err_mps_tab_in_field5

Corresponding constant:
MSK_RES_ERR_MPS_TAB_IN_FIELD5

Description:
A tab char occurred in field 5.

Response message string:
"A tab char occurred in field 5."

- err_mps_undef_con_name

Corresponding constant:
MSK_RES_ERR_MPS_UNDEF_CON_NAME

Description:
An undefined constraint name occurred in an MPS file.

Response message string:
"`'%s' is an undefined constraint name.`"

- `err_mps_undef_var_name`

  Corresponding constant:
  `MSK_RES_ERR_MPS_UNDEF_VAR_NAME`

  Description:
  An undefined variable name occurred in an MPS file.

  Response message string:
  "`'%s' is an undefined variable name.`"

- `err_mul_a_element`

  Corresponding constant:
  `MSK_RES_ERR_MUL_A_ELEMENT`

  Description:
  An element in A is defined multiple times.

  Response message string:
  "`Multiple elements in row %d of A at column %d.`"

- `err_name_max_len`

  Corresponding constant:
  `MSK_RES_ERR_NAME_MAX_LEN`

  Description:
  A name is longer than the buffer that is supposed to hold it.

  Response message string:
  "`A name('%s') of length %d is longer than the buffer of length %d that is supposed to hold it.`"

- `err_nan_in_blc`

  Corresponding constant:
  `MSK_RES_ERR_NAN_IN_BLC`

  Description:
  l^c contains an invalid floating point value, i.e. a `NaN`.

  Response message string:
  "`The bound vector blc contains an invalid value for constraint '%s' (%d).`"

- `err_nan_in_blx`

  Corresponding constant:
  `MSK_RES_ERR_NAN_IN_BLX`

Description:
l^x contains an invalid floating point value, i.e. a NaN.

Response message string:
"The bound vector blx contains an invalid value for variable '%s' (%d)."

- err_nan_in_buc

Corresponding constant:
MSK_RES_ERR_NAN_IN_BUC

Description:
u^c contains an invalid floating point value, i.e. a NaN.

Response message string:
"The bound vector buc contains an invalid value for constraint '%s' (%d)."

- err_nan_in_bux

Corresponding constant:
MSK_RES_ERR_NAN_IN_BUX

Description:
u^x contains an invalid floating point value, i.e. a NaN.

Response message string:
"The bound vector bux contains an invalid value for variable '%s' (%d)."

- err_nan_in_c

Corresponding constant:
MSK_RES_ERR_NAN_IN_C

Description:
c contains an invalid floating point value, i.e. a NaN.

Response message string:
"The objective vector c contains an invalid value for variable '%s' (%d)."

- err_nan_in_double_data

Corresponding constant:
MSK_RES_ERR_NAN_IN_DOUBLE_DATA

Description:
An invalid floating point value was used in some double data.

Response message string:
"The parameter '%s' contained an invalid floating value."

- err_negative_append

Corresponding constant:
MSK_RES_ERR_NEGATIVE_APPEND

Description:
Cannot append a negative number.

Response message string:
"Cannot append a negative number of %d."

- err_negative_surplus

Corresponding constant:
MSK_RES_ERR_NEGATIVE_SURPLUS

Description:
Negative surplus.

Response message string:
"Negative surplus."

- err_newer_dll

Corresponding constant:
MSK_RES_ERR_NEWER_DLL

Description:
The dynamic link library is newer than the specified version.

Response message string:
"The dynamic link library version %d.%d.%d.%d is newer than version %d.%d.%d.%d."

- err_no_basis_sol

Corresponding constant:
MSK_RES_ERR_NO_BASIS_SOL

Description:
No basic solution is defined.

Response message string:
"No basic solution is defined."

- err_no_dual_for_itg_sol

Corresponding constant:
MSK_RES_ERR_NO_DUAL_FOR_ITG_SOL

Description:
No dual information is available for the integer solution.

Response message string:
"No dual information is available for the integer solution."

- err_no_dual_infeas_cer

Corresponding constant:
MSK_RES_ERR_NO_DUAL_INFEAS_CER

Description:
A certificate of infeasibility is not available.

Response message string:
"A certificate of dual infeasibility is not available."

- `err_no_init_env`

  Corresponding constant:
  `MSK_RES_ERR_NO_INIT_ENV`

  Description:
  `env` is not initialized.

  Response message string:
  "`Environment is not initialized.`"

- `err_no_optimizer_var_type`

  Corresponding constant:
  `MSK_RES_ERR_NO_OPTIMIZER_VAR_TYPE`

  Description:
  No optimizer is available for this class of optimization problems.

  Response message string:
  "`No optimizer is available for optimization problems containing variables of type '%s'.`"

- `err_no_primal_infeas_cer`

  Corresponding constant:
  `MSK_RES_ERR_NO_PRIMAL_INFEAS_CER`

  Description:
  A certificate of primal infeasibility is not available.

  Response message string:
  "`A certificate of primal infeasibility is not available.`"

- `err_no_solution_in_callback`

  Corresponding constant:
  `MSK_RES_ERR_NO_SOLUTION_IN_CALLBACK`

  Description:
  The required solution is not available.

  Response message string:
  "`The required solution is not available.`"

- `err_nonconvex`

  Corresponding constant:
  `MSK_RES_ERR_NONCONVEX`

  Description:
  The optimization problem is nonconvex.

  Response message string:
  "`The optimization problem is nonconvex.`"

- `err_nonlinear_equality`

Corresponding constant:
MSK_RES_ERR_NONLINEAR_EQUALITY

Description:
The model contains a nonlinear equality which defines a nonconvex set.

Response message string:
"Non convex model detected. Constraint '%s'(%d) is a nonlinear equality."

- err_nonlinear_ranged

Corresponding constant:
MSK_RES_ERR_NONLINEAR_RANGED

Description:
The model contains a nonlinear ranged constraint which by definition defines a nonconvex set.

Response message string:
"Constraint '%s(%d)' is nonlinear and ranged constraint i.e. it has finite lower and upper bound."

- err_nr_arguments

Corresponding constant:
MSK_RES_ERR_NR_ARGUMENTS

Description:
Incorrect number of function arguments.

Response message string:
"Incorrect number of %s arguments. Got %d expected %d."

- err_null_env

Corresponding constant:
MSK_RES_ERR_NULL_ENV

Description:
env is a NULL pointer.

Response message string:
"env is a NULL pointer."

- err_null_name

Corresponding constant:
MSK_RES_ERR_NULL_NAME

Description:
An all blank name has been specified.

Response message string:
"An all blank name has been specified."

- err_null_pointer

Corresponding constant:
`MSK_RES_ERR_NULL_POINTER`

Description:
An argument to a function is unexpectedly a NULL pointer.

Response message string:
"`Argument '%s' is unexpectedly a NULL pointer.`"

- `err_null_task`

Corresponding constant:
`MSK_RES_ERR_NULL_TASK`

Description:
`task` is a NULL pointer.

Response message string:
"`task is a NULL pointer.`"

- `err_numconlim`

Corresponding constant:
`MSK_RES_ERR_NUMCONLIM`

Description:
Maximum number of constraints limit is exceeded.

Response message string:
"`Maximum number of constraints limit is exceeded.`"

- `err_numvarlim`

Corresponding constant:
`MSK_RES_ERR_NUMVARLIM`

Description:
Maximum number of variables limit is exceeded.

Response message string:
"`Maximum number of variables limit is exceeded.`"

- `err_obj_q_not_nsd`

Corresponding constant:
`MSK_RES_ERR_OBJ_Q_NOT_NSD`

Description:
The quadratic coefficient matrix in the objective is not negative semi-definite as expected for a maximization problem.

Response message string:
"`The quadratic coefficient matrix in the objective is not negative semi-definite as expected for a maximization problem.`"

- `err_obj_q_not_psd`

Corresponding constant:
MSK_RES_ERR_OBJ_Q_NOT_PSD

Description:
The quadratic coefficient matrix in the objective is not positive semi-definite as expected for a minimization problem.

Response message string:
"The quadratic coefficient matrix in the objective is not positive semi-definite as expected for a minimization problem."

- err_objective_range

Corresponding constant:
MSK_RES_ERR_OBJECTIVE_RANGE

Description:
Empty objective range.

Response message string:
"Empty objective range."

- err_older_dll

Corresponding constant:
MSK_RES_ERR_OLDER_DLL

Description:
The dynamic link library is older than the specified version.

Response message string:
"The dynamic link library version %d.%d.%d.%d is older than expected version %d.%d.%d.%d."

- err_open_dl

Corresponding constant:
MSK_RES_ERR_OPEN_DL

Description:
A dynamic link library could not be opened.

Response message string:
"A dynamic link library '%s' could not be opened."

- err_opf_format

Corresponding constant:
MSK_RES_ERR_OPF_FORMAT

Description:
Syntax error in an OPF file

Response message string:
"Syntax error in an OPF file at line number: %d."

- err_optimizer_license

  Corresponding constant:
  MSK_RES_ERR_OPTIMIZER_LICENSE

  Description:
  The optimizer required is not licensed.

  Response message string:
  "The optimizer required is not licensed."

- err_ord_invalid

  Corresponding constant:
  MSK_RES_ERR_ORD_INVALID

  Description:
  Invalid content in branch ordering file.

  Response message string:
  "Invalid content in branch ordering file"

- err_ord_invalid_branch_dir

  Corresponding constant:
  MSK_RES_ERR_ORD_INVALID_BRANCH_DIR

  Description:
  An invalid branch direction key is specified.

  Response message string:
  "'%s' is an invalid branch direction key is specified."

- err_param_index

  Corresponding constant:
  MSK_RES_ERR_PARAM_INDEX

  Description:
  Parameter index is out of range.

  Response message string:
  "The parameter index %d is invalid for a parameter of type %s."

- err_param_is_too_large

  Corresponding constant:
  MSK_RES_ERR_PARAM_IS_TOO_LARGE

  Description:
  The parameter value is too large.

  Response message string:
  "The parameter value %s is too large for parameter '%s'."

- err_param_is_too_small

Corresponding constant:
MSK_RES_ERR_PARAM_IS_TOO_SMALL

Description:
The parameter value is too small.

Response message string:
"The parameter value %s is too small for parameter '%s'."

- err_param_name

Corresponding constant:
MSK_RES_ERR_PARAM_NAME

Description:
The parameter name is not correct.

Response message string:
"The parameter name '%s' is invalid."

- err_param_name_dou

Corresponding constant:
MSK_RES_ERR_PARAM_NAME_DOU

Description:
The parameter name is not correct for a double parameter.

Response message string:
"The parameter name '%s' is invalid for a double parameter."

- err_param_name_int

Corresponding constant:
MSK_RES_ERR_PARAM_NAME_INT

Description:
The parameter name is not correct for an integer parameter.

Response message string:
"The parameter name '%s' is invalid for an int parameter."

- err_param_name_str

Corresponding constant:
MSK_RES_ERR_PARAM_NAME_STR

Description:
The parameter name is not correct for a string parameter.

Response message string:
"The parameter name '%s' is invalid for a string parameter."

- err_param_type

Corresponding constant:
MSK_RES_ERR_PARAM_TYPE

Description:
The parameter type is invalid.

Response message string:
"The parameter type %d is invalid."

- err_param_value_str

Corresponding constant:
MSK_RES_ERR_PARAM_VALUE_STR

Description:
The parameter value string is incorrect.

Response message string:
"The parameter value string '%s' for parameter %s is incorrect."

- err_platform_not_licensed

Corresponding constant:
MSK_RES_ERR_PLATFORM_NOT_LICENSED

Description:
A requested license feature is not available for the required platform.

Response message string:
"No license feature '%s' for the required platform is available."

- err_postsolve

Corresponding constant:
MSK_RES_ERR_POSTSOLVE

Description:
An error occurred during the postsolve. Please contact MOSEK support.

Response message string:
"An error occurred during the postsolve."

- err_pro_item

Corresponding constant:
MSK_RES_ERR_PRO_ITEM

Description:
An invalid problem is used.

Response message string:
"'%d' is an invalid problem item."

- err_prob_license

Corresponding constant:
MSK_RES_ERR_PROB_LICENSE

Description:
The software is not licensed to solve the problem.

Response message string:
"The software is not licensed to solve the problem."

- err_qcon_subi_too_large

Corresponding constant:
MSK_RES_ERR_QCON_SUBI_TOO_LARGE

Description:
Invalid value in qcsubi.

Response message string:
"Invalid value %d at qcsubi[%d]. It should be < %d."

- err_qcon_subi_too_small

Corresponding constant:
MSK_RES_ERR_QCON_SUBI_TOO_SMALL

Description:
Invalid value in qcsubi.

Response message string:
"Invalid value %d at qcsubi[%d]. It should be >= %d."

- err_qcon_upper_triangle

Corresponding constant:
MSK_RES_ERR_QCON_UPPER_TRIANGLE

Description:
An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.

Response message string:
"The element q[%d,%d] in the upper triangle of the quadratic term in the %dth constraint is specified."

- err_qobj_upper_triangle

Corresponding constant:
MSK_RES_ERR_QOBJ_UPPER_TRIANGLE

Description:
An element in the upper triangle of Q^o is specified. Only elements in the lower triangle should be specified.

Response message string:
"The element q[%d,%d] in the upper triangle of the quadratic term in the objective is specified."

- err_read_format

Corresponding constant:
MSK_RES_ERR_READ_FORMAT

Description:
The specified format cannot be read.

Response message string:
"`The specified format cannot be read. The format code is %d.`"

- `err_read_lp_nonexisting_name`

Corresponding constant:
`MSK_RES_ERR_READ_LP_NONEXISTING_NAME`

Description:
A variable never occurred in objective or constraints.

Response message string:
"`The variable name '%s' did not occur in objective or constraints.`"

- `err_remove_cone_variable`

Corresponding constant:
`MSK_RES_ERR_REMOVE_CONE_VARIABLE`

Description:
A variable cannot be removed because it will make a cone invalid.

Response message string:
"`If variable %d ('%s') is removed, then cone %d ('%s') will be invalid.`"

- `err_sen_bound_invalid_lo`

Corresponding constant:
`MSK_RES_ERR_SEN_BOUND_INVALID_LO`

Description:
Analysis of lower bound requested for an index, where no lower bound exists.

Response message string:
"`No lower bound for index '%d' given in line %d.`"

- `err_sen_bound_invalid_up`

Corresponding constant:
`MSK_RES_ERR_SEN_BOUND_INVALID_UP`

Description:
Analysis of upper bound requested for an index, where no upper bound exists.

Response message string:
"`No upper bound for index '%d' given in line %d.`"

- `err_sen_format`

Corresponding constant:
`MSK_RES_ERR_SEN_FORMAT`

Description:
Syntax error in sensitivity analysis file.

Response message string:
"Syntax error in sensitivity analysis file at line number: %d. %s"

- err_sen_index_invalid

  Corresponding constant:
  MSK_RES_ERR_SEN_INDEX_INVALID

  Description:
  Invalid range given in the sensitivity file.

  Response message string:
  "The index range %d-%d in line %d is invalid."

- err_sen_index_range

  Corresponding constant:
  MSK_RES_ERR_SEN_INDEX_RANGE

  Description:
  Index out of range in the sensitivity analysis file.

  Response message string:
  "Index '%d' out of range at line %d."

- err_sen_invalid_regexp

  Corresponding constant:
  MSK_RES_ERR_SEN_INVALID_REGEXP

  Description:
  Syntax error in regexp or regexp longer than 1024.

  Response message string:
  "Syntax error in regexp on line %d: %s."

- err_sen_numerical

  Corresponding constant:
  MSK_RES_ERR_SEN_NUMERICAL

  Description:
  Numerical difficulties encountered performing the sensitivity analysis.

  Response message string:
  "Numerical difficulties encountered performing the sensitivity analysis."

- err_sen_solution_status

  Corresponding constant:
  MSK_RES_ERR_SEN_SOLUTION_STATUS

  Description:
  No optimal solution found to the original problem given for sensitivity analysis.

Response message string:
"No optimal solution found to the original problem given for sensitivity analysis. Solution status = %d."

- err_sen_undef_name

Corresponding constant:
MSK_RES_ERR_SEN_UNDEF_NAME

Description:
An undefined name was encountered in the sensitivity analysis file.

Response message string:
"Name '%s' on line %d not defined."

- err_size_license

Corresponding constant:
MSK_RES_ERR_SIZE_LICENSE

Description:
The problem is bigger than the license.

Response message string:
"The problem is bigger than the license."

- err_size_license_con

Corresponding constant:
MSK_RES_ERR_SIZE_LICENSE_CON

Description:
The problem has too many constraints to be solved with the available license.

Response message string:
"The problem has %d constraint(s) but the license allows only %d constraint(s) for feature '%s'."

- err_size_license_intvar

Corresponding constant:
MSK_RES_ERR_SIZE_LICENSE_INTVAR

Description:
The problem contains too many integer variables to be solved with the available license.

Response message string:
"The problem contains %d integer variable(s) but the license allows only %d integer variable(s) for feature '%s'."

- err_size_license_var

Corresponding constant:
MSK_RES_ERR_SIZE_LICENSE_VAR

Description:
The problem has too many variables to be solved with the available license.

Response message string:
"The problem has %d variable(s) but the license allows only %d variable(s) for feature '%s'."

- err_sol_file_number

  Corresponding constant:
  MSK_RES_ERR_SOL_FILE_NUMBER

  Description:
  An invalid number is specified in a solution file.

  Response message string:
  "An invalid number '%s' is specified in a solution file."

- err_solitem

  Corresponding constant:
  MSK_RES_ERR_SOLITEM

  Description:
  The solution item number solitem is invalid. Please note MSK_SOL_ITEM_SNX is invalid for the basic solution.

  Response message string:
  "%d is not a valid solution item code for solution %d."

- err_solver_probtype

  Corresponding constant:
  MSK_RES_ERR_SOLVER_PROBTYPE

  Description:
  Problem type does not match the chosen optimizer.

  Response message string:
  "Problem type does not match the chosen optimizer."

- err_space

  Corresponding constant:
  MSK_RES_ERR_SPACE

  Description:
  Out of space.

  Response message string:
  "Out of space."

- err_space_leaking

  Corresponding constant:
  MSK_RES_ERR_SPACE_LEAKING

Description:
MOSEK is leaking memory. This can be due to either an incorrect use of MOSEK or a bug.

Response message string:
"MOSEK is leaking memory."

- err_space_no_info

Corresponding constant:
MSK_RES_ERR_SPACE_NO_INFO

Description:
No available information about the space usage.

Response message string:
"No available information about the space usage."

- err_thread_cond_init

Corresponding constant:
MSK_RES_ERR_THREAD_COND_INIT

Description:
Could not initialize a condition.

Response message string:
"Could not initialize a condition."

- err_thread_create

Corresponding constant:
MSK_RES_ERR_THREAD_CREATE

Description:
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.

Response message string:
"Could not create a thread. System error code: %d"

- err_thread_mutex_init

Corresponding constant:
MSK_RES_ERR_THREAD_MUTEX_INIT

Description:
Could not initialize a mutex.

Response message string:
"Could not initialize a mutex."

- err_thread_mutex_lock

Corresponding constant:
MSK_RES_ERR_THREAD_MUTEX_LOCK

Description:
Could not lock a mutex.

Response message string:
"`Could not lock a mutex.`"

- `err_thread_mutex_unlock`

  Corresponding constant:
  `MSK_RES_ERR_THREAD_MUTEX_UNLOCK`

  Description:
  Could not unlock a mutex.

  Response message string:
  "`Could not unlock a mutex.`"

- `err_too_small_maxnumanz`

  Corresponding constant:
  `MSK_RES_ERR_TOO_SMALL_MAXNUMANZ`

  Description:
  Maximum number of non-zeros allowed in A is too small.

  Response message string:
  "`Maximum number of non-zeros allowed in A is too small. %d is required.`"

- `err_unb_step_size`

  Corresponding constant:
  `MSK_RES_ERR_UNB_STEP_SIZE`

  Description:
  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.

  Response message string:
  "`A step-size in an optimizer was unexpectedly unbounded.`"

- `err_undef_solution`

  Corresponding constant:
  `MSK_RES_ERR_UNDEF_SOLUTION`

  Description:
  The required solution is not defined.

  Response message string:
  "`The solution with code %d is not defined.`"

- `err_undefined_objective_sense`

  Corresponding constant:
  `MSK_RES_ERR_UNDEFINED_OBJECTIVE_SENSE`

Description:
The objective sense has not been specified before the optimization.

Response message string:
"The objective sense has not been specified before the optimization."

- err_unknown

  Corresponding constant:
  MSK_RES_ERR_UNKNOWN

  Description:
  Unknown error.

  Response message string:
  "Unknown error."

- err_user_func_ret

  Corresponding constant:
  MSK_RES_ERR_USER_FUNC_RET

  Description:
  An user function reported an error.

  Response message string:
  "An user function returned a non-zero error code %d."

- err_user_func_ret_data

  Corresponding constant:
  MSK_RES_ERR_USER_FUNC_RET_DATA

  Description:
  An user function returned invalid data.

  Response message string:
  "An user function returned invalid data for '%s'."

- err_user_nlo_eval

  Corresponding constant:
  MSK_RES_ERR_USER_NLO_EVAL

  Description:
  The user-defined nonlinear function reported an error.

  Response message string:
  "The user-defined nonlinear function reported the error '%s'."

- err_user_nlo_eval_hessubi

  Corresponding constant:
  MSK_RES_ERR_USER_NLO_EVAL_HESSUBI

  Description:
  The user-defined nonlinear function reported an invalid subscript in the Hessian.

Response message string:
"The user-defined nonlinear function reported the invalid hessubi[%d]: %d.'"

- err_user_nlo_eval_hessubj

  Corresponding constant:
  MSK_RES_ERR_USER_NLO_EVAL_HESSUBJ

  Description:
  The user-defined nonlinear function reported an invalid subscript in the Hessian.

  Response message string:
  "The user-defined nonlinear function reported the invalid subscript hessubj[%d]: %d.'"

- err_user_nlo_func

  Corresponding constant:
  MSK_RES_ERR_USER_NLO_FUNC

  Description:
  The user-defined nonlinear function reported an error.

  Response message string:
  "The user-defined nonlinear function reported an error."

- err_whichitem_not_allowed

  Corresponding constant:
  MSK_RES_ERR_WHICHITEM_NOT_ALLOWED

  Description:
  whichitem is unacceptable.

  Response message string:
  "%d is an unacceptable whichitem."

- err_whichsol

  Corresponding constant:
  MSK_RES_ERR_WHICHSOL

  Description:
  The solution defined by compwhichsol does not exists.

  Response message string:
  "%d is not a valid solution code."

- err_write_lp_format

  Corresponding constant:
  MSK_RES_ERR_WRITE_LP_FORMAT

  Description:
  Problem cannot be written as an LP file.

Response message string:
“Problem cannot be written as an LP file because of: %s.”

- err_write_lp_non_unique_name

Corresponding constant:
MSK_RES_ERR_WRITE_LP_NON_UNIQUE_NAME

Description:
An auto-generated name is not unique.

Response message string:
“The auto-generated name ’%s’ is not unique.”

- err_write_mps_invalid_name

Corresponding constant:
MSK_RES_ERR_WRITE_MPS_INVALID_NAME

Description:
An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.

Response message string:
“The name ’%s’ is not a valid MPS name.”

- err_write_opf_invalid_var_name

Corresponding constant:
MSK_RES_ERR_WRITE_OPF_INVALID_VAR_NAME

Description:
Empty variable names cannot be written to OPF files.

Response message string:
“Name of variable index %d is empty and cannot be written to an OPF file.”

- err_xml_invalid_problem_type

Corresponding constant:
MSK_RES_ERR_XML_INVALID_PROBLEM_TYPE

Description:
The problem type is not supported by the XML format.

Response message string:
“The problem type %s is not supported by the XML format.”

- err_y_is_undefined

Corresponding constant:
MSK_RES_ERR_Y_IS_UNDEFINED

Description:
The solution item y is undefined.

Response message string:
"The solution term y is undefined."

- ok

Corresponding constant:
MSK_RES_OK

Description:
No error occurred.

Response message string:
"No error occurred."

- trm_internal

Corresponding constant:
MSK_RES_TRM_INTERNAL

Description:
The optimizer terminated due to some internal reason. Please contact MOSEK support.

Response message string:
"The optimizer terminated due to some internal reason."

- trm_internal_stop

Corresponding constant:
MSK_RES_TRM_INTERNAL_STOP

Description:
The optimizer terminated for internal reasons. Please contact MOSEK support.

Response message string:
"The optimizer terminated for internal reasons."

- trm_max_iterations

Corresponding constant:
MSK_RES_TRM_MAX_ITERATIONS

Description:
The optimizer terminated at the maximum number of iterations.

Response message string:
"Maximum number of iterations is exceeded."

- trm_max_num_setbacks

Corresponding constant:
MSK_RES_TRM_MAX_NUM_SETBACKS

Description:
The optimizer terminated as the maximum number of set-backs was reached. This indicates numerical problems and a possibly badly formulated problem.

Response message string:
"The optimizer terminated as the maximum number of set-backs was reached."

- trm_max_time

  Corresponding constant:
  MSK_RES_TRM_MAX_TIME

  Description:
  The optimizer terminated at the maximum amount of time.

  Response message string:
  "Maximum amount of time exceeded."

- trm_mio_near_abs_gap

  Corresponding constant:
  MSK_RES_TRM_MIO_NEAR_ABS_GAP

  Description:
  The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied.

  Response message string:
  "The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied."

- trm_mio_near_rel_gap

  Corresponding constant:
  MSK_RES_TRM_MIO_NEAR_REL_GAP

  Description:
  The mixed-integer optimizer terminated because the near optimal relative gap tolerance was satisfied.

  Response message string:
  "The mixed-integer optimizer terminated because the near optimal relative gap tolerance was satisfied."

- trm_mio_num_branches

  Corresponding constant:
  MSK_RES_TRM_MIO_NUM_BRANCHES

  Description:
  The mixed-integer optimizer terminated as to the maximum number of branches was reached.

  Response message string:
  "The mixed-integer optimizer terminated as to the maximum number of branches was reached."

- trm_mio_num_relaxs

  Corresponding constant:
  MSK_RES_TRM_MIO_NUM_RELAXS

Description:
The mixed-integer optimizer terminated as the maximum number of relaxations was reached.

Response message string:
"The mixed-integer optimizer terminated as the maximum number of relaxations was reached."

- trm_num_max_num_int_solutions

Corresponding constant:
MSK_RES_TRM_NUM_MAX_NUM_INT_SOLUTIONS

Description:
The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.

Response message string:
"The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached."

- trm_numerical_problem

Corresponding constant:
MSK_RES_TRM_NUMERICAL_PROBLEM

Description:
The optimizer terminated due to numerical problems.

Response message string:
"The optimizer terminated due to a numerical problem."

- trm_objective_range

Corresponding constant:
MSK_RES_TRM_OBJECTIVE_RANGE

Description:
The optimizer terminated on the bound of the objective range.

Response message string:
"The optimal solution has an objective value outside the objective range."

- trm_stall

Corresponding constant:
MSK_RES_TRM_STALL

Description:
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, in case of nonlinear problems, the nonlinear call-back functions are incorrect.

Please contact MOSEK support if this happens.

Response message string:
"The optimizer terminated due to slow progress."

- trm_user_break

  Corresponding constant:
  MSK_RES_TRM_USER_BREAK

  Description:
  The optimizer terminated due to a user break.

  Response message string:
  "Control break was pressed."

- trm_user_callback

  Corresponding constant:
  MSK_RES_TRM_USER_CALLBACK

  Description:
  The optimizer terminated due to the return of the user-defined call-back function.

  Response message string:
  "The user-defined progress call-back function terminated the optimization."

- wrn_dropped_nz_qobj

  Corresponding constant:
  MSK_RES_WRN_DROPPED_NZ_QOBJ

  Description:
  One or more non-zero elements were dropped in the Q matrix in the objective.

  Response message string:
  "'%d' non-zero element(s) are dropped in the Q matrix in the objective."

- wrn_eliminator_space

  Corresponding constant:
  MSK_RES_WRN_ELIMINATOR_SPACE

  Description:
  The eliminator is skipped at least once due to lack of space.

  Response message string:
  "The eliminator is skipped at least once due to lack of space."

- wrn_empty_name

  Corresponding constant:
  MSK_RES_WRN_EMPTY_NAME

  Description:
  A variable or constraint name is empty. The output file may be invalid.

  Response message string:
  "A variable or constraint name is empty. The output file may be invalid."

- wrn_fixed_bound_values

Corresponding constant:
MSK_RES_WRN_FIXED_BOUND_VALUES

Description:
A fixed constraint/variable has been specified using the bound keys but the numerical bounds are different. The variable is fixed at the lower bound.

Response message string:
"For the bound key MSK_BK_FX the specified lower %24.16e and upper bound %24.16e are different."

- wrn_ignore_integer

Corresponding constant:
MSK_RES_WRN_IGNORE_INTEGER

Description:
Ignored integer constraints.

Response message string:
"Ignored integer constraints."

- wrn_large_aij

Corresponding constant:
MSK_RES_WRN_LARGE_AIJ

Description:
A numerically large value is specified for one $a_{i,j}$.

Response message string:
"A large value of %8.1e has been specified in A for variable '%s' (%d) in constraint '%s' (%d)."

- wrn_large_bound

Corresponding constant:
MSK_RES_WRN_LARGE_BOUND

Description:
A very large bound in absolute value has been specified.

Response message string:
"A large bound of value %8.1e has been specified for %s '%s' (%d)."

- wrn_large_cj

Corresponding constant:
MSK_RES_WRN_LARGE_CJ

Description:
A numerically large value is specified for one c_j.

Response message string:
"A large value of %8.1e has been specified in cx for variable '%s' (%d)."

- wrn_large_lo_bound

  Corresponding constant:
  MSK_RES_WRN_LARGE_LO_BOUND

  Description:
  A large but finite lower bound in absolute value has been specified.

  Response message string:
  "A large lower bound of value %8.1e has been specified for %s '%s' (%d)."

- wrn_large_up_bound

  Corresponding constant:
  MSK_RES_WRN_LARGE_UP_BOUND

  Description:
  A large but finite upper bound in absolute value has been specified.

  Response message string:
  "A large upper bound of value %8.1e has been specified for %s '%s' (%d)."

- wrn_license_expire

  Corresponding constant:
  MSK_RES_WRN_LICENSE_EXPIRE

  Description:
  The license expires.

  Response message string:
  "The license expires in %ld days."

- wrn_license_feature_expire

  Corresponding constant:
  MSK_RES_WRN_LICENSE_FEATURE_EXPIRE

  Description:
  The license expires.

  Response message string:
  "The license feature '%s' expires in %ld days."

- wrn_license_server

  Corresponding constant:
  MSK_RES_WRN_LICENSE_SERVER

  Description:
  The license server is not responding.

  Response message string:
  "The license server is not responding."

- wrn_lp_drop_variable

Corresponding constant:
MSK_RES_WRN_LP_DROP_VARIABLE

Description:
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.

Response message string:
"`The variable '%s' is ignored because the variable was not previously defined.`"

- `wrn_lp_old_quad_format`

Corresponding constant:
MSK_RES_WRN_LP_OLD_QUAD_FORMAT

Description:
Missing '/2' after quadratic expressions in bound or objective.

Response message string:
"`Missing '/2' after quadratic expressions in bound or objective.`"

- `wrn_mio_infeasible_final`

Corresponding constant:
MSK_RES_WRN_MIO_INFEASIBLE_FINAL

Description:
The final mixed integer problem with all the integer variables fixed at their optimal values is infeasible.

Response message string:
"`The '%s' solution reports that final problem with all the integer variables fixed is infeasible while an integer solution has been found.`"

- `wrn_mps_split_bou_vector`

Corresponding constant:
MSK_RES_WRN_MPS_SPLIT_BOU_VECTOR

Description:
A BOUNDS vector is split into several nonadjacent parts in an MPS file.

Response message string:
"`The BOUNDS vector '%s' is split into several nonadjacent parts.`"

- `wrn_mps_split_ran_vector`

Corresponding constant:
MSK_RES_WRN_MPS_SPLIT_RAN_VECTOR

Description:
A RANGE vector is split into several nonadjacent parts in an MPS file.

Response message string:
"`The RANGE vector '%s' is split into several nonadjacent parts.`"

- wrn_mps_split_rhs_vector

  Corresponding constant:
  MSK_RES_WRN_MPS_SPLIT_RHS_VECTOR

  Description:
  An RHS vector is split into several nonadjacent parts in an MPS file.

  Response message string:
  "The RHS vector '%s' is split into several nonadjacent parts."

- wrn_name_max_len

  Corresponding constant:
  MSK_RES_WRN_NAME_MAX_LEN

  Description:
  A name is longer than the buffer that is supposed to hold it.

  Response message string:
  "A name of length %d is longer than the buffer of length %d that is supposed to hold it."

- wrn_no_global_optimizer

  Corresponding constant:
  MSK_RES_WRN_NO_GLOBAL_OPTIMIZER

  Description:
  No global optimizer is available.

  Response message string:
  "No global optimizer is available (%s)."

- wrn_noncomplete_linear_dependency_check

  Corresponding constant:
  MSK_RES_WRN_NONCOMPLETE_LINEAR_DEPENDENCY_CHECK

  Description:
  The linear dependency check(s) was not completed and therefore the A matrix may contain linear dependencies.

  Response message string:
  "The linear dependency check(s) is incomplete."

- wrn_nz_in_upr_tri

  Corresponding constant:
  MSK_RES_WRN_NZ_IN_UPR_TRI

  Description:
  Non-zero elements specified in the upper triangle of a matrix were ignored.

  Response message string:
  "Non-zero elements in the upper triangle of variable '%s' are ignored."

- wrn_open_param_file

  Corresponding constant:
  MSK_RES_WRN_OPEN_PARAM_FILE

  Description:
  The parameter file could not be opened.

  Response message string:
  "Could not open the parameter file '%s'."

- wrn_presolve_bad_precision

  Corresponding constant:
  MSK_RES_WRN_PRESOLVE_BAD_PRECISION

  Description:
  The presolve estimates that the model is specified with insufficient precision.

  Response message string:
  "The presolve estimates that the model is specified with insufficient precision."

- wrn_presolve_outofspace

  Corresponding constant:
  MSK_RES_WRN_PRESOLVE_OUTOFSPACE

  Description:
  The presolve is incomplete due to lack of space.

  Response message string:
  "The presolve is incomplete due to lack of space."

- wrn_sol_filter

  Corresponding constant:
  MSK_RES_WRN_SOL_FILTER

  Description:
  Invalid solution filter is specified.

  Response message string:
  "'%s' is an invalid solution filter is specified."

- wrn_spar_max_len

  Corresponding constant:
  MSK_RES_WRN_SPAR_MAX_LEN

  Description:
  A value for a string parameter is longer than the buffer that is supposed to hold it.

  Response message string:
  "A value for a string parameter is longer than the buffer that is supposed to hold it."

- wrn_too_few_basis_vars

Corresponding constant:
MSK_RES_WRN_TOO_FEW_BASIS_VARS

Description:
An incomplete basis has been specified. Too few basis variables are specified.

Response message string:
"%d number of basis variables are specified but %d are expected."

- wrn_too_many_basis_vars

Corresponding constant:
MSK_RES_WRN_TOO_MANY_BASIS_VARS

Description:
A basis with too many variables has been specified.

Response message string:
"%d basis variables are specified but %d are expected."

- wrn_undef_sol_file_name

Corresponding constant:
MSK_RES_WRN_UNDEF_SOL_FILE_NAME

Description:
Undefined name occurred in a solution.

Response message string:
"'%s' is an undefined %s name."

- wrn_using_generic_names

Corresponding constant:
MSK_RES_WRN_USING_GENERIC_NAMES

Description:
The file writer reverts to generic names because a name is blank.

Response message string:
"The file writer reverts to generic names because a name is blank."

- wrn_write_discarded_cfix

Corresponding constant:
MSK_RES_WRN_WRITE_DISCARDED_CFIX

Description:
The fixed objective term could not be converted to a variable and was discarded in the output file.

Response message string:
"The fixed objective term was discarded in the output file."

- wrn_zero_aij

Corresponding constant:
MSK_RES_WRN_ZERO_AIJ

Description:
One or more zero elements are specified in A.

Response message string:
"%d zero element(s) in A are specified."

- wrn_zeros_in_sparse_data

Corresponding constant:
MSK_RES_WRN_ZEROS_IN_SPARSE_DATA

Description:
One or more almost zero elements are specified in sparse input data.

Response message string:
"%d zero elements are specified in sparse input data."

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 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.

18.3 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.

18.4 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 optimizer decides which branch to choose.

(1) MSK_BRANCH_DIR_UP
The mixed integer optimizer always chooses the up branch first.

18.5 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_INFEAS_ANA
The call-back function is called when the infeasibility analyzer is started.

(8) MSK_CALLBACK_BEGIN_INTPNT
The call-back function is called when the interior-point optimizer is started.

(9) MSK_CALLBACK_BEGIN_LICENSE_WAIT
Begin waiting for license.

(10) MSK_CALLBACK_BEGIN_MIO
The call-back function is called when the mixed integer optimizer is started.

(11) MSK_CALLBACK_BEGIN_NETWORK_DUAL_SIMPLEX
The call-back function is called when the dual network simplex optimizer is started.

(12) `MSK_CALLBACK_BEGIN_NETWORK_PRIMAL_SIMPLEX`
The call-back function is called when the primal network simplex optimizer is started.

(13) `MSK_CALLBACK_BEGIN_NETWORK_SIMPLEX`
The call-back function is called when the simplex network optimizer is started.

(14) `MSK_CALLBACK_BEGIN_NONCONVEX`
The call-back function is called when the nonconvex optimizer is started.

(15) `MSK_CALLBACK_BEGIN_PRESOLVE`
The call-back function is called when the presolve is started.

(16) `MSK_CALLBACK_BEGIN_PRIMAL_BI`
The call-back function is called from within the basis identification procedure when the primal phase is started.

(17) `MSK_CALLBACK_BEGIN_PRIMAL_SENSITIVITY`
Primal sensitivity analysis is started.

(18) `MSK_CALLBACK_BEGIN_PRIMAL_SETUP_BI`
The call-back function is called when the primal BI setup is started.

(19) `MSK_CALLBACK_BEGIN_PRIMAL_SIMPLEX`
The call-back function is called when the primal simplex optimizer is started.

(20) `MSK_CALLBACK_BEGIN_SIMPLEX`
The call-back function is called when the simplex optimizer is started.

(21) `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.

(22) `MSK_CALLBACK_BEGIN_SIMPLEX_NETWORK_DETECT`
The call-back function is called when the network detection procedure is started.

(23) `MSK_CALLBACK_CONIC`
The call-back function is called from within the conic optimizer after the information database has been updated.

(24) `MSK_CALLBACK_DUAL_SIMPLEX`
The call-back function is called from within the dual simplex optimizer.

(25) `MSK_CALLBACK_END_BI`
The call-back function is called when the basis identification procedure is terminated.

(26) `MSK_CALLBACK_END_CONCURRENT`
Concurrent optimizer is terminated.

(27) `MSK_CALLBACK_END_CONIC`
The call-back function is called when the conic optimizer is terminated.

(28) MSK_CALLBACK_END_DUAL_BI
The call-back function is called from within the basis identification procedure when the dual phase is terminated.

(29) MSK_CALLBACK_END_DUAL_SENSITIVITY
Dual sensitivity analysis is terminated.

(30) MSK_CALLBACK_END_DUAL_SETUP_BI
The call-back function is called when the dual BI phase is terminated.

(31) MSK_CALLBACK_END_DUAL_SIMPLEX
The call-back function is called when the dual simplex optimizer is terminated.

(32) MSK_CALLBACK_END_INFEAS_ANA
The call-back function is called when the infeasibility analyzer is terminated.

(33) MSK_CALLBACK_END_INTPNT
The call-back function is called when the interior-point optimizer is terminated.

(34) MSK_CALLBACK_END_LICENSE_WAIT
End waiting for license.

(35) MSK_CALLBACK_END_MIO
The call-back function is called when the mixed integer optimizer is terminated.

(36) MSK_CALLBACK_END_NETWORK_DUAL_SIMPLEX
The call-back function is called when the dual network simplex optimizer is terminated.

(37) MSK_CALLBACK_END_NETWORK_PRIMAL_SIMPLEX
The call-back function is called when the primal network simplex optimizer is terminated.

(38) MSK_CALLBACK_END_NETWORK_SIMPLEX
The call-back function is called when the simplex network optimizer is terminated.

(39) MSK_CALLBACK_END_NONCONVEX
The call-back function is called when the nonconvex optimizer is terminated.

(40) MSK_CALLBACK_END_PRESOLVE
The call-back function is called when the presolve is completed.

(41) MSK_CALLBACK_END_PRIMAL_BI
The call-back function is called from within the basis identification procedure when the primal phase is terminated.

(42) MSK_CALLBACK_END_PRIMAL_SENSITIVITY
Primal sensitivity analysis is terminated.

(43) MSK_CALLBACK_END_PRIMAL_SETUP_BI
The call-back function is called when the primal BI setup is terminated.

(44) MSK_CALLBACK_END_PRIMAL_SIMPLEX
The call-back function is called when the primal simplex optimizer is terminated.

(45) MSK_CALLBACK_END_SIMPLEX
The call-back function is called when the simplex optimizer is terminated.

(46) 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.

(47) MSK_CALLBACK_END_SIMPLEX_NETWORK_DETECT
The call-back function is called when the network detection procedure is terminated.

(48) MSK_CALLBACK_IGNORE_VALUE
This code means that the call-back does not indicate a new phase in the optimization, but is simply a time-triggered call-back.

(49) MSK_CALLBACK_IM_BI
The call-back function is called from within the basis identification procedure at an intermediate point.

(50) 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.

(51) 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.

(52) MSK_CALLBACK_IM_DUAL_SENSIVITY
The call-back function is called at an intermediate stage of the dual sensitivity analysis.

(53) MSK_CALLBACK_IM_DUAL_SIMPLEX
The call-back function is called at an intermediate point in the dual simplex optimizer.

(54) 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.

(55) MSK_CALLBACK_IM_LICENSE_WAIT
MOSEK is waiting for a license.

(56) MSK_CALLBACK_IM_MIO
The call-back function is called at an intermediate point in the mixed integer optimizer.

(57) 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.

(58) 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.

(59) MSK_CALLBACK_IM_MIO_PRESOLVE
The call-back function is called at an intermediate point in the mixed integer optimizer while running the presolve.

(60) 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.

(61) MSK_CALLBACK_IM_NETWORK_DUAL_SIMPLEX
The call-back function is called at an intermediate point in the dual network simplex optimizer.

(62) MSK_CALLBACK_IM_NETWORK_PRIMAL_SIMPLEX
The call-back function is called at an intermediate point in the primal network simplex optimizer.

(63) 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.

(64) MSK_CALLBACK_IM_PRESOLVE
The call-back function is called from within the presolve procedure at an intermediate stage.

(65) 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.

(66) MSK_CALLBACK_IM_PRIMAL_SENSIVITY
The call-back function is called at an intermediate stage of the primal sensitivity analysis.

(67) MSK_CALLBACK_IM_PRIMAL_SIMPLEX
The call-back function is called at an intermediate point in the primal simplex optimizer.

(68) 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.

(69) MSK_CALLBACK_INTPNT
The call-back function is called from within the interior-point optimizer after the information database has been updated.

(70) MSK_CALLBACK_NEW_INT_MIO
The call-back function is called after a new integer solution has been located by the mixed integer optimizer.

(71) MSK_CALLBACK_NONCOVEX
The call-back function is called from within the nonconvex optimizer after the information database has been updated.

(72) MSK_CALLBACK_PRIMAL_SIMPLEX
The call-back function is called from within the primal simplex optimizer.

(73) MSK_CALLBACK_QCONE
The call-back function is called from within the Qcone optimizer.

(74) 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.

(75) MSK_CALLBACK_UPDATE_DUAL_SIMPLEX
The call-back function is called in the dual simplex optimizer.

(76) MSK_CALLBACK_UPDATE_NETWORK_DUAL_SIMPLEX
The call-back function is called in the dual network simplex optimizer.

(77) MSK_CALLBACK_UPDATE_NETWORK_PRIMAL_SIMPLEX
The call-back function is called in the primal network simplex optimizer.

(78) 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.

(79) MSK_CALLBACK_UPDATE_PRESOLVE
The call-back function is called from within the presolve procedure.

(80) 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.

(81) MSK_CALLBACK_UPDATE_PRIMAL_SIMPLEX
The call-back function is called in the primal simplex optimizer.

(82) MSK_CALLBACK_UPDATE_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.

18.6 Types of convexity checks.

(0) MSK_CHECK_CONVEXITY_NONE
No convexity check.

(1) MSK_CHECK_CONVEXITY_SIMPLE
Perform simple and fast convexity check.

18.7 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.

18.8 Cone types

(0) MSK_CT_QUAD
The cone is a quadratic cone.

(1) MSK_CT_RQUAD
The cone is a rotated quadratic cone.

18.9 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.

18.10 Data format types

(0) MSK_DATA_FORMAT_EXTENSION
The file extension is used to determine the data file format.

(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.

18.11 Double information items

(0) MSK_DINF_BI_CLEAN_CPUTIME
Time (in CPU seconds) spent within the clean-up phase of the basis identification procedure since its invocation.

(1) MSK_DINF_BI_CPUTIME
Time (in CPU seconds) spent within the basis identification procedure since its invocation.

(2) MSK_DINF_BI_DUAL_CPUTIME
Time (in CPU seconds) spent within the dual phase basis identification procedure since its invocation.

(3) MSK_DINF_BI_PRIMAL_CPUTIME
Time (in CPU seconds) spent within the primal phase of the basis identification procedure since its invocation.

(4) MSK_DINF_CONCURRENT_CPUTIME
Time (in CPU seconds) spent within the concurrent optimizer since its invocation.

(5) MSK_DINF_CONCURRENT_REALTIME
Time (in wall-clock seconds) within the concurrent optimizer since its invocation.

(6) MSK_DINF_INTPNT_CPUTIME
Time (in CPU seconds) spent within the interior-point optimizer since its invocation.

(7) 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.)

(8) MSK_DINF_INTPNT_DUAL_OBJ
Dual objective value reported by the interior-point or Qcone optimizer.

(9) MSK_DINF_INTPNT_FACTOR_NUM_FLOPS
An estimate of the number of flops used in the factorization.

(10) 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.

(11) MSK_DINF_INTPNT_ORDER_CPUTIME
Order time (in CPU seconds).

(12) 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).

(13) MSK_DINF_INTPNT_PRIMAL_OBJ
Primal objective value reported by the interior-point or Qcone optimizer.

(14) MSK_DINF_INTPNT_REALTIME
Time (in wall-clock seconds) spent within the interior-point optimizer since its invocation.

(15) 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.

(16) MSK_DINF_MIO_CPUTIME
Time spent in the mixed integer optimizer.

(17) 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.

(18) MSK_DINF_MIO_OBJ_BOUND
The best bound objective value corresponding to the best integer feasible solution is located. Please note that at least one integer feasible solution must be located i.e. check MSK_IINF_MIO_NUM_INT_SOLUTIONS.

(19) 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.

(20) 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

$$\frac{|(\text{objective value of feasible solution}) - (\text{objective bound})|}{\max(1, |(\text{objective value of feasible solution})|)}.$$

Otherwise it has the value -1.0.

(21) MSK_DINF_MIO_USER_OBJ_CUT
If the objective cut is used, then this information item has the value of the cut.

(22) MSK_DINF_OPTIMIZER_CPUTIME
Total time (in CPU seconds) spent in the optimizer since it was invoked.

(23) MSK_DINF_OPTIMIZER_REALTIME
Total time (in wall-clock seconds) spent in the optimizer since it was invoked.

(24) MSK_DINF_PRESOLVE_CPUTIME
Total time (in CPU seconds) spent in the presolve since it was invoked.

(25) MSK_DINF_PRESOLVE_ELI_CPUTIME
Total time (in CPU seconds) spent in the eliminator since the presolve was invoked.

(26) MSK_DINF_PRESOLVE_LINDEP_CPUTIME
Total time (in CPU seconds) spent in the linear dependency checker since the presolve was invoked.

(27) MSK_DINF_RD_CPUTIME
Time (in CPU seconds) spent reading the data file.

(28) MSK_DINF_SIM_CPUTIME
Time (in CPU seconds) spent in the simplex optimizer since invoking it.

(29) MSK_DINF_SIM_FEAS
Feasibility measure reported by the simplex optimizer.

(30) MSK_DINF_SIM_OBJ
Objective value reported by the simplex optimizer.

(31) MSK_DINF_SOL_BAS_DUAL_OBJ
Dual objective value of the basic solution. Updated at the end of the optimization.

(32) MSK_DINF_SOL_BAS_MAX_DBI
Maximal dual bound infeasibility in the basic solution. Updated at the end of the optimization.

(33) MSK_DINF_SOL_BAS_MAX_DEQI
Maximal dual equality infeasibility in the basic solution. Updated at the end of the optimization.

(34) MSK_DINF_SOL_BAS_MAX_PBI
Maximal primal bound infeasibility in the basic solution. Updated at the end of the optimization.

(35) MSK_DINF_SOL_BAS_MAX_PEQI
Maximal primal equality infeasibility in the basic solution. Updated at the end of the optimization.

(36) MSK_DINF_SOL_BAS_MAX_PINTI
Maximal primal integer infeasibility in the basic solution. Updated at the end of the optimization.

(37) MSK_DINF_SOL_BAS_PRIMAL_OBJ
Primal objective value of the basic solution. Updated at the end of the optimization.

(38) MSK_DINF_SOL_INT_MAX_PBI
Maximal primal bound infeasibility in the integer solution. Updated at the end of the optimization.

(39) MSK_DINF_SOL_INT_MAX_PEQI
Maximal primal equality infeasibility in the basic solution. Updated at the end of the optimization.

(40) MSK_DINF_SOL_INT_MAX_PINTI
Maximal primal integer infeasibility in the integer solution. Updated at the end of the optimization.

(41) MSK_DINF_SOL_INT_PRIMAL_OBJ
Primal objective value of the integer solution. Updated at the end of the optimization.

(42) MSK_DINF_SOL_ITR_DUAL_OBJ
Dual objective value of the interior-point solution. Updated at the end of the optimization.

(43) MSK_DINF_SOL_ITR_MAX_DBI
Maximal dual bound infeasibility in the interior-point solution. Updated at the end of the optimization.

(44) MSK_DINF_SOL_ITR_MAX_DCNI
Maximal dual cone infeasibility in the interior-point solution. Updated at the end of the optimization.

(45) MSK_DINF_SOL_ITR_MAX_DEQI
Maximal dual equality infeasibility in the interior-point solution. Updated at the end of the optimization.

(46) MSK_DINF_SOL_ITR_MAX_PBI
Maximal primal bound infeasibility in the interior-point solution. Updated at the end of the optimization.

(47) MSK_DINF_SOL_ITR_MAX_PCNI
Maximal primal cone infeasibility in the interior-point solution. Updated at the end of the optimization.

(48) MSK_DINF_SOL_ITR_MAX_PEQI
Maximal primal equality infeasibility in the interior-point solution. Updated at the end of the optimization.

(49) MSK_DINF_SOL_ITR_MAX_PINTI
Maximal primal integer infeasibility in the interior-point solution. Updated at the end of the optimization.

(50) MSK_DINF_SOL_ITR_PRIMAL_OBJ
Primal objective value of the interior-point solution. Updated at the end of the optimization.

18.12 Double values

(1e+30) MSK_INFINITY
Definition of infinity.

18.13 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.

18.14 Integer information items.

(0) MSK_IINF_BI_ITER
Number of simplex pivots performed since invoking the basis identification procedure.

(1) MSK_IINF_CACHE_SIZE_L1
L1 cache size used.

(2) MSK_IINF_CACHE_SIZE_L2
L2 cache size used.

(3) MSK_IINF_CONCURRENT_FASTEST_OPTIMIZER
The type of the optimizer that finished first in a concurrent optimization.

(4) MSK_IINF_CPU_TYPE
The type of cpu detected.

(5) MSK_IINF_INTPNT_FACTOR_NUM_NZ
Number of non-zeros in factorization.

(6) MSK_IINF_INTPNT_FACTOR_NUM_OFFCOL
Number of columns in the constraint matrix (or Jacobian) that has an offending structure.

(7) MSK_IINF_INTPNT_ITER
Number of interior-point iterations since invoking the interior-point optimizer.

(8) MSK_IINF_INTPNT_NUM_THREADS
Number of threads that the interior-point optimizer is using.

(9) MSK_IINF_INTPNT_SOLVE_DUAL
Non-zero if the interior-point optimizer is solving the dual problem.

(10) 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.

(11) MSK_IINF_MIO_INITIAL_SOLUTION
Is non-zero if an initial integer solution is specified.

(12) MSK_IINF_MIO_NUM_ACTIVE_NODES
Number of active nodes in the branch and bound tree.

(13) MSK_IINF_MIO_NUM_BRANCH
Number of branches performed during the optimization.

(14) MSK_IINF_MIO_NUM_CUTS
Number of cuts generated by the mixed integer optimizer.

(15) MSK_IINF_MIO_NUM_INT_SOLUTIONS
Number of integer feasible solutions that has been found.

(16) MSK_IINF_MIO_NUM_INTPNT_ITER
Number of interior-point iterations performed by the mixed-integer optimizer.

(17) MSK_IINF_MIO_NUM_RELAX
Number of relaxations solved during the optimization.

(18) MSK_IINF_MIO_NUM_SIMPLEX_ITER
Number of simplex iterations performed by the mixed-integer optimizer.

(19) MSK_IINF_MIO_NUMCON
Number of constraints in the problem solved be the mixed integer optimizer.

(20) MSK_IINF_MIO_NUMINT
Number of integer variables in the problem solved be the mixed integer optimizer.

(21) MSK_IINF_MIO_NUMVAR
Number of variables in the problem solved be the mixed integer optimizer.

(22) MSK_IINF_MIO_TOTAL_NUM_BASIS_CUTS
Number of basis cuts.

(23) MSK_IINF_MIO_TOTAL_NUM_BRANCH
Number of branches performed during the optimization.

(24) `MSK_IINF_MIO_TOTAL_NUM_CARDGUB_CUTS`
Number of cardgub cuts.

(25) `MSK_IINF_MIO_TOTAL_NUM_CLIQUE_CUTS`
Number of clique cuts.

(26) `MSK_IINF_MIO_TOTAL_NUM_COEF_REDC_CUTS`
Number of coef. redc. cuts.

(27) `MSK_IINF_MIO_TOTAL_NUM_CONTRA_CUTS`
Number of contra cuts.

(28) `MSK_IINF_MIO_TOTAL_NUM_CUTS`
Total number of cuts generated by the mixed integer optimizer.

(29) `MSK_IINF_MIO_TOTAL_NUM_DISAGG_CUTS`
Number of diasagg cuts.

(30) `MSK_IINF_MIO_TOTAL_NUM_FLOW_COVER_CUTS`
Number of flow cover cuts.

(31) `MSK_IINF_MIO_TOTAL_NUM_GCD_CUTS`
Number of gcd cuts.

(32) `MSK_IINF_MIO_TOTAL_NUM_GOMORY_CUTS`
Number of Gomory cuts.

(33) `MSK_IINF_MIO_TOTAL_NUM_GUB_COVER_CUTS`
Number of GUB cover cuts.

(34) `MSK_IINF_MIO_TOTAL_NUM_KNAPSUR_COVER_CUTS`
Number of knapsack cover cuts.

(35) `MSK_IINF_MIO_TOTAL_NUM_LATTICE_CUTS`
Number of lattice cuts.

(36) `MSK_IINF_MIO_TOTAL_NUM_LIFT_CUTS`
Number of lift cuts.

(37) `MSK_IINF_MIO_TOTAL_NUM_OBJ_CUTS`
Number of obj cuts.

(38) `MSK_IINF_MIO_TOTAL_NUM_PLAN_LOC_CUTS`
Number of loc cuts.

(39) `MSK_IINF_MIO_TOTAL_NUM_RELAX`
Number of relaxations solved during the optimization.

(40) `MSK_IINF_MIO_USER_OBJ_CUT`
If it is non-zero, then the objective cut is used.

(41) `MSK_IINF_OPT_NUMCON`
Number of constraints in the problem solved when the optimizer is called.

(42) `MSK_IINF_OPT_NUMVAR`
Number of variables in the problem solved when the optimizer is called

(43) `MSK_IINF_OPTIMIZE_RESPONSE`
The reponse code returned by optimize.

(44) `MSK_IINF_RD_NUMANZ`
Number of non-zeros in A that is read.

(45) `MSK_IINF_RD_NUMCON`
Number of constraints read.

(46) `MSK_IINF_RD_NUMCONE`
Number of conic constraints read.

(47) `MSK_IINF_RD_NUMINTVAR`
Number of integer constrained variables read.

(48) `MSK_IINF_RD_NUMQ`
Number of nonempty Q matrices read.

(49) `MSK_IINF_RD_NUMQNZ`
Number of Q non-zeros.

(50) `MSK_IINF_RD_NUMVAR`
Number of variables read.

(51) `MSK_IINF_RD_PROTYPE`
Problem type.

(52) `MSK_IINF_SIM_DUAL_DEG_ITER`
The number of dual degenerate iterations.

(53) `MSK_IINF_SIM_DUAL_HOTSTART`
If 1 then the dual simplex algorithm is solving from an advance basis.

(54) `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.

(55) `MSK_IINF_SIM_DUAL_INF_ITER`
The number of iterations taken with dual infeasibility.

(56) `MSK_IINF_SIM_DUAL_ITER`
Number of dual simplex iterations during the last optimization.

(57) `MSK_IINF_SIM_NUMCON`
Number of constraints in the problem solved by the simplex optimizer.

(58) `MSK_IINF_SIM_NUMVAR`
Number of variables in the problem solved by the simplex optimizer.

(59) MSK_IINF_SIM_PRIMAL_DEG_ITER
The number of primal degenerate iterations.

(60) MSK_IINF_SIM_PRIMAL_HOTSTART
If 1 then the primal simplex algorithm is solving from an advance basis.

(61) 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.

(62) MSK_IINF_SIM_PRIMAL_INF_ITER
The number of iterations taken with primal infeasibility.

(63) MSK_IINF_SIM_PRIMAL_ITER
Number of primal simplex iterations during the last optimization.

(64) MSK_IINF_SIM_SOLVE_DUAL
Is non-zero if dual problem is solved.

(65) MSK_IINF_SOL_BAS_PROSTA
Problem status of the basic solution. Updated after each optimization.

(66) MSK_IINF_SOL_BAS_SOLSTA
Solution status of the basic solution. Updated after each optimization.

(67) MSK_IINF_SOL_INT_PROSTA
Problem status of the integer solution. Updated after each optimization.

(68) MSK_IINF_SOL_INT_SOLSTA
Solution status of the integer solution. Updated after each optimization.

(69) MSK_IINF_SOL_ITR_PROSTA
Problem status of the interior-point solution. Updated after each optimization.

(70) MSK_IINF_SOL_ITR_SOLSTA
Solution status of the interior-point solution. Updated after each optimization.

(71) 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.

(72) 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.

(73) 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.

18.15 Information item types

(0) MSK_INF_DOU_TYPE
Is a double information type.

(1) MSK_INF_INT_TYPE
Is an integer.

18.16 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.

18.17 Bound keys

(0) MSK_MARK_LO
The lower bound is selected for sensitivity analysis.

(1) MSK_MARK_UP
The upper bound is selected for sensitivity analysis.

18.18 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.

18.19 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.

18.20 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.

18.21 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.

18.22 Message keys

(1100) MSK_MSG_MPS_SELECTED

(1000) MSK_MSG_READING_FILE

(1001) MSK_MSG_WRITING_FILE

18.23 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.

18.24 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.

18.25 On/off

(0) MSK_OFF
Switch the option off.

(1) MSK_ON
Switch the option on.

18.26 Optimizer types

(9) MSK_OPTIMIZER_CONCURRENT
The optimizer for nonconvex nonlinear problems.

(2) MSK_OPTIMIZER_CONIC
Another cone optimizer.

(5) MSK_OPTIMIZER_DUAL_SIMPLEX
The dual simplex optimizer is used.

(0) MSK_OPTIMIZER_FREE
The optimizer is chosen automatically.

(6) MSK_OPTIMIZER_FREE_SIMPLEX
Either the primal or the dual simplex optimizer is used.

(1) MSK_OPTIMIZER_INTPNT
The interior-point optimizer is used.

(7) MSK_OPTIMIZER_MIXED_INT
The mixed integer optimizer.

(8) MSK_OPTIMIZER_NONCONVEX
The optimizer for nonconvex nonlinear problems.

(4) MSK_OPTIMIZER_PRIMAL_SIMPLEX
The primal simplex optimizer is used.

(3) MSK_OPTIMIZER_QCONE
The Qcone optimizer is used.

18.27 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.

18.28 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.

18.29 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.

18.30 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.

18.31 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.

18.32 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.

18.33 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.

18.34 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.

18.35 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.

18.36 Sensitivity types

(0) MSK_SENSITIVITY_TYPE_BASIS
Basis sensitivity analysis is performed.

(1) MSK_SENSITIVITY_TYPE_OPTIMAL_PARTITION
Optimal partition sensitivity analysis is performed.

18.37 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.

18.38 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.

18.39 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).

18.40 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.

18.41 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.

18.42 Solution types

(1) MSK_SOL_BAS
The basic solution.

(2) MSK_SOL_ITG
The integer solution.

(0) MSK_SOL_ITR
The interior solution.

18.43 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.

18.44 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.

18.45 Starting point types

(1) `MSK_STARTING_POINT_CONSTANT`
The starting point is set to a constant. This is more reliable than a non-constant starting point.

(0) `MSK_STARTING_POINT_FREE`
The starting point is chosen automatically.

18.46 Stream types

(2) MSK_STREAM_ERR
Error stream.

(0) MSK_STREAM_LOG
Log stream.

(1) MSK_STREAM_MSG
Message stream.

(3) MSK_STREAM_WRN
Warning stream.

18.47 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.

18.48 Variable types

(0) MSK_VAR_TYPE_CONT
Is a continuous variable.

(1) MSK_VAR_TYPE_INT
Is an integer variable.

18.49 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

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.

B.1 The MPS file format

The version of the MPS format supported by MOSEK allows specification of an optimization problem on the form

$$
\begin{array}{lccl}
l^c & \leq & Ax + q(x) & \leq & u^c, \\
l^x & \leq & x & \leq & u^x, \\
 & & x \in \mathcal{C}, & & \\
 & & x_{\mathcal{J}} \text{ integer}, & &
\end{array}
\tag{B.1}
$$

where

- $x \in R^n$ is the vector of decision variables.
- $A \in R^{m \times n}$ is the constraint matrix.
- $l^c \in R^m$ is the lower limit on the activity for the constraints.
- $u^c \in R^m$ is the upper limit on the activity for the constraints.
- $l^x \in R^n$ is the lower limit on the activity for the variables.
- $u^x \in R^n$ is the upper limit on the activity for the variables.
- $q : R^n \rightarrow R$ is a vector of quadratic functions. Hence,

  $$q_i(x) = 1/2 x^T Q^i x$$

  where it is assumed that

  $$Q^i = (Q^i)^T. \tag{B.2}$$

  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
*23456789012345678901234567890123456789012345678901234567890
NAME          [name]
OBJSENSE
    [objsense]
OBJNAME
    [objname]
ROWS
 ?  [cname1]
COLUMNS
    [vname1]  [cname1]    [value1]     [vname3]  [value2]
RHS
    [name]    [cname1]    [value1]     [cname2]  [value2]
RANGES
    [name]    [cname1]    [value1]     [cname2]  [value2]
QSECTION      [cname1]
    [vname1]  [vname2]    [value1]     [vname3]  [value2]
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 B.5 for details.

B.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.

B.1.2 NAME

In this section a name (`[name]`) is assigned to the problem.

B.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.

B.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.

B.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	Re-quired	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.

B.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.

B.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.

B.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			

B.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	Re-quired	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

$$\begin{array}{lrcl} \text{minimize} & -x_2 + 0.5(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & & \\ \text{subject to} & x_1 + x_2 + x_3 & \geq & 1, \\ & x \geq 0 & & \end{array}$$

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 QSECTIONs can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- All entries specified in a QSECTION are assumed to belong to the lower triangular part of the quadratic term of Q.

B.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	Re-quired	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].

B.1.11 CSECTION (optional)

The purpose of the `CSECTION` is to specify the constraint

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

in (B.1).

It is assumed that $\mathcal{C}$ satisfies the following requirements. Let

$$x^t \in R^{n^t},\ t = 1, \ldots, k$$

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

$$\mathcal{C} := \left\{ x \in R^n : \ x^t \in \mathcal{C}_t,\ t = 1, \ldots, k \right\}$$

where $\mathcal{C}_t$ must have one of the following forms

- R set:
$$\mathcal{C}_t = \{x \in R^{n^t}\}.$$
- Quadratic cone:
$$\mathcal{C}_t = \left\{ x \in R^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \tag{B.3}$$
- Rotated quadratic cone:
$$\mathcal{C}_t = \left\{ x \in R^{n^t} : 2x_1 x_2 \geq \sum_{j=3}^{n^t} x_j^2,\ x_1, x_2 \geq 0 \right\}. \tag{B.4}$$

In general, only quadratic and rotated quadratic cones are specified in the MPS file whereas membership of the 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 R cone.

Next, let us study an example. Assume that the quadratic cone

$$x_4 \geq \sqrt{x_5^2 + x_8^2} \tag{B.5}$$

and the rotated quadratic cone

$$2x_3 x_7 \geq x_1^2 + x_8^2,\ x_3, x_7 \geq 0, \tag{B.6}$$

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
*234567890123456789012345678901234567890123456789012345678901
CSECTION      konea       0.0          QUAD
    x4
    x5
    x8
CSECTION      koneb       0.0          RQUAD
    x7
    x3
    x1
    x0
```

This first CSECTION specifies the cone (B.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 (B.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	Re-quired	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. (B.3).
RQUAD	≥ 2	Rotated quadratic cone i.e. (B.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	Re-quired	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.

B.1.12 ENDATA

This keyword denotes the end of the MPS file.

B.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.

B.3 General limitations

- An MPS file should be an ASCII file.

B.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.

B.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 C

The LP file format

MOSEK supports the LP file format with some extensions i.e. MOSEK can read and write LP formatted files.

C.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.

C.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

- $x \in R^n$ is the vector of decision variables.
- $c \in R^n$ is the linear term in the objective.
- $q^o :\in R^n \rightarrow R$ is the quadratic term in the objective where

  $$q^o(x) = x^T Q^o x$$

  and it is assumed that

  $$Q^o = (Q^o)^T. \tag{C.1}$$

- $A \in R^{m\times n}$ is the constraint matrix.
- $l^c \in R^m$ is the lower limit on the activity for the constraints.
- $u^c \in R^m$ is the upper limit on the activity for the constraints.
- $l^x \in R^n$ is the lower limit on the activity for the variables.
- $u^x \in R^n$ is the upper limit on the activity for the variables.
- $q : R^n \to R$ is a vector of quadratic functions. Hence,

  $$q_i(x) = x^T Q^i x$$

  where it is assumed that

  $$Q^i = (Q^i)^T. \tag{C.2}$$
- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer constrained variables.

C.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.

C.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.

C.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.

$$-5 \leq x_1 + x_2 \leq 5 \tag{C.3}$$

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 (C.3) may be written as

$$x_1 + x_2 - sl_1 = 0, \ -5 \leq sl_1 \leq 5.$$

C.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
```

C.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.

C.2.1.5 Terminating section

Finally, an LP formatted file must be terminated with the keyword

```
end
```

C.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
```

C.2.2 LP format peculiarities

C.2.2.1 Comments

Anything on a line after a "\" is ignored and is treated as a comment.

C.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.

C.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.

C.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.

C.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.

C.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.

C.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
```

C.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 D

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.

D.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.

D.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 + y + x^2 + 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]
```

D.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

$$-10 \leq x, y \leq 10. \tag{D.1}$$

  – `[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 thing is always to set to status `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`.

- [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.

D.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]+)?
```

D.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"
```

D.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]
```

D.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.

D.5 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

D.5.1 Linear example `lo1.opf`

Consider the example:

$$\begin{array}{lrcrcr} \text{minimize} & -10x_1 & & -9x_2, & & \\ \text{subject to} & 7/10x_1 & + & 1x_2 & \leq & 630, \\ & 1/2x_1 & + & 5/6x_2 & \leq & 600, \\ & 1x_1 & + & 2/3x_2 & \leq & 708, \\ & 1/10x_1 & + & 1/4x_2 & \leq & 135, \\ & x_1, & & x_2 & \geq & 0. \end{array} \tag{D.2}$$

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]
```

D.5.2 Quadratic example `qo1.opf`

An example of a quadratic optimization problem is

$$\begin{array}{lccc} \text{minimize} & & & x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\ \text{subject to} & 1 & \leq & x_1 + x_2 + x_3, \\ & & & x \geq 0. \end{array} \tag{D.3}$$

This can be formulated in `opf` as shown below.

```
[comment]
  Example qo1.mps conterted 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]
```

D.5.3 Conic quadratic example `cqo1.opf`

Consider the example:

$$
\begin{array}{lccl}
\text{minimize} & 1x_1 + 2x_2 & & \\
\text{subject to} & 2x_3 + 4x_4 & = & 5, \\
& x_5^2 & \leq & 2x_1x_3, \\
& x_6^2 & \leq & 2x_2x_4, \\
& x_5 & = & 1, \\
& x_6 & = & 1, \\
& x \geq 0. & &
\end{array}
\tag{D.4}
$$

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 conterted 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]
```

D.5.4 Mixed integer example milo1.opf

Consider the mixed integer problem:

$$\begin{array}{lccl} \text{maximize} & x_0 + 0.64x_1 & & \\ \text{subject to} & 50x_0 + 31x_1 & \leq & 250, \\ & 3x_0 - 2x_1 & \geq & -4, \\ & x_0, x_1 \geq 0 & & \text{and integer} \end{array} \tag{D.5}$$

This can be implemented in OPF with:

```
[comment]
   Written by MOSEK version 5.0.0.7
   Date 20-11-106
   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 E

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 F

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 F.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
*23456789012345678901234567890123456789012345678901234567890
NAME          [name]
 ?? [vname1]             [value1]
ENDATA
```

Figure F.1: The standard ORD format.

F.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 G

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.

G.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 G.1: Status keys.

`NAME` The name of the constraint assigned by the user.

`AT` The status of the constraint. In Table G.1 the possible values of the status keys and their interpretation are shown.

`ACTIVITY` Given the ith constraint on the form

$$l_i^c \leq \sum_{j=1}^{n} a_{ij} x_j \leq u_i^c, \tag{G.1}$$

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 (G.1)).

`UPPER LIMIT` Is the quantity u_i^c (see (G.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.

G.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] K. D. Andersen. A Modified Schur Complement Method for Handling Dense Columns in Interior-Point Methods for Linear Programming. *ACM Trans. Math. Software*, 22(3):348–356, 1996.

[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. Kennington 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, 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] R. J. Vanderbei. *Linear Programming. Foundations and Extensions*. Kluwer Academic Publishers, Boston/London/Dordrect, 1997.

[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