# The MOSEK C API manual.
# Version 5.0 (Revision 45).

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<br>C/O Symbion Science Park<br>Fruebjergvej 3, Box 16<br>2100 Copenhagen Ø<br>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 15 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 12 which deals with analysis of infeasible problems,
- Chapter 13 about the sensitivity analysis interface, and
- Chapter 14 which contains a few larger case studies.

Finally, the C API reference material is located in

- Chapter 16 which lists all types and functions,
- Chapter 17 which lists all available parameters,
- Chapter 18 which lists all response codes, and
- Chapter 19 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 [14] or "Computer Solution of Linear Programs" by Nazareth [19]. For more theoretical aspects see for example "Nonlinear programming: Theory and algorithms" by Bazaraa, Shetty, and Sherali [10]. Finally the book "Model building in mathematical programming" by Williams [25] 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:\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:\mosek\5\licenses
PATH                     : c:\local\python24;c:\local\bin;C:\mosek\5\tools\platform\win\bin

*** 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:\mosek\5
```

and that the platform-specific files are located in

```
c:\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:\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:\mosek\5\tools\platform\<platform>\h\mosek.h
  ```

- The MOSEK lib file located in

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

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

1. Change directory:

   ```
   c:
   cd \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:\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:\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 is 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 16.4.

#### 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 17.

### 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:

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

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 is 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:

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

| 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 is 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:

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

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 $k$th 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:

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

This can be written equivalently as

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

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 is 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:

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

This is equivalent to

$$\begin{array}{lll} \text{minimize} & 1/2x^T Q^o x + c^T x & \\ \text{subject to} & 1/2x^T Q^0 x + Ax & \geq \quad b, \end{array} \tag{5.19}$$

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 is 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

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

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 is 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 is 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 is 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:

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

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 inputing 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 wish to change the time required for assembly of product 0 to 3 minutes. This corresponds to setting $a_{0,0} = 3$. We do this 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:

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

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) | Polish (minutes) | Pack (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 varaible 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 varaible */
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 look like:

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

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 be automatically used to reduce solution time of the new problem if possible.

In this case an optimal solution to problem (5.28) was found and we then added a column 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 freely chose the simplex optimizer when optimizing.

```
/* Change optimizer to simplex free 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 wish to add a new stage to the production called “Quality control” to which we have 30000 minutes 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, there are situations where the user has valuable knowledge about the problem, which may be used to improve 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 framentation. One way to avoid this is to give MOSEK an estimated size of your problem using the functions:

- `MSK_putmaxnumvar` which set estimate for the number of variables.
- `MSK_putmaxnumcon` which set estimate for the number of constraints.
- `MSK_putmaxnumcone` which set estimate for the number of cones.
- `MSK_putmaxnumanz` which set estimate for the number of nonzeros in $A$.
- `MSK_putmaxnumqnz` which set estimate for the number of nonzeros in quadratic terms (both objective and constraints).

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 may 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 if 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 for instance be worthwhile to add many variables instead of one. Initially let the unused variable be fixed at zero, then later unfix them as needed. Similarly, you can add multiple free constraints and then use them as needed.

**Only use one environment (env):** If possible share the environment (`env`) between several tasks. For most application it is only needed to create a single `env`.

**Do not remove basic variables:** Instead of removing a basic variable it may be more efficient to fix the variable at zero and then remove it later when 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 then it means the number of constraints.

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

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:

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

- The quadratic terms in the constraints:

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

- The linear terms in the constraints:

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

| C name | C type | Dimension | Related problem parameter |
|---|---|---|---|
| numcon | int | | $m$ |
| numvar | int | | $n$ |
| numcone | int | | $t$ |
| numqonz | int | | $q_{ij}^o$ |
| qosubi | int[] | numqonz | $q_{ij}^o$ |
| qosubj | int[] | numqonz | $q_{ij}^o$ |
| qoval | double* | numqonz | $q_{ij}^o$ |
| c | double[] | numvar | $c_j$ |
| cfix | double | | $c^f$ |
| numqcnz | int | | $q_{ij}^k$ |
| qcsubk | int[] | qcnz | $q_{ij}^k$ |
| qcsubi | int[] | qcnz | $q_{ij}^k$ |
| qcsubj | int[] | qcnz | $q_{ij}^k$ |
| qcval | double* | qcnz | $q_{ij}^k$ |
| aptrb | int[] | numvar | $a_{ij}$ |
| aptre | int[] | numvar | $a_{ij}$ |
| asub | int[] | aptre[numvar-1] | $a_{ij}$ |
| aval | double[] | aptre[numvar-1] | $a_{ij}$ |
| bkc | MSKboundkeye* | numcon | $l_k^c$ and $u_k^c$ |
| blc | double[] | numcon | $l_k^c$ |
| buc | double[] | numcon | $u_k^c$ |
| bkx | MSKboundkeye * | numvar | $l_k^x$ and $u_k^x$ |
| blx | double[] | numvar | $l_k^x$ |
| bux | double[] | numvar | $u_k^x$ |

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 bounds on the constraints are specified using the variables `bkc`, `blc`, and `buc`. The components of the integer array `bkc` specifies the type of the bounds according to Table 5.3. 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

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

and

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

- The bounds on the variables are specified using the variables `bkx`, `blx`, and `bux`. The components in the integer array `bkx` specifies the type of the bounds according to Table 5.3. The numerical values for the lower bounds on the variables are given by

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

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

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

#### 5.8.1.1 Bounds

A bound on a variable or 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, 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 for 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);
```

Note that items in MOSEK are numbered from 0, such 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 on constraint 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 whole or in parts, in a sparse format. There are basically two different format 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);
```

Note that in some cases (like `MSK_putaijlist`), *only* the specified indexes are 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 nonzero 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 nonzeros 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 nonzero entries are given in column or row order.

A sparse matrix in column ordered format consist of:

**asub:** List of row indexes.

**aval:** List of nonzero 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 such that for

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

We define

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

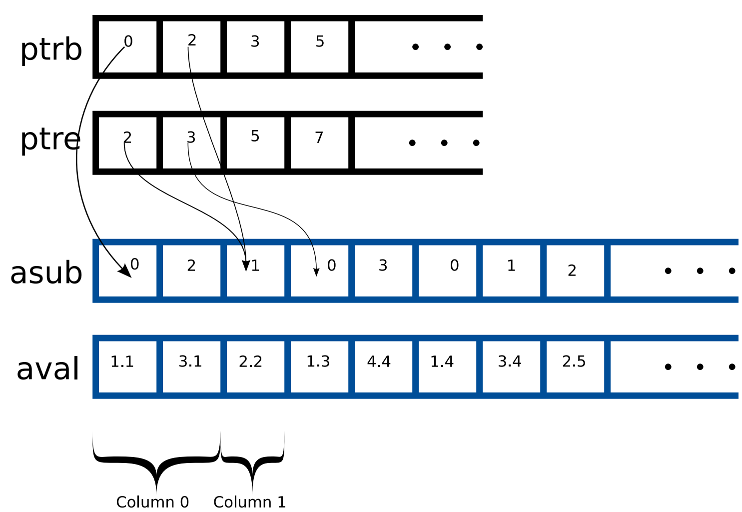


Figure 5.1: The matrix $A$ (5.43) represented in column ordered matrix format.

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 format as

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

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:

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

# 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 ourself to separable functions.

### 6.1.1 The problem

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

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

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 part linear 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 $i$th 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 only depends 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

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

to demonstrate the solution of a convex separable optimization problem using MOSEK.

First observe the problem (6.2) is not a separable optimization problem due to the logarithmic term in objective is not a function of a single variable. However, by introducing one additional constraint and variable then 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 all nonlinear functions are well defined for values of $x$ satisfying the variable bounds strictly i.e.

$$x_3 > 0.$$

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

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

### 6.1.3 `scopt` an optimizer for separable convex optimization

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

#### 6.1.3.1 Design principles of `scopt`

All the linear data of the problem such as $c$ and $A$ are 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 specifies the type of the nonlinear expressions and where they should be added.

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

| `oprc[k]` | `opric[k]` | `oprjc[k]` | `oprfc[k]` | `oprgc[k]` | `oprhc[k]` | expression added in 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 to. `oprfc[k]` and `oprgc[k]` are parameters used when the nonlinear expression are evaluated. This implies that nonlinear expressions can be add to an arbitrary constraint and hence you can have multiple nonlinear constraints.

Using the same idea 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 wish to add a nonlinear expresion $-ln(x_3)$ to the objective. This is an expresion on the form $f \ln(gx_j + h)$ with $f = -1$, $g = 1$, $h = 0$ and $j = 3$. This can be represented with:

```
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 which defines the two functions `MSK_scbegin` and `MSK_scend`. These two functions are used to initialize and remove the nonlinear function data respectively.
- `scopt.c` : This file implements the nonlinear function initialization and evaluation. Hence, in this module all the information about the nonlinear functions required are computed and inputted to MOSEK.
- `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 `scopt.c` which includes comments. However, we will show the driver program `tstscopt.c` which solves the example (6.2).

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

   File     : tstscopt.c

   Purpose  : Solves 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 nonzeros 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;

  /* Make 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 )
    {
      /* Setup 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 Building and linking

In order to build `tstscopt`, then the object file `scopt.obj` should first be created by compiling the file `scopt.c`. Next the file `tstscopt.c` should be compiled and linked with `scopt.obj` and the appropriate MOSEK library.

#### 6.1.3.5 Adding more nonlinear expressions types

`scopt` handles only a limited number of nonlinear expressions 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` which 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 the a nonlinear expression. After the modification the function has the form:

```
static int evalopr(int    opr,
                   double f,
                   double g,
                   double h,
                   double xj,
                   double *fxj,
                   double *grdfxj,
                   double *hesfxj)
/* Purpose: Evaluates 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 gradiant. */


      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

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

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

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

then the problem (6.4) is a convex optimization 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

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

which could be solved using the scopt interface discussed in Section 6.1. 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.

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 Souce code

Included in the MOSEK distribution is the source code for a program that enables you to:

1. Read (and write) a data file stating an exponential optimization problem.
2. Verifies that the input data is reasonable.
3. Solves the problem via the exponential optimization problem (6.5) or the dual geometric optimization problem (6.12).
4. Writes a solution file.

### 6.2.3 Solving from the command line.

Subsequently 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. It 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. The next three lines defines the number of constraints ($m$), the number of variables ($n$), and the number of terms $T$ in the problem. Next three different sections follows.

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 term number $t$ belongs to the objective.

The third section

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

defines the nonzero $a_{t,j}$ values. For instance the entry

$$1 \quad 3 \quad 3.3$$

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

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

### 6.2.4 Choosing primal or dual form

One can select to solve the exponential optimization problem directly in the primal form (6.5) or on the dual form. By default `mskexpopt` solves problem on dual form because usually that is the most efficient way. The command line option

```
-primal
```

chooses the primal form.

### 6.2.5 An example

Consider the problem:

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

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

```
* File : expopt1.eo

1   * numcon
3   * numvar
5   * numter

* Coefficients of terms

40
20
40
0.3333333
1.3333333

* Constraints each term belong to

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 file.
- **MSK_expoptsetup:** Sets up a problem. The function take the arguments:
  - expopttask: A MOSEK task structure.
  - solveform: If 0, then the optimizer will chose 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.

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

  - *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:** Implementation of 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 is reserved.

   File     : tstexpopt.c

   Purpose  : Demonstrate 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 hold 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

Suppose a column $j$ of the coefficients matrix $A$ contains only positive values. If all other variables are fixed at some value, then the terms $x_j$ occurs in will vanish if $x_j \rightarrow 0$. Such a problem is most likely ill-posed and MOSEK issues a warning if it is encountered. It is best to modify the problem in such a case so the $x_j$ variable is removed from the problem.

A similar problem occurs if a column of the coefficient matrix $A$ contains only negative values. Then

$x_j \to \infty$ in the optimal solution and the terms the $x_j$ occur in vanish.

## 6.3 General convex optimization

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

### 6.3.1 A warning

Using the general convex optimization facaility in MOSEK is very difficult. It is therefore recommended to use conic optimization or the scopt interface insted. Alternatively the GAMS or AMPL links is also 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

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

Observe this problem is a generalization of linear and quadratic optimization. This implies that the parameters $c$, $A$, $Q^o$, $Q$, and so forth has the same meaning 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 described by two functions:

**$f(x)$:** A general nonlinear function which must be twice differentiable.

**$g_k(x)$:** A general nonlinear function which must be twice differentiable.

### 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. This can be stated more precisely as $f(x)$ and $g(x)$ must be at least twice differentiable for all $x$ such that

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

The second assumption is that

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

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

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

must be a convex function if

$$u_k^c < \infty$$

and a concave function if

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

Note this implies 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 using the general convex framework see Section 6.4.

## 6.4 Dual geometric optimization

Dual geometric is special class of nonlinear optimization problem involing a nonlinear and nonseparable objective function. In this section we will show how to solve dual geometric optimization problems using MOSEK.

Please observe geometric optimization can be solved using exponential optimization discussed previously and that is the recommened method for solving geometric optimization problems.

### 6.4.1 The problem

the dual geometric optimization problem

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

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

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

Using this notation 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 positive vector meaning all the components of $v$ are positive.

Given the assumptions then it can be shown that $f$ is concave function and therefore the dual geometric optimization problem can be solved using MOSEK.

For a through discussion of geometric optimization see [10, pp. 531-538].

Formulas involving sums are a bit tedious to write therefore 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

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

Observe that the Hessian is a block diagonal matrix and in particular if $t$ is large then it is very sparse. Both of these facts are advantageous because MOSEK will automatically exploit them to speed up the computations. Moreover, the Hessian can be computed cheaply. In fact it can be computed in

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

operations.

### 6.4.2 A numerical example

For the purpose of demonstration the data

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

and the function $f$ given by

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

will be used subsequently.

### 6.4.3 dgopt a program for dual geometric optimization

The generic dual geometric optimization problem and a numerical example has now been presented and we will therefore turn the attention to the development of a program which using the MOSEK API solves the dual geometric optimization problem.

#### 6.4.3.1 Data input

The first problem that arises is how to feed the problem data into MOSEK. However, due to the constraints of the optimization problem are linear, then they can be specified fully using a MPS file as in the linear case. For the numerical example mentioned above the MPS file has the format:

```
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 and for that purpose a file having the format

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

is used. 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

Next 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` consist of the files:

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

These files are available electronically at:

```
mosek\<verson>\tools\examp\
```

and are listed below:

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

   File:      dgopt.c

   Purpose:   This program solves dual geometric
              optimization problems using the
              MOSEK API.

              It works as follows

              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 nonzeros 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
       * strong the Hessian.
       */

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

          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: Evalute 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 gradiant of the Lagrangian.
     * Note 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 Lagrangien
     * 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 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);

    /* setup 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 nonzeros
           * 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,
                 coresponding 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 allocate 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 is reserved.

   File:      mskdgopt.c

   Purpose:
              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;


  /* Make 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 reads the linear part of the problem from a MPS file into the task. Next the nonlinear part of the problem is read from a files 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 setup 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 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 and it reports back numerical information about $f$ such as the objective value and objective gradient value 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 identical to 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 nonsingular meaning

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

or equivalently that $B^{-1}$ exists. This implies 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

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

If variable $x_j$ is a basis variable, then the $j$'th column of $A$ denoted $a_{:,j}$ will appear in $B$. Similarly, if $x^c_i$ is a basis variable, then the $i$'th column of $-I$ will appear in the basis. The order of the basis variables and therefor 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 initialize data structures for later use and return the indexes of the basis variables in the array `basis`. The interpretation of `basis` is as follows. If

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

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

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

then the $i$'th basis variable is variable

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

and the $i$'th column of $B$ is the column

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

For instance if $\texttt{basis}[0] = 4$ and $\texttt{numcon} = 5$, then since $\texttt{basis}[0] < \texttt{numcon}$ we have that the first basis variable is $x^c_4$. 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:

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

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

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

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

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix}. \tag{6.17}$$

Observe 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 is reserved.


  Purpose:   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 variabels 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 nr. %d is xc%d.\n",i, basis[i]);
  else
    printf ("Basis variable nr. %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 occured.)\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. The solution to the linear system is thus the optimal solution to the problem. When running the example program the following output is produced.

```
basis[0] = 1
Basis variable nr. 0 is xc1.
basis[1] = 2
Basis variable nr. 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
```

Observe the ordering of the basis variables is

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

and the basis is thus given by:

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

It can be verified that

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

indeed 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

$$Ax = b$$

with MSK_solvewithbasis without optimizing the problem as done in the previous example. Instead of optimizing the problem (and thereby defining a basic solution) then MOSEK should be informed by the user which variables should be basic variables. The corresponding linear system can then be solved with MSK_solvewithbasis. This makes it possible to solve any linear system by defining the coefficient matrix $A$ in MOSEK and selecting all variables as basic.

Below we demonstrate how to solve the linear system

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

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

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

   File     :  solvelinear.c

   Purpose  :  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.
     Only call this function 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 with the call to MSK_putsolutioni. Here we define the status key for each variable. The actual value of the variables are not important and can be selected arbitrarily, we set them to zero. All variables corresponding to columns in the linear system we wish to solve are selected as basic and the slack variables for constraints are non-basic and set at 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 the system.

## 6.6 The progress callback

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

- to obtain information the solution process,
- report about the progress the optimizer makes,
- and if desired ask MOSEK to terminate.

### 6.6.1 Source code example

In the subsequent source code example it is documented how the progress callback function is used.

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

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

   File:      callback.c

   Purpose:   1. Demonstrate how to use the progress
                 callback.
              2. The program reads an optimization
                 problem from an user specified
                 MPS file.
              3. Optimizes the problem.
 */


#include "mosek.h"


/* Note: This function is declared using MSKAPI,
         so the correct calling convention is
         used. */
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);

      if ( cputime>=maxtime[0] )
      {
        /* mosek is using too much time.
           Terminate it. */
        return ( 1 );
      }
      break;
    case MSK_CALLBACK_IM_INTPNT:
      if ( cputime>=maxtime[0] )
      {
        /* mosek is using too much time.
           Terminate it. */
        return ( 1 );
      }
      break;
    case MSK_CALLBACK_END_INTPNT:
      printf("Interior-point optimizer finished.\n");
      break;
    case MSK_CALLBACK_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("  Elasped time: %6.2f(%.2f)\n",
             cputime,scputime);
      printf("Obj.: %-18.6e\n",pobj);

      if ( cputime>=maxtime[0] )
      {
        /* mosek is using too much time.
           Let us stop. */
        return ( 1 );
      }
      break;
    case MSK_CALLBACK_BEGIN_BI:
      printf("Basis identifcation started.\n");
      break;
    case MSK_CALLBACK_END_BI:
      printf("Basis identification done.\n");
      break;
  }

  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<2 )
  {
    printf("No input file spcified\n");
    exit(0);
  }

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

  /* Make 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[1].
     */

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

It is possible to customize the warning and error reporting in the `C` API. The function `MSK_putresponsefunc` may 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 may then handle the error/warning as desired.

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


   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. Note that

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

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

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

### 6.8.1 A source code example

The example below documents how to convert a `wchat_t *` string to an UTF8 string.

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


   Purpose:   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" specifing a file name
         is converted to 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

Observe that the MPS and LP format are based ASCII formats whereas the OPF, MBT, and XML UTF8 formats. This implies if the problem should be written to a MPS or a LP formatted file, then all names on constraints, variables etc. should be ASCII strings.

# 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 found to be infeasible.
- A set of examples showing different ways to formulate commonly occurring problems such that they can be solved using MOSEK.
- Recomendations 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 part linear 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

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

If a bound in the primal problem is plus or minus infinity, the corresponding dual variable is fixed at 0, and we use the convension 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 said to be *(dual) feasible*, otherwise the problem is said to be *(dual) infeasible*.

We will denote a solution

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

such 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 *primal-dual* solution. A solution which is both primal and dual feasible if denoted a *primal-dual feasible* solution.

[1] We will use the words bounds 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 non-negative. 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.

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

Note that the duality gap will always be non-negative.

#### 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 exists feasible primal and dual solutions such that the duality gap is zero, or, equivalently, that the *complementarity conditions*

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

are satisfied.

If (7.1) has an optimal solution and MOSEK successfully solves the problem, then both the primal and dual solution is reported, including a status telling the exact state of the solution.

#### 7.1.1.3 Primal infeasible problems

If the problem (7.1) is infeasible (has no feasible solution), then 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

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

such 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) such 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 will imply 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, then so is the original primal problem.

#### 7.1.1.4 Dual infeasible problems

If the problem (7.2) is infeasible (has no feasible solution), then 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

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

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

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

such that the objective value $c^T x$ is negative. Such a solution will imlpy that (7.4) is unbounded, and that dual corresponding (7.4) to 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, then so is the original dual problem.

### 7.1.2 Primal and dual infeasible case

In the case 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 that is found).

## 7.2 Linear network flow problems

A network flow problems is a very special class of linear optimization problems which has many applications. The class of network flow problems can be specfied 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 capcity $[l^x_{ij}, u^x_{ij}]$. 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 network flow problem can be stated as follows

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

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

$$
\begin{array}{lcccccl}
\text{minimize} & & & \frac{1}{2}x^TQ^ox + c^Tx + c^f & & & \\
\text{subject to} & l^c_k & \leq & \frac{1}{2}x^TQ^kx + \sum\limits_{j=0}^{n-1} a_{k,i}x_j & \leq & u^c_k, & k = 0,\ldots,m-1, \\
 & l^x & \leq & x & \leq & u^x, & j = 0,\ldots,n-1,
\end{array} \tag{7.6}
$$

where the convexity requirement implies:

- $Q^o$ 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 positive 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 recommend to apply MOSEK 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 expirience that for the majority of pratical 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 the form

$$x_j^2$$

and not the form

$$x_j x_i.$$

The separable form has the advantages

- it is very easy to check the convexity assumption (why?)
- and the simpler structure in a separable problem usually makes them easier to solve.

It is well known that a positive semi-definite matrix $Q$ can always be factorized, that is, there always exist a matrix $F$ such that

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

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 that produced 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 seperable. Normally, if $F$ has fewer rows than columns, then the reformulation as separable problem is worthwhile. Indeed consider the extreme case where $F$ has one dense row and hence $Q$ will be dense matrix.

The idea presented above can also be applied to quadratic constraints. 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

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

It should be obvious how to generalize this idea to make any convex quadratic problem separable.

Next, consider the constraint

$$1/2x^T(D + F^T F)x \leq b$$

where $D$ is positive semi-definite matrix, $F$ is a matrix, and $b$ is a scalar. We will assume $D$ has a simple structure i.e. $D$ is for instance a diagonal or a block diagonal matrix. If that is the case, then the reformulation

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

may be worthwhile to perform.

Now the question may a appear when should a quadratic problem be reformulated to make it separable or near separable. The simplest rule of thumb, is it should be reformulated if the number non-zeros used to represent the problem decrease 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 $x_t \in R^{n_t}$. We note that the $n$-dimensional Euclidian space $R^n$ is itself a cone, 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 cone $\mathcal{C}_t$ 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:

$$\mathcal{C}_t = \left\{ x \in R^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2, \ x_1, x_2 \geq 0 \right\}.$$

Although these cones may seem to provide only limited expressive power then those 3 cones 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

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

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

$$\mathcal{C}^* = \mathcal{C}_1^* \times \cdots \mathcal{C}_p^*$$

where the 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 are 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:

$$\mathcal{C}_t := \left\{ x \in R^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2, \ x_1, x_2 \geq 0 \right\}. \quad \Leftrightarrow \quad \mathcal{C}_t^* = \mathcal{C}_t.$$

## 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 infeaibility. 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 a 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

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

where $F$ and $G$ are matrices. $c$ and $a$ are vectors. Here we for simplicity assume there is only one constraint. It should be obvious how to generalize the presented ideas to an arbitrary number of constraints.

Problem (7.13) can be reformulated as

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

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.

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

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

If we assume that $F$ is a nonsingular matrix — for instance if it is a diagonal matrix — then we have

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

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

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

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

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

where

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

This problem is equivalent to

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

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 cones $\mathcal{C}^i$ 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

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

Obviously this problem is equivalent to

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

Note in this case due to the special structure of the primal problem, then the dual problem can be reduced to an "ordinary" convex quadratically constrained optimization 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

An arbitrary polynomial term of the form

$$
f x^g
$$

cannot be represented with conic quadratic constraints but subsequently we will some special cases that can.

A particular simple polynomial term is the reciprocical 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 with

$$
\begin{array}{rcl}
z & = & \sqrt{2}, \\
z^2 & \leq & 2xy.
\end{array}
$$

The last formulation is a conic constraint plus a simple linear equality.

For example, consider the problem

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

where it is assumed $f_j > 0$ 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

$$
\frac{1}{x_j} \leq z_j \text{ and } \sum_{j=1}^{n} \frac{f_j}{x_j} \leq \sum_{j=1}^{n} f_j z_j = b.
$$

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 the 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 so-scalled market impact term is portfolio optimization then 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

$$x^{1.5} = \frac{x^2}{\sqrt{x}} \leq \frac{x^2}{2s} \leq t.$$

Observe $s$ can be chosen freely and $\sqrt{x} = 2s$ 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). Observe the $x \geq 0$ constraint does not appear explicitly in (7.25) and (7.26) but appears implicitly because $x = v \geq 0$.

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, p. 12-13]

#### 7.4.4.4 Further reading

If you want to know more about what can be modelled as a conic optimization problem then we can recommend the references [17, 12, 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.

#### 7.4.5.1 Nonattainment 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 is never attained because we implicitly assume the optimal $y$ should be finite.

#### 7.4.5.2 Nonattainment 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

$$x_1^* = 0,\ x_2^* = 1,\ x_3^* = 0 \text{ and } x_4^* = 1$$

which implies 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

$$2s_1^*s_2^* \geq (s_3^*)^2 = 1$$

and hence $s_2^* > 0$. Given this fact we can conclude

$$\begin{array}{rcl} y_1^* + y_3^* & = & 1 - s_2^* \\ & < & 1 \end{array}$$

implying the optimal dual objective value is 1 but is never attained. Hence, there does not exist a primal and dual bounded optimal solution that has zero duality gap. It is of course 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 somewhat strange constraint $x_1 = 0$ which implies $x_3 = 0$. 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 then it becomes easy to solve.

## 7.5 Nonlinear convex optimization

MOSEK is capable of solving smooth convex non-linear 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 constraint or slack variables.
- $c \in R^n$ is the part linear 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 bounds and limit interchangeably.

- $g : R^n \rightarrow R^m$ is a nonlinear vector function.

This means that the $i$th constraint has the form

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

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 non-linear 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

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

### 7.5.1 Duality

, we have not discussed what happens when MOSEK is used to solve a primal or dual infeasible. In the subsequent section those issues are addressed.

Similarly to the linear case, then MOSEK also reports dual information in 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

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

which is equivalent to

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

## 7.6 Recomendations

### 7.6.1 General

An optimization problem can often 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 may mean the ability to solve the problem or not.

Below we list several issues that are beneficial to be aware when developing agood formulation.

1. Sparsity is very important. The constraint matrix $A$ is assumed to be sparse matrix where by sparse it is meant that it contains many zeros i.e. typically less than 10% non-zeros. Normally the sparse $A$ is the less storage is required to store the problem and the faster the problem is solved.

2. Avoid large bounds because large bounds can introduce all sorts of numerical problems. Assume 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 nonbinding at optimum and therefore that the large number $1.0e16$ will not cause problems. Unfortunately, this a naive assumption because the large number $1.0e16$ 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 suffcient i.e. $1.0e16$ is just a way 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 can and will most likely cause numerical problems.

4. On a computer all computations are performed in finite precision. This implies
$$1 = 1 + \varepsilon$$
where $\varepsilon$ is about $10^{-16}$. This has the implication that the result of all computations are truncated leading to the introduction of rounding errors. The upshot is that small numbers and large numbers should be avoided. For instance it is recommended that all elements in $A$ is either zero or belongs 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 non-linear optimization it might be 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. Note that network problems and multi-commodity network flow problems usually contain one or more linearly dependent rows.

7. Finally, it can be recommended to consult some of the papers about preprocessing to obtain some ideas about efficient formulations. See e.g. [4, 5, 15, 16].

### 7.6.2 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 an extremely small 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. $f \in R^n$ and $g, b \in R$ are all 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.

Observe equalities involving absolute value such as

$$|x| = 1$$

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

### 7.7.2 The Markowitz portfolio model

In this section we 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 then 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 $j$ assumed to be a random variable. $r$ has a known mean $\bar{r}$ and covariance $\Sigma$.

The variable $x_j$ will denote the amount of asset $j$ traded in the period and have 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 $\Sigma$ 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 $\Sigma$ is advantageous from a computational point of view. If $L$ is not known, then it can be computed using the Cholesky factorization.

#### 7.7.2.1 Minimizing variance for a given return

In our first model we wish to minimize the variance while selecting a portfolio having a specified expected target return $t$. Additionally our 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 can be expressed in the model

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

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

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

#### 7.7.2.2 Conic quadratic reformulation.

An equivalent conic quadratic reformulation is given by:

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

here we minimize the standard deviation instead of variance. 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 cost

We will now expand our model to include transaction costs as a fraction of the traded volume. [1, pp. 445-475] argues transactions cost are important to incorporate and has the form

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

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 so called market impact term

$$\theta\sqrt{\frac{\text{trade volume}}{\text{daily volume}}} \approx m_j\sqrt{|x_j|}.$$

The $\theta$ and "daily volume" has 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 some amount $x_j$ is given by

$$|x_j|(m_j|x_j|^{\frac{1}{2}}).$$

This leads us to the model:

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

Now, defining the variable transformation

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

we obtain

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

As shown in Section 7.4.4.3 the set

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

can be modeled by

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

#### 7.7.2.4 Further reading

For further readinge we refer the reader to [18] in particular. The references [22] and [1] also contains a lot of interesting 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 non-linear problems. The purpose of the present chapter is to discuss which optimizers that are available for the continuous problem classes and how the performance of an optimizer can be tuned if needed.

This chapter deal with the optimizers for *continuous problems* with no integer variables.

## 8.1 How an optimizer works

When the optimizer is called, it performs roughly the following steps:

**Presolve:** Preprocessing to reduce the size of the problem.

**Dualizer:** Choose whether to solve the primal or the dual form of the problem.

**Scaling:** Scale the problem for better numerical stability.

**Optimize:** Solve the actual optimization.

The first three preprocessing steps are transparent to the user, but are 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

- to remove redundant constraints,
- eliminate fixed variables,
- remove linear dependencies,
- substitute out free variables,
- and in general to reduce the size of the optimization problem.

After the presolved problem is optimized then the solution is automatically postsolved so that the returned solution is valid for the original problem. Hence, the presolve is completely transparent. Further details about the presolve phase can be seen in [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 can be done by setting the parameter `MSK_IPAR_PRESOLVE_USE` to `MSK_PRESOLVE_MODE_OFF`.

The two most time consuming steps of the presolve is usually the

- eliminator
- and the linear dependency check.

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

The purpose of the eliminator is to eliminate free and implied free variables from the problem using substitution. For instance, given the constraints

$$\begin{array}{rcl} y & = & \sum_j x_j, \\ y, x & \geq & 0, \end{array}$$

then $y$ is an implied free variable that can substituted out of the problem if deemed worthwhile. By implied free variable it 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 that 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 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 are in general an extremely good idea because it reduces the size of the problem. Moreover, the linear dependencies is likely to introduce numerical problems in the optimization phase. Therefore, it is strongly recommended to build models without linear dependencies. If the linear dependencies have been removed at the modelling stage, then 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 problem is solved instead of the primal problem, then it is possible to recover the solution to the original primal problem. Therefore, if it is faster to solve the dual problem than the primal, then an optimizer can solve the dual problem instead of the 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 that try decides which of the two problems that is better to solve. Which form of the problem (primal or dual) that is solved can be seen in the MOSEK log. Note that the dualizer is transparent, all solution values returned by the optimizer refers to the original primal problem.

By default MOSEK chooses which form of the problem that should be solved. The dualizer can be controlled manually by setting the parameter:

- `MSK_IPAR_INTPNT_SOLVE_FORM`: In the case of the interior-point optimizer.
- `MSK_IPAR_SIM_SOLVE_FORM`: In the case the simplex optimizer.

Finally, observe that currently only linear problems may be dualized.

### 8.1.3 Scaling

Problems containing data with large or/and 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 make calculations the optimizer rely on inaccurate. Since computers work in finite precision extreme coefficients should be avoided. In general it is preferred to have data around the same "order of magnitude", 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 observe that the optimizer terminates when the stopping criteria is meet on the scaled problem, therefor it is possible that significant primal or dual infeasibilities occurs after unscaling for badly scaled problems. The best solution to this problem is to reformulate the problem such that it becomes better scaled.

MOSEK will by default heuristically choose 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 using the interior-point optimizer can take advantage multiple CPU's.

By default MOSEK use one thread to solve the problem. By changing the parameter

MSK_IPAR_INTPNT_NUM_THREADS

the number of threads (and thereby CPU's) used can be set. This should never be more than the number of CPU's on the machine.

The speed-up obtained when using multiple CPUs is highly problem and hardware dependent. It is therefore advisable that the user compare single threaded and multi threaded performance for the given problem type to determine the optimal settings.

For small problems, using multiple threads will most likely 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 optimizer for linear problems is a so-called interior-point variant. However, as an alternative the simplex optimizer can be employed.

We refer the curious reader to [23] 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 so-called homogeneous and self-dual algorithm. For a detailed description of the algorithm we refer the reader to [7].

#### 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 postprocessing step that computes an optimal basic solution starting from the optimal interior-point solution. A lot of information about the basis identification procedure can be located in [7].

Observe a basic solution is often more accurate than an interior-point solution.

By default MOSEK will perform a basic identification. However, if a basic solution is not needed, the basic identification procedure can be turned off. The parameters

- MSK_IPAR_INTPNT_BASIS,

| 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 criteria.

- MSK_IPAR_BI_IGNORE_MAX_ITER,
- and MSK_IPAR_BI_IGNORE_NUM_ERROR

controls when basis identification is performed.

#### 8.2.2.2 Interior-point termination criteria

The parameters in Table 8.1 controls 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 to solve an problem. Contrary to the interior-point optimizer the simplex optimizer can exploit a guess for the optimal solution to decrease the solution time. Depending on the problem it might 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 dual variant of the simplex optimizer more about this later.

#### 8.2.3.1 Simplex termination criteria

The simplex optimizer terminates when it finds a 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 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 (basis) solution and thereby reduce the solution time significantly. When a simplex optimizer starts form an existing solution we say that it performs a “hotstart”. If the user are solving a sequence of optimization problems by solving the problem, making modifications, and the solving again, MOSEK will automatically hotstart.

Setting the parameter MSK_IPAR_OPTIMIZER to MSK_OPTIMIZER_FREE_SIMPLEX instructs MOSEK to automatically select 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 will also use presolve when performing a hotstart. 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 carefully designed to minimize numerical difficulties, still it is possible the optimizer in rare cases will have a hard time solving a problem. MOSEK counts a numerical unexpected behaviour inside the optimizer as a "setback". The user can control how many setbacks the optimizer is allowed to have, and it will abort if this number is exceeded. Setbacks is implemented to avoid long sequences where the optimizer tries to recover from an unstable situation. But what counts as a setback ? It is hard to say without getting very technical but obvious cases could be 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 setbacks then the problem is usually badly scaled. In such a situation then try to reformulate into a more well scaled problem. If a lot of setbacks still occur, then trying one of more of the following suggestions might be worthwhile.

- Raise tolerances for allowed dual or primal feasibility: Hence, increase the value of
  - MSK_DPAR_BASIS_TOL_X
  - and MSK_DPAR_BASIS_TOL_S.
- Raise or lower pivot tolerance: Change the parameter MSK_DPAR_SIMPLEX_ABS_TOL_PIV.
- 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 hotstarts, it might in very rare case be more stable to switch off this feature controlled by MSK_IPAR_SIM_HOTSTART.
- Increase maximum setbacks 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 then which optimizer is the best. The primal simplex, the dual simplex or the interior-point optimizer? Unfortunately it is impossible to answer this question in general. However, the interior-point optimizer is the one of the three that behaves most predictably. The interior-point optimizer tends to use somewhere between 20 and 100 iterations almost independently of problem size, while the number of iterations used by the simplex optimizer is far more unpredictable. Therefore, the interior-point optimizer is the default optimizer.

On other hand the interior-point optimizer cannot exploit an existing guess for a solution. This implies that the interior-point optimizer must always start from scratch. On the other hand the simplex optimizer is excellent at exploiting an existing solution. Therefore, if hotstart is possible, then it is likely to be advantageous to use the simplex optimizer rather than the interior-point optimizer.

### 8.2.5 The primal or the dual simplex variant?

MOSEK provides both a primal and dual simplex optimizer. Predicting the fastest simplex optimizer (primal or dual) is simply impossible. However in the recent years the dual optimizer has made several algorithmic and computational improvements, which makes it on average faster than the primal simplex optimizer in our experience. But both optimizers have a significant number of "wins", depending on the problem structure and size.

Setting `MSK_IPAR_OPTIMIZER` to `MSK_OPTIMIZER_FREE_SIMPLEX` will instruct MOSEK to guess whether the primal or the dual simplex optimizer is the best one to employ.

To summarize if you want to know which optimizer that is the best for a given problem then you should try all the optimizers on the given problem structure if possible.

Or alternatively use the concurrent optimizer presented in Section 10.3.

## 8.3 Linear network optimization

### 8.3.1 Solving network flow problems

Linear optimization problems have the network flow structure specified in Section 7.2 can in most cases be solved extremely fast with a specialized version of the simplex method [2].

MOSEK includes a highly tuned network simplex implementation which often solves network problems one or two orders of magnitude faster than the standard standard simplex optimizers implemented in MOSEK.

The network flow optimizer in MOSEK is easy to use. Indeed just follow the procedure

- Input the network flow problem as an ordinary linear optimization problem.
- Optimize the problem using the parameter setting

  ```
  MSK_IPAR_SIM_NETWORK_DETECT 0
  ```

| 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 problems declared infeasible |
| MSK_DPAR_INTPNT_CO_TOL_MU_RED | Controls when the complementarity is reduced enough |

Table 8.2: Parameters employed in termination criteria.

```
MSK_IPAR_OPTIMIZER          MSK_OPTIMIZER_FREE_SIMPLEX
```

Given this parameter setting MOSEK will automatically discover the problem is a network flow problem and apply the specialized simplex optimizer.

### 8.3.2 Solving embedded network problems

Often problems consist of large parts of network structure plus some extra non network constraints or variables, such problems are said to have embedded network structure. It is possible to extract the network structure and solve this subproblem with a specialized network optimizer. The network solution can then be used as a hotstart to the original problem which in some cases may lead to improved solution times.

MOSEK is capable of finding and exploiting embedded network structure within the simplex optimizer. Now finding the largest possible embedded network structure in a problem is in general very difficult and therefore MOSEK employs an extraction heuristic which often finds large parts of network structure if present. If the embedded network consist of more than roughly $p$% of the total problem then the special network optimizer is employed. The number $p$ can be specified using the parameter MSK_IPAR_SIM_NETWORK_DETECT. found network is larger than X then MOSEK will solve the embedded network problem with the network optimizer and hotstart the standard simplex. In general it is only recommended to use the network optimizer on problems where the embedded network part is substantial.

## 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 we refer the reader to [6].

#### 8.4.1.1 Interior-point termination criteria

The parameters that controls when the conic interior-point optimizer terminates is shown in Table 8.2.

| 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 problemis declared infeasible |
| MSK_DPAR_INTPNT_NL_TOL_MU_RED | Controls when the complementarity is reduced enough |

Table 8.3: Parameters employed in termination criteria.

## 8.5 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 so-called homogeneous and self-dual algorithm. For a detailed description of the algorithm we refer the reader to [8, 9].

#### 8.5.1.1 Interior-point termination criteria

The parameters that controls when the conic interior-point optimizer terminates is shown in Table 8.3.

# 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 having conic constraints is not allowed to have quadratic objective or constraints.

Readers unfamiliar with integer optimization are strongly recommended to consult relevant literature. The book [26] by Wolsey is a good introduction to integer optimization.

## 9.1 Some notation

In general an integer optimization problem have the form

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

where $\mathcal{J}$ is an index set specifying which variables that 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 **extremely** important observation because assume that it is not possible to solve the mixed integer optimization problem within a reasonable timeframe but a feasible solution can be located. Then a natural question is how far is the feasible solution from the optimal solution. The answer is obvious because there can be no feasible solution that has an objective value better than $\underline{z}$. This implies $\overline{z} - \underline{z}$ is a conservative measure for how far the feasible solution is from the optimal solution.

## 9.2 An important fact about integer optimization problems

An important fact to understand is that the time taken to solve an integer optimization problem may in the worst case grows exponentially with the size of the problem. For instance assume a problem contains $n$ binary variables, then the time taken to solve the problem 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 at computing near optimal solution quickly rather than at locating an optimal solution. Of course if an optimal solution can be located we are delighted but in general we cannot expect to find the 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 a heuristic 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 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 parameter MSK_IPAR_MIO_PRESOLVE_USE.

### 9.3.2 Heuristic

The integer optimizer will initially try to guess a good feasible solution using different heuristics:

- First a very simple heuristic i.e. a rounding heuristic is employed.
- Next, if deemed worthwhile, the so-called feasibility pump heuristic is used.
- Finally, if the two previous stages did not produce a good initial solution, a more sophisticated heuristic is used.

The following parameters can be used to control the effort the integer optimizer spends on finding an initial feasible solution.

- MSK_IPAR_MIO_HEURISTIC_LEVEL: Controls how sophisticated (and computationally expensive) heuristic to employ.
- MSK_DPAR_MIO_HEURISTIC_TIME: The minimum amount of time to be used 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 criteria

In general it is impossible to find an exact feasible and optimal solution to an integer optimization problem in a reasonable amount of time. (In many practical cases it might be though.) Therefore, the integer optimizer employs a relaxed feasibility and optimality criteria 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 criteria

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

is satisfied. Hence, such a solution is defined to be a feasible solution to (9.1).

Whenever the integer optimizer locates an integer feasible solution then it will check if the criteria

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

| Tolerance | Parameter name |
|---|---|
| $\delta_1$ | MSK_DPAR_MIO_TOL_ABS_RELAX_INT |
| $\delta_2$ | MSK_DPAR_MIO_TOL_REL_RELAX_INT |
| $\delta_3$ | MSK_DPAR_MIO_TOL_ABS_GAP |
| $\delta_4$ | MSK_DPAR_MIO_TOL_REL_GAP |
| $\delta_5$ | MSK_DPAR_MIO_NEAR_TOL_ABS_GAP |
| $\delta_6$ | MSK_DPAR_MIO_NEAR_TOL_REL_GAP |

Table 9.1: Integer optimizer tolerances.

| Parameter name | Delayed | Explanation |
|---|---|---|
| MSK_IPAR_MIO_MAX_NUM_BRANCHES | Yes | Limits the maximum number branches allowed. |
| MSK_IPAR_MIO_MAX_NUM_RELAXS | Yes | Limits the maximum number relaxations allowed. |

Table 9.2: Parameters affecting the termination of the integer optimizer.

is satisfied. If that is the case, then integer optimizer terminates and reports the integer feasible solution as an optimal solution. Note $\underline{z}$ 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 $\delta$ tolerances can be specified using parameters, see Table 9.1. If an optimal solution cannot be located within a reasonable time, it may be advantageous to use a relaxed termination criteria after some time. this is possible in the integer optimizer. Whenever the integer optimizer locates an integer feasible solution and has spend at least MSK_DPAR_MIO_DISABLE_TERM_TIME seconds on solving the problem, it will check if the criteria

$$\overline{z} - \underline{z} \leq \max(\delta_5, \delta_6 \max(1, |\overline{z}|))$$

is satisfied. If it is satisfied,the optimizer will report that the candidate solution is **near optimal** and terminate. All $\delta$ tolerances can be adjusted using suitable parameters, see Table 9.1. In Table 9.2 some other parameters affecting the termination of the integer optimizer is shown. Note if the effect of a parameter is delayed then associated termination criteria is first applied after some time specified by the parameter MSK_DPAR_MIO_DISABLE_TERM_TIME.

## 9.5 How to speedup the solution process

As previously mentioned, 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 stopping criteria: In the case the run time is not acceptable, then the first thing to do is to relax the stopping criteria, see Section 9.4 for details.

- Specify a good initial solution: In many cases a good feasible solution is either known or can easily be computed using problem specific knowledge. If a good feasible solution is known, then it is almost always worthwhile to inform the integer optimizer about such a solution.
- Improve the formulation: A mixed integer optimization problem can be impossible to solve in one formulation and quite easy in another formulation. However, it is beyond the scope of this manual to discuss good formulations for mixed integer problems. Therefore, we refer the reader to [26].

# Chapter 10

# Solving problems in parallel

If a computer has multiple CPUs (or a CPU with multiple cores), then it might be advantageous to use the multiple CPUs to solve the optimization problem. For instance if you have two CPUs you may want to exploit the two CPUs to solve the problem in the half time. MOSEK can exploit multiple CPUs.

## 10.1 Thread safety

The MOSEK API is thread safe provided that a task is only modified from one thread at any given point in time. Sharing an environment between threads is safe.

## 10.2 The parallelized interior-point optimizer

The interior-point optimizer has been parallelized. This implies that whenever the interior-point optimizer should solve an optimization problem, then it will try to divide the work so each CPU gets a share of the work. The user decides how many CPUs MOSEK should exploit. Unfortunately, it is not always easy to divide the work and some of the coordination work must occur in sequential. 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, then it is not advantageous of to use the parallel option.

The parameter `MSK_IPAR_INTPNT_NUM_THREADS` sets the number of threads (and therefore the number of CPU's) that the interior-point optimizer will use.

| 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 10.1: Default priorities for optimizer selection in concurrent optimization.

## 10.3 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 the interior-point and the dual simplex optimizers may be applied to an linear optimization problem concurrently. The concurrent optimizer terminates when the first optimizer has completed and reports the solution of the fastest optimizer. That way a new optimizer has been created which essentially has the best performance of the interior-point and the dual simplex optimizer.

Hence, the concurrent optimizer is the best one to use if multiple optimizers are available for the problem and you cannot say beforehand which one is the best one. Note that any solution present in the task will also be used for hotstarting the simplex algorithms. One possible scenario would therefore be running a hotstart dual simplex optimizer in parallel with the interior-point optimizer.

MOSEK provides a special optimizer called the concurrent optimizer which makes it possible to apply several optimizers to an optimization concurrently. By setting the parameter

```
MSK_IPAR_OPTIMIZER
```

to

```
MSK_OPTIMIZER_CONCURRENT
```

the concurrent optimizer is used.

The number optimizers used in parallel is determined by the parameter

```
MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS.
```

Moreover, the optimizers are selected according to a preassigned priority with optimizers having the highest priority been selected first. The default priority for each optimizer is shown in Table 10.3.

### 10.3.1 An example

As an example the setting

```
MSK_IPAR_OPTIMIZER MSK_OPTIMIZER_CONCURRENT
MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS  2
```

implies that the interior-point and the free simplex optimizers is both applied to the optimization problem concurrently.

#### 10.3.1.1 Using the command line tool

The command line

```
  mosek afiro.mps  -d MSK_IPAR_OPTIMIZER MSK_OPTIMIZER_CONCURRENT \
       -d MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS 2
```

produces the following (edited) output:

```
...

Number of concurrent optimizers          : 2
Optimizer selected for thread number 0   : interior-point (threads = 1)
Optimizer selected for thread number 1   : free simplex
Total number of threads required         : 2

...

Thread number 1 (free simplex) terminated first.

...

Concurrent optimizer terminated. CPU Time: 0.03. Real Time: 0.00.
```

As seen from the log information the interior-point and the free simplex optimizers are employed concurrently. However, only the output from the optimizer having the highest priority is printed to the screen. In the example this is the interior point optimizer.

The line

```
Total number of threads required         : 2
```

indicates the number of threads used. For concurrent optimization to be effective, this should be lower than the number of CPUs.

In the above example the simplex optimizer finishes first as indicated in the log information.

#### 10.3.1.2 From the API

The following example shows how to call the concurrent optimizer from the API.

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

```
   File:      concurrent1.c

   Purpose:   Demonstrates 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;

  /* Make 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 */
```

### 10.3.2 A more flexible concurrent optimizer

MOSEK also provides a more flexible method of concurrent optimization by using the function MSK_optimizeconcurrent. The main advantage of this function is that it allows the calling application to assign arbitrary values to the parameters of each tasks and that a callback functions can be attached to each task. This may be useful in following situation. Assume you know the primal simplex optimizer is the best optimizer for your problem but you do not know which of the available incoming selection strategy is the best. In such case you can solve the problem concurrently with the primal simplex optimizer but using different selection strategies.

The function MSK_optimizeconcurrent is documented in the API reference and in the example below.

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

  File:      concurrent2.c

  Purpose:   Demonstrates 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];

  /* Make 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.
     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 diffrent 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 callback 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 11

# Analyzing infeasible problems

When creating a new optimization model the first attempt is often (primal) infeasible. This is caused by specifying inconsistent constraints. A model might also be unbounded which usually is caused by having left out important constraints of the problem.

The purpose of the present chapter is to

- Present the relevant theory about infeasible problems.
- Discuss how to find the cause of the infeasibility with the use of MOSEK infeasibility report. MOSEK infeasibility report is a tool for locating a smaller subset of constraints that are still infeasible (Section 11.3).

MOSEK can also find the wighted sum of infeasibility. For information on this see chapter 12.

## 11.1 A motivating example

We go through an example and discuss some procedures for diagnosing the cause of the infeasible status of an optimization problem. As the example we will use a simple transportation problem.

Consider the problem of minimizing the cost of transportation between a number of production plant 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 is given in Figure 11.1. It easy to see that problem represented in Figure 11.1 is infeasible, since the total demand

$$2300 = 1100 + 200 + 500 + 500. \tag{11.1}$$

exceeds the total supply

$$2200 = 200 + 1000 + 1000 \tag{11.2}$$

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

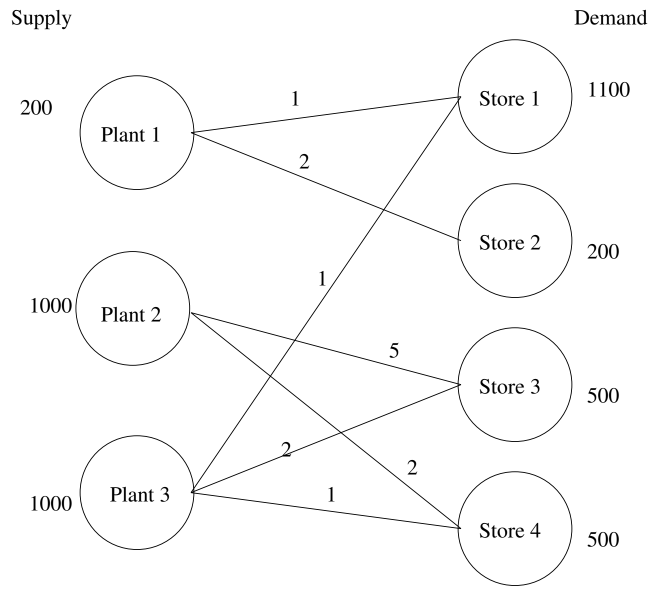


Figure 11.1: Supply, demand and cost of transportation.

formulated as the LP:

$$
\begin{array}{lcccccccccccccccl}
\text{minimize} & x_{11} & + & 2x_{12} & + & 5x_{23} & + & 2x_{24} & + & x_{31} & + & 2x_{33} & + & x_{34} & & \\
\text{subject to} & x_{11} & + & x_{12} & & & & & & & & & & & \leq & 200, \\
 & & & & & x_{23} & + & x_{24} & & & & & & & \leq & 1000, \\
 & & & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\
 & x_{11} & & & & & & & + & x_{31} & & & & & = & 1100, \\
 & & & x_{12} & & & & & & & & & & & = & 200, \\
 & & & & & x_{23} & + & & & & & x_{33} & & & = & 500, \\
 & & & & & & & x_{24} & + & & & & & x_{34} & = & 500, \\
 & x_{ij} \geq 0. & & & & & & & & & & & & & &
\end{array}
\tag{11.3}
$$

Solving the problem (11.3) using MOSEK will result in a solution, a solution status and a problem status. Among the log output from running 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 has been found. The certificate is returned in place of the solution to the problem.

## 11.2 Locating the cause of the infeasibility

Usually an infeasible problem status is caused by a mistake 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 the 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. Furthermore, after removing the objective, the problem is guaranteed to be bounded.
- 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 infeasible after the above steps, then some of the constraints must be relaxed or even removed completely. If the relaxation results in a feasible problem, then this provide a hint about what is causing the infeasibility. For example if removing the bounds on a variable results in feasibility, the problem might be incorrect bounds on that variable or a constraint containing that variable. The MOSEK infeasibility report (Section 11.3) may be of assistance to you in finding the constraints that cause the infeasibility.

Possible ways of relaxing your problem includes:

- 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

$$x_{12} = 200 \tag{11.4}$$

makes the problem feasible.

This constraint models the demand of the second store. Examining the problem, we discover that `store 2` can only get its supply satisfied by receiving 200 goods from `plant 1`. Since `plant 1` now has used its supply, `store 1` can only get goods from `plant 3`. But `plant 3` only has a supply of 1000 and `store 1` needs 1100. This explains the cause of the infeasibility.

Lowering the demand of store 2 from 200 to 100 makes the problem feasible.

### 11.2.1 A warning about what is possible

The problem

$$\begin{array}{lll} \text{minimize} & 0 & \\ \text{subject to} & 0 \leq x_1, & \\ & x_j \leq x_{j+1}, & j = 1, \ldots, n-1, \\ & x_n \leq -1 & \end{array} \tag{11.5}$$

is clearly infeasible. Moreover, if any one of the constraints are dropped, then the problem becomes feasible.

Therefore, in the worst case many constraints can be involved in an infeasibility. Hence, it is not always easy or possible to pinpoint a few constraints which is causing the infeasibility.

## 11.3 The infeasibility report

MOSEK has some facilities for diagnosing the cause of a primal or dual infeasibility. They can be turned on using the parameter setting

`MSK_IPAR_INFEAS_REPORT_AUTO MSK_ON`

This causes MOSEK to print a report about an infeasible subset of the constraints, when an infeasibility is encountered. Moreover, the parameter

`MSK_IPAR_INFEAS_REPORT_LEVEL`

controls the amount info presented in the infeasibility report. The default value is 1.

### 11.3.1 Examples

#### 11.3.1.1 The case of a primal infeasibility

We will reuse the example (11.3) which can be represented in

```
\
\ 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
```

which indicates which constraints and bounds that are important for the infeasibility i.e. causing the infeasibility.

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, because if the dual lower value is different from zero for a constraint, then it implies that the lower bound on the constraint is important for the infeasibility. Similarly, if the dual upper value is different from zero on a constraint, then this implies the upper bound on the constraint is important for infeasibility.

It also possible to obtain the infeasible subproblem. The command line

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

will produce the files `rinfeas.bas.inf.lp` and `rinfeas.itr.inf.lp`. In this case the file `rinfeas.-bas.inf.lp` has the content

```
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. Observe this optimization problem is identical to (11.3), except the objective and some of the constraints and bounds has been removed. Nevertheless the command line

```
mosek -d MSK_IPAR_INFEAS_REPORT_AUTO MSK_ON rinfeas.bas.inf.lp
```

demonstrates the reduced problem is **primal infeasible**. However, it should be much easier to locate the cause of the infeasibility in the reduced problem, rather in the original problem (11.3) because it is smaller.

By inspection of the infeasible subproblem it can be observed that $x12 = 200$ is implied by the constraints and hence $x11 = 0$. This implies that $x31 = 1.1e3$ which cannot be satisfied due to the constraint ``s2'' and $x31, x33 \geq 0$.

#### 11.3.1.2 The case of a 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 dual infeasibility certificate. In the case of the example produces the infeasibility report(slightly edited):

```
MOSEK DUAL INFEASIBILITY REPORT.

Problem status: The problem is dual infeasible

The following constraints are involved in the infeasibility.

Index    Name             Activity         Objective        Lower bound      Upper bound
5        x33              -1.000000e+000                    NONE             2.000000e+000
6        x44              -1.000000e+000                    NONE             1.000000e+000

The following variables are involved in the infeasibility.

Index    Name             Activity         Objective        Lower bound      Upper bound
0        y1               -1.000000e+000   -2.000000e+002   NONE             0.000000e+000
2        y3               -1.000000e+000   -1.000000e+003   NONE             0.000000e+000
3        y4               1.000000e+000    -1.100000e+003   NONE             NONE
4        y5               1.000000e+000    -2.000000e+002   NONE             NONE
Interior-point solution
Problem status  : DUAL_INFEASIBLE
Solution status : DUAL_INFEASIBLE_CER
Primal - objective: -1.0000000000e+002 eq. infeas.: 0.00e+000 max bound infeas.: 0.00e+000
                                       cone infeas.: 0.00e+000
Dual   - objective: 0.0000000000e+000  eq. infeas.: 0.00e+000 max bound infeas.: 0.00e+000
                                       cone infeas.: 0.00e+000

Basic solution
Problem status  : DUAL_INFEASIBLE
Solution status : DUAL_INFEASIBLE_CER
Primal - objective: -1.0000000000e+002 eq. infeas.: 0.00e+000 max bound infeas.: 0.00e+000
Dual   - objective: 0.0000000000e+000  eq. infeas.: 0.00e+000 max bound infeas.: 0.00e+000
```

Comments:

- MOSEK states the problem is dual infeasible. Moreover, MOSEK states that the solution is a dual infeasibility certificate. This implies the primal objective value

$$c^T x^* \tag{11.6}$$

(see (11.54)) should be positive[1] and the primal infeasibility measures should be approximately equal to zero. Due to this is the case MOSEK has computed a correct infeasibility certificate[2].

- It can be seen that the variables `y1`, `y3`, `y4`, and `y5` are involved in the dual infeasibility because those variables has a nonnegative activity. I.e. the values of the variables are reported in the "`Activity`" column.

- Due to the variables `y1` and `y3` has negative activity then adding a lower bound to those variable or increasing their the objective coefficient might help because this invalidates the infeasibility certificate. Note that

$$\begin{aligned} c^T x^* &= -1*(-200) - 1*(-1000) + 1*(-1100) + 1*(-200) \\ &= -100 \end{aligned} \tag{11.7}$$

  where $x^*$ is the dual infeasibility certificate. Therefore, one possibility to repair the dual infeasible status is to change the objective so $c^T x^*$ becomes nonnegative. For instance changing $c_{y1}$ from -200 to -100. This will in fact resolve the infeasibility in this case.

  Alternatively due to the variables `y1` and `y3` has a positive activity then adding an upper bound to those variable or decreasing their their objective coefficient for those variables might help.

- In addition it can be seen that only the constraints `x33` and `x44` has a nonzero activity. So it is only worthwhile to change the bounds for these constraints. Once again it holds that if the activity is negative then a lower bound on the constraint should introduced whereas if the activity is positive then an upper bound is introduced.

In summary we have seen how the dual infeasibility certificate provides some hints to how to repair the dual infeasibility.

## 11.4 Theory concerning infeasible problems

The following chapter discuss the theory of infeasibility certificates and how MOSEK use the certificate to pinpoint the cause of infeasibility in the infeasibility report (11.3).

In general a constrained optimization problem such as

$$\begin{array}{lrcl} \text{minimize} & c^T x & & \\ \text{subject to} & Ax & = & b, \\ & x \geq 0 & & \end{array} \tag{11.8}$$

is said to be *primal feasible* if there exist a solution $x$ that satisfies all the constraints. On the other hand if no such solution exists i.e.

$$\{x : \ Ax = b, \ x \geq 0\} = \emptyset \tag{11.9}$$

then the problem is said to be *primal infeasible*.

[1] The objective value should be negative if the problem is maximized.

[2] If this is not the case you should contact MOSEK support.

Given a problem is feasible then it is said to be *unbounded* if the optimal objective value is not finite. For instance the problem

$$\begin{array}{lccl} \text{minimize} & -x_2 & & \\ \text{subject to} & x_1 - x_2 & = & 0, \\ & x_1, x_2 \geq 0 & & \end{array} \tag{11.10}$$

is unbounded.

Linear, conic, and convex optimization problems have an associated dual problem. For instance the dual problem corresponding to (11.8) is

$$\begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & A^T y + s = c \\ & s \geq 0 \end{array} \tag{11.11}$$

An optimization problem is said to be *dual feasible* if there exists solution that satisfies all the constraints of the dual problem. If no such solution exists, then the problem is said to be *dual infeasible*. The problem

$$\begin{array}{lccl} \text{minimize} & x_1 - x_2 & & \\ \text{subject to} & x_1 & = & -1, \\ & x_1, x_2 \geq 0 & & \end{array} \tag{11.12}$$

has the dual problem

$$\begin{array}{lccl} \text{maximize} & -y_1 & & \\ \text{subject to} & y_1 + s_1 & = & 1, \\ & s_2 & = & -1, \\ & s_1, s_2 \geq 0 & & \end{array} \tag{11.13}$$

which obviously is infeasible because $s_2 = -1$ and $s_2 \geq 0$ are contradictory. Hence, (11.12) is dual infeasible.

The problem

$$\begin{array}{lccl} \text{minimize} & -x_2 & & \\ \text{subject to} & x_1 - x_2 & = & 0, \\ & x_1, x_2 \geq 0 & & \end{array} \tag{11.14}$$

is unbounded. Unbounded problems are always dual infeasible, but, dual infeasible problems are not always unbounded, as the example (11.12) shows.

If a problem is primal (dual) infeasible, there is not meaningfull primal (dual) solution. Instead, if a problem is found to be infeasible, MOSEK will report a certificate of infeasibility (a proof that the problem is infeasible).

### 11.4.1 Primal infeasibility certificate

Let the problem

$$\begin{array}{lccl} \text{minimize} & c^T x & & \\ \text{subject to} & Ax & = & b, \\ & x \geq 0 & & \end{array} \tag{11.15}$$

be given where $A \in R^{m \times n}$.

The well known Farkas Lemma states that (11.15) is primal infeasible if and only if

$$\exists y : A^T y \leq 0,\ b^T y > 0. \tag{11.16}$$

Therefore, any $y$ satisfying (11.16) is said to be a *certificate of the primal infeasible* status of (11.15). Note that given a particular $y^*$ it is easy to verify it is a certificate of the infeasible status by verifying

$$A^T y^* \leq 0 \text{ and } b^T y^* > 0 \tag{11.17}$$

is satisfied.

Let us try to demonstrate that $y^*$ is in fact a certificate of the infeasibility. First assume on the contrary that (11.15) has a solution $x^*$ which implies

$$(y^*)^T A x^* = (y^*)^T b \tag{11.18}$$

and hence

$$\begin{array}{rcl} 0 & \geq & \sum\limits_{j=1}^{n} \left( \sum\limits_{i=1}^{m} a_{ij} y_i^* \right) x_j^* \\ & = & \sum\limits_{i=1}^{m} \left( \sum\limits_{j=1}^{n} a_{ij} x_j^* \right) y_i^* \\ & = & \sum\limits_{i=1}^{m} b_i y_i^* \\ & = & (y^*)^T b \\ & > & 0 \end{array} \tag{11.19}$$

which is a contradiction. The first inequality follows from the assumption:

$$x_j^* \geq 0 \text{ and } \sum_{i=1}^{m} a_{ij} y_i^* \leq 0. \tag{11.20}$$

Therefore, we can conclude that (11.15) can **not** have a feasible solution, if an infeasibility certificate exists.

An alternative way of stating Farkas Lemma is

$$\exists y : A^T y + s = 0,\ s \geq 0,\ b^T y > 0. \tag{11.21}$$

which is same as saying that the problem

$$\begin{array}{lrcl} \text{maximize} & b^T y & & \\ \text{subject to} & A^T y + s & = & 0, \\ & s \geq 0 & & \end{array} \tag{11.22}$$

is unbounded. The problem (11.22) is almost equivalent to the problem of (11.15) except the $c$ has been replaced by 0 in the right hand side. This has the implication that a primal infeasibility certificate can be reported in the dual solution of an optimization problem. This is exactly what MOSEK does when a problem is primal infeasible.

Next we will show that an infeasibility certificate provides important information about which constraints causes the infeasibility.

Let

$$\mathcal{I} := \{i : \ y_i^* \neq 0\} \text{ and } \mathcal{J} := \{j : \ \sum_{i=1}^{m} a_{ij} y_i^* \neq 0\} \tag{11.23}$$

and define the relaxed problem

$$\begin{array}{lrcll} \text{minimize} & c^T x & & & \\ \text{subject to} & \sum\limits_{j=1}^{n} a_{ij} x_j & = & b_i, & \forall i \in \mathcal{I}, \\ & x_j \geq 0, & & & \forall j \in \mathcal{J}. \end{array} \tag{11.24}$$

Obviously if the relaxation (11.24) of (11.15) is infeasible, then (11.15) is also infeasible. Now assume (11.24) has the feasible solution $x^+$ then

$$\begin{array}{rcl} 0 & \geq & \sum\limits_{j=1}^{n} \left( \sum\limits_{i=1}^{m} a_{ij} y_i^* \right) x_j^+ \\ & = & \sum\limits_{i \in \mathcal{I}}^{m} \left( \sum\limits_{j \in \mathcal{J}} a_{ij} x_j^+ \right) y_i^* \\ & = & (y^*)^T b \\ & > & 0. \end{array} \tag{11.25}$$

which is a contradiction. Hence, a feasible solution $x^+$ cannot exists.

To summarize we have shown that the infeasibility certificate can be used to create a relaxation of the original problem which is infeasible. The relaxation is identical to the original problem except that (hopefully) many constraints has been dropped. As long as the relaxation is infeasible, then the original problem must also be infeasible. Therefore, the infeasibility in the relaxation should be repaired first and this should be easier if the relaxation does not contain too many constraints. Or to state it differently, the two index set $\mathcal{I}$ and $\mathcal{J}$, computed based on an infeasibility certificate can be used to pinpoint a set of constraints that is causing an infeasibility.

#### 11.4.1.1 An example

The problem

$$\begin{array}{lrcl} \text{minimize} & 0.5x_1 - x_2 & & \\ \text{subject to} & x_1 - x_2 - x_3 & = & 1, \\ & x_1, x_2, x_3 \geq 0 & & \end{array} \tag{11.26}$$

has the dual problem

$$\begin{array}{lrcl} \text{maximize} & y_1 & & \\ \text{subject to} & y_1 + s_1 & = & 0.5, \\ & -y_1 + s_2 & = & -1, \\ & -y_1 + s_3 & = & 0, \\ & s_1, s_2, s_3 \geq 0 & & \end{array} \tag{11.27}$$

The dual problem is equivalent to

$$\begin{array}{lrcl} \text{maximize} & y_1 & & \\ \text{subject to} & y_1 & \leq & 0.5, \\ & y_1 & \geq & 1, \\ & y_1 & \geq & 0 \end{array} \tag{11.28}$$

which clearly is infeasible. This is also confirmed by that

$$x_1^* = 1,\ x_2^* = 1 \text{ and } x_3^* = 0 \tag{11.29}$$

is certificate of dual infeasibility. Recall we can use the infeasibility certificate to state a reduced (relaxed) dual problem which is also infeasible. In this case the reduced problem only contains the

$$\begin{array}{lccl} \text{minimize} & 0.5x_1 - x_2 & & \\ \text{subject to} & x_1 - x_2 & = & 1, \\ & x_1, x_2 \geq 0 & & \end{array} \tag{11.30}$$

which has the corresponding dual problem

$$\begin{array}{lccl} \text{maximize} & y_1 & & \\ \text{subject to} & y_1 + s_1 & = & 0.5, \\ & -y_1 + s_2 & = & -1, \\ & s_1, s_2 \geq 0. & & \end{array} \tag{11.31}$$

Since $x_3^* = 0$, variable $x_3$ is not included in the reduced dual problem.

(11.31) is a relaxation of (11.27) and is infeasible too. We can therefore conclude that variables $x_1$ and $x_2$ is causing the dual infeasibility. Hence, we should for instance add a constraint to the problem that bounds $x_1$ and $x_2$ or perhaps change the objective. For instance any $c_1 > 1$ will make the problem dual feasible.

#### 11.4.1.2 The case of general bounds

In general MOSEK solves the problem

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x \end{array} \tag{11.32}$$

where the corresponding dual problem is

$$\begin{array}{lccl} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & \end{array} \tag{11.33}$$

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

$$l_j^x = -\infty \ \Rightarrow \ (s_l^x)_j = 0 \text{ and } l_j^x (s_l^x)_j = 0. \tag{11.34}$$

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

In this case an infeasibility certificate is *any* solution proving that the problem

$$
\begin{array}{lccl}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\
 & +(l^x)^T s_l^x - (u^x)^T s_u^x & & \\
\text{subject to} & A^T y + s_l^x - s_u^x & = & 0, \\
 & -y + s_l^c - s_u^c & = & 0, \\
 & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & &
\end{array} \tag{11.35}
$$

is unbounded, i.e. any feasible solution that has a positive objective value.

Let $(s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*})$ be a dual infeasibility certificate then

$$
(s_l^{c*})_i > 0 \quad ((s_u^{c*})_i > 0) \tag{11.36}
$$

implies that the lower (upper) bound on the $i$th constraint is important for the infeasibility. Furthermore,

$$
(s_l^{x*})_j > 0 \quad ((s_u^{x*})_i > 0) \tag{11.37}
$$

implies that the lower (upper) bound on the $j$th variable is important for the infeasibility.

### 11.4.2 Dual infeasibility certificate

As stated in Section 11.4, a necessary condition for a linear optimization problem to have an optimal solution is that the dual problem is feasible. Therefore, MOSEK might sometimes state that a problem is dual infeasible when it cannot solve the problem.

If you are not familiar with duality, then it might be beneficial to consult a basic text book on linear optimization.

In this Section we will discuss how MOSEK diagnoses dual infeasibility and the kind of information MOSEK reports in the dual infeasible case. To keep it as simple as possible then consider the problem

$$
\begin{array}{lccl}
\text{minimize} & c^T x & & \\
\text{subject to} & Ax & = & b, \\
 & x \geq 0 & &
\end{array} \tag{11.38}
$$

which has the dual problem

$$
\begin{array}{ll}
\text{maximize} & b^T y \\
\text{subject to} & A^T y + s = c, \\
 & s \geq 0.
\end{array} \tag{11.39}
$$

A necessary, but not sufficient condition, for (11.38) to have an optimal solution is that (11.39) is feasible. Therefore, if MOSEK is not capable of solving a linear optimization problem it might report the problem is dual infeasible i.e. (11.39) is infeasible. In such a case you will have to relax some of the constraints in the dual problem or equivalently modify variables in the primal problem for instance by adding bounds on variables.

Due to the fact that the dual problem (11.39) is a linear optimization problem, we could essentially stop the discussion here, since all the theory, ideas, and methods we have developed for the primal infeasible case can be applied to analyze the infeasibility in the dual problem. However, for the convenience of the reader we will discuss the dual infeasible case in some detail.

Similar to the primal infeasible case a *certificate of the dual infeasibility* exists, when a problem is dual infeasible. Indeed the problem (11.39) is infeasible if and only if

$$\exists x : \ Ax = 0, \ c^T x < 0, x \geq 0. \tag{11.40}$$

We say that any $x$ satisfying (11.40) is a certificate of dual infeasibility. Note the problem (11.40) is a homogenized version of (11.38) since $b$ has been replaced by the 0 vector.

It is easy to prove that any $x^*$ satisfying (11.40) is indeed a certificate of the dual infeasibility: Assume on the contrary that $y$ is a feasible solution to (11.39), then

$$\begin{array}{rcl} 0 & = & y^T(Ax^*) \\ & = & (x^*)^T(A^T y) \\ & = & (x^*)^T(c - s) \\ & = & c^T x^* - s^T x^* \\ & < & 0 \end{array} \tag{11.41}$$

which is a contradiction, implying that (11.39) is infeasible.

Observe that if (11.38) has a feasible solution $x^0$, and $x^*$ is a dual infeasibility certificate, then

$$A(x^0 + \alpha x^*) = b \text{ and } x^0 + \alpha x^* \geq 0 \tag{11.42}$$

for all $\alpha \geq 0$. Moreover,

$$\lim_{\alpha \to \infty} c^T(x^0 + \alpha x^*) = -\infty. \tag{11.43}$$

This shows that if a problem is both dual infeasible and primal feasible, then it must be unbounded. Moreover, the dual infeasibility certificate is a feasible direction along which the objective value tends to minus infinity. Hence, the primal problem is unbounded.

The converse is also true i.e. if a linear optimization problem is unbounded, then it is also dual infeasible.

Whenever MOSEK states a linear optimization is dual infeasible MOSEK returns a dual infeasibility certificate, which as demonstrated above proves the dual problem is infeasible.

In addition to proving the primal problem cannot have an optimal solution the dual infeasibility certificate provides information about which dual constraints are causing the infeasibility. As there is a one-to-one correspondence between primal variables and dual constraints, the dual infeasibility certificate provides information about which variables are causing the dual infeasibility. This can be seen as follows: Assume $x^*$ is a dual infeasibility certificate, and define the set

$$\mathcal{J} := \{j : \ x_j^* > 0\}, \tag{11.44}$$

and the reduced dual problem

$$\begin{array}{lll} \text{maximize} & b^T y & \\ \text{subject to} & A_{:j}^T y + s_j = c_j, & \forall j \in \mathcal{J}, \\ & s_j \geq 0 & \forall j \in \mathcal{J}. \end{array} \tag{11.45}$$

Clearly, the problem (11.45) is equivalent to problem (11.39), except that some of the constraints have been removed. Therefore, (11.40) is a relaxation of (11.39). This implies if (11.45) is infeasible , then

(11.39) is also infeasible. The fact that the reduced problem (11.45) is infeasible follows from

$$\begin{array}{rcl} 0 & > & c^T x^* \\ & = & \sum_{j \in \mathcal{J}} c_j x_j^*, \\ Ax^* & = & \sum_{j \in \mathcal{J}} A_{:j} x_j^*, \end{array} \tag{11.46}$$

and

$$x_j^* \geq 0, \ \forall j \in \mathcal{J} \tag{11.47}$$

because it shows that $x_{\mathcal{J}}^*$ is a certificate of dual infeasibility of problem (11.45).

Problem (11.45) (hopefully) contains fewer constraints than (11.39), and it should be easier to locate the cause of infeasibility when inspecting (11.45) rather than the full problem (11.39).

#### 11.4.2.1 The case of general bounds

In general MOSEK solves the problem

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x \end{array} \tag{11.48}$$

where the corresponding dual problem is

$$\begin{array}{llcl} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & \end{array} \tag{11.49}$$

Here we use the convention that if a bound is infinite then the corresponding dual variable $s$ is zero and the product of the bound and the $s$ variable is always zero. For example

$$l_j^x = -\infty \ \Rightarrow \ (s_l^x)_j = 0 \text{ and } l_j^x (s_l^x)_j = 0. \tag{11.50}$$

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

In this case a dual infeasibility certificate is any solution proving that the problem

$$\begin{array}{lccccc} \text{minimize} & & & c^T x & & \\ \text{subject to} & \bar{l}^c & \leq & Ax & \leq & \bar{u}^c, \\ & \bar{l}^x & \leq & x & \leq & \bar{u}^x \end{array} \tag{11.51}$$

is unbounded, where we use the definitions

$$\bar{l}_i^c := \left\{ \begin{array}{ll} 0, & l_i^c > -\infty, \\ -\infty, & \text{otherwise}, \end{array} \right. \quad \bar{u}_i^c := \left\{ \begin{array}{ll} 0, & u_i^c < \infty, \\ \infty, & \text{otherwise}, \end{array} \right. \tag{11.52}$$

and

$$\bar{l}_i^x := \left\{ \begin{array}{ll} 0, & l_i^x > -\infty, \\ -\infty, & \text{otherwise}, \end{array} \right. \quad \text{and } \bar{u}_i^x := \left\{ \begin{array}{ll} 0, & u_i^x < \infty, \\ \infty, & \text{otherwise}. \end{array} \right. \tag{11.53}$$

Therefore, (11.51) is the homogenized dual problem i.e. the constant in the objective and all finite bounds have been replaced by 0. Hence, a dual infeasibility certificate is any $x^*$ such that

$$\begin{array}{ccccc} & & c^T x^* & < & 0, \\ \bar{l}^c & \leq & Ax^* & \leq & \bar{u}^c, \\ \bar{l}^x & \leq & x^* & \leq & \bar{u}^x \end{array} \tag{11.54}$$

Let $x^*$ be any dual infeasibility certificate then it is easy to see if $x_j^* \neq 0$ then

$$-\infty < l_j^x \tag{11.55}$$

and

$$u_j^x < \infty \tag{11.56}$$

cannot both be the case. This is a mathematical statement of the fact that variable that has both a finite lower and upper bound cannot cause dual infeasibility.

Finally, we can state the important observation if $x^*$ is dual infeasibility certificate then for any $j$ such that

$$x_j^* \neq 0 \tag{11.57}$$

variable $j$ is important for the dual infeasibility.

#### 11.4.2.2 Rules of thumb for repairing a dual infeasible model

In general it is impossible for MOSEK to say precisely how a dual infeasible model should be repaired. For instance it can be repaired by changing the objective or the bounds or by adding new constraints.

However, some ideas for how to repair a dual infeasible model can be deduced from a dual infeasibility certificate. Indeed a dual infeasibility certificate must satisfy the conditions (11.54). Therefore, one way to repair a dual infeasible model is to change the model such that $x^*$ no long is a valid infeasibility certificate.

For instance the objective coefficients $c$ can be changed so

$$c^T x^* \geq 0 \tag{11.58}$$

is the case. Hence, the dual infeasibility certificate will no longer satisfy condition (11.54). It easy to see it is not worthwhile to change those $c_j$'s for for which $x_j^* = 0$. Moreover, if $x_j^* < 0$ then $c_j$ should be decreased and if $x_j^* > 0$, then $c_j$ should be increased.

Rather than changing the objective $c$ then changing the bounds might resolve the infeasibility. In general it is not worthwhile to change only the value of the finite bounds and therefore at least one infinite bound must be made finite. (This observation follows from that only the finiteness of the bounds plays a role in (11.54) and not their actual values.) It easy to see that if $x_j^* < 0$, then $l_j = \infty$ must be the case. Therefore on such variable it might be a good to introduce a finite lower bound. Similarly, if $x_j^* > 0$, then introducing a finite upper bound on $x_j$ might help removing the infeasibility.

In summary by inspection of the infeasibility certificate and (11.54) some rules of thumb can be developed that helps resolving the dual infeasibility.

# Chapter 12

# Feasibility repair

## 12.1 The primal case

In Chapter 11.2 it is discussed 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 constraints 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 optimisation. MOSEK can also repair infeasibilitys in quadratic and conic optimization problems possibly having integer constrained variables. Infeasibilitys in general nonlinear optimization problems can't be repaired using the method described below.

### 12.1.1 The main idea

Consider the linear optimization problem with $m$ constraints and $n$ variables

$$\begin{array}{lccccl} \mbox{minimize} & & & c^T x + c^f & & \\ \mbox{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \tag{12.1}$$

which we will assume is infeasible. We will assume that

$$(l^c)_i \leq (u^c)_i, \ \forall i \tag{12.2}$$

and

$$(l^x)_j \leq (u^x)_j, \ \forall j \tag{12.3}$$

because otherwise the problem (12.1) is trivially infeasible. Note checking whether these assumptions are fulfilled is very easy.

One way to make the problem feasible is to reduce the lower bounds and increase the upper bounds. For a large enough change in the bounds the problem becomes feasible.

One obvious question is: What is the smallest change to the bounds that will make the problem feasible?

Associate with each bound the set of weights:

- $w_l^c \in R^m$ (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 $u^x$),

The problem

$$\begin{array}{lrcccl} \text{minimize} & & & p & & \\ \text{subject to} & l^c & \leq & Ax + v_l^c - v_u^c & \leq & u^c, \\ & l^x & \leq & x + v_l^x - v_u^x & \leq & u^x, \\ & & & (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq & 0, \\ & & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0 & & \end{array} \tag{12.4}$$

computes the minimal 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 so-called *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. Now due to $p$ is minimized and the constraint

$$(w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p \leq 0, \tag{12.5}$$

then 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 (12.4) is always feasible given the assumptions (12.2) (12.3).

Observe that if a weight is negative then the problem (12.4) is unbounded. MOSEK allows some weights to be negative, but then fixes the associated elasticity variables to zero. Constraints associated with negative weights are therefore not relaxed. This is sometimes a useful feature for marking certain constraints as not being a candidate for relaxation. Please see Section 12.1.2 for more details.

The weights $w_l^c$, $w_u^c$, $w_l^x$, and $w_u^x$ can be thought of as a costs(penalties) for violating the constraint associated with the weights. Thus a higher weight imply that higher priority is put on satisfying the constraint.

The main idea can be now presented as follows. If you have an infeasible problem, then form the problem (12.4) and optimize it. Next inspect the optimal solution $(v_l^c)^*, (v_u^c)^*, (v_l^x)^*$, and $(v_u^x)^*$ to problem (12.4). This solution provides a suggested relaxation of the bounds that will make the problem feasible.

Assume $p^*$ is an optimal objective value to (12.4) An extension of the idea given above is to solve the problem

$$
\begin{array}{lcccll}
\text{minimize} & & & c^T x & & \\
\text{subject to} & l^c & \leq & Ax + v_l^c - v_u^c & \leq & u^c, \\
 & l^x & \leq & x + v_l^x - v_u^x & \leq & u^x, \\
 & & & (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq & 0, \\
 & & & p & = & p^*, \\
 & & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0 & &
\end{array} \tag{12.6}
$$

which minimize the true objective while making sure that total weighted violations of the bounds is minimal i.e. equal to $p^*$.

### 12.1.2 The usage of negative weights

As the problem (12.4) is presented it does not make sense to use negative weights because that makes the problem unbounded. Therefore, if a weight is negative MOSEK fixes the associated elasticity variable to zero. Hence, if for instance

$$(w_l^c)_i < 0$$

then MOSEK imposes the bound

$$(v_l^c)_i \leq 0.$$

This implies that the lower bound on the $i$th constraint will not violated. (Clearly, this could also imply the problem is infeasible so negative weight should be used with care).

Therefore, negative weights can be used to indicate that some constraints must not be relaxed.

### 12.1.3 Feasibility repair in MOSEK

MOSEK makes some tools available that helps you construct the problem (12.4). This tool can be used for linear, quadratic, or conic optimization problems, possibly having integer constrained variables.

In particular MOSEK can automatically create a new problem of the form (12.4) starting from an existing problem. Hence, MOSEK will add the elasticity variables and the additional constraint.

To be specific the variables $v_l^c$, $v_u^c$, $v_l^x$, $v_u^x$, and $p$ are append to existing variable vector $x$ in their natural order. Moreover, the constraint (12.5) is appended as the last constraint.

The new variables are automatically given a name. For instance assume the 9th constraint is named `c9`, then $(v_l^c)_9$ and $(v_u^c)_9$ are given the names `LO*c9` and `UP*c9` respectively. The string `*` is a user definable separator, that should be chosen such that all names are unique. The default separator value is `*` and can be changed using the parameter `MSK_SPAR_FEASREPAIR_NAME_SEPARATOR`.

The additional constraints

$$l^x \leq x + v_l^x - v_u^x \leq u^x$$

are given a name as follows. Assume the first constraint is associated with a variable name `x1`, then the corresponding constraint is given the name

```
MSK-x1.
```

MSK- is a user definable prefix given by the value of the parameter MSK_SPAR_FEASREPAIR_NAME_PREFIX. The variable $p$ is given the name WSUMVIOLVAR and the constraint (12.5) is given the name WSUMVIOLCON. The parameter MSK_SPAR_FEASREPAIR_NAME_WSUMVIOL can be used to change WSUMVIOL in WSUMVIOLCON and WSUMVIOLVAR respectively.

#### 12.1.3.1 From the API

The API provide the function

MSK_relaxprimal

which creates a new task containing the problem (12.4). Moreover, if requested this function can solve the problems (12.4) or (12.6) automatically.

The parameter

```
MSK_IPAR_FEASREPAIR_OPTIMIZE
```

controls whether the function returns the problem (12.4) or the problem (12.6). In the case

```
MSK_IPAR_FEASREPAIR_OPTIMIZE
```

is equal to

```
MSK_FEASREPAIR_OPTIMIZE_NONE
```

then (12.4) is returned, but the problem is not solved. For MSK_FEASREPAIR_OPTIMIZE_PENALTY the problem (12.4) is returned and solved. Finally for MSK_FEASREPAIR_OPTIMIZE_COMBINED (12.6) is returned and solved.

Please see the description of the function MSK_relaxprimal for details.

### 12.1.4 An example

Consider the example linear optimization

$$
\begin{array}{lccccc}
\text{minimize} & -10x_1 & & -9x_2, & & \\
\text{subject to} & 7/10x_1 & + & 1x_2 & \leq & 630, \\
 & 1/2x_1 & + & 5/6x_2 & \leq & 600, \\
 & 1x_1 & + & 2/3x_2 & \leq & 708, \\
 & 1/10x_1 & + & 1/4x_2 & \leq & 135, \\
 & x_1, & & x_2 & \geq & 0. \\
 & & x_2 \geq 650 & & &
\end{array} \tag{12.7}
$$

This is an infeasible problem. Now suppose we wish to use MOSEK to suggest a modification to the bounds that makes the problem feasible. The following code example performs this task.

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

  File:      feasrepairex1.c

  Purpose:   Demonstrates how to use the MSK_relaxprimal function to
             locate the course 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 13

# Sensitivity analysis

## 13.1 Introduction

Given an optimization problem it is often useful to obtain information about how the optimal objective value change when the problem parameters are perturbed. For instance assume that a bound represents a capacity of a machine. Now it might be possible to expand the capacity for a certain cost and hence it worthwhile to know what the value of additional capacity is. This is precisely the type of questions sensitivity analysis deals with.

Analyzing how the optimal objective value changes when the problem data is changed is called sensitivity analysis.

## 13.2 Restrictions

Currently, sensitivity analysis is only available for continuous linear optimization problems. Moreover, MOSEK can only deal with perturbations in bounds or objective coefficients.

## 13.3 References

The book [14] discusses the classical sensitivity analysis in Chapter 10 whereas the book [21, Chapter 19] presents a modern introduction to sensitivity analysis. Finally, it is recommended to read the short paper [24] to avoid some of the pitfalls associated with sensitivity analysis.

## 13.4 Sensitivity analysis for linear problems

### 13.4.1 The optimal objective value function

Assume we are given the problem

$$\begin{array}{rclcccl} z(l^c, u^c, l^x, u^x, c) & = & \text{minimize} & & c^T x & & \\ & & \text{subject to} & l^c \leq & Ax & \leq & u^c, \\ & & & & l^x \leq x \leq u^x, & & \end{array} \tag{13.1}$$

and we want to know how the optimal objective value changes as $l_i^c$ is perturbed. In order to answer this question then define the perturbed problem for $l_i^c$ as follows

$$\begin{array}{rclccl} f_{l_i^c}(\beta) & = & \text{minimize} & & c^T x & \\ & & \text{subject to} & l^c + \beta e_i \leq & Ax & \leq u^c, \\ & & & & l^x \leq x \leq u^x, & \end{array} \tag{13.2}$$

where $e_i$ is the $i$th column of the identity matrix. The function

$$f_{l_i^c}(\beta) \tag{13.3}$$

shows the optimal objective value as a function of $\beta$. Note a change in $\beta$ corresponds to a perturbation in $l_i^c$ and hence (13.3) shows the optimal objective value as a function of $l_i^c$.

It is possible to prove that the function (13.3) is a piecewise linear and convex function i.e. the function may look like the illustration in Figure 13.1.

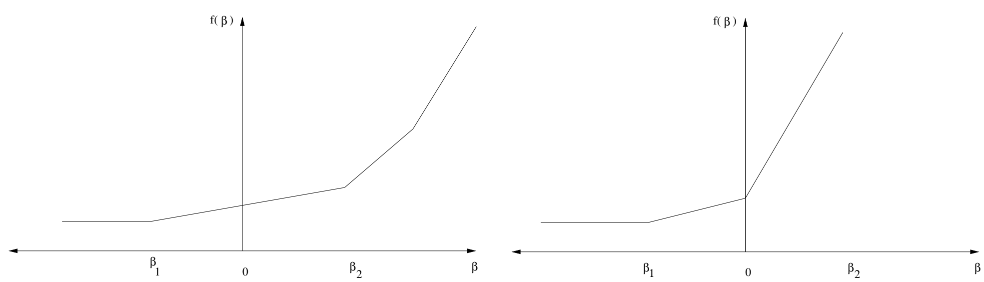


Figure 13.1: The optimal value function $f_{l_i^c}(\beta)$. Left: $\beta = 0$ is in the interior of linearity interval. Right: $\beta = 0$ is a breakpoint.

Clearly, if the function $f_{l_i^c}(\beta)$ does not change much when $\beta$ is changed, then we can conclude that the optimal objective value is insensitive to changes in $l_i^c$. Therefore, we are interested in how $f_{l_i^c}(\beta)$ changes for small changes in $\beta$. Now define

$$f'_{l_i^c}(0) \tag{13.4}$$

to be the so called *shadow price* related to $l_i^c$. The shadow price specifies how the objective value changes for small changes in $\beta$ around zero. Moreover, we are interested in the so called *linearity interval*

$$\beta \in [\beta_1, \beta_2] \tag{13.5}$$

for which

$$f'_{l_i^c}(\beta) = f'_{l_i^c}(0). \tag{13.6}$$

To summarize the sensitivity analysis provides a shadow price and the linearity interval in which the shadow price is constant.

The reader may have noticed that we are sloppy in the definition of the shadow price. The reason is that the shadow price is not defined in the right example in Figure 13.1 because the function $f_{l_i^c}(\beta)$ is not differentiable for $\beta = 0$. However, in that case we can define a left and a right shadow price and a left and a right linearity interval.

In the above discussion we only discussed changes in $l_i^c$. We define the other optimal objective value functions as follows

$$\begin{array}{rcll} f_{u_i^c}(\beta) & = & z(l^c, u^c + \beta e_i, l^x, u^x, c), & i = 1, \ldots, m, \\ f_{l_j^x}(\beta) & = & z(l^c, u^c, l^x + \beta e_j, u^x, c), & j = 1, \ldots, n, \\ f_{u_j^x}(\beta) & = & z(l^c, u^c, l^x, u^x + \beta e_j, c), & j = 1, \ldots, n, \\ f_{c_j}(\beta) & = & z(l^c, u^c, l^x, u^x, c + \beta e_j), & j = 1, \ldots, n. \end{array} \tag{13.7}$$

Given these definitions it should be clear how linearity intervals and shadow prices are defined for the parameters $u_i^c$ etc.

#### 13.4.1.1 Equality constraints

In MOSEK a constraint can be specified as either an equality constraints or a ranged constraints. Suppose constraint $i$ is an equality constraint. We then define the optimal value function for constraint $i$ by

$$f_{e_i^c}(\beta) = z(l^c + \beta e_i, u^c + \beta e_i, l^x, u^x, c) \tag{13.8}$$

Thus for a equality constraint the upper and lower bound (which are equal) are perturbed simultaneously. From the point of view of MOSEK sensitivity analysis a ranged constrain with $l_i^c = u_i^c$ therefore differs from an equality constraint.

### 13.4.2 The basis type sensitivity analysis

The classical sensitivity analysis discussed in most textbooks about linear optimization, e.g. [14, Chapter 10], is based on an optimal basic solution or equivalently on an optimal basis. This method may produce misleading results [21, 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 then the basis type sensitivity analysis computes the linearity interval $[\beta_1, \beta_2]$ such 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 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.

### 13.4.3 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 to the optimal partition type sensitivity analysis is it is computationally expensive. This type of sensitivity analysis is currently provided as an experimental feature in MOSEK.

Given optimal primal and dual solutions to (13.1) i.e. $x^*$ and $((s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ then the optimal objective value is given by

$$z^* := c^T x^*. \tag{13.9}$$

The left and right shadow prices $\sigma_1$ and $\sigma_2$ for $l_i^c$ is given by the pair of optimization problems

$$\begin{array}{lllll} \sigma_1 & = & \text{minimize} & e_i^T s_l^c & & \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x & = & c, \\ & & & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) & = & z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0 & & \end{array} \tag{13.10}$$

and

$$\begin{array}{lllll} \sigma_2 & = & \text{maximize} & e_i^T s_l^c & & \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x & = & c, \\ & & & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) & = & z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0. & & \end{array} \tag{13.11}$$

The above two optimization problems makes it easy to interpret-ate the shadow price. Indeed assume that $((s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ is an arbitrary optimal solution then it must hold

$$(s_l^c)_i^* \in [\sigma_1, \sigma_2]. \tag{13.12}$$

Next the linearity interval $[\beta_1, \beta_2]$ for $l_i^c$ is computed by solving the two optimization problems

$$\begin{array}{lllllll} \beta_1 & = & \text{minimize} & & & \beta & & \\ & & \text{subject to} & l^c + \beta e_i & \leq & Ax & \leq & u^c, \\ & & & & & c^T x - \sigma_1 \beta & = & z^*, \\ & & & & & l^x \leq x \leq u^x, & & \end{array} \tag{13.13}$$

and

$$\begin{array}{lllllll} \beta_2 & = & \text{maximize} & & & \beta & & \\ & & \text{subject to} & l^c + \beta e_i & \leq & Ax & \leq & u^c, \\ & & & & & c^T x - \sigma_2 \beta & = & z^*, \\ & & & & & l^x \leq x \leq u^x. & & \end{array} \tag{13.14}$$

The linearity intervals and shadow prices for $u_i^c$, $l_j^x$, and $u_j^x$ can be computed in a similar way to how it is computed for $l_i^c$.

The left and right shadow price for $c_j$ denoted $\sigma_1$ and $\sigma_2$ respectively is given by the pair optimization problems

$$\begin{array}{rclcccl} \sigma_1 & = & \text{minimize} & & e_j^T x & & \\ & & \text{subject to} & l^c + \beta e_i \leq & Ax & \leq & u^c, \\ & & & & c^T x & = & z^*, \\ & & & & l^x \leq x \leq & u^x & \end{array} \tag{13.15}$$

and

$$\begin{array}{rclcccl} \sigma_2 & = & \text{maximize} & & e_j^T x & & \\ & & \text{subject to} & l^c + \beta e_i \leq & Ax & \leq & u^c, \\ & & & & c^T x & = & z^*, \\ & & & & l^x \leq x \leq & u^x. & \end{array} \tag{13.16}$$

Once again the above two optimization problems makes it easy to interpret-ate the shadow prices. Indeed assume that $x^*$ is an arbitrary primal optimal solution then it must hold

$$x_j^* \in [\sigma_1, \sigma_2]. \tag{13.17}$$

The linearity interval $[\beta_1, \beta_2]$ for a $c_j$ is computed as follows

$$\begin{array}{rclccl} \beta_1 & = & \text{minimize} & \beta & & \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x & = & c + \beta e_j, \\ & & & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) - \sigma_1 \beta & \leq & z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0 & & \end{array} \tag{13.18}$$

and

$$\begin{array}{rclccl} \beta_2 & = & \text{maximize} & \beta & & \\ & & \text{subject to} & A^T(s_l^c - s_u^c) + s_l^x - s_u^x & = & c + \beta e_j, \\ & & & (l_c)^T(s_l^c) - (u_c)^T(s_u^c) + (l_x)^T(s_l^x) - (u_x)^T(s_u^x) - \sigma_2 \beta & \leq & z^*, \\ & & & s_l^c, s_u^c, s_l^c, s_u^x \geq 0. & & \end{array} \tag{13.19}$$

### 13.4.4 An example

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 13.2.

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

$$1x_{11} + 2x_{12} + 5x_{23} + 2x_{24} + 1x_{31} + 2x_{33} + 1x_{34} \tag{13.20}$$

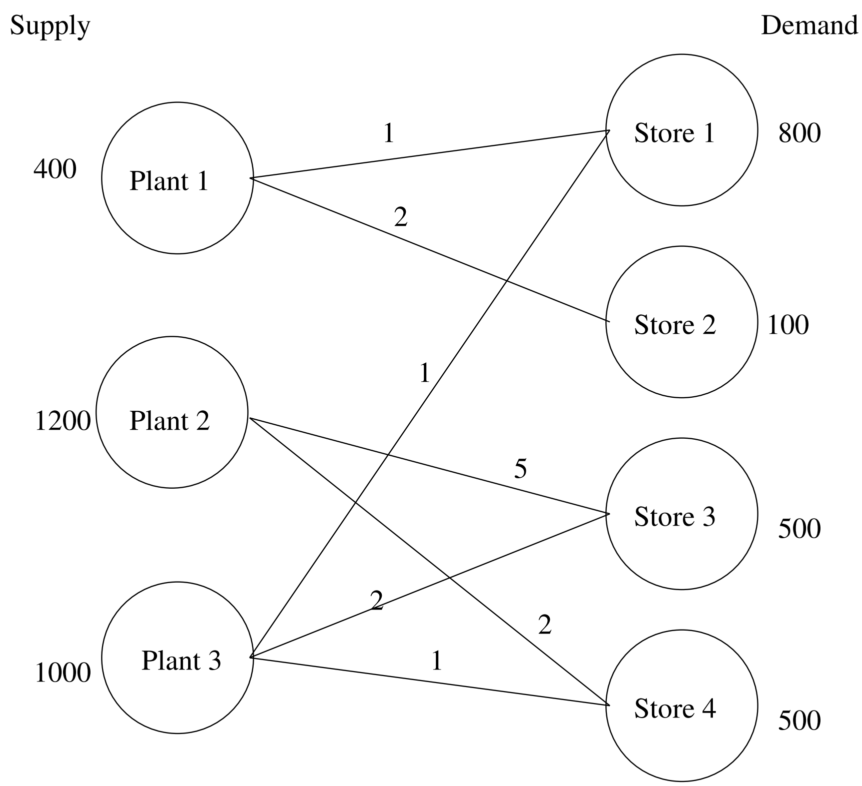


Figure 13.2: Supply, demand and cost of transportation.

subject to

$$
\begin{array}{cccccccccccccccc}
x_{11} & + & x_{12} & & & & & & & & & & & \leq & 400, \\
& & & & x_{23} & + & x_{24} & & & & & & & \leq & 1200, \\
& & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\
x_{11} & & & & & & & + & x_{31} & & & & & = & 800, \\
& & x_{12} & & & & & & & & & & & = & 100, \\
& & & & x_{23} & + & & & & & x_{33} & & & = & 500, \\
& & & & & & x_{24} & + & & & & & x_{34} & = & 500, \\
x_{11}, & & x_{12}, & & x_{23}, & & x_{24}, & & x_{31}, & & x_{33}, & & x_{34} & \geq & 0.
\end{array}
\tag{13.21}
$$

The basis type and the optimal partition type sensitivity results for the transportation problem is shown in Table 13.1 and 13.2 respectively.

**Basis type**

| Con. | $\beta_1$ | $\beta_2$ | $\sigma_1$ | $\sigma_2$ |
|---|---|---|---|---|
| 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. | $\beta_1$ | $\beta_2$ | $\sigma_1$ | $\sigma_2$ |
| $x_{11}$ | $-\infty$ | 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. | $\beta_1$ | $\beta_2$ | $\sigma_1$ | $\sigma_2$ |
|---|---|---|---|---|
| 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. | $\beta_1$ | $\beta_2$ | $\sigma_1$ | $\sigma_2$ |
| $x_{11}$ | $-\infty$ | 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 13.1: Ranges and shadow prices related to bounds on constraints and variables. Left: Results for basis type sensitivity analysis. Right: Results for the optimal partition type sensitivity analysis.

Looking at the results from the optimal partition type sensitivity analysis we see that for the 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 price is 3 and 1 respectively. This implies if the upper bound on constraint 1 increases by

$$\beta \in [0, \beta_1] = [0, 500] \tag{13.22}$$

then the optimal objective value will decrease by the value

$$\sigma_2 \beta = 1\beta. \tag{13.23}$$

Correspondingly, if the upper bound on constraint 1 is decreased by

$$\beta \in [0, 300] \tag{13.24}$$

**Basis type**

| Var. | $\beta_1$ | $\beta_2$ | $\sigma_1$ | $\sigma_2$ |
|---|---|---|---|---|
| $c_1$ | $-\infty$ | 3.00 | 300.00 | 300.00 |
| $c_2$ | $-\infty$ | $\infty$ | 100.00 | 100.00 |
| $c_3$ | $-2.00$ | $\infty$ | 0.00 | 0.00 |
| $c_4$ | $-\infty$ | 2.00 | 500.00 | 500.00 |
| $c_5$ | $-3.00$ | $\infty$ | 500.00 | 500.00 |
| $c_6$ | $-\infty$ | 2.00 | 500.00 | 500.00 |
| $c_7$ | $-2.00$ | $\infty$ | 0.00 | 0.00 |

**Optimal partition type**

| Var. | $\beta_1$ | $\beta_2$ | $\sigma_1$ | $\sigma_2$ |
|---|---|---|---|---|
| $c_1$ | $-\infty$ | 3.00 | 300.00 | 300.00 |
| $c_2$ | $-\infty$ | $\infty$ | 100.00 | 100.00 |
| $c_3$ | $-2.00$ | $\infty$ | 0.00 | 0.00 |
| $c_4$ | $-\infty$ | 2.00 | 500.00 | 500.00 |
| $c_5$ | $-3.00$ | $\infty$ | 500.00 | 500.00 |
| $c_6$ | $-\infty$ | 2.00 | 500.00 | 500.00 |
| $c_7$ | $-2.00$ | $\infty$ | 0.00 | 0.00 |

Table 13.2: Ranges and shadow prices related to the objective coefficients. Left: Results for basis type sensitivity analysis. Right: Results for the optimal partition type sensitivity analysis.

then the optimal objective value will increased by the value

$$\sigma_1\beta = 3\beta. \tag{13.25}$$

## 13.5 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 is reserved.

  File:      sensitivity.c

  Purpose:   Demonstrates 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 nonzeros 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;

  /* Make 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 */
```

## 13.6 Sensitivity analysis with the command line tool

A sensitivity analysis can be performed with the MOSEK command line tool sensitivity analysis 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 problem problem parameters the sensitivity analysis should be performed for.

Results are by default written to the file `myproblem.sen`. This default can be changed by setting the parameter

```
MSK_SPAR_SENSITIVITY_RES_FILE_NAME
```

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. This parameter can take values

```
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.

### 13.6.1 Sensitivity analysis specification file

MOSEK employs a MPS like file format to specify on which model parameters the sensitivity analysis should be performed on. Due to the optimal partition type sensitivity analysis can be computational expensive then it is important to be able to limit the sensitivity analysis.

The format of the sensitivity specification file is shown in Figure 13.3. Capitalized names are keywords

```
* 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 13.3: The sensitivity analysis file format.

in the format whereas names appearing in brackets are user defined names of constraints and variables.

The sensitivity specification file has three sections i.e.

- `BOUNDS CONSTRAINTS`: Specifies on which bounds on constraints the sensitivity analysis should be performed on.
- `BOUNDS VARIABLES`: Specifies on which bounds on variables the sensitivity analysis should be performed on.
- `OBJECTIVE VARIABLES`: Specifies on which objective coefficients the sensitivity analysis should be performed on.

A line in the body of a section must begin with a whitespace. In the `BOUNDS` sections one of the the keys `L`, `U`, and `LU` must appear next. These keys specifies whether the sensitivity analysis is performed on the lower bound, on the upper bound, or both on the lower and upper bound respectively. Next a single constraint (variable) or range of constraints (variables) is specified.

Recall from Section 13.4.1.1 that equality constraints are handled in a special way. Sensitivity analysis for an equality constraint can be specified with either `L`, `U`, or `LU`, all having the same meaning, namely that the upper and lower bound (which are equal) are perturbed simultaneously.

As an example consider

```
BOUNDS CONSTRAINTS
 L  "cons1"
 U  "cons2"
 LU "cons3"-"cons6"
```

which says sensitivity analysis should be performed on the lower bound on the constraint named `cons1` and on the upper bound for the constraint named `cons2`. Finally, sensitivity analysis should be performed for both the lower and the upper bounds on the constraints named `cons3` to `cons6`.

It is allowed to use indexes instead of names i.e. for instance

```
BOUNDS CONSTRAINTS
 L  "cons1"
 U  2
 LU 3 - 6
```

The character "*" indicates that the line contains a comment and is ignored.

### 13.6.2 An example

As an example consider the file `sensitivity.ssp` shown in Figure 13.4.

```
* 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 13.4: 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 file `transport.sen` 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
```

### 13.6.3 Controlling log output

Setting the parameter

`MSK_IPAR_LOG_SENSITIVITY`

to 1 or 0 (default) control whether or not the results from sensitivity calculations are printed to the message stream.

The parameter

`MSK_IPAR_LOG_SENSITIVITY_OPT`

control the amount of debug information on internal calculations from sensitivity analysis.

# Chapter 14

# Case Studies

## 14.1 The traveling salesman problem

### 14.1.1 Weak versus strong formulations

When solving mixed integer optimization problems it is important to use a strong formulation of the problem. Otherwise MOSEK may take a very long time to solve the optimization problem. This is not only true for MOSEK but in branch and bound based solution method. To illustrate the impact of the strength of the formulation we have solved a series of traveling salesman problems (TSP) using formulations of varying strength.

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 [20].

#### 14.1.1.1 The TSP formulations

The TSP is the problem of finding a tour (a directed cycle containing all nodes) of minimal length in a complete directed graph. We use the variables $x_{ij}$ to indicate whether the arc $(i,j)$ is included in the tour.

The core of the formulation is

$$
\begin{array}{lll}
\text{minimize} & \sum\limits_{i,j} c_{ij}x_{ij} & \\
\text{subject to} & \sum\limits_{i} x_{ij} = 1 & \forall j, \\
 & \sum\limits_{j} x_{ij} = 1 & \forall i, \\
 & 0 \leq x_{ij} \leq 1, & x_{ij} \text{ integer.}
\end{array}
\tag{14.1}
$$

These constraints are called the assignment constraints. The assignment constraints however does not constitute the entire formulation as groups of disjoint cycles, called subtours, are feasible as well as the complete tours.

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

$$\begin{aligned} u_1 &= 1, & \\ 2 \le u_i &\le n & \forall i \neq 1, \\ u_i - u_j + 1 &\le (n-1)(1 - x_{ij}) & \forall i \neq 1, j \neq 1. \end{aligned} \tag{14.2}$$

The idea of this formulation is to assign the numbers 1 through $n$ to the nodes with the extra variables $u_i$ such 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.

$$\sum_{i,j \in S} x_{ij} \le |S| - 1 \quad (S \subsetneq V, |S| > 1) \tag{14.3}$$

As the subtour inequality for $V \setminus S$ is a linear combination of the inequality for $S$ and the assignment constraints, it is enough to use the subtour inequalities with $S$ having size at most $n/2$. 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. We will try for each problem 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 pseudocode is (the complete C implementation is included below in Section 14.1.1.3):

1. Let the IP formulation consist of the assignment constraints (14.1) 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 at most 1000 subtour constraints, in which $S$ is the union of several $S_i$ sets and $|S| \le \lfloor n/2 \rfloor$.
3. Add the MTZ arc constraints to the formulation, and solve the IP to optimality.

We add at most 1000 constraints each round 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 14.1: Solving TSP using increasingly stronger formulations.

#### 14.1.1.2 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 is recorded in Table 14.1. The entry "***" means that the problem was unsolvable within a time window of 5000 seconds. The time spent solving the relaxed IP's 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.

#### 14.1.1.3 Example code

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

## 14.2 Geometric (posynomial) optimization

### 14.2.1 The problem

A so-called geometric optimization problem can be stated as follows

$$
\begin{array}{lll}
\text{minimize} & \sum_{k\in J_0} c_k \prod_{j=0}^{n-1} t_j^{a_{kj}} & \\
\text{subject to} & \sum_{k\in J_i} c_k \prod_{j=0}^{n-1} t_j^{a_{kj}} \quad \leq \quad 1, \quad i=1,\ldots,m, & \\
& t > 0, &
\end{array}
\tag{14.4}
$$

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 an $T \times n$ 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 (14.4) 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

$$t_j = e^{x_j} \tag{14.5}$$

we obtain the problem

$$\begin{array}{lll} \text{minimize} & \sum\limits_{k \in J_0} c_k e^{\sum\limits_{j=0}^{n-1} a_{kj} x_j} & \\ \text{subject to} & \sum\limits_{k \in J_i} c_k e^{\sum\limits_{j=0}^{n-1} a_{kj} x_j} & \leq \quad 1, \quad i = 1, \ldots, m, \end{array} \tag{14.6}$$

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 (14.6) is nonseparable. However, using

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

we obtain the separable problem

$$\begin{array}{llll} \text{minimize} & \sum\limits_{t \in J_0} e^{v_t} & & \\ \text{subject to} & \sum\limits_{t \in J_i} e^{v_t} & \leq \quad 1, & i = 1, \ldots, m, \\ & \sum\limits_{j=0}^{n-1} a_{tj} x_j - v_t & = \quad -\log(c_t), & t = 0, \ldots, T, \end{array} \tag{14.7}$$

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.

### 14.2.2 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 [13] and the references therein.

### 14.2.3 Modelling tricks

A lot of tricks that can be used modelling posynomial optimization problems can be seen in [13]. Therefore, in this section we only cover one important case.

#### 14.2.3.1 Equalities

In general equalities are not allowed in (14.4) 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

$$\begin{array}{lcll} xyz^{-1} & & \leq & 1, \\ \frac{1}{xyz^{-1}} & = & x^{-1}y^{-1}z \quad \leq & 1. \end{array}$$

Hence, it is possible to model a monomial equality using two inequalities.

### 14.2.4 Problematic formulations

Consider the problem

$$\begin{array}{lccc} \text{minimize} & x^2y & & \\ \text{subject to} & xy & \leq & 1, \\ & x, y > 0. & & \end{array}$$

Clearly, the optimal objective value is 0. But this is never attained the constraint $x, y > 0$ this may causes problems the algorithm solving the problem. Observer this problem does not occur in

$$\begin{array}{lccc} \text{minimize} & x^2y^{-1} & & \\ \text{subject to} & x^{-1}y & \leq & 1, \\ & x, y > 0, & & \end{array}$$

because in this case neither $x$ nor $y$ can or will be arbitrary small. It should now be clear what the issue is. If a variable $x$ has a nonnegative $a_{kj}$ for all $k$, then this variable can be fixed at zero but this causes problems related to

$$x > 0$$

constraint. Or an alternative problem can be that a constraint of the form

$$lx^{-1} \leq 1$$

where $l$ is a positive constant i.e. this constraints implies $x \geq l$ and hence $x$ cannot be arbitrary small.

Therefore, avoid formulating problems where

$$a_{kj} \geq 0, \ \forall k.$$

In fact a similar problem occurs if

$$a_{kj} \leq 0, \ \forall k.$$

In this case $x$ identical to plus infinity is a solution but usually this cannot be case in practice and hence some constraints has been let of formulation.

### 14.2.5 An example

Consider the example

$$\begin{array}{lccc} \text{minimize} & x^{-1}y & & \\ \text{subject to} & x^2y^{-\frac{1}{2}} + 3y^{\frac{1}{2}}z^{-1} & \leq & 1, \\ & xy^{-1} & = & z^2, \\ & -x & \leq & -\frac{1}{10}, \\ & x & \leq & 3, \\ & x, y, z > 0, & & \end{array}$$

which is not a geometric optimization problem. However, using the obvious transformations we obtain the problem

$$\begin{array}{lccc} \text{minimize} & x^{-1}y & & \\ \text{subject to} & x^2y^{-\frac{1}{2}} + 3y^{\frac{1}{2}}z^{-1} & \leq & 1, \\ & xy^{-1}z^{-2} & \leq & 1, \\ & x^{-1}yz^2 & \leq & 1, \\ & \frac{1}{10}x^{-1} & \leq & 1, \\ & \frac{1}{3}x & \leq & 1, \\ & x, y, z > 0, & & \end{array} \tag{14.8}$$

which is a geometric optimization problem.

### 14.2.6 Solving from the command line tool

MOSEK provides the command line tool `mskexpopt` to solve a problem on the form (14.7). As seen previusly an optimal solution to this problem can be transformed to an optimal solution to the geometric optimization problem (14.4) by using the transform:

$$t_j = e^{x_j}.$$

A more detailed description of `mskexpopt` and the definition of the input format used can be found in Section 6.2. The source code is also included in the MOSEK distribution.

#### 14.2.6.1 An example

The problem 14.8 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 to a solution to the geometric optimization problem as follows

$$t_0 = e^{-2.302585e+00} = 0.1 \tag{14.9}$$

$$t_1 = e^{-9.208438e+00} = 1.0019^{-4} \tag{14.10}$$

$$t_1 = e^{3.452927e+00} = 31.5927. \tag{14.11}$$

### 14.2.7 Further information

More information about geometric optimization problems can located in [10, 11, 13].

Chapter 15

# API developer guidelines

The purpose of this chapter is to present some guidelines that are useful to follow when developing an application which employs the MOSEK API.

## 15.1 Turn logging on

While developing a new application it is beneficial to turn logging on so error and diagnostic messages can be seen.

Using the function MSK_linkfiletotaskstream a file can be linked to a task stream. This implies all the messages send to a task stream is also send 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 user defined function to a stream using the function MSK_linkfunctotaskstream. The user defined function may send the stream messages to the screen.

## 15.2 Turn data checks on

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 warns about them.

## 15.3 Debugging an optimization task

If something is wrong with a problem or a solution, then it is useful to write the problem to a file using the function MSK_writedata because then the problem can be inspected. For instance the code fragment

```
MSK_writedata(task,"taskdump.lp");
MSK_optimize(task);
```

demonstrates how to write the problem to the file taskdump.lp immediately before optimizing the problem. An inspection of the text file taskdump.lp may reveal that MOSEK has been fed the wrong problem data.

## 15.4 Error handling

Almost all functions in the C API returns a so-called repsonse code which indicates whether an error occured. It is recommended to check to the response code and in case it is indicatating an error then an appropriate action should be taken.

## 15.5 Fatal error handling

In case MOSEK encounter a fatal error either due to an internal bug or an user error, then a so called exit function is called. It is possible to inform MOSEK about a user defined exit function using the function MSK_putexitfunc. The user defined exit function will then be called in case a fatal occurs.

The purpose of an exit function is to print out a suitable message that can help diagnose the cause of the error.

## 15.6 Checking for memory leaks and overwrites

If you suspect MOSEK or your own application overwrites memory or leaks memory, then we suggest you use external tools such as Purify[1] or valgrind[2] to pinpoint the cause of the problem.

MOSEK has a memory check feature that can be enabled by letting the argument dbugfile by a non null pointer when calling the function MSK_makeenv. If dgbfile is valid file name, then MOSEK will write some memory debug information to the file specified by the string.

Assuming memory debugging is turned on, then MOSEK will warn about MOSEK specific memory leaks when a MOSEK environment or task is deleted.

[1] Purify is a commercial product available from IBM. Purify runs on Windows and various UNIXes. You may need to upgrade to latest version of Purify.

[2] valgrind is open source product available for Linux on X86. It is an excellent tool that is highly recommended

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 of those functions finds that the memory has been corrupted, then a fatal error is generated.

## 15.7 Check the problem and solution statuses

In the MOSEK primal or dual infeasible problem is **not** considered an error. Hence, no error or exception is generated in the case of either primal or dual infeasible problems.

Therefore, it is important to check the problem status and solution status after the optimization optimization been perfomed. using the MSK_getsolutionstatus or MSK_getsolutioninf. function.

## 15.8 Important API limitations

### 15.8.1 Thread safety

The MOSEK API is thread safe provided that a task is modified from one thread only.

### 15.8.2 Unicoded strings

The `C` API supports the usage of unicoded strings. Indeed all `char *` are allowed to be UTF8 encoded strings.

#### 15.8.2.1 Limitations

Please note that the MPS and LP file formats are ASCII formats. Therefore, it might be advantageous to limit all names for constraints, variables etc. to ASCII strings.

## 15.9 Bug reporting

If you think MOSEK is solving your problem incorrectly, then please contact MOSEK support at support@mosek.com with 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 file `taskfile.mbt` contains the problem data in binary form which is very useful when reproducing a problem.

# Chapter 16

# API reference

This chapter lists all functionality in the MOSEK C API.

## 16.1 Type definitions

- `MSKbooleant`

  **Description:**
  A signed integer interpreted as a boolean value.

- `MSKenv_t`

  **Description:**
  The MOSEK Environment type.

- `MSKidxt`

  **Description:**
  A signed integer used for indexing, usually 32 bits. This is used as indexer into arrays which are guaranteed to not exceed $2^3 2$ 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 are may on some platforms exceed $2^3 2$ bits in length. In 32bit architectures it will always be a signed 32bit integer, while on 64bit architectures it may be either a 32bit or 64bit signed integer.

- `MSKlintt`

  **Description:**

  A signed large integer. On 32bit architectures it is always 32 bits, while on 64bit 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 generic userdefined handle.

- `MSKwchart`

  **Description:**

  Wide char type. The actual type may differ depending on the platform; it is either a 16bit or 32bit 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 interior-point optimizer. For the simplex optimizers then `MSK_IPAR_LOG_SIM_FREQ` controls how frequent 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 because if the progress call-back function returns a nonzero value, then 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 accesses the task information database. The items in task information database are updated during the optimization process.

**Syntax:**
```
MSKintt MSKcallbackfunc (
        MSKtask_t task,
        MSKuserhandle_t usrptr,
        MSKcallbackcodee caller);
```

**Arguments:** `task` (input)
An optimization task.

`usrptr` (input/output)
A user defined handle which is passed to the user defined function.

`caller` (input)
This is an integer which will denote where the function was called from. See Section 19.5 for the possible values of this argument.

- `MSKctrlcfunc`

**Description:** Definition of a user defined `ctrl-c` function. If the function returns a nonzero value, then MOSEK assumes `ctrl-c` has been pressed.

**Syntax:** `MSKintt MSKctrlcfunc (MSKuserhandle_t usrptr)`

**Arguments:** `usrptr` (input/output)
A user defined handle which is passed to the user defined function.

- `MSKexitfunc`

**Description:** A user defined exit function which is called in case of fatal errors to handle an error message and terminate the program. The function should never return.

**Syntax:**
```
void MSKexitfunc (
        MSKuserhandle_t usrptr,
        MSKCONST char * file,
        MSKintt line,
        MSKCONST char * msg);
```

**Arguments:** `usrptr` (input/output)
A user defined handle which is passed to the user defined function.

`file` (input)
The name of the file where the fatal error occurred.

`line` (input)
The line number in file where the fatal error occurred.

`msg` (input)
A message about the error.

- `MSKfreefunc`

**Description:** A user defined `free` function.

**Syntax:**
```
void MSKfreefunc (
        MSKuserhandle_t usrptr,
        MSKuserhandle_t buffer);
```

**Arguments:** `usrptr` (input)
A user defined handle which is passed to the user defined function.

`buffer` (input)
A pointer to the buffer which should be freed.

- `MSKmallocfunc`

**Description:** A user defined `malloc` function.

**Syntax:**
```
void * MSKmallocfunc (
    MSKuserhandle_t usrptr,
    MSKCONST size_t size);
```

**Arguments:** `usrptr` (input)
A user defined handle which is passed to the user defined function.

`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 MSKnlgetspfunc (
    MSKuserhandle_t nlhandle,
    MSKintt * numgrdobjnz,
    MSKidxt * grdobjsub,
    MSKidxt i,
    MSKintt * 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. The pointer is passed to MOSEK when the function `MSK_putnlfunc` is called.

`numgrdobjnz` (output)
If required, then `numgrdobjnz` should be assigned the number of non-zero elements in the gradient of $f$.

`grdobjsub` (output)
If required, then it should contain the position 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.

`convali` (output)
If a non-null pointer, then

$$\texttt{convali}[0] = \begin{cases} 0, & g_i(x) = 0, \ \forall x, \\ 1, & \text{otherwise.} \end{cases}$$

`grdconinz` (output)
If required, then `grdconinz` should be assigned the number of non-zero elements in $\nabla g_i(x)$.

`grdconisub` (output)
If a non-null pointer, then

$$\texttt{grdconisub}[0, .., \texttt{grdconinz}[0] - 1]$$

should be identical to the positions of the non-zeros in $\nabla g_i(x)$.

`yo` (input)
If non-zero, then the $f$ should be included when the gradient and the Hessian of the Lagrangian is computed.

`numycnz` (input)
Number of constraint functions which are included in the definition of the Lagrangian. See (16.1).

`ycsub` (input)
Index of constraint functions which are included in the definition of the Lagrangian. See (16.1).

`maxnumhesnz` (input)
Length of the arguments `hessubi` and `hessubj`.

`numhesnz` (output)
If required, then `numhesnz` should be assigned the number of non-zero elements in the lower triangular part of the Hessian of the Lagrangian:

$$L := \texttt{yo} f(x) - \sum_{k=0}^{\texttt{numycnz}-1} g_{\texttt{ycsub[k]}}(x) \tag{16.1}$$

`hessubi` (output)
If a non-null pointer, then `hessubi` and `hessubj` are used to convey the position of the non-zeros in the Hessian of the Lagrangian $L$ (see (16.1)) as follows

$$\nabla^2 L_{\texttt{hessubi}[k],\texttt{hessubj}[k]}(x) \neq 0.0 \tag{16.2}$$

for $k = 0, \ldots, \texttt{numhesnz} - 1$. All other positions in $L$ are assumed to be zero. Note it is sufficient to return the lower or the upper triangular part of the Hessian.

`hessubj` (output)
See the argument `hessubi`.

- `MSKnlgetvafunc`

**Description:** Type definition of the call-back function which is used to provide structural as well as numerical information about the nonlinear functions $f$ and $g$ in the optimization problem.

For later use we need the definition of the Lagrangian $L$ which is given by

$$L := \mathtt{yo} * f(\mathtt{xx}) - \sum_{i=0}^{\mathtt{numi}-1} \mathtt{yc}_{\mathtt{subi[k]}} g_{\mathtt{subi[k]}}(\mathtt{xx}). \tag{16.3}$$

**Syntax:**

```
MSKintt MSKnlgetvafunc (
    MSKuserhandle_t nlhandle,
    MSKCONST MSKrealt * xx,
    MSKrealt yo,
    MSKCONST MSKrealt * yc,
    MSKrealt * objval,
    MSKintt * numgrdobjnz,
    MSKidxt * grdobjsub,
    MSKrealt * grdobjval,
    MSKintt numi,
    MSKCONST MSKidxt * subi,
    MSKrealt * conval,
    MSKCONST 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 required, then `objval` should be be assigned $f(x)$ evaluated at $xx$.

`numgrdobjnz` (output)

If required, then `numgrdobjnz` should be assigned the number of non-zero elements in the gradient of $f$.

`grdobjsub` (output)
: If a non-null pointer, then it should 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 required, then it should contain the numerical value of the gradient of $f$ evaluated at `xx`. The following data structure

$$\texttt{grdobjval[k]} = \frac{\partial f}{\partial x_{\texttt{grdobjsub[k]}}}(\texttt{xx})$$

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.

$$\texttt{conval[k]} = g_{\texttt{subi[k]}}(\texttt{xx})$$

for $k = 0,\ldots,\texttt{numi}-1$.

`grdconptrb` (input)
: If required, then it is used to specify the gradients of the constraints. See the argument `grdconval` for details.

`grdconptre` (input)
: If required, then it is used to specify the gradients of the constraints. See the argument `grdconval` for details.

`grdconsub` (output)
: If required, then it is used to specify the position of the non-zeros in gradients of the constraints. See the argument `grdconval` for details.

`grdconval` (output)
: `grdconptrb`, `grdconptre`, and `grdconsub` are used to specify the gradients of the constraint functions. **`grdconptrb`** and `grdconptre` are specified by the calling function. **Please note that both `grdconsub` and `grdconval` should be updated when required.**
The gradient data are stored as follows

$$\begin{array}{l}\texttt{grdconval[k]} = \frac{\partial g_{\texttt{subi[i]}}(xx)}{\partial xx_{\texttt{grdconsub[k]}}}, \quad \text{for} \\ \quad k = \texttt{grdconptrb}[i],\ldots,\texttt{grdconptre}[i]-1, \\ \quad i = 0,\ldots,\texttt{numi}-1.\end{array}$$

`grdlag` (output)
: If required, then `grdlag` should be identical to 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 required, then `numhesnz` should be assigned the number of non-zeros elements in the Hessian of the Lagrangian $L$, see (16.3).

`hessubi` (output)
See the argument `hesval`.

`hessubj` (output)
See the argument `hesval`.

`hesval` (output)
`hessubi`, `hessubj`, and `hesval` are used to store the Hessian of the Lagrangian function $L$ defined by (16.3).
The following data structure

$$\texttt{hesval}[k] = \nabla^2 L_{\min(\texttt{hessubi}[k],\texttt{hessubj}[k]),\max(\texttt{hessubi}[k],\texttt{hessubj}[k])}$$

for $k = 0, \ldots, \texttt{numhesnz}[0] - 1$ is used. Note if one 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 then this function is called. The argument `r` contains the code of the error/warning and the argument `msg` contains the corresponding error/warning message. This function should always return `MSK_RES_OK`.

**Syntax:**
```
MSKrescodee MSKresponsefunc (
    MSKuserhandle_t handle,
    MSKrescodee r,
    MSKCONST char * msg);
```

**Arguments:** `handle` (input/output)
A pointer to a user defined data structure (or a null pointer).

`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 if a message is send to the stream to which the function is linked, then the function is called by MOSEK and the argument `str` will be identical to the message. Hence, the user can decide what should happen to message.

**Syntax:**
```
void MSKstreamfunc (
    MSKuserhandle_t handle,
    MSKCONST char * str);
```

**Arguments:** `handle` (input/output)
A pointer to a user defined data structure (or a null pointer).

`str` (input)
A string containing a message to a stream.

## 16.2 API Functionality

Functions in the interface grouped by functionality.

### 16.2.1 Reading and writing data to files.

Reading and writing data to files.

**MSK_readbranchpriorities** (page 342)
Reads branching priority data from a file.

**MSK_readdata** (page 342)
Reads problem data from a file.

**MSK_readparamfile** (page 343)
Reads a parameter file.

**MSK_readsolution** (page 343)
Reads a solution from a file.

**MSK_readsummary** (page 343)
Prints information about last read.

**MSK_writebranchpriorities** (page 353)
Writes branching priority data to a file.

**MSK_writeparamfile** (page 354)
Writes all the parameters to a parameter file.

**MSK_writesolution** (page 354)
Write a solution to a file.

### 16.2.2 Solutions.

Obtain or define a solution.

**MSK_deletesolution** (page 279)
Undefine a solution and free the memory it uses.

**MSK_getdualobj** (page 290)
Obtains the dual objective value.

**MSK_getprimalobj** (page 302)
Obtains the primal objective value.

**MSK_getreducedcosts** (page 304)
Obtains the difference of slx-sux for a sequence of variables.

**MSK_getsolution** (page 305)
Obtains the complete solution.

**MSK_getsolutioni** (page 306)
Obtains the solution for single constraint or variable.

**MSK_getsolutionincallback** (page 307)
Obtains the whole or a part of the solution from with the progress callback function.

**MSK_getsolutioninf** (page 308)
Obtains information about a solution.

**MSK_getsolutionslice** (page 309)
Obtains slice of the solution.

**MSK_getsolutionstatus** (page 310)
Obtains information about the problem and solution statuses.

**MSK_getsolutionstatuskeyslice** (page 311)
Obtains a slice of the solution status keys.

**MSK_makesolutionstatusunknown** (page 318)
Set the solution status to unknown.

**MSK_putsolution** (page 338)
Inserts a solution.

**MSK_putsolutioni** (page 339)
Sets the primal and dual solution information for a single constraint or variable.

**MSK_readsolution** (page 343)
Reads a solution from a file.

**MSK_solstatostr** (page 348)
Obtains a solution status string.

**MSK_solutiondef** (page 348)
Checks whether a solution defined.

**MSK_solutionsummary** (page 349)
Prints a short summary about a solution.

**MSK_undefsolution** (page 352)
Undefines a solution.

**MSK_writedata** (page 353)
Write problem data to a file.

### 16.2.3 Callbacks (put/get).

Manipulating callbacks.

**MSK_getcallbackfunc** (page 287)
Obtains the call-back function and the associated user handle.

**MSK_getnlfunc** (page 298)
Get nonlinear callback functions.

**MSK_getsolutionincallback** (page 307)
Obtains the whole or a part of the solution from with the progress callback function.

**MSK_linkfunctoenvstream** (page 256)
Connects a user defined function to a stream.

**MSK_linkfunctotaskstream** (page 318)
Connects a user defined function to a task stream.

**MSK_putcallbackfunc** (page 328)
Input the progress call back function.

**MSK_putctrlcfunc** (page 258)
Set a user defined function which is called when ctrl-c is pressed.

**MSK_putexitfunc** (page 259)
Inputs a user defined exit function which is called in case of fatal errors.

**MSK_putnlfunc** (page 334)
Input of nonlinear function information.

**MSK_putresponsefunc** (page 338)
Inputs a user defined error callback function.

**MSK_unlinkfuncfromenvstream** (page 261)
Disconnects a user defined function from a stream.

**MSK_unlinkfuncfromtaskstream** (page 352)
Disconnects a user defined function from a task stream.

### 16.2.4 Memory allocation and deallocation.

Memory allocation and deallocation.

**MSK_callocdbgenv** (page 250)
A replacement for the system function `callocenv`.

**MSK_callocdbgtask** (page 250)
A replacement for the system function `calloc`.

### 16.2.5 Change problem specification.

Input or change problem specification

**MSK_inputdata** (page 315)
Input the linear part of an optimization task in one function call.

**MSK_putaij** (page 323)
Changes a coefficient in $A$.

**MSK_putaijlist** (page 324)
Changes one or more coefficients in $A$.

**MSK_putavec** (page 324)
Replaces all elements in one rows or columns in $A$ by new values.

**MSK_putaveclist** (page 325)
Replaces all elements in one or more rows or columns in $A$ by new values.

**MSK_putbound** (page 326)
Changes the bound for either one constraint or one variable.

**MSK_putboundlist** (page 326)
Changes the bounds of constraints or variables.

**MSK_putboundslice** (page 327)
Modifies bounds.

**MSK_putcfix** (page 328)
Replaces the fixed term in the objective.

**MSK_putcj** (page 329)
Modifies a part of $c$.

**MSK_putclist** (page 329)
Modifies a part of $c$.

**MSK_putcone** (page 329)
Replaces a conic constraint with a new conic constraint.

**MSK_putobjsense** (page 334)
Set the objective sense.

**MSK_putqcon** (page 335)
Replaces all quadratic terms in constraints.

**MSK_putqconk** (page 336)
Replaces all quadratic terms in a single constraint.

**MSK_putqobj** (page 337)
Replaces all quadratic terms in the objective.

**MSK_putqobjij** (page 338)
Replaces one of the coefficients in the quadratic term in the objective.

**MSK_putvartype** (page 341)
Sets the variable type of one variable.

### 16.2.6 Delete problem elements (variables,constraints,cones).

Functionality for deleting problem elements such as variables, constraints or cones.



### 16.2.7 Add problem elements (variables,constraints,cones).

Functionality for adding problem elements such as variables, constraints or cones.



### 16.2.8 Inspect problem specification.

Functionality for inspecting the problem specification ($A$,$Q$, bounds, objective e.t.c).

**MSK_getc** (page 287)
Obtains all objective coefficients $c$.

**MSK_getcfix** (page 288)
Obtains the fixed term in the objective.

**MSK_getcone** (page 288)
Obtains a conic constraint.

**MSK_getconeinfo** (page 288)
Obtains information about a conic constraint.

**MSK_getcslice** (page 289)
Obtains a part of $c$.

**MSK_getnumanz** (page 298)
Obtains the number of non-zeros in $A$.

**MSK_getnumcon** (page 299)
Obtains the number of constraints.

**MSK_getnumcone** (page 299)
Obtains the number of cones.

**MSK_getnumconemem** (page 299)
Obtains the number of members in a cone.

**MSK_getnumintvar** (page 299)
Obtains the number of integer constrained variables.

**MSK_getnumqconnz** (page 300)
Obtains the number of nonzero quadratic terms in a constraint.

**MSK_getnumqobjnz** (page 300)
Obtains the number of nonzero quadratic terms in the objective.

**MSK_getnumvar** (page 301)
Obtains the number of variables.

**MSK_getobjsense** (page 301)
Get the objective sense.

**MSK_getprobtype** (page 302)
Obtains the problem type.

**MSK_getqconk** (page 303)
Obtains all the quadratic terms in a constraint.

**MSK_getqobj** (page 303)
Obtains all the quadratic terms in the objective.

**MSK_getqobjij** (page 304)
Obtains one coefficient in the quadratic term of the objective

**MSK_getvartype** (page 314)
Gets the variable type of one variable.

**MSK_getvartypelist** (page 314)
Obtains the variable type for one or more variables..

### 16.2.9 Conic constraints.

Functionality related to conic terms in the problem.

**MSK_appendcone** (page 273)
Appends a new cone constraint to the problem.

**MSK_getcone** (page 288)
Obtains a conic constraint.

**MSK_getconeinfo** (page 288)
Obtains information about a conic constraint.

**MSK_getnumcone** (page 299)
Obtains the number of cones.

**MSK_putcone** (page 329)
Replaces a conic constraint with a new conic constraint.

**MSK_removecone** (page 346)
Remove a conic constraint from the problem.

### 16.2.10 Bounds.

Functionality related to changing or inspecting bounds on variables or constraints.

**MSK_chgbound** (page 278)
Changes the bounds for one constraint or variable.

**MSK_getbound** (page 286)
Obtains bound information for one constraint or variable.

**MSK_getboundslice** (page 287)
Obtains bounds.

**MSK_putbound** (page 326)
Changes the bound for either one constraint or one variable.

**MSK_putboundlist** (page 326)
Changes the bounds of constraints or variables.

**MSK_putboundslice** (page 327)
Modifies bounds.

### 16.2.11 Task initialization and deletion.

Task initialization and deletion.

**MSK_deletetask** (page 280)
Deletes an optimization task.

**MSK_makeemptytask** (page 257)
Creates a new and empty optimization task.

**MSK_maketask** (page 258)
Creates a new optimization task.

### 16.2.12 Error handling.

Error handling.

**MSK_exceptiontask** (page 281)
Echo a response code to a task stream.

**MSK_getcodedisc** (page 254)
Obtains a short description of the response code.

**MSK_getresponseclass** (page 254)
Obtain the class of a response code.

**MSK_putresponsefunc** (page 338)
Inputs a user defined error callback function.

### 16.2.13 Output stream functions.

Output stream functions.

**MSK_echoenv** (page 252)
Sends a message to a given environment stream.

**MSK_echointro** (page 252)
Prints a short intro to message stream.

**MSK_echotask** (page 281)
Prints a format string to a task stream.

**MSK_exceptiontask** (page 281)
Echo a response code to a task stream.

**MSK_linkfiletoenvstream** (page 256)
Directs all output from a stream to a file.

### 16.2.14 Objective function.

Change or inspect objective function.

**MSK_getprimalobj** (page 302)
Obtains the primal objective value.

**MSK_getqobj** (page 303)
Obtains all the quadratic terms in the objective.

**MSK_getqobjij** (page 304)
Obtains one coefficient in the quadratic term of the objective

**MSK_putcfix** (page 328)
Replaces the fixed term in the objective.

**MSK_putcj** (page 329)
Modifies a part of $c$.

**MSK_putclist** (page 329)
Modifies a part of $c$.

**MSK_putobjsense** (page 334)
Set the objective sense.

**MSK_putqobj** (page 337)
Replaces all quadratic terms in the objective.

**MSK_putqobjij** (page 338)
Replaces one of the coefficients in the quadratic term in the objective.

### 16.2.15 Inspect statistics from the optimizer.

Inspect statistics from the optimizer.

**MSK_appendstat** (page 275)
Appends a record the statistics file.

**MSK_getdouinf** (page 290)
Obtains a double information item.

**MSK_getinfindex** (page 291)
Obtains the index of a named information item.

**MSK_getinfmax** (page 292)
Obtains the maximum index of an information of a given type `inftype` plus 1.

**MSK_getinfname** (page 292)
Obtains the name of an information item.

**MSK_getintinf** (page 292)
Obtains an integer information item.

**MSK_getnadouinf** (page 295)
Obtains a double information item.

**MSK_getnaintinf** (page 296)
Obtains an integer information item.

**MSK_getnaintparam** (page 296)
Obtains an integer parameter.

**MSK_startstat** (page 350)
Starts the statistics file.

**MSK_stopstat** (page 350)
Stops the statistics file.

### 16.2.16 Parameters (set/get).

Setting and inspecting solver parameters.

**MSK_getdouparam** (page 290)
Obtains a double parameter.

**MSK_getintparam** (page 293)
Obtains an integer parameter.

**MSK_getnadouparam** (page 295)
Obtains a double parameter.

**MSK_getnastrparam** (page 297)
Obtains a string parameter.

**MSK_getnastrparamal** (page 297)
Obtains the value of a string parameter.

**MSK_getnumparam** (page 300)
Obtains the number of parameters of a given type.

**MSK_getparammax** (page 301)
Obtains the maximum index of a parameter of a given type plus 1.

**MSK_getparamname** (page 302)
Obtains the name of a parameter.

**MSK_getstrparam** (page 311)
Obtains the value of a string parameter.

**MSK_getstrparamal** (page 312)
Obtains the value a string parameter.

**MSK_getsymbcondim** (page 255)
Obtains dimensional information for the defined symbolic constants.

**MSK_iparvaltosymnam** (page 255)
Obtains the symbolic name corresponding to a value that can be assigned to a integer parameter.

**MSK_isdouparname** (page 316)
Checks a double parameter name.

**MSK_isintparname** (page 317)
Checks an integer parameter name.

**MSK_isstrparname** (page 317)
Checks a string parameter name.

**MSK_putdouparam** (page 330)
Sets a double parameter.

**MSK_putintparam** (page 330)
Sets an integer parameter.

**MSK_putnadouparam** (page 332)
Sets a double parameter.

**MSK_putnaintparam** (page 333)
Sets an integer parameter.

**MSK_putnastrparam** (page 333)
Sets a string parameter.

**MSK_putparam** (page 335)
Modifies the value of parameter.

**MSK_putstrparam** (page 340)
Sets a string parameter.

**MSK_setdefaults** (page 348)
Resets all parameters values.

**MSK_symnamtovalue** (page 261)
Obtains the value corresponding to a symbolic name defined by MOSEK.

**MSK_whichparam** (page 352)
Checks a parameter name.

### 16.2.17 Naming.

Functionality related to naming.

**MSK_getconname** (page 289)
Obtains a name of a constraint.

**MSK_getmaxnamelen** (page 293)
Obtains the maximum length of any objective, constraint, variable, or cone name.

**MSK_getname** (page 296)
Obtains the name assigned to a constraint or a variable.

**MSK_getnameindex** (page 297)
Checks whether a name has been assigned and returns the index corresponding to the name.

**MSK_getobjname** (page 301)
Obtains the name assigned to the objective function.

**MSK_gettaskname** (page 312)
Obtains the task name.

**MSK_getvarname** (page 314)
Obtains a name of a variable.

**MSK_putname** (page 333)
Assigns the name `name` to a problem item such as a constraint.

**MSK_putobjname** (page 334)
Assigns a new name to the objective.

**MSK_puttaskname** (page 341)
Assigns a new name to the task.

### 16.2.18 Preallocating space for problem data.

Functionality related to preallocating space for problem data.

**MSK_getmaxnumanz** (page 293)
Obtains number of preallocated non-zeros for $A$.

**MSK_getmaxnumcon** (page 293)
Obtains the number of preallocated constraints in the optimization task.

**MSK_getmaxnumcone** (page 294)
Obtains the number of preallocated cones in the optimization task.

**MSK_getmaxnumqnz** (page 294)
Obtains number of preallocated non-zeros for $Q$ (both objective and constraints).

**MSK_getmaxnumvar** (page 294)
Obtains the maximum number variables allowed.

**MSK_putmaxnumanz** (page 330)
The function changes the size of the preallocated storage for $A$.

**MSK_putmaxnumcon** (page 331)
Sets the number of preallocated constraints in the optimization task.

**MSK_putmaxnumcone** (page 331)
Sets the number of preallocated constraints in the optimization task.

**MSK_putmaxnumqnz** (page 332)
The function changes the size of the preallocated storage for $Q$.

**MSK_putmaxnumvar** (page 332)
Sets the number of preallocated variables in the optimization task.

### 16.2.19 Integer variables.

Functionality related to integer variables.

**MSK_getnumintvar** (page 299)
Obtains the number of integer constrained variables.

**MSK_getvarbranchdir** (page 313)
Obtains the branching direction for a variable.

**MSK_getvarbranchorder** (page 313)
Obtains the branching priority for a variable.

**MSK_getvarbranchpri** (page 313)
Obtains the branching priority for a variable.

**MSK_getvartype** (page 314)
Gets the variable type of one variable.

**MSK_getvartypelist** (page 314)
Obtains the variable type for one or more variables..

**MSK_putvarbranchorder** (page 341)
Assigns a branching priority and direction to a variable.

**MSK_putvartype** (page 341)
Sets the variable type of one variable.

**MSK_putvartypelist** (page 342)
Sets the variable type for one or more variables.

### 16.2.20 Quadratic terms.

Functionality related to quadratic terms.

**MSK_getqconk** (page 303)
Obtains all the quadratic terms in a constraint.

**MSK_getqobj** (page 303)
Obtains all the quadratic terms in the objective.

**MSK_getqobjij** (page 304)
Obtains one coefficient in the quadratic term of the objective

**MSK_putqcon** (page 335)
Replaces all quadratic terms in constraints.

### 16.2.21 Diagnosing infeasibility.

Functions for diagnosing infeasibility.



### 16.2.22 Optimization.

Functions for optimization.



### 16.2.23 Sensitivity analysis.

Functions for sensitivity analysis.

### 16.2.24 Testing data validity.

Functions for testing data validity.

**MSK_checkconvexity** (page 277)
Checks if a quadratic optimization problem is convex.

### 16.2.25 Solving with the basis.

Functions for solving linear systems with the basis matrix.

**MSK_initbasissolve** (page 315)
This function must be called immediately before the first usage of the function `MSK_solvewithbasis`.

**MSK_solvewithbasis** (page 349)
Solve a linear equation system involving a basis matrix.

### 16.2.26 Initialization of environment.

Creation and initialization of environment.

**MSK_deleteenv** (page 252)
Deletes the MOSEK environment.

**MSK_initenv** (page 255)
Initialize a MOSEK environment.

**MSK_makeenv** (page 257)
Creates a new MOSEK environment.

**MSK_putlicensedefaults** (page 260)
Set defaults used by the license manager.

### 16.2.27 Change $A$.

Change elements in the coefficient ($A$) matrix.

**MSK_appendcons** (page 274)
Appends one or more constraints and specify bounds and $A$ coefficients.

**MSK_appendvars** (page 275)
Appends one or more variables and specify bounds on variables, $c$ coefficients and $A$ coefficients.

**MSK_commitchanges** (page 279)
It will commit all cached problem changes.

**MSK_putaij** (page 323)
Changes a coefficient in $A$.

**MSK_putaijlist** (page 324)
Changes one or more coefficients in $A$.

**MSK_putavec** (page 324)
Replaces all elements in one rows or columns in $A$ by new values.

**MSK_putaveclist** (page 325)
Replaces all elements in one or more rows or columns in $A$ by new values.

## 16.3 Mosek Env

**Description:**

A Mosek Environment

### 16.3.1 Methods

- MSK_callocdbgenv

**Syntax:**

```
void * MSK_callocdbgenv (
    MSKenv_t env,
    MSKCONST size_t number,
    MSKCONST size_t size,
    MSKCONST char * file,
    MSKCONST unsigned line);
```

**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 * 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 * 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 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 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 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 in general a good idea to call the function `MSK_unlinkfuncfromenvstream` for each stream that has have had function linked to it.

- `MSK_echoenv`

**Syntax:**

```
MSKrescodee MSK_echoenv (
    MSKenv_t env,
    MSKstreamtypee whichstream,
    MSKCONST char * format,
    ...);
```

**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 MSK_echointro (
    MSKenv_t env,
    MSKintt longver);
```

**Arguments:**

`env` **(input)** The MOSEK environment.

`longver` **(input)** If nonzero, then the intro is slightly longer.

**Description:** Prints a intro to message stream.

- `MSK_freedbgenv`

**Syntax:**

```
void 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:** Free space allocated by a MOSEK function. Must not be applied to the MOSEK environment and task.

- `MSK_freeenv`

**Syntax:**

```
void MSK_freeenv (
    MSKenv_t env,
    MSKCONST void * buffer);
```

**Arguments:**

`env` **(input)** The MOSEK environment.

`buffer` **(input)** A pointer.

**Description:** Free space allocated by a MOSEK function. Must not be applied to the MOSEK environment and task.

- `MSK_getbuildinfo`

**Syntax:**

```
MSKrescodee MSK_getbuildinfo (
    char * buildstate,
    char * builddate,
    char * buildtool);
```

**Arguments:**

buildstate **(output)** State of binaries i.e. a debug, candidate, or final release build.

builddate **(output)** Date the binaries was build.

buildtool **(output)** Tool(s) used to build the binaries.

**Description:** Obtains build information.

- MSK_getcodedisc

**Syntax:**

```
MSKrescodee 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 a response code code.

- MSK_getglbdllname

**Syntax:**

```
MSKrescodee 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 MSK_getresponseclass (
    MSKrescodee r,
    MSKrescodetypee * rc);
```

**Arguments:**

r **(input)** A response code indicating the status of function call.

rc **(output)** The return response class

**Description:** Obtain the class of a response code.

- `MSK_getsymbcondim`

  **Syntax:**

  ```
  MSKrescodee MSK_getsymbcondim (
      MSKenv_t env,
      MSKintt * num,
      size_t * maxlen);
  ```

  **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 name of any symbolic constant.

- `MSK_getversion`

  **Syntax:**

  ```
  MSKrescodee MSK_getversion (
      MSKintt * major,
      MSKintt * minor,
      MSKintt * build,
      MSKintt * revision);
  ```

  **Arguments:**

  `major` **(output)** Major version number. Only modified if a non-null pointer.

  `minor` **(output)** Minor version number. Only modified if a non-null pointer.

  `build` **(output)** Build number. Only modified if a non-null pointer.

  `revision` **(output)** Revision number. Only modified if a non-null pointer.

  **Description:** Obtains information about the version of MOSEK.

- `MSK_initenv`

  **Syntax:**

  ```
  MSKrescodee MSK_initenv (MSKenv_t env)
  ```

  **Arguments:**

  `env` **(input)** The MOSEK environment.

  **Description:** This function initializes the MOSEK environment, for instance the function contacts the license server. Error messages from the license manager can be captured by linking to the environment message stream.

- `MSK_iparvaltosymnam`

  **Syntax:**

```
MSKrescodee 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 a integer parameter.

- `MSK_isinfinity`

**Syntax:**

```
MSKbooleant MSK_isinfinity (MSKrealt value)
```

**Arguments:**

`value`

**Description:** Return true if `value` considered infinity by MOSEK

- `MSK_linkfiletoenvstream`

**Syntax:**

```
MSKrescodee MSK_linkfiletoenvstream (
    MSKenv_t env,
    MSKstreamtypee whichstream,
    MSKCONST char * filename,
    MSKintt append);
```

**Arguments:**

`env` **(input)** The MOSEK environment.

`whichstream` **(input)** Index of the stream.

`filename` **(input)** Sends all output to the stream `whichstream` to the file named `filename`.

`append` **(input)** If this argument is nonzero, then the output is append to the file.

**Description:** Direct all output to a stream to a file.

- `MSK_linkfunctoenvstream`

**Syntax:**

```
MSKrescodee 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 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 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 DLL/shared libraries that MOSEK are loading.

`MSK_putlicensedefaults` Set defaults used by the license manager.

MSK_putcpudefaults Set defaults default CPU type and cache sizes.

- MSK_maketask

**Syntax:**

```
MSKrescodee MSK_maketask (
    MSKenv_t env,
    MSKintt maxnumcon,
    MSKintt maxnumvar,
    MSKtask_t * task);
```

**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 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:** Set defaults CPU type and cache sizes. This function should be called before MSK_initenv.

- MSK_putctrlcfunc

**Syntax:**

```
MSKrescodee 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 nonzero value, then MOSEK terminates the optimization process and returns with the return code `MSK_RES_TRM_USER_BREAK`.

Note the function is only called if the parameter `MSK_IPAR_CHECK_CTRL_C` is set to `MSK_ON`.

- `MSK_putdllpath`

**Syntax:**

```
MSKrescodee 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 identical to `NULL`, then MOSEK assumes that operating system can locate the libraries.

**Description:** Sets the path to DLL/shared libraries that MOSEK are loading. If needed, then it should normally be called before `MSK_initenv`.

- `MSK_putexitfunc`

**Syntax:**

```
MSKrescodee MSK_putexitfunc (
    MSKenv_t env,
    MSKexitfunc exitfunc,
    MSKuserhandle_t handle);
```

**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 the case MOSEK has a fatal error, then an exit function is called. The exit function should terminate MOSEK. In general it is not necessary to define an exit function.

- `MSK_putkeepdlls`

**Syntax:**

```
MSKrescodee MSK_putkeepdlls (
    MSKenv_t env,
    MSKintt keepdlls);
```

**Arguments:**

`env` **(input)** Size of the L2 cache.

`keepdlls` **(input)** Controls whether explicit loaded DLLs should be kept.

**Description:** Controls whether explicit loaded DLLs should be kept even after they no longer are in use.

- `MSK_putlicensedefaults`

**Syntax:**

```
MSKrescodee MSK_putlicensedefaults (
    MSKenv_t env,
    MSKCONST char * licensefile,
    MSKCONST MSKintt * licensebuf,
    MSKintt licwait,
    MSKintt licdebug);
```

**Arguments:**

`env` **(input)** The MOSEK environment.

`licensefile` **(input)** A NULL pointer 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 nonzero, 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 nonzero, then MOSEK will print debug info regarding the license checkout.

**Description:** Set defaults used by the license manager. This function should be called before `MSK_initenv`.

- `MSK_replacefileext`

**Syntax:**

```
void 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 * 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 * 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 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 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 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 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.

## 16.4 Mosek Task

**Description:**

A Mosek Optimization task

### 16.4.1 Methods

- MSK_append

  **Syntax:**

```
MSKrescodee MSK_append (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKintt num);
```

  **Arguments:**

  task **(input)** An optimization task.

  accmode **(input)** Determine if constraints (MSK_ACC_CON) or variables (MSK_ACC_VAR) are appended.

  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. Note MOSEK will automatically expand the problem dimension to accommodate the additional constraints and variables.

  **See also:**

  MSK_remove The function removes a number of constraints or variables.

- MSK_appendcone

**Syntax:**

```
MSKrescodee MSK_appendcone (
    MSKtask_t task,
    MSKconetypee conetype,
    MSKrealt conepar,
    MSKintt nummem,
    MSKCONST MSKidxt * submem);
```

**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 specified by the argument `submem`. 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 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 (16.5).

`aptre` **(input)** See (16.5).

`asub` **(input)** Variables subscripts of the new $A$ coefficients. See (16.5).

`aval` **(input)** $A$ coefficients of the new constraints. See (16.5).

`bkc` **(input)** Bound keys for constraints to be appended. See (16.4).

blc **(input)** Lower bounds on constraints to be appended. See (16.4).

buc **(input)** Upper bounds on constraints to be appended. See (16.4).

**Description:** The function appends one or more constraints to the optimization task. The bounds and $A$ are modified as follows

$$\begin{array}{rcl} l^c_{\mathtt{numcon+k}} & = & \mathtt{blc}[\mathtt{k}], \quad \mathtt{k} = 0,\ldots,\mathtt{num}-1, \\ u^c_{\mathtt{numcon+k}} & = & \mathtt{buc}[\mathtt{k}], \quad \mathtt{k} = 0,\ldots,\mathtt{num}-1, \end{array} \tag{16.4}$$

and

$$a_{\mathtt{numcon+k},\mathtt{asub[l]}} = \mathtt{aval}[\mathtt{l}], \quad k = 0,\ldots,\mathtt{num}-1, \quad l = \mathtt{aptrb}[\mathtt{k}],\ldots,\mathtt{aptre}[\mathtt{k}]-1. \tag{16.5}$$

**See also:**

MSK_putmaxnumcon Sets the number of preallocated constraints in the optimization task.

- MSK_appendstat

**Syntax:**

```
MSKrescodee MSK_appendstat (MSKtask_t task)
```

**Arguments:**

task **(input)** An optimization task.

**Description:** Appends a record to the statistics file.

- MSK_appendvars

**Syntax:**

```
MSKrescodee 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 (16.6).

aptrb **(input)** See (16.7).

aptre **(input)** See (16.7).

asub **(input)** Constraint subscripts of the $A$ coefficients to be added. See (16.7).

`aval` **(input)** The $A$ coefficients corresponding to appended variables. See (16.7).

`bkx` **(input)** Bound Keys on variables to be appended. See (16.6).

`blx` **(input)** Lower bounds on variables to be appended. See (16.6).

`bux` **(input)** Upper bounds on variables to be appended. See (16.6).

**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

$$\begin{array}{rcll} c_{\texttt{numvar+k}} & = & \texttt{cval}[\texttt{k}], & \texttt{k} = 0, \ldots, \texttt{num} - 1, \\ l^x_{\texttt{numvar+k}} & = & \texttt{blx}[\texttt{k}], & \texttt{k} = 0, \ldots, \texttt{num} - 1, \\ u^x_{\texttt{numvar+k}} & = & \texttt{bux}[\texttt{k}], & \texttt{k} = 0, \ldots, \texttt{num} - 1, \end{array} \tag{16.6}$$

and

$$a_{\texttt{asub}[\texttt{l}],\texttt{numvar+k}} = \texttt{aval}[\texttt{l}], \quad k = 0, \ldots, \texttt{num} - 1, \quad l = \texttt{aptrb}[\texttt{k}], \ldots, \texttt{aptre}[\texttt{k}] - 1 \tag{16.7}$$

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 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 a identifier string corresponding to a bound key.

- `MSK_callbackcodetostr`

**Syntax:**

```
MSKrescodee MSK_callbackcodetostr (
    MSKcallbackcodee code,
    char * callbackcodestr);
```

**Arguments:**

`code` **(input)** A callback code.

`callbackcodestr` **(output)** String corresponding to the callback code .

**Description:** Obtains a the string representation of a corresponding to a callback code.

- `MSK_calloctask`

**Syntax:**

```
void * 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 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 MSK_checkdata (MSKtask_t task)
```

**Arguments:**

`task` **(input)** An optimization task.

**Description:** Checks the data of the optimization task.

- `MSK_checkmemtask`

**Syntax:**

```
MSKrescodee MSK_checkmemtask (
    MSKtask_t task,
    MSKCONST char * file,
    MSKintt line);
```

**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 MSK_chgbound (
    MSKtask_t task,
    MSKaccmodee con,
    MSKidxt i,
    MSKintt lower,
    MSKintt finite,
    MSKrealt value);
```

**Arguments:**

`task` **(input)** An optimization task.

`con` **(input)** Defines if bounds for constraints (`MSK_ACC_CON`) or bounds for variables (`MSK_ACC_VAR`) are changed.

`i` **(input)** Index of the constraint or variable for which the bounds should be changed.

`lower` **(input)** If nonzero, then the lower bound is changed. Otherwise the upper bound is changed.

`finite` **(input)** If nonzero, then `value` is assumed to be finite.

`value` **(input)** New value for the bound.

**Description:** Changes the bounds for one constraint or variable. If `con` equals `MSK_ACC_CON`, then the bounds are changed for a bound on a constraint. If `con` equals `MSK_ACC_VAR` the bounds is changed for a variable.

If `lower` is nonzero, 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

$$\text{new upper bound} = \begin{cases} \infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Note this function automatically updates the bound key for bound.

**See also:**

`MSK_putbound` Changes the bound for either one constraint or one variable.

`MSK_DPAR_DATA_TOL_BOUND_INF`

`MSK_DPAR_DATA_TOL_BOUND_WRN`

- `MSK_clonetask`

**Syntax:**

```
MSKrescodee MSK_clonetask (
    MSKtask_t task,
    MSKtask_t * clonedtask);
```

**Arguments:**

`task` **(input)** An optimization task.

`clonedtask` **(output)** The cloned task i.e. an clone of `task`.

**Description:** Creates a clone (or copy if you wish) of an existing task. Callback functions are not included in the cloned task. A task that has nonlinear function callbacks cannot be cloned.

- `MSK_commitchanges`

**Syntax:**

```
MSKrescodee MSK_commitchanges (MSKtask_t task)
```

**Arguments:**

`task` **(input)** An optimization task.

**Description:** It will commit all cached problem changes to the task. It is never really required to call this function.

- `MSK_conetypetostr`

**Syntax:**

```
MSKrescodee MSK_conetypetostr (
    MSKtask_t task,
    MSKconetypee conetype,
    char * str);
```

**Arguments:**

`task` **(input)** An optimization task.

`conetype` **(input)** Cone type.

`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 MSK_deletesolution (
    MSKtask_t task,
    MSKsoltypee whichsol);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichsol` **(input)** The solution index which can take the values listed in Appendix 19.42.

**Description:** Undefine a solution and free the memory it uses.

- `MSK_deletetask`

**Syntax:**

```
MSKrescodee MSK_deletetask (MSKtask_t * task)
```

**Arguments:**

`task` **(input/output)** An optimization task.

**Description:** Deletes a task.

- `MSK_dualsensitivity`

**Syntax:**

```
MSKrescodee 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 $\beta_1$ for the coefficient with index `subj`[j].

`rightrangej` **(output)** `leftrangej`[$j$] is the right range $\beta_2$ for the coefficient with index `subj`[j].

**Description:** Calculate 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 such that e.g `leftprice`[$j$] is the left shadow price of the objective coefficient with index `subj`[$j$].

The type of sensitivity analysis to performed (basis or optimal partition) is controlled by the parameter MSK_IPAR_SENSITIVITY_TYPE.

Example code can be found in section 13.5.

**See also:**

MSK_primalsensitivity Perform sensitivity analysis on bounds.

MSK_sensitivityreport Create a sensitivity report.

MSK_IPAR_SENSITIVITY_TYPE
MSK_IPAR_LOG_SENSITIVITY
MSK_IPAR_LOG_SENSITIVITY_OPT

- MSK_echotask

**Syntax:**

```
MSKrescodee MSK_echotask (
    MSKtask_t task,
    MSKstreamtypee whichstream,
    MSKCONST char * format,
    ...);
```

**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 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 18. Moreover, the corresponding response string listed in Appendix 18 is printed. It is the users responsibility to provide appropriate arguments for the response string listed in Appendix 18 too.

- MSK_freedbgtask

**Syntax:**

```
void 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:** Free space allocated by a MOSEK function. Must not be applied to the MOSEK environment and task.

- `MSK_freetask`

**Syntax:**

```
void MSK_freetask (
    MSKtask_t task,
    MSKCONST void * buffer);
```

**Arguments:**

`task` **(input)** An optimization task.

`buffer` **(input)** A pointer.

**Description:** Free space allocated by a MOSEK function. Must not be applied to the MOSEK environment and task.

- `MSK_getaij`

**Syntax:**

```
MSKrescodee MSK_getaij (
    MSKtask_t task,
    MSKidxt i,
    MSKidxt j,
    MSKrealt * aij);
```

**Arguments:**

`task` **(input)** An optimization task.

`i` **(input)** Row index of coefficient to be returned.

`j` **(input)** Column index of coefficient to be returned.

`aij` **(output)** The required coefficient $a_{i,j}$.

**Description:** Obtains a single coefficient in $A$.

- `MSK_getapiecenumnz`

**Syntax:**

```
MSKrescodee MSK_getapiecenumnz (
    MSKtask_t task,
    MSKidxt firsti,
    MSKidxt lasti,
    MSKidxt firstj,
    MSKidxt lastj,
    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 nonzero $A$ elements in the rectangular piece.

**Description:** Obtains the number nonzeros of 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 a an efficient way to obtain the number of nonzeros in one row or column. In that case use the function `MSK_getavecnumnz`.

**See also:**

`MSK_getavecnumnz` Obtains the number of nonzero elements in one row or column of $A$.

`MSK_getaslicenumnz` Obtains the number of nonzeros in a row or column slice of $A$.

- `MSK_getaslice`

**Syntax:**

```
MSKrescodee 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]`. On return `surp` has been decremented by the total number of nonzero elements in the obtained rows and columns.

`ptrb` **(output)** `ptrb[t]` is a index pointing to the first element in the $t$th row or column obtained.

`ptre` **(output)** `ptre[t]` is a index pointing to the last element plus one in the $t$th row or column obtained.

`sub` **(output)** Contains the row or columns 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 nonzeros in a row or column slice of $A$.

- MSK_getaslicenumnz

**Syntax:**

```
MSKrescodee 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 nonzeros in the slice.

**Description:** Obtains the number of nonzeros in a row or column slice of $A$.

- MSK_getaslicetrip

**Syntax:**

```
MSKrescodee 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 nonzero elements in the obtained rows and columns.

`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 $p^1$ by

$$p^1 = \texttt{maxnumnz} - \texttt{surp}[0]$$

when the function is called and $p^2$ by

$$p^2 = \texttt{maxnumnz} - \texttt{surp}[0],$$

where `surp[0]` is the value upon termination. Using this notation then

$$\texttt{val}[k] = a_{\texttt{subi[k]},\texttt{subj[k]}}, \quad k = p^1, \ldots, p^2 - 1.$$

**See also:**

`MSK_getaslicenumnz` Obtains the number of nonzeros in a row or column slice of $A$.

**Comments:**

- `MSK_getavec`

**Syntax:**

```
MSKrescodee MSK_getavec (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKintt * nzi,
    MSKidxt * subi,
    MSKrealt * vali);
```

**Arguments:**

`task` **(input)** An optimization task.

`accmode` **(input)** Determine if a column (`MSK_ACC_VAR`) or a row `MSK_ACC_CON` is requested.

`i` **(input)** Index of the row or column.

`nzi` **(output)** Number of nonzeros in the vector obtained.

`subi` **(output)** Index of the nonzeros in the vector obtained.

`vali` **(output)** Numerical values of the vector to be obtained.

**Description:** Obtains one row or column of $A$ in a sparse format. If accmode equals MSK_ACC_CON a row is returned and hence:

$$\texttt{vali}[k] = a_{i,\texttt{subi}[k]}, \quad k = 0, \ldots, \texttt{nzi}[0] - 1$$

If accmode equals MSK_ACC_VAR a column is returned, that is:

$$\texttt{vali}[k] = a_{\texttt{subi}[k],i}, \quad k = 0, \ldots, \texttt{nzi}[0] - 1.$$

- MSK_getavecnumnz

**Syntax:**

```
MSKrescodee MSK_getavecnumnz (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKintt * nzj);
```

**Arguments:**

task **(input)** An optimization task.

accmode **(input)** Defines whether nonzeros are counted by columns or by rows.

i **(input)** Index of the row or column.

nzj **(output)** Number of nonzeros in the $i$th row or column of $A$.

**Description:** Obtains the number of nonzero elements in one row or column of $A$.

- MSK_getbound

**Syntax:**

```
MSKrescodee MSK_getbound (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKboundkeye * bk,
    MSKrealt * bl,
    MSKrealt * bu);
```

**Arguments:**

task **(input)** An optimization task.

accmode **(input)** Defines whether bounds for constraints (MSK_ACC_CON) or variables MSK_ACC_VAR should be obtained.

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 bounds information for one constraint or variable.

- `MSK_getboundslice`

**Syntax:**

```
MSKrescodee MSK_getboundslice (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt first,
    MSKidxt last,
    MSKboundkeye * bk,
    MSKrealt * bl,
    MSKrealt * bu);
```

**Arguments:**

`task` **(input)** An optimization task.

`accmode` **(input)** Defines whether bounds for constraints (`MSK_ACC_CON`) or variables (`MSK_ACC_VAR`) should be obtained.

`first` **(input)** First index in the sequence.

`last` **(input)** Last index plus 1 in the sequence.

`bk` **(output)** Bound keys.

`bl` **(output)** Values for lower bounds.

`bu` **(output)** Values for upper bounds.

**Description:** Obtains bounds information for a sequence of variables or constraints.

- `MSK_getc`

**Syntax:**

```
MSKrescodee MSK_getc (
    MSKtask_t task,
    MSKrealt * c);
```

**Arguments:**

`task` **(input)** An optimization task.

`c` **(output)** Linear term in the objective.

**Description:** Obtains all objective coefficients $c$.

- `MSK_getcallbackfunc`

**Syntax:**

```
MSKrescodee 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 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 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)** Is the type of the cone.

conepar **(output)** Is the parameter of the cone.

nummem **(output)** Is the number of members in the cone.

submem **(output)** Variable subscripts of the cone members.

**Description:** Obtains a conic constraint.

- MSK_getconeinfo

**Syntax:**

```
MSKrescodee 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)** Is the type of the cone.

`conepar` **(output)** Is the parameter of the cone.

`nummem` **(output)** Is the number of members in the cone.

**Description:** Obtains information about a conic constraint.

- `MSK_getconname`

**Syntax:**

```
MSKrescodee MSK_getconname (
    MSKtask_t task,
    MSKidxt i,
    MSKCONST size_t maxlen,
    char * name);
```

**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 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 term in the objective.

**Description:** Obtains a sequence of elements in $c$.

- MSK_getdouinf

**Syntax:**

```
MSKrescodee MSK_getdouinf (
    MSKtask_t task,
    MSKdinfiteme whichdinf,
    MSKrealt * dvalue);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichdinf` **(input)** An double information item. See Section 19.11 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_getdouparam

**Syntax:**

```
MSKrescodee 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 MSK_getdualobj (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * dualobj);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichsol` **(input)** The solution index which can take the values listed in Appendix 19.42.

`dualobj` **(output)** Objective value corresponding to the dual solution.

**Description:** Obtains the current objective value of the dual problem for `whichsol`.

- MSK_getenv

**Syntax:**

```
MSKrescodee MSK_getenv (
    MSKtask_t task,
    MSKenv_t * env);
```

**Arguments:**

task **(input)** An optimization task.

env **(output)** The MOSEK environment.

**Description:** Obtains the environment used to create the task.

- MSK_getinfeasiblesubproblem

**Syntax:**

```
MSKrescodee MSK_getinfeasiblesubproblem (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKtask_t * inftask);
```

**Arguments:**

task **(input)** An optimization task.

whichsol **(input)** Which solution to use for determining the infeasible subproblem.

inftask **(output)** A new task containing the infeasible subproblem.

**Description:** Obtains a infeasible sub problem. The infeasible subproblem is a problem consisting of smaller subset of constraints such that the problem is still infeasible. For more information see section 11.

**See also:**

MSK_IPAR_INFEAS_PREFER_PRIMAL

MSK_relaxprimal Create a problem that finds the minimal change to the bounds that makes an infeasible problem feasible.

- MSK_getinfindex

**Syntax:**

```
MSKrescodee MSK_getinfindex (
    MSKtask_t task,
    MSKinftypee inftype,
    MSKCONST char * infname,
    MSKintt * infindex);
```

**Arguments:**

task **(input)** An optimization task.

inftype **(input)** Type of the information item.

infname **(input)** Name of the information item.

infindex **(output)**

**Description:** Obtains the index of a named information item.

- MSK_getinfmax

**Syntax:**

```
MSKrescodee 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 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 19.11 and Section 19.14 for the possible values.

`infname` **(output)** Name of the information item.

**Description:** Obtains the name of an information item.

- MSK_getintinf

**Syntax:**

```
MSKrescodee MSK_getintinf (
    MSKtask_t task,
    MSKiinfiteme whichiinf,
    MSKintt * ivalue);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichiinf` **(input)** An integer information type. See Section 19.14 for the possible values.

`ivalue` **(output)** The value of the required integer information item.

**Description:** Obtains an integer information item from the task information database.

- MSK_getintparam

**Syntax:**

```
MSKrescodee MSK_getintparam (
    MSKtask_t task,
    MSKiparame param,
    MSKintt * parvalue);
```

**Arguments:**

task **(input)** An optimization task.

param **(input)** Which parameter.

parvalue **(output)** Parameter value.

**Description:** Obtains the value of an integer parameter.

- MSK_getmaxnamelen

**Syntax:**

```
MSKrescodee 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 MSK_getmaxnumanz (
    MSKtask_t task,
    MSKlintt * maxnumanz);
```

**Arguments:**

task **(input)** An optimization task.

maxnumanz **(output)** Number of preallocated nonzero 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 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 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 MSK_getmaxnumqnz (
    MSKtask_t task,
    MSKintt * maxnumqnz);
```

**Arguments:**

`task` **(input)** An optimization task.

`maxnumqnz` **(output)** Number of nonzero elements preallocated in quadratic coefficient matrices.

**Description:** Obtains number of preallocated non-zeros for $Q$ (both objective and constraints). When this number of non-zeros is reached MOSEK will automatically allocate more space for $Q$.

- `MSK_getmaxnumvar`

**Syntax:**

```
MSKrescodee MSK_getmaxnumvar (
    MSKtask_t task,
    MSKintt * maxnumvar);
```

**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 MSK_getmemusagetask (
    MSKtask_t task,
    size_t * meminuse,
    size_t * maxmemuse);
```

**Arguments:**

`task` **(input)** An optimization task.

`meminuse` **(output)** Amount of memory/space used the `task` currently.

`maxmemuse` **(output)** Maximum amount of memory/space by used the `task` until now.

**Description:** Obtains information about the amount of memory use by a task.

- MSK_getnadouinf

**Syntax:**

```
MSKrescodee MSK_getnadouinf (
    MSKtask_t task,
    MSKCONST char * whichdinf,
    MSKrealt * dvalue);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichdinf` **(input)** An double information item. See Section 19.11 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 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 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 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 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 which can take the values listed in Appendix 19.30.

`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 item i.e. for instance a problem name.

**See also:**

MSK_getmaxnamelen Obtains the maximum length of any objective, constraint, variable, or cone name.

- MSK_getnameindex

**Syntax:**

```
MSKrescodee MSK_getnameindex (
    MSKtask_t task,
    MSKproblemiteme whichitem,
    MSKCONST char * name,
    MSKintt * asgn,
    MSKidxt * index);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichitem` **(input)** Problem item which can take the values listed in Appendix 19.30.

`name` **(input)** The name which should be checked.

`asgn` **(output)** Is nonzero if `name` is assigned.

`index` **(output)** If the name is assigned, assigned the required name.

**Description:** Checks whether a name has already been assigned to an item i.e. a constraint, a variable, or a cone. If the name has already been assigned, then the index of the item that name has been assigned to is returned.

- MSK_getnastrparam

**Syntax:**

```
MSKrescodee 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 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 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 callback functions. If NULL no nonlinear callback function exists.

- `MSK_getnumanz`

**Syntax:**

```
MSKrescodee MSK_getnumanz (
    MSKtask_t task,
    MSKlintt * numanz);
```

**Arguments:**

`task` **(input)** An optimization task.

`numanz` **(output)** Number of nonzero elements in $A$.

**Description:** Obtains the number of non-zeros in $A$.

- `MSK_getnumcon`

**Syntax:**

```
MSKrescodee 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 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 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 members in the cone.

**Description:** Obtains the number of members in a cone.

- `MSK_getnumintvar`

**Syntax:**

```
MSKrescodee 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 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 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 nonzero quadratic terms in a constraint.

- `MSK_getnumqobjnz`

**Syntax:**

```
MSKrescodee MSK_getnumqobjnz (
    MSKtask_t task,
    MSKlintt * numqonz);
```

**Arguments:**

`task` **(input)** An optimization task.

`numqonz` **(output)** Number of nonzero elements in $Q^o$.

**Description:** Obtains the number of nonzero quadratic terms in the objective.

- MSK_getnumvar

  **Syntax:**

  ```
  MSKrescodee 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 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 objective name.

  objname **(output)** Assigned the objective name.

  **Description:** Obtains the name assigned to the objective function.

- MSK_getobjsense

  **Syntax:**

  ```
  MSKrescodee MSK_getobjsense (
      MSKtask_t task,
      MSKobjsensee * sense);
  ```

  **Arguments:**

  task **(input)** An optimization task.

  sense **(output)** The returned objective sense.

  **Description:** Get the objective sense of the task.

  **See also:**

  MSK_putobjsense Set the objective sense.

- MSK_getparammax

  **Syntax:**

```
MSKrescodee 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 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 MSK_getprimalobj (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * primalobj);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichsol` **(input)** The solution index which can take the values listed in Appendix 19.42.

`primalobj` **(output)** Objective value corresponding to the primal solution.

**Description:** Obtains the primal objective value for a solution.

- `MSK_getprobtype`

**Syntax:**

```
MSKrescodee MSK_getprobtype (
    MSKtask_t task,
    MSKproblemtypee * probtype);
```

**Arguments:**

`task` **(input)** An optimization task.

`probtype` **(output)** The problem type.

**Description:** Obtains the problem type.

- `MSK_getqconk`

**Syntax:**

```
MSKrescodee MSK_getqconk (
    MSKtask_t task,
    MSKidxt k,
    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)** On entering the function it is assumed that last `qcsurp[0]` positions in `qcsubi`, `qcsubj`, and `qcval` are free. Hence, the quadratic terms are stored in this area. On return `qcsurp` is identical to the 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 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)** Is the length of the arrays `qosubi`, `qosubj`, and `qoval`.

`qosurp` **(input/output)** Initially `qosurp[0]` is the number of free positions at end of the arrays `qosubi`, `qosubj`, and `qoval`. On return `qosurp` is the updated number of free positions left in those arrays.

`numqonz` **(output)** Number of nonzero elements in $Q^o$.

`qosubi` **(output)** $i$ subscript for $q_{ij}^o$.

`qosubj` **(output)** $j$ subscript for $q_{ij}^o$.

`qoval` **(output)** Numerical value for $q_{ij}^o$.

**Description:** Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in `qosubi`, `qosubj`, and `qoval`.

- `MSK_getqobjij`

**Syntax:**

```
MSKrescodee MSK_getqobjij (
    MSKtask_t task,
    MSKidxt i,
    MSKidxt j,
    MSKrealt * qoij);
```

**Arguments:**

`task` **(input)** An optimization task.

`i` **(input)** $i$ index for the coefficient.

`j` **(input)** $j$ index for coefficient.

`qoij` **(output)** The required coefficient.

**Description:** Obtains one coefficient in the quadratic term of the objective i.e. obtains

$$q_{ij}^o.$$

- `MSK_getreducedcosts`

**Syntax:**

```
MSKrescodee MSK_getreducedcosts (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKidxt first,
    MSKidxt last,
    MSKrealt * redcosts);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichsol` **(input)** The solution index which can take the values listed in Appendix 19.42.

`first` **(input)** See formula (16.8) for the definition.

`last` **(input)** See formula (16.8) 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.

$$\texttt{redcosts}[j - \texttt{first} + 0] = (s_l^x)_j - (s_u^x)_j, \ j = \texttt{first}, \ldots, \texttt{last} - 1. \tag{16.8}$$

- `MSK_getsolution`

**Syntax:**

```
MSKrescodee MSK_getsolution (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKprostae * prosta,
    MSKsolstae * solsta,
    MSKstakeye * skc,
    MSKstakeye * skx,
    MSKstakeye * skn,
    MSKrealt * xc,
    MSKrealt * xx,
    MSKrealt * y,
    MSKrealt * slc,
    MSKrealt * suc,
    MSKrealt * slx,
    MSKrealt * sux,
    MSKrealt * snx);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichsol` **(input)** The solution index which can take the values listed in Appendix 19.42.

`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)** 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 programing. The primal problem is given by

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x. \end{array} \tag{16.9}$$

and the corresponding dual problem is

$$\begin{array}{lccc} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & & \end{array} \tag{16.10}$$

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`. Some possible values of `solsta` and their meaning 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 single constraint or variable.

`MSK_getsolutionslice` Obtains slice of the solution.

- `MSK_getsolutioni`

**Syntax:**

```
MSKrescodee MSK_getsolutioni (
    MSKtask_t task,
    MSKaccmodee accmode,
```

```
    MSKidxt i,
    MSKsoltypee whichsol,
    MSKstakeye * sk,
    MSKrealt * x,
    MSKrealt * sl,
    MSKrealt * su,
    MSKrealt * sn);
```

**Arguments:**

`task` **(input)** An optimization task.

`accmode` **(input)** If nonzero, then solution information for a constraint are obtained. Otherwise for a variable.

`i` **(input)** Index of the constraint or variable.

`whichsol` **(input)** The solution index which can take the values listed in Appendix 19.42.

`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 slice of the solution.

- `MSK_getsolutionincallback`

**Syntax:**

```
MSKrescodee MSK_getsolutionincallback (
    MSKtask_t task,
    MSKcallbackcodee where,
    MSKsoltypee whichsol,
    MSKprostae * prosta,
    MSKsolstae * solsta,
    MSKstakeye * skc,
    MSKstakeye * skx,
    MSKstakeye * skn,
    MSKrealt * xc,
    MSKrealt * xx,
    MSKrealt * y,
    MSKrealt * slc,
    MSKrealt * suc,
    MSKrealt * slx,
    MSKrealt * sux,
    MSKrealt * snx);
```

**Arguments:**

task **(input)** An optimization task.

where **(input)** The callback-key from the current callback

whichsol **(input)** The solution index which can take the values listed in Appendix 19.42.

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)** 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 callback. This function must only be called from a progress callback function.

This is an experimental feature. Please contact MOSEK support before using this function.

- MSK_getsolutioninf

**Syntax:**

```
MSKrescodee MSK_getsolutioninf (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKprostae * prosta,
    MSKsolstae * solsta,
    MSKrealt * primalobj,
    MSKrealt * maxpbi,
    MSKrealt * maxpcni,
    MSKrealt * maxpeqi,
    MSKrealt * maxinti,
    MSKrealt * dualobj,
    MSKrealt * maxdbi,
    MSKrealt * maxdcni,
    MSKrealt * maxdeqi);
```

**Arguments:**

task **(input)** An optimization task.

whichsol **(input)** The solution index which can take the values listed in Appendix 19.42.

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 MSK_getsolutionslice (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKsoliteme solitem,
    MSKidxt first,
    MSKidxt last,
    MSKrealt * values);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichsol` **(input)** The solution index which can take the values listed in Appendix 19.42.

`solitem` **(input)** What part of the solution should be retrieved.

`first` **(input)** Index of the first value in the slice.

`last` **(input)** Index of the last index+1 in the slice. I.e. if $xx[5,\ldots,9]$ is required `last` should be equal to 10.

`values` **(output)** The value 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 programing. The primal problem is given by

$$
\begin{array}{lccccc}
\text{minimize} & & & c^T x + c^f & & \\
\text{subject to} & l^c & \leq & Ax & \leq & u^c, \\
 & l^x & \leq & x & \leq & u^x.
\end{array}
\tag{16.11}
$$

and the corresponding dual problem is

$$
\begin{array}{llcl}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\
 & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\
\text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\
 & -y + s_l^c - s_u^c & = & 0, \\
 & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & &
\end{array}
\tag{16.12}
$$

Which part of the solution to return in `values` is determined based on the value of the argument `solitem`:

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:

$$\begin{array}{lccl} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & & \\ & +(l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x + s_n^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, \\ & s_n^x \in \mathcal{C}^* & & \end{array} \tag{16.13}$$

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 depend 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 single constraint or variable.

- `MSK_getsolutionstatus`

**Syntax:**

```
MSKrescodee MSK_getsolutionstatus (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKprostae * prosta,
    MSKsolstae * solsta);
```

**Arguments:**

`task` **(input)** An optimization task.

whichsol **(input)** The solution index which can take the values listed in Appendix 19.42.

prosta **(output)** Problem status.

solsta **(output)** Solution status.

**Description:** Obtains information about the problem and solution statuses.

- MSK_getsolutionstatuskeyslice

**Syntax:**

```
MSKrescodee MSK_getsolutionstatuskeyslice (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKsoltypee whichsol,
    MSKidxt first,
    MSKidxt last,
    MSKstakeye * sk);
```

**Arguments:**

task **(input)** An optimization task.

accmode **(input)** Defines whether problem data is accessed row-wise or column-wise.

whichsol **(input)** The solution index which can take the values listed in Appendix 19.42.

first **(input)** Index of the first value in the slice.

last **(input)** Index of the last index+1 in the slice. I.e. if $xx[5,\ldots,9]$ is required last should be equal to 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 single constraint or variable.

- MSK_getstrparam

**Syntax:**

```
MSKrescodee 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 a non NULL pointer, then the parameter value is stored in `parvalue`.

**Description:** Obtains the value of a string parameter.

- `MSK_getstrparamal`

**Syntax:**

```
MSKrescodee MSK_getstrparamal (
    MSKtask_t task,
    MSKsparame param,
    MSKCONST size_t numaddchr,
    MSKstring_t * value);
```

**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 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)** Maximum length allowed of name including terminating null char.

`name` **(output)** Name of the $i$th symbolic constant.

`value` **(output)** The corresponding value.

**Description:** Obtains the name and corresponding value for the $i$th symbolic constant.

- `MSK_gettaskname`

**Syntax:**

```
MSKrescodee 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 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 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 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 MSK_getvarname (
    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 variable.

**See also:**

`MSK_getmaxnamelen` Obtains the maximum length of any objective, constraint, variable, or cone name.

- `MSK_getvartype`

**Syntax:**

```
MSKrescodee 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 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 for one or more variables. I.e. variable `vartype[k]` is assigned the variable type of variable `subj[k]`.

- `MSK_initbasissolve`

**Syntax:**

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

**Arguments:**

`task` **(input)** An optimization task.

`basis` **(output)** This is an array of basis indexes which shows the ordering of the basic variables employed in MOSEK. If

$$\texttt{basis}[i] \leq \texttt{numcon} - 1,$$

then this implies that $x^c_{\texttt{basis}[i]}$ is in the basis at position $i$. Otherwise if $x_{\texttt{basis}[i]-\texttt{numcon}}$ is in the basis at position $i$.

**Description:** This function must be called before the first usage of the function `MSK_solvewithbasis`. The function initialize various internal data structures which are required by `MSK_solvewithbasis`.

Moreover, if the optimization task is modified between two calls of `MSK_solvewithbasis`, then `MSK_initbasissolve` should called again immediately before the second call to `MSK_solvewithbasis`.

- `MSK_inputdata`

**Syntax:**

```
MSKrescodee 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 term in the objective.

`cfix` **(input)** Fixed term in the objective.

`aptrb` **(input)** Pointer to the first element in columns of $A$. See (5.37).

`aptre` **(input)** Pointer to the last element + 1 in columns of $A$. See (5.37).

`asub` **(input)** Constraint subscripts. See (5.37).

`aval` **(input)** Constraint values. See (5.37).

`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:** The procedure is used to input the linear part of an optimization task. 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.

- `MSK_isdouparname`

**Syntax:**

```
MSKrescodee 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 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 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 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)** Sends all output to the stream `whichstream` to the file named `filename`.

`append` **(input)** If this argument is nonzero, then the output is append to the file.

**Description:** Direct all output to a task stream to a file.

- MSK_linkfunctotaskstream

**Syntax:**

```
MSKrescodee 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 MSK_makesolutionstatusunknown (
    MSKtask_t task,
    MSKsoltypee whichsol);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichsol` **(input)** The solution index which can take the values listed in Appendix 19.42.

**Description:** Set the solution status to unknown. Also all the status keys for the constraints and the variables are set to unknown.

- MSK_optimize

**Syntax:**

```
MSKrescodee MSK_optimize (MSKtask_t task)
```

**Arguments:**

`task` **(input)** An optimization task.

**Description:** Call 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 single constraint or variable.

`MSK_getsolutioninf` Obtains information about a solution.

MSK_IPAR_OPTIMIZER

- MSK_optimizeconcurrent

**Syntax:**

```
MSKrescodee MSK_optimizeconcurrent (
    MSKtask_t task,
    MSKCONST MSKtask_t * taskarray,
    MSKintt num);
```

**Arguments:**

`task` **(input)** An optimization task.

`taskarray` **(input)** An array of `num` tasks.

`num` **(input)** Length of `taskarray`

**Description:** Solves several instances of the same problem in parallel, with unique parameter settings for each task. The argument `task` contains the problem to be solved. `taskarray` is a pointer to a 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 and other solution information of the task which finished first with return code MSK_RES_OK or MSK_RES_TRM_USER_BREAK.

If all the concurrent optimizations finished with an error code different from MSK_RES_OK or MSK_RES_TRM_USER_BREAK, then 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 user defined parameters (`IPAR`, `DPAR` and `SPAR`) in `task` are copied to each of the tasks in `taskarray`. In particular this means that any solution in `task` is copied to the other tasks. Callback functions are not copied.
2. The tasks `task` and the `num` tasks in `taskarray` are started in parallel.
3. When a tasks finishes with return code MSK_RES_OK or MSK_RES_TRM_USER_BREAK it's solution is copied to `task` and all other tasks are stopped.

For an explained code example see section 10.3.2.

- MSK_primalsensitivity

**Syntax:**

```
MSKrescodee 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]` specify 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]` specify 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]`) on the constraint with index `subi[i]`.

`leftrangei` **(output)** `leftrangei[i]` is the left range for the upper/lower bound (indicated by `marki[i]`) on the constraint with index `subi[i]`.

`rightrangei` **(output)** `rightrangei[i]` is the right range for the upper/lower bound (indicated by `marki[i]`) on 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:** Calculate sensitivity information for bounds on variables or constraints.

For details on sensitivity analysis and the definition of *shadow price* and *linearity interval* see chapter 13.

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` then the upper bound of constraint `subi[i]` is analyzed if `marki[i] = MSK_MARK_LO` then the lower bound is analyzed. If `subi[i]` is an equality constraint, one can use either `MSK_MARK_LO` or `MSK_MARK_UP` to select the constraint for sensitivity analysis.

Consider the problem:

$$\begin{array}{lccc} \text{minimize} & x_1 + x_2 & & \\ \text{subject to} -1 \leq & x_1 - x_2 & \leq & 1, \\ & x_1 & = & 0, \\ & x_1 \geq 0, x_2 \geq 0 & & \end{array} \tag{16.14}$$

Suppose:

```
numi = 1;
subi = [0];
marki = [MSK_MARK_UP]
```

then
`leftpricei[0]`,`rightpricei[0]`,`leftrangei[0]`,`rightrangei[0]` will contain the sensitivity information for the upper bound on constraint 0 given by the expression:

$$x_1 - x_2 \leq 1 \tag{16.15}$$

Similarly the variables for which to performs sensitivity analysis is 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` then the upper bound of constraint `subi[j]` is analyzed if `markj[j] = MSK_MARK_LO` then the lower bound is analyzed. If `subi[j]` is an equality constraint, one can use either `MSK_MARK_LO` or `MSK_MARK_UP` to select the constraint for sensitivity analysis.

Example code can be found in section 13.5.

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` Perform sensitivity analysis on objective coefficients.

`MSK_sensitivityreport` Create a sensitivity report.

`MSK_IPAR_SENSITIVITY_TYPE`

`MSK_IPAR_LOG_SENSITIVITY`

`MSK_IPAR_LOG_SENSITIVITY_OPT`

- `MSK_printdata`

**Syntax:**

```
MSKrescodee MSK_printdata (
    MSKtask_t task,
    MSKstreamtypee whichstream,
    MSKidxt firsti,
    MSKidxt lasti,
    MSKidxt firstj,
    MSKidxt lastj,
    MSKidxt firstk,
    MSKidxt lastk,
    MSKintt c,
    MSKintt qo,
    MSKintt a,
    MSKintt qc,
    MSKintt bc,
    MSKintt bx,
    MSKintt vartype,
    MSKintt cones);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichstream` **(input)** Index of the stream.

`firsti` **(input)** Index of first constraint for data should be printed.

`lasti` **(input)** Index of last constraint plus 1 for which data is to be printed.

`firstj` **(input)** Index of first variable for data should be printed.

`lastj` **(input)** Index of last variable plus 1 for which data is to be printed.

`firstk` **(input)** Index of first cone for data should be printed.

`lastk` **(input)** Index of last cone plus 1 for which data is to be printed.

`c` **(input)** If nonzero, then $c$ is printed.

`qo` **(input)** If nonzero, then $Q^o$ is printed.

`a` **(input)** If nonzero, then $A$ is printed.

`qc` **(input)** If nonzero, then $Q^k$ is printed for the relevant constraints.

`bc` **(input)** If nonzero, then constraints bounds are printed.

`bx` **(input)** If nonzero, then variable bounds are printed.

`vartype` **(input)** If nonzero, then variable types are printed.

`cones` **(input)** If nonzero, then conic data are printed.

**Description:** Prints a part of the problem data to a stream. This function is normally used for debugging purpose only. I.e. to verify that the correct data has been inputted.

- `MSK_printparam`

**Syntax:**

```
MSKrescodee 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 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 explanatory string corresponding to a problem type.

- `MSK_prostatostr`

**Syntax:**

```
MSKrescodee 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 explanatory string corresponding to a problem status.

- `MSK_putaij`

**Syntax:**

```
MSKrescodee 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 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 rows or columns in $A$ by new values.

MSK_putaijlist Changes one or more coefficients in $A$.

**Comments:**

- MSK_putaijlist

**Syntax:**

```
MSKrescodee 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 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 coefficients values for $a_{i,j}$.

**Description:** Changes one or more coefficients in $A$ using the method

$$a_{\mathtt{subi[k]},\mathtt{subj[k]}} = \mathtt{valij[k]}, \quad k = 0, \ldots, \mathtt{num} - 1.$$

**See also:**

MSK_putavec Replaces all elements in one rows or columns in $A$ by new values.

MSK_putaij Changes a coefficient in $A$.

**Comments:**

- MSK_putavec

**Syntax:**

```
MSKrescodee MSK_putavec (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKidxt i,
    MSKlintt nzi,
    MSKCONST MSKidxt * subi,
    MSKCONST MSKrealt * vali);
```

**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 nonzeros in the vector.

subi **(input)** Index of the $a_{i,j}$ values that should be changed.

vali **(input)** New $a_{i,j}$ values.

**Description:** Replaces all elements in one row or column of $A$ with user defined values. If accmode equals MSK_ACC_CON a row is replaced changing $A$ as shown below.

$$a_{i,\mathtt{subi}[k]} = \mathtt{vali}[k], \quad k = 0, \ldots, \mathtt{nzi} - 1$$

If accmode equals MSK_ACC_VAR a column is replaced such that:

$$a_{\mathtt{subi}[k],i} = \mathtt{vali}[k], \quad k = 0, \ldots, \mathtt{nzi} - 1.$$

The above formulas assumes there are no duplicates in subi. If that is not the case, then the duplicate elements are added together. For an explanation of the meaning of ptrb and ptre see 5.8.3.2.

- MSK_putaveclist

**Syntax:**

```
MSKrescodee 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 whether columns or rows are changed.

num **(input)** Number of rows or columns of $A$ that should be replaced.

sub **(input)** sub contain indexes of rows or columns that should be replaced. sub should not contain duplicate values.

ptrb **(input)** Pointer 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.

ptre **(input)** Pointer 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.

asub **(input)** In the case accmode equals MSK_ACC_CON, then asub contains the new variable indexes. Otherwise it contains the new constraint indexes.

**aval (input)** Constraint values. See (5.37).

**Description:** The function replaces all elements in one or more rows or columns of $A$ with another set of specified elements.

Assume accmode equals MSK_ACC_CON then for $i = 0, \ldots, \mathtt{num} - 1$ let

$$\begin{array}{rcl} j & = & \mathtt{sub}[i], \\ a_{j,\mathtt{asub}[k]} & = & \mathtt{val}[k], \quad k = \mathtt{aptrb}[i], \ldots, \mathtt{aptre}[i] - 1. \end{array}$$

Otherwise assume accmode equals MSK_ACC_VAR then for $i = 0, \ldots, \mathtt{num} - 1$ let

$$\begin{array}{rcl} j & = & \mathtt{sub}[i], \\ a_{\mathtt{asub}[k],j} & = & \mathtt{val}[k], \quad k = \mathtt{aptrb}[i], \ldots, \mathtt{aptre}[i] - 1. \end{array}$$

- MSK_putbound

**Syntax:**

```
MSKrescodee 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 produced, 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 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. In the case multiple bound changes are specified for a constraint or a variable, then only the last change has any effect.

**See also:**

`MSK_putbound` Changes the bound for either one constraint or one variable.

`MSK_DPAR_DATA_TOL_BOUND_INF`

`MSK_DPAR_DATA_TOL_BOUND_WRN`

- `MSK_putboundslice`

**Syntax:**

```
MSKrescodee MSK_putboundslice (
    MSKtask_t task,
    MSKaccmodee con,
    MSKidxt first,
    MSKidxt last,
    MSKCONST MSKboundkeye * bk,
    MSKCONST MSKrealt * bl,
    MSKCONST MSKrealt * bu);
```

**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 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 call-back function removed. The progress function has the type `MSK_callbackfunc`.

`handle` **(input)** A pointer to a user defined data structure. Whenever the function `callbackfunc` is called, then `handle` is passed to the function.

**Description:** The function is used to input a user defined progress call-back function of type `MSK_callbackfunc`. The call-back function is called frequently during the optimization process.

**See also:**

`MSK_IPAR_LOG_SIM_FREQ`

- `MSK_putcfix`

**Syntax:**

```
MSKrescodee MSK_putcfix (
    MSKtask_t task,
    MSKrealt cfix);
```

**Arguments:**

`task` **(input)** An optimization task.

`cfix` **(input)** Fixed term in the objective.

**Description:** Replaces the a fixed term in the objective by a new one.

- MSK_putcj

**Syntax:**

```
MSKrescodee 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 element in the coefficient $c$ of the linear term in the objective. $c$ is changed such that:

$$c_{\mathtt{j}} = \mathtt{cj}.$$

- MSK_putclist

**Syntax:**

```
MSKrescodee 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

$$c_{\mathtt{subj[t]}} = \mathtt{val}[\mathtt{t}], \quad t = 0, \ldots, \mathtt{num} - 1.$$

If a variable index is specified multiple times in `subj`, then the corresponding elements of `val` are added together.

- MSK_putcone

**Syntax:**

```
MSKrescodee 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)** The parameter of the cone.

`nummem` **(input)** Number of members in the cone.

`submem` **(input)** Variable subscripts of member in the cone.

**Description:** Replaces a conic constraint with a new conic constraint.

- `MSK_putdouparam`

**Syntax:**

```
MSKrescodee 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 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 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 nonzero 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 non-zeros to preallocate for storing $A$.

It may be advantageous to reserve more nonzeros for $A$ than actually needed because it may improve the internal efficiency of MOSEK. However, it is never worthwhile to specify more than the double of the anticipated number of nonzeros in $A$.

It is never mandatory to call this function, it's only function is to give a hint of the amount of data to preallocate for efficiency reasons.

**See also:**

`MSK_IPAR_MAXNUMANZ_DOUBLE_TRH`

`MSK_IINF_STO_NUM_A_REALLOC`

- `MSK_putmaxnumcon`

**Syntax:**

```
MSKrescodee 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, it's only function is to give a hint of the amount of data to preallocate for efficiency reasons. Please note that `maxnumcon` must be larger than the current number of constraints in the task.

- `MSK_putmaxnumcone`

**Syntax:**

```
MSKrescodee MSK_putmaxnumcone (
    MSKtask_t task,
    MSKintt maxnumcone);
```

**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, it's only function is to give a hint of the amount of data to preallocate for efficiency reasons. Please note that `maxnumcon` must be larger than the current number of constraints in the task.

The function changes the maximum number of conic constraints allowed in a task. Please note that `maxnumcone` must be larger than the number of cones in the task.

- `MSK_putmaxnumqnz`

**Syntax:**

```
MSKrescodee MSK_putmaxnumqnz (
    MSKtask_t task,
    MSKlintt maxnumqnz);
```

**Arguments:**

`task` **(input)** An optimization task.

`maxnumqnz` **(input)** Number of nonzero elements preallocated in quadratic coefficient matrices.

**Description:** MOSEK stores only the nonzero 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 nonzeros for $A$ than actually needed because it may improve the internal efficiency of MOSEK. However, it is never worthwhile to specify more than the double of the anticipated number of nonzeros in $A$.

It is never mandatory to call this function, it's only function is to give a hint of the amount of data to preallocate for efficiency reasons.

- `MSK_putmaxnumvar`

**Syntax:**

```
MSKrescodee MSK_putmaxnumvar (
    MSKtask_t task,
    MSKintt maxnumvar);
```

**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, it's only function is to give a hint of the amount of data to preallocate for efficiency reasons. Please Note that `maxnumvar` must be larger than the current number of variables in the task.

- `MSK_putnadouparam`

**Syntax:**

```
MSKrescodee MSK_putnadouparam (
    MSKtask_t task,
    MSKCONST char * paramname,
    MSKrealt parvalue);
```

**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 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 MSK_putname (
    MSKtask_t task,
    MSKproblemiteme whichitem,
    MSKidxt i,
    MSKCONST char * name);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichitem` **(input)** Problem item which can take the values listed in Appendix 19.30.

`i` **(input)** Index.

`name` **(input)** New name to be assigned to the item.

**Description:** Assigns the name `name` to a problem item such as a constraint.

- `MSK_putnastrparam`

**Syntax:**

```
MSKrescodee 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 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 MSK_putobjname (
    MSKtask_t task,
    MSKCONST char * objname);
```

**Arguments:**

task **(input)** An optimization task.

objname **(input)** Name of the objective.

**Description:** Assigns the name objname to the objective function.

- MSK_putobjsense

**Syntax:**

```
MSKrescodee 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 the the problem is maximized or minimized respectively. The value MSK_OBJECTIVE_SENSE_UNDEFINED means the objective sense is taken from the parameter MSK_IPAR_OBJECTIVE_SENSE.

**Description:** Set the objective sense of the task.

**See also:**

MSK_getobjsense Get the objective sense.

- MSK_putparam

**Syntax:**

```
MSKrescodee 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 parname is valid parameter name. If yes then, then parameter is set to value specified by parvalue.

- MSK_putqcon

**Syntax:**

```
MSKrescodee 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:** It is recommended to use the function MSK_putqconk to set quadratic constraints. Replaces all quadratic entries in the constraints. Consider constraints on the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{\texttt{numvar}-1} \sum_{j=0}^{\texttt{numvar}-1} q_{ij}^k x_i x_j + \sum_{j=0}^{\texttt{numvar}-1} a_{kj} x_j \leq u_k^c, \; k = 0, \ldots, m-1. \tag{16.16}$$

The function assigns values to $q$ such that:

$$q_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}}^{\texttt{qcsubk[t]}} = \texttt{qcval[t]}, \; t = 0, \ldots, \texttt{numqcnz} - 1. \tag{16.17}$$

and

$$q_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}}^{\texttt{qcsubk[t]}} = \texttt{qcval[t]}, \; t = 0, \ldots, \texttt{numqcnz} - 1. \tag{16.18}$$

Values not assigned are set to zero.

- `MSK_putqconk`

**Syntax:**

```
MSKrescodee 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 $k$ on the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{\texttt{numvar}-1} \sum_{j=0}^{\texttt{numvar}-1} q_{ij}^k x_i x_j + \sum_{j=0}^{\texttt{numvar}-1} a_{kj} x_j \leq u_k^c. \tag{16.19}$$

It is assumed that $Q^k$ is symmetric i.e. $q_{ij}^k = q_{ji}^k$. Therefore, only the values of $q_{ij}^k$ for which $i \geq j$ should be inputted to MOSEK. In order to be precise then MOSEK use the following procedure

1. $Q^k = 0$
2. for $t = 0$ to $\texttt{numqonz} - 1$
3. $\quad q_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}}^k = q_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}}^k + \texttt{qcval[t]}$
3. $\quad q_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}}^k = q_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}}^k + \texttt{qcval[t]}$

Please Note that:

– Only the lower triangular part should be specified. $Q^k$ is symmetric so it is only necessary to specify the lower triangular part. Specifying values for $q_{ij}^k$ where $i < j$ is an error.
– Only non-zero elements are specified.
– The order in which the non-zero elements are specified is insignificant.
– Please note duplicate elements are added to together. Hence, it is recommended not to specify the same element multiple times in `qosubi`, `qosubj`, and `qoval`.

For an explained code example see section 5.3.2.

- `MSK_putqobj`

**Syntax:**

```
MSKrescodee 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 nonzero elements in $Q^o$.
`qosubi` **(input)** $i$ subscript for $q_{ij}^o$.
`qosubj` **(input)** $j$ subscript for $q_{ij}^o$.
`qoval` **(input)** Numerical value for $q_{ij}^o$.

**Description:** Replaces all the quadratic terms in the objective

$$\frac{1}{2} \sum_{i=0}^{\texttt{numvar}-1} \sum_{j=0}^{\texttt{numvar}-1} q_{ij}^o x_i x_j + \sum_{j=0}^{\texttt{numvar}-1} c_j x_j + c^f. \tag{16.20}$$

It is assumed that $Q^o$ is symmetric i.e. $q_{ij}^o = q_{ji}^o$. Therefore, only the values of $q_{ij}^o$ for which $i \geq j$ should be inputted to MOSEK. In order to be precise then MOSEK use 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 $q_{ij}^o$ where $i < j$ is an error.
– Only nonzero elements should be specified.
– The order in which the non-zero elements are specified is insignificant.
– Please note that duplicate elements are added to together. Hence, it is recommended not to specify the same element multiple times in `qosubi`, `qosubj`, and `qoval`.

For an explained code example see Section 5.3.1.

- MSK_putqobjij

**Syntax:**

```
MSKrescodee MSK_putqobjij (
    MSKtask_t task,
    MSKidxt i,
    MSKidxt j,
    MSKrealt qoij);
```

**Arguments:**

`task` **(input)** An optimization task.

`i` **(input)** $i$ index for the coefficient to be replaced.

`j` **(input)** $j$ 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. I.e. the procedure performs the assignment

$$q^o_{\mathtt{ij}} = \mathtt{qoij}.$$

- MSK_putresponsefunc

**Syntax:**

```
MSKrescodee 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 callback which is called when an error or warning occurs.

- MSK_putsolution

**Syntax:**

```
MSKrescodee MSK_putsolution (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKCONST MSKstakeye * skc,
    MSKCONST MSKstakeye * skx,
    MSKCONST MSKstakeye * skn,
```

```
    MSKCONST MSKrealt * xc,
    MSKCONST MSKrealt * xx,
    MSKCONST MSKrealt * y,
    MSKCONST MSKrealt * slc,
    MSKCONST MSKrealt * suc,
    MSKCONST MSKrealt * slx,
    MSKCONST MSKrealt * sux,
    MSKCONST MSKrealt * snx);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichsol` **(input)** The solution index which can take the values listed in Appendix 19.42.

`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)** 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 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 nonzero, then the solution information for a constraint is modified. Otherwise for a variable.

`i` **(input)** Index of the constraint or variable.

whichsol **(input)** The solution index which can take the values listed in Appendix 19.42.

sk **(input)** Status key of the constraint of variable.

x **(input)** Solution value of the primal 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. If a sequence of function calls MSK_putsolutioni is used defined to a new solution, then normally the function MSK_makesolutionstatusunknown should be called before the first call of the function MSK_putsolutioni.

**See also:**

MSK_makesolutionstatusunknown Set the solution status to unknown.

- MSK_putsolutionyi

**Syntax:**

```
MSKrescodee 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)** The solution index which can take the values listed in Appendix 19.42.

y **(input)** Solution value of the dual variable.

**Description:** Input the dual variable of a solution.

**See also:**

MSK_makesolutionstatusunknown Set the solution status to unknown.

MSK_putsolutioni Sets the primal and dual solution information for a single constraint or variable.

- MSK_putstrparam

**Syntax:**

```
MSKrescodee MSK_putstrparam (
    MSKtask_t task,
    MSKsparame param,
    MSKCONST char * parvalue);
```

**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 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 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 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 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 which should have their variable type changed.

`vartype` **(input)** A list of variable types that should be assigned to the variables specified by `subj`. See Section 19.48 for the possible values of vartype.

**Description:** Sets the variable type for one or more variables. I.e. variable `subj[k]` is assigned the variable type `vartype[k]`.

- MSK_readbranchpriorities

**Syntax:**

```
MSKrescodee 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 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 problem data associated with the optimization task from a file.

The expected format of the data file is determined based on parameter MSK_IPAR_READ_DATA_FORMAT. If this parameter has the (default) value MSK_DATA_FORMAT_EXTENSION, then the extension of the file name is used to determine the file format. I.e. if file name has the extension `.lp.gz` then it is assumed to be a compressed LP formatted file is written.

**See also:**

MSK_writedata Write problem data to a file.

MSK_IPAR_READ_DATA_FORMAT

- MSK_readparamfile

**Syntax:**

```
MSKrescodee MSK_readparamfile (MSKtask_t task)
```

**Arguments:**

`task` **(input)** An optimization task.

**Description:** Reads a parameter file.

- MSK_readsolution

**Syntax:**

```
MSKrescodee MSK_readsolution (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKCONST char * filename);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichsol` **(input)** The solution index which can take the values listed in Appendix 19.42.

`filename` **(input)** A valid file name.

**Description:** Reads a solution file and inserts the solution into the solution `whichsol`.

- MSK_readsummary

**Syntax:**

```
MSKrescodee MSK_readsummary (
    MSKtask_t task,
    MSKstreamtypee whichstream);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichstream` **(input)** Index of the stream.

**Description:** Prints a short summary related to the last MPS file that was read.

- MSK_relaxprimal

**Syntax:**

```
MSKrescodee 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 lower 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:** This function creates a problem that computes a minimal (weighted) relaxation of the bounds that will make an infeasible problem feasible.

Given an existing task describing the problem

$$\begin{array}{lccccl} \text{minimize} & & & c^T x & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \tag{16.21}$$

then the function forms a new task `relaxedtask` having the form

$$\begin{array}{lcccll} \text{minimize} & & & p & & \\ \text{subject to} & l^c & \leq & Ax + v_l^c - v_u^c & \leq & u^c, \\ & l^x & \leq & x + v_l^x - v_u^x & \leq & u^x, \\ & & & (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq & 0, \\ & & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0. & & \end{array} \tag{16.22}$$

Hence, the function adds so-called elasticity variables to all the constraints which relaxes the constraints i.e. for instance $(v_l^c)_i$ and $(v_u^c)_i$ relaxes $(l^c)_i$ and $(u^c)_i$ respectively. It should be obvious that (16.22) 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, if the problem (16.22) is optimized, then the minimal change to the bounds in a weighted sense is computed that will make the problem feasible.

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 (16.22), then a natural thing to do is to solve the optimization problem

$$\begin{array}{lccccl} \text{minimize} & & & c^T x & & \\ \text{subject to} & l^c & \leq & A x + v_l^c - v_u^c & \leq & u^c, \\ & l^x & \leq & x + v_l^x - v_u^x & \leq & u^x, \\ & & & (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x - p & \leq & 0, \\ & & & p & = & p^*, \\ & & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0, & & \end{array} \tag{16.23}$$

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 (16.22) or the problem (16.23).The parameter can take one of the following values.

`MSK_FEASREPAIR_OPTIMIZE_NONE` : The returned task `relaxedtask` contains problem (16.22) and is not optimized.

`MSK_FEASREPAIR_OPTIMIZE_PENALTY` : The returned task `relaxedtask` contains problem (16.22) and is optimized.

`MSK_FEASREPAIR_OPTIMIZE_COMBINED` : The returned task `relaxedtask` contains problem (16.23) and is optimized.

Note that the $v$ variables are appended to the $x$ variables in the order

$$(v_u^c)_1, (v_l^c)_1, (v_u^c)_2, (v_l^c)_2, \ldots, (v_u^c)_m, (v_l^c)_m, \quad (v_u^x)_1, (v_l^x)_1, (v_u^x)_2, (v_l^x)_2, \ldots, (v_u^x)_n, (v_l^x)_n$$

in the returned task.

If `NAME_CON` (`NAME_VAR`) is the name of the $i$th 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 "`*`" is a user defined separator string given by the parameter `MSK_SPAR_FEASREPAIR_NAME_SEPARATOR`.

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 an linear optimization problem. However, the function also work for quadratic and conic optimization problems but it cannot be used for general nonlinear optimization problems.

**See also:**

MSK_DPAR_FEASREPAIR_TOL

MSK_IPAR_FEASREPAIR_OPTIMIZE

MSK_SPAR_FEASREPAIR_NAME_SEPARATOR

MSK_SPAR_FEASREPAIR_NAME_PREFIX

- MSK_remove

**Syntax:**

```
MSKrescodee MSK_remove (
    MSKtask_t task,
    MSKaccmodee accmode,
    MSKintt num,
    MSKCONST MSKintt * sub);
```

**Arguments:**

`task` **(input)** An optimization task.

`accmode` **(input)** Defines whether constraints or variables are removed.

`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 forward.

**See also:**

`MSK_append` Appends a number of variables or constraints to the optimization task.

- MSK_removecone

**Syntax:**

```
MSKrescodee MSK_removecone (
    MSKtask_t task,
    MSKidxt k);
```

**Arguments:**

`task` **(input)** An optimization task.

`k` **(input)** Index of the conic constraint that should be removed.

**Description:** Remove a conic constraint from the problem. Please note this implies that all the conic constraints appearing after the removed cone is renumbered. I.e. their index is decreased by one.

In general it is much more efficient to remove a cone with a high index than a low index.

- MSK_resizetask

**Syntax:**

```
MSKrescodee MSK_resizetask (
    MSKtask_t task,
    MSKintt maxnumcon,
    MSKintt maxnumvar,
    MSKintt maxnumcone,
    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 non-zeros in $A$.

`maxnumqnz` **(input)** New maximum number non-zeros in all $Q$ matrices.

**Description:** Sets the amount of preallocated space assigned for each type of data in an optimization task.

Note the procedure is **destructive** meaning the data stored in the task is destroyed.

It is never mandatory to call this function, it's only function is to give a hint of the amount of data to preallocate for efficiency reasons.

**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 constraints in the optimization task.

`MSK_putmaxnumanz` The function changes the size of the preallocated storage for $A$.

`MSK_putmaxnumqnz` The function changes the size of the preallocated storage for $Q$.

- `MSK_sensitivityreport`

**Syntax:**

```
MSKrescodee MSK_sensitivityreport (
    MSKtask_t task,
    MSKstreamtypee whichstream);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichstream` **(input)** Index of the stream.

**Description:** Read a sensitivity format file from location given by `MSK_SPAR_SENSITIVITY_FILE_NAME` and write 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` Perform sensitivity analysis on objective coefficients.

`MSK_primalsensitivity` Perform sensitivity analysis on bounds.

MSK_IPAR_LOG_SENSITIVITY
MSK_IPAR_LOG_SENSITIVITY_OPT
MSK_IPAR_SENSITIVITY_TYPE

- MSK_setdefaults

**Syntax:**

```
MSKrescodee MSK_setdefaults (MSKtask_t task)
```

**Arguments:**

`task` **(input)** An optimization task.

**Description:** Resets all the parameters to their default values.

- MSK_sktostr

**Syntax:**

```
MSKrescodee MSK_sktostr (
    MSKtask_t task,
    MSKintt sk,
    char * str);
```

**Arguments:**

`task` **(input)** An optimization task.
`sk` **(input)** A valid status key.
`str` **(output)** String corresponding to the status key `sk`.

**Description:** Obtains a explanatory string corresponding to a status key.

- MSK_solstatostr

**Syntax:**

```
MSKrescodee 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 a explanatory string corresponding to a solution status.

- MSK_solutiondef

**Syntax:**

```
MSKrescodee MSK_solutiondef (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKintt * isdef);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichsol` **(input)** The solution index which can take the values listed in Appendix 19.42.

`isdef` **(output)** Is nonzero if the requested solution is defined.

**Description:** Checks whether a solution defined.

- `MSK_solutionsummary`

**Syntax:**

```
MSKrescodee MSK_solutionsummary (
    MSKtask_t task,
    MSKstreamtypee whichstream);
```

**Arguments:**

`task` **(input)** An optimization task.

`whichstream` **(input)** Index of the stream.

**Description:** Prints a short summary related to the current solution.

- `MSK_solvewithbasis`

**Syntax:**

```
MSKrescodee 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 nonzero, then (16.25) is solved. Otherwise the system (16.24) is solved.

`numnz` **(input/output)** As input it is the number of nonzeros in $b$. As output it is the number of nonzeros in $\bar{x}$.

`sub` **(input/output)** As input it contains the position of the nonzeros in $b$ i.e.

$$b[\texttt{sub}[k]] \neq 0, \; k = 0, \ldots, \texttt{numnz}[0] - 1.$$

As output it contains the position of the nonzeros in $\bar{x}$. It is important `sub` has room for `numcon` elements.

`val` **(input/output)** As input it is the vector $b$. Although the positions of the nonzero elements are specified in `sub`, then 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 and **not** a packed sparse vector. This implies `val` has room for `numcon` elements.

**Description:** If a basic solution is available, then exactly `numcon` basis variables are defined. Those `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

$$B\bar{x} = b \tag{16.24}$$

or the system

$$B^T\bar{x} = b \tag{16.25}$$

for the unknowns $\bar{x}$. $b$ is 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` a ordering of the basis variables is obtained. This ordering can be used to deduce how MOSEK has constructed $B$. Indeed if the $k$th basis variable is variable $x_j$ then this implies

$$B_{i,k} = A_{i,j}, \; i = 0, \ldots, \texttt{numcon} - 1.$$

Otherwise if the $k$th basis variable is variable $x_j^c$ then this implies

$$B_{i,k} = \begin{cases} -1, & i = j, \\ 0, & i \neq j. \end{cases}$$

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` This function must be called immediately before the first usage of the function `MSK_solvewithbasis`.

- `MSK_startstat`

**Syntax:**

```
MSKrescodee MSK_startstat (MSKtask_t task)
```

**Arguments:**

`task` **(input)** An optimization task.

**Description:** Starts the statistics file.

- `MSK_stopstat`

**Syntax:**

```
MSKrescodee MSK_stopstat (MSKtask_t task)
```

**Arguments:**

`task` **(input)** An optimization task.

**Description:** Stops the statistics file.

- `MSK_strdupdbgtask`

**Syntax:**

```
char * 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 * 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 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 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 a explanatory string.

- MSK_undefsolution

**Syntax:**

```
MSKrescodee MSK_undefsolution (
    MSKtask_t task,
    MSKsoltypee whichsol);
```

**Arguments:**

task **(input)** An optimization task.

whichsol **(input)** The solution index which can take the values listed in Appendix 19.42.

**Description:** Undefines a solution. Purges all information regarding whichsol.

- MSK_unlinkfuncfromtaskstream

**Syntax:**

```
MSKrescodee 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 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 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 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 from the file specified by `MSK_SPAR_DATA_FILE_NAME`.

**Description:** Write 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.

The type of the data file written is determined either based on parameter MSK_IPAR_WRITE_DATA_FORMAT. If the parameter MSK_IPAR_WRITE_DATA_FORMAT has the (default) value MSK_DATA_FORMAT_EXTENSION, then the extension of the file name is used to determine the file format. I.e. if file name has the extension .lp.gz then a compressed LP formatted file is written.

Note in the case no names has been inputted into the task an anonymous names are required in the data file, then the option MSK_IPAR_WRITE_GENERIC_NAMES should be turned on.

**See also:**

MSK_readdata Reads problem data from a file.

MSK_IPAR_WRITE_DATA_FORMAT

- MSK_writeparamfile

**Syntax:**

```
MSKrescodee MSK_writeparamfile (
    MSKtask_t task,
    MSKCONST char * filename);
```

**Arguments:**

task **(input)** An optimization task.

filename **(input)** is the name of parameter file.

**Description:** Writes all the parameters to a parameter file.

- MSK_writesolution

**Syntax:**

```
MSKrescodee MSK_writesolution (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKCONST char * filename);
```

**Arguments:**

task **(input)** An optimization task.

whichsol **(input)** The solution index which can take the values listed in Appendix 19.42.

filename **(input)** A valid file name.

**Description:** Saves the current basic, interior-point, or integer solution to a file.

# Chapter 17

# Parameter reference

## 17.1 Parameter groups

Parameters grouped by meaning and functionality.

### 17.1.1 Basis identification parameters

### 17.1.2 Interior-point method parameters

Parameters defining the behavior of the interior-point method for linear, conic and convex problems.

### 17.1.3 Simplex optimizer parameters

Parameters defining the behavior of the simplex optimizer for linear problems.

### 17.1.4 Primal simplex optimizer parameters

Parameters defining the behavior of the primal simplex optimizer for linear problems.



### 17.1.5 Dual simplex optimizer parameters

Parameters defining the behavior of the dual simplex optimizer for linear problems.

### 17.1.6 Network simplex optimizer parameters

Parameters defining the behavior of the network simplex optimizer for linear problems.



### 17.1.7 Non-linear convex method parameters

Parameters defining the behavior of the interior-point method for non-linear convex problems.

### 17.1.8 Conic interior-point method parameters

Parameters defining the behavior of the interior-point method for conic problems.



### 17.1.9 Mixed integer optimization parameters

### 17.1.10 Presolve parameters

### 17.1.11 Termination criterion parameters

Parameters which define termination and optimality criterions and related information.

### 17.1.12 Progress callback parameters



### 17.1.13 Non-convex solver parameters



### 17.1.14 Feasibility repair parameters



### 17.1.15 Optimization system parameters

Parameters defining the overall solver system environment. This includes system and platform related information and behavior.

### 17.1.16 Output information parameters

- MSK_IPAR_MAX_NUM_WARNINGS . . . . . . . . . . . . . . . . . . . . . 429
  Waning level. A higher value implies more warnings.
- MSK_IPAR_WARNING_LEVEL . . . . . . . . . . . . . . . . . . . . . 459
  Warning level.

### 17.1.17 Extra information about the optimization problem

- MSK_IPAR_OBJECTIVE_SENSE . . . . . . . . . . . . . . . . . . . . . 437
  If the objective sense for task is undefined, then the value of this parameter is used as the default objective sense.

### 17.1.18 Overall solver parameters

- MSK_IPAR_BI_CLEAN_OPTIMIZER . . . . . . . . . . . . . . . . . . . . . 410
  Controls which simplex optimizer that is used in the clean up phase.
- MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS . . . . . . . . . . . . . . . . . . . . . 413
  The maximum number of simultaneous optimizations that will be started by the concurrent optimizer.
- MSK_IPAR_CONCURRENT_PRIORITY_DUAL_SIMPLEX . . . . . . . . . . . . . . . . . . . . . 413
  Priority of the dual simplex algorithm when selecting solvers for concurrent optimization.
- MSK_IPAR_CONCURRENT_PRIORITY_FREE_SIMPLEX . . . . . . . . . . . . . . . . . . . . . 413
  Priority of free simplex optimizer when selecting solvers for concurrent optimization.
- MSK_IPAR_CONCURRENT_PRIORITY_INTPNT . . . . . . . . . . . . . . . . . . . . . 413
  Priority of the interior point algorithm when selecting solvers for concurrent optimization.
- MSK_IPAR_CONCURRENT_PRIORITY_PRIMAL_SIMPLEX . . . . . . . . . . . . . . . . . . . . . 414
  Priority of the primal simplex algorithm when selecting solvers for concurrent optimization.
- MSK_IPAR_DATA_CHECK . . . . . . . . . . . . . . . . . . . . . 414
  Enable data checking for debug purposes.
- MSK_IPAR_FEASREPAIR_OPTIMIZE . . . . . . . . . . . . . . . . . . . . . 415
  Controls which type of feasibility analysis is to be performed.
- MSK_IPAR_INFEAS_PREFER_PRIMAL . . . . . . . . . . . . . . . . . . . . . 416
  Use only primal or dual certificate.
- MSK_IPAR_LICENSE_WAIT . . . . . . . . . . . . . . . . . . . . . 422
  Controls if MOSEK should queue for a license if one is not available.
- MSK_IPAR_MIO_CONT_SOL . . . . . . . . . . . . . . . . . . . . . 431
  Controls the meaning of interior and basic solutions in MIP problems.
- MSK_IPAR_MIO_LOCAL_BRANCH_NUMBER . . . . . . . . . . . . . . . . . . . . . 433
  Turn on/off mixed integer mode.

### 17.1.19 Behavior of the optimization task

Parameters defining the behavior of an optimization task, men data is fed into it.

### 17.1.20 Data input/output parameters

Parameters defining the behavior of data readers and writers.

### 17.1.21 Solution input/output parameters

Parameters defining the behavior of solution reader and writer.

### 17.1.22 Infeasibility report parameters



### 17.1.23 License manager parameters



### 17.1.24 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.

## 17.2 Double parameters

- 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 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 relaxations in branch and bound search.

MSK_IPAR_MIO_MAX_NUM_BRANCHES Maximum number 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 spend 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 spend 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 spend 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 $\alpha$ be the value of this parameter and $m$ the number constraints, then mixed integer optimizer is allowed to $\alpha m$ 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 spend 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 $\infty$.

**Possible Values:**
Any number between -inf and +inf.

**Default value:**
0.5e30

## 17.3 Integer parameters

- MSK_IPAR_WRITE_XML_MODE ........ 466
  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 made room for when numanz exceeds maxnumqnz during addition of new $Q$ entries.

  **Possible Values:**
  Any nonnegative integer.

  **Default value:**
  5000

- bi_clean_optimizer

  **Corresponding constant:**
  MSK_IPAR_BI_CLEAN_OPTIMIZER

  **Description:**
  Controls which simplex optimizer that 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 choice of optimizer is made 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 terminated due to max 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 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 the case it has, then control is returned to the user program.

  In the 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. NaN's. 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 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 A 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 MOSEK down but on the other 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

MSK_FEASREPAIR_OPTIMIZE_COMBINED

MSK_FEASREPAIR_OPTIMIZE_PENALTY

**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 it is never flushed. Otherwise a larger value implies 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 a primal and dual infeasibility certificate is 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 info presented in an infeasibility report. Higher values implies more information.

**Possible Values:**
Any nonnegative value.

**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 though 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 that says 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 the case the interior-point optimizer is terminated due to max iterations.

MSK_IPAR_BI_IGNORE_NUM_ERROR Turns on basis identification in the 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 nonnegative integer.

  **Default value:**
  400

- intpnt_max_num_cor

  **Corresponding constant:**
  MSK_IPAR_INTPNT_MAX_NUM_COR

  **Description:**
  Controls the maximum number of corrector's allowed to be used by the multiple corrector procedure. A negative value means the 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 the optimizer is choosing 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 nonnegative integer.

**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 automatically chosen.

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 choose 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 used by the interior-point optimizer.

**Possible Values:**

`MSK_STARTING_POINT_CONSTANT` The starting point is chosen to 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 later reuse. Checking out a license from the license server has a small overhead. Therefore, if a large number of optimizations are performed within a small amount of time, then it is useful (read efficient) to cache the license in the MOSEK environment for later use. This way a number of license check outs from the license server are avoided.

If a license has not been used in the given amount of time, then MOSEK will automatically check in the license. To disable license caching set to 0.

**Possible Values:**
All nonnegative integers

**Default value:**
5

- `license_check_time`

**Corresponding constant:**
`MSK_IPAR_LICENSE_CHECK_TIME`

**Description:**
The parameter specifies the number seconds between all the active licenses in the MOSEK environment license cache are checked to determine if they 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 nonnegative value.

**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, this parameter can used to turn on that MOSEK will wait for a license until becomes available i.e. MOSEK queue for a 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.

**Possible Values:**
Any number between 0 and +inf.

**Default value:**
10

- `log_bi`

**Corresponding constant:**
`MSK_IPAR_LOG_BI`

**Description:**
Controls amount of output printed by the basis identification procedure.

**Possible Values:**
Any number between 0 and +inf.

**Default value:**
1

- `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 nonnegative number.

**Default value:**
1

- log_cut_second_opt

**Corresponding constant:**
MSK_IPAR_LOG_CUT_SECOND_OPT

**Description:**

**Possible Values:**
Any number between 0 and +inf.

**Default value:**
1

- log_factor

**Corresponding constant:**
MSK_IPAR_LOG_FACTOR

**Description:**
If turned on, then factor 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 amount of output printed when performing feasibility repair.

**Possible Values:**
Any nonnegative number.

**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.

**Possible Values:**
Any nonnegative number.

**Default value:**
1

- `log_intpnt`

**Corresponding constant:**
`MSK_IPAR_LOG_INTPNT`

**Description:**
Controls amount of output printed by the interior-point optimizer.

**Possible Values:**
Any nonnegative number.

**Default value:**
4

- `log_mio`

**Corresponding constant:**
`MSK_IPAR_LOG_MIO`

**Description:**
Controls the print level for the mixed integer optimizer

**Possible Values:**
Any number between 0 and +inf.

**Default value:**
2

- 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 nonnegative number.

**Default value:**
1

- log_optimizer

**Corresponding constant:**
MSK_IPAR_LOG_OPTIMIZER

**Description:**
If turned on, then optimizer lines are added to the log.

**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 nonnegative integer.

  **Default value:**
  0

- `log_presolve`

  **Corresponding constant:**
  `MSK_IPAR_LOG_PRESOLVE`

  **Description:**
  Controls amount of output printed by the presolve procedure.

  **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.

  **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.

**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:**
Controls whether some of the less important log information simplex optimizer is outputted.

**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:**

`MSK_ON` Switch the option on.

`MSK_OFF` Switch the option off.

**Default value:**
MSK_OFF

- `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 implies more warnings.

**Possible Values:**
Any nonnegative integer.

**Default value:**
10

- `maxnumanz_double_trh`

**Corresponding constant:**
`MSK_IPAR_MAXNUMANZ_DOUBLE_TRH`

**Description:**
Whenever MOSEK runs out of storage for the $A$ matrix then it will double the value for `maxnumanz` until compmaxnumnza reaches the value of this parameter. After this threshold is reaches it will use a slower increase.

**Possible Values:**
Any nonnegative integer.

**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 up branch first.

`MSK_BRANCH_DIR_UP` The mixed integer optimizer always chooses the down 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 has 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 continues 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 which problem the interior and basis solutions are solutions to when the problem is optimized using the mixed integer optimizer.

**Possible Values:**

MSK_MIO_CONT_SOL_ITG The reported interior-point and basis 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 the case the problem has a primal feasible solution.

MSK_MIO_CONT_SOL_NONE No interior or basis solutions are reported when the mixed integer optimizer is used.

MSK_MIO_CONT_SOL_ROOT The reported interior-point and basis solutions are a solution to the root node problem when mixed integer optimizer is used.

MSK_MIO_CONT_SOL_ITG_REL In the case the problem is primal feasible then the reported interior-point and basis 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 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 large value than 0 means a gradually more sophisticated heuristic is used which is computationally more expensive. A negative value implies that the optimizer chooses the heuristic to be used. 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 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 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 have been located. If this parameter has the value $n$ and $n$ strictly positive, then 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 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 that is employed 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 choice of optimizer is made 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_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 that 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 choice of optimizer is made 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 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 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 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 non-negative integer, where 0 means unlimited

**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 basis solution is defined, include the basis 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_OFF

- opf_write_solutions

**Corresponding constant:**
MSK_IPAR_OPF_WRITE_SOLUTIONS

**Description:**
Enable inclusion of solutions in 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 choice of optimizer is made 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 considered to be 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 eliminations 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 eliminator 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 is 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 nonnegative integer.

**Default value:**
1

- presolve_use

  **Corresponding constant:**
  MSK_IPAR_PRESOLVE_USE

  **Description:**
  Controls whether 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 automatically decided whether the presolved 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 nonnegative integer.

  **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 nonnegative integer.

  **Default value:**
  0

- read_add_cone

  **Corresponding constant:**
  MSK_IPAR_READ_ADD_CONE

  **Description:**
  Additional number of constraints that is made room for in the problem.

  **Possible Values:**
  Any nonnegative integer.

**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 nonnegative integer.

**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 nonnegative integer.

**Default value:**
0

- read_anz

**Corresponding constant:**
MSK_IPAR_READ_ANZ

**Description:**
Expected maximum number of $A$ nonzeros to be read. The option is only used by fast MPS and LP file readers.

**Possible Values:**
Any nonnegative integer.

**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 nonnegative integer.

**Default value:**
10000

- read_cone

**Corresponding constant:**
MSK_IPAR_READ_CONE

**Description:**
Expected maximum number of conic constraints to be read. The option is only used by fast MPS and LP file readers.

**Possible Values:**
Any nonnegative integer.

**Default value:**
2500

- read_data_compressed

**Corresponding constant:**
MSK_IPAR_READ_DATA_COMPRESSED

**Description:**
If the this option is turned on, then it is assumed 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 a XML formatted file.

MSK_DATA_FORMAT_EXTENSION The extension of the file name 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 a 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, 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_format`

  **Corresponding constant:**
  `MSK_IPAR_READ_MPS_FORMAT`

  **Description:**
  Controls how strict the MPS file reader is regarding 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 the input file satisfies the free MPS format. This implies spaces are not allowed names. On the other hand 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, then 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:**
MOSEK cannot solve integer programming problems, but only the continuous relaxation. If this option is turned on, then the relaxation 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 $Q$ nonzeros to be read. The option is only used by fast MPS and LP file readers.

  **Possible Values:**
  Any nonnegative integer.

  **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.

  **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 only used by fast MPS and LP file readers.

  **Possible Values:**
  Any nonnegative integer.

  **Default value:**
  10000

- `sensitivity_all`

  **Corresponding constant:**
  `MSK_IPAR_SENSITIVITY_ALL`

  **Description:**

  If set to `MSK_ON` then `MSK_sensitivityreport` analyze all bounds and variables instead of reading a specification from 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 choice of optimizer is made 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

MSK_SENSITIVITY_TYPE_BASIS

**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 optimize should use no degeneration strategy.

MSK_SIM_DEGEN_MODERATE The simplex optimize should use a moderate degeneration strategy.

MSK_SIM_DEGEN_MINIMUM The simplex optimize should use minimum degeneration strategy.

MSK_SIM_DEGEN_AGGRESSIVE The simplex optimize should use a aggressive degeneration strategy.

MSK_SIM_DEGEN_FREE The simplex optimize 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 that (100-this parameter value)% fixed variables, then a crash will be performed.

**Possible Values:**
Any nonnegative integer value.

**Default value:**
90

- sim_dual_restrict_selection

**Corresponding constant:**
MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION

**Description:**
The dual simplex can use a so called restricted selection/pricing strategy to choose the outgoing variable. Hence, if restricted selection is applied, then the dual simplex first choose a subset of all the potential outgoing variables. Next it will for some time only choose the outgoing among the subset. Of course from time to time the subset is redefined.

A large value of this parameter implies the optimizer will be more aggressive in its restriction strategy. I.e. a value of 0 implies 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 an 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 choose 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 hotstart 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 hotstart.

MSK_SIM_HOTSTART_FREE The simplex optimize chooses the hotstart type.

**Default value:**
MSK_SIM_HOTSTART_FREE

- sim_max_iterations

**Corresponding constant:**
MSK_IPAR_SIM_MAX_ITERATIONS

**Description:**
Maximum number of iterations that can 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 that 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 nonzero integer.

  **Default value:**
  250

- `sim_network_detect`

  **Corresponding constant:**
  `MSK_IPAR_SIM_NETWORK_DETECT`

  **Description:**
  The simplex optimizer has the capability of exploiting that a problem contains a network flow component. It is only worthwhile to exploit the network flow component if it is sufficient large. This parameter controls has large the network component in “relative” terms has to be 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 nonnegative integer.

  **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 hotstart. 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 nonnegative integer.

  **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 ensure 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 that (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 can use a so called restricted selection/pricing strategy to choose the incoming variable. Hence, if restricted selection is applied, then the primal simplex first choose a subset of all the potential incoming variables. Next it will for some time only choose the incoming among the subset. Of course from time to time the subset is defined.

A large value of this parameter implies the optimizer will be more aggressive in its restriction strategy. I.e. a value of 0 implies 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 an 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 choose 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 when the best point of refactorization is.

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 choose 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 simplex optimizers.

**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 big priority the numerical stability should be given.

**Possible Values:**
Any nonnegative integer value.

**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 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 the simplex optimizer is chosen to be the primal (dual) one then it is switch 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 a it will strip off a leading and trailing quote.

**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 solution 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 callbacks 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 nonnegative integer.

  **Default value:**
  1

- `write_bas_constraints`

  **Corresponding constant:**
  `MSK_IPAR_WRITE_BAS_CONSTRAINTS`

  **Description:**
  Controls whether the constraint section is written to the basis 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 basis 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 basis 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 and higher values means more compression.

  **Possible Values:**
  Any nonnegative value

  **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 a file using `MSK_writedata`.

  **Possible Values:**

  `MSK_DATA_FORMAT_XML` The data file is a XML formatted file.

  `MSK_DATA_FORMAT_EXTENSION` The extension of the file name 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 a 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, then 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 is 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 generic names or the 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 a 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 formated output files satisfies 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 nonnegative number.

**Default value:**
10

- write_mps_int

  **Corresponding constant:**
  MSK_IPAR_WRITE_MPS_INT

  **Description:**
  Controls whether 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 on, the object 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 contain 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 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 also stored in the task file.

**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

## 17.4 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 that is used. An empty name means 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 that is used as objective function. An empty name means 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 that is used. An empty name means 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 that is used. An empty name means 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 read this file as 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 nonempty string, then MSK_sensitivityreport write 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 that used to determine which constraints that should be listed in the solution file. A value of "0.5" means all constraints that has `xc[i]>0.5` should be printed. Whereas "+0.5" means all constraints that has `xc[i]>=blc[i]+0.5` should be listed. An empty filter means 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 that is used to determine which constraints that should be listed in the solution file. A value of "0.5" means all constraints that has `xc[i]<0.5` should be printed. Whereas "-0.5" means all constraints that has `xc[i]<=buc[i]-0.5` should be listed. An empty filter means 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 that is used to determine which variables that should be listed in the solution file. A value of "0.5" means all constraints that has `xx[j]>=0.5` should be printed. Whereas "+0.5" means all constraints that has `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 that is used to determine which variables that should be listed in the solution file. A value of "0.5" means all constraints that has `xx[j]<0.5` should be printed. Whereas "-0.5" means all constraints that has `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:**
Named 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 then additional variables must be inserted. They will have the prefix denoted by this parameter.

**Possible Values:**
Any valid string.

**Default value:**
"xmskgen"

# Chapter 18

# Response codes

A BOUNDS vector is split into several nonadjacent parts in a MPS file.


Missing '/2' after quadratic expressions in bound or objective.


Ignore a variable because the variable has not been previously defined. Usually this implies a variable has been deigned in the bound section but not in the objective or the constraints.


Nonzero elements is specified in the upper triangle of a matrix which is ignored by the code.


One or more nonzero elements are dropped from the Q matrix in the objective.


Ignored integer constraints.


No global optimizer is available.


When the MOSEK mixed integer optimizer reoptimizes a mixed integer problem with all the integer variables fixed at their "optimal value" the then problem becomes infeasible. Sometimes the problem can be resolved by reducing the tolerances MSK_DPAR_MIO_TOL_ABS_RELAX_INT and MSK_DPAR_MIO_TOL_REL_RELAX_INT.


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.


Invalid solution filter is specified.


Undefined name occurred in a solution.


An incomplete basis has been specified. Too few basis variables are specified.


A basis with too many variables has been specified. Too few basis variables are specified.


The license expires.


The license server is not responding.


A variable or constraint name is empty. The output file may be invalid.

(503) MSK_RES_WRN_USING_GENERIC_NAMES ........ 553
The file writer reverts to generic names because a name is blank.

(505) MSK_RES_WRN_LICENSE_FEATURE_EXPIRE ........ 549
The license expires.

(700) MSK_RES_WRN_ZEROS_IN_SPARSE_DATA ........ 553
One or more almost zero elements are specified in sparse input data.

(800) MSK_RES_WRN_NONCOMPLETE_LINEAR_DEPENDENCY_CHECK ........ 551
The linear dependency check(s) was not completed and therefore the $A$ matrix may contain linear dependencies.

(801) MSK_RES_WRN_ELIMINATOR_SPACE ........ 547
The eliminator is skipped at least once due to lack of space.

(802) MSK_RES_WRN_PRESOLVE_OUTOFSPACE ........ 552
The presolve is incomplete due to lack of space.

(803) MSK_RES_WRN_PRESOLVE_BAD_PRECISION ........ 551
The presolve estimates that the model is specified in too low precision.

(804) MSK_RES_WRN_WRITE_DISCARDED_CFIX ........ 553
The fixed objective term could not be converted to a variable and was discarded in the output file.

(1000) MSK_RES_ERR_LICENSE ........ 512
Invalid license.

(1001) MSK_RES_ERR_LICENSE_EXPIRED ........ 513
The license has expired.

(1002) MSK_RES_ERR_LICENSE_VERSION ........ 514
The license is valid for another version of MOSEK.

(1005) MSK_RES_ERR_SIZE_LICENSE ........ 536
The problem is bigger than the license.

(1006) MSK_RES_ERR_PROB_LICENSE ........ 533
The software is not licensed to solve the problem.

(1007) MSK_RES_ERR_FILE_LICENSE ........ 499
Invalid license file.

(1008) MSK_RES_ERR_MISSING_LICENSE_FILE ........ 518
MOSEK cannot find the license file or license server. Usually this happens if the operating system variable MOSEKLM_LICENSE_FILE is not appropriately setup. Please see the MOSEK installation manual for details.

(1010) MSK_RES_ERR_SIZE_LICENSE_CON ........ 537
The problem has too many constraints to be solved with the available license.

(1011) MSK_RES_ERR_SIZE_LICENSE_VAR ........ 537
The problem has too many variables to be solved with the available license.

(1012) MSK_RES_ERR_SIZE_LICENSE_INTVAR ........ 537
The problem contains too many integer variables to be solved with the available license.

(1013) MSK_RES_ERR_OPTIMIZER_LICENSE ........ 530
The optimizer required is not licensed.

(1014) MSK_RES_ERR_FLEXLM ........ 500
The FLEXlm license manager reported an error.

(1015) MSK_RES_ERR_LICENSE_SERVER ........ 514
The license server is not responding.

(1016) MSK_RES_ERR_LICENSE_MAX ........ 514
Maximum number of licenses are reached.

(1017) MSK_RES_ERR_LICENSE_MOSEKLM_DAEMON ........ 514
The MOSEKLM license manager daemon is not up and running.

(1018) MSK_RES_ERR_LICENSE_FEATURE ........ 513
A feature is not available in the license file(s). This is mosek likely due to an incorrect setup of the license system.

(1019) MSK_RES_ERR_PLATFORM_NOT_LICENSED ........ 533
A license feature is not available for the required platform is not licensed a feature.

(1020) MSK_RES_ERR_LICENSE_CANNOT_ALLOCATE ........ 513
The license system cannot allocate the memory it requires.

(1021) MSK_RES_ERR_LICENSE_CANNOT_CONNECT ........ 513
MOSEK cannot connect to the license server. Most likely the license server is not up and running.

(1025) MSK_RES_ERR_LICENSE_INVALID_HOSTID ........ 514
The hostid specified in the license file does not match the hostid of the computer.

(1030) MSK_RES_ERR_OPEN_DL ........ 530
A dynamic link library could not be opened.

(1035) MSK_RES_ERR_OLDER_DLL ........ 530
The dynamic link library is older than the specified version.

(1036) MSK_RES_ERR_NEWER_DLL ........ 526
The dynamic link library is newer than the specified version.

(1040) MSK_RES_ERR_LINK_FILE_DLL ........ 514
A file cannot be linked to a stream in the DLL version.

(1045) MSK_RES_ERR_THREAD_MUTEX_INIT ........ 539
Could not initialize a mutex.

(1046) MSK_RES_ERR_THREAD_MUTEX_LOCK ........ 539
Could not lock a mutex.

(1047) MSK_RES_ERR_THREAD_MUTEX_UNLOCK ........ 539
Could not unlock a mutex.

(1048) MSK_RES_ERR_THREAD_CREATE ........ 539
Could not create a thread. This error may happen if a large number of environments are created and not deleted again. In any case it is a good practice to minimize the number of environments created.

(1049) MSK_RES_ERR_THREAD_COND_INIT ........ 538
Could not initialize a condition.

(1050) MSK_RES_ERR_UNKNOWN ........ 540
Unknown error.

(1051) MSK_RES_ERR_SPACE ........ 538
Out of space.

(1052) MSK_RES_ERR_FILE_OPEN ........ 499
Error while opening a file.

(1053) MSK_RES_ERR_FILE_READ ........ 499
File read error.

(1054) MSK_RES_ERR_FILE_WRITE ........ 500
File write error.

(1055) MSK_RES_ERR_DATA_FILE_EXT ........ 498
The data file format cannot be determined from the file name.

(1056) MSK_RES_ERR_INVALID_FILE_NAME ........ 510
An invalid file name has been specified.

(1057) MSK_RES_ERR_INVALID_SOL_FILE_NAME ........ 511
An invalid file name has been specified.

(1058) MSK_RES_ERR_INVALID_MBT_FILE ........ 510
A MOSEK binary task file is invalid.

(1059) MSK_RES_ERR_END_OF_FILE ........ 498
End of file reached.

(1060) MSK_RES_ERR_NULL_ENV ........ 528
`env` is a NULL pointer.

(1061) MSK_RES_ERR_NULL_TASK ........ 528
`task` is a NULL pointer.

(1062) MSK_RES_ERR_INVALID_STREAM ........ 511
An invalid stream is referenced.

(1150) MSK_RES_ERR_LP_INCOMPATIBLE ........ 516
The problem cannot be written to an LP formatted file.

(1151) MSK_RES_ERR_LP_EMPTY ........ 515
The problem cannot be written to an LP formatted file.

(1152) MSK_RES_ERR_LP_DUP_SLACK_NAME ........ 515
The name of the slack variable added to a ranged constraint already exists.

(1153) MSK_RES_ERR_WRITE_MPS_INVALID_NAME ........ 542
An invalid name is created while writing an MPS file. This will usually make the MPS file unreadable.

(1154) MSK_RES_ERR_LP_INVALID_VAR_NAME ........ 516
A variable name is invalid when used in an LP formatted file.

(1155) MSK_RES_ERR_LP_FREE_CONSTRAINT ........ 515
Free constraints cannot be written in LP file format.

(1156) MSK_RES_ERR_WRITE_OPF_INVALID_VAR_NAME ........ 543
Empty variable names cannot be written to OPF files.

(1157) MSK_RES_ERR_LP_FILE_FORMAT ........ 515
Syntax error in LP file.

(1158) MSK_RES_ERR_WRITE_LP_FORMAT ........ 542
Problem cannot be written as an LP file.

(1160) MSK_RES_ERR_LP_FORMAT ........ 515
Syntax error in LP file.

(1161) MSK_RES_ERR_WRITE_LP_NON_UNIQUE_NAME ........ 542
An auto generated name is not unique.

(1162) MSK_RES_ERR_READ_LP_NONEXISTING_NAME ........ 534
A variable never occurred in objective or constraints.

(1163) MSK_RES_ERR_LP_WRITE_CONIC_PROBLEM ........ 516
The problem contains cones that cannot be written to a LP formatted file.

(1164) MSK_RES_ERR_LP_WRITE_GECO_PROBLEM ........ 516
The problem contains general convex terms that cannot be written to a LP formatted file.

(1165) MSK_RES_ERR_NAME_MAX_LEN ........ 524
A name is longer than the buffer that is supposed to hold it.

(1168) MSK_RES_ERR_OPF_FORMAT ........ 530
Syntax error in OPF file

(1170) MSK_RES_ERR_INVALID_NAME_IN_SOL_FILE ........ 510
An invalid name occurred in a solution file.

(1197) MSK_RES_ERR_ARGUMENT_LENNEQ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 494
Wrong length of arguments.

(1198) MSK_RES_ERR_ARGUMENT_TYPE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 495
Wrong argument type.

(1199) MSK_RES_ERR_NR_ARGUMENTS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 528
Nr. of function arguments are wrong.

(1200) MSK_RES_ERR_IN_ARGUMENT . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 501
A function argument is incorrect.

(1201) MSK_RES_ERR_ARGUMENT_DIMENSION . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 494
A function argument is of incorrect dimension.

(1203) MSK_RES_ERR_INDEX_IS_TOO_SMALL . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 502
An index in an argument is too small.

(1204) MSK_RES_ERR_INDEX_IS_TOO_LARGE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 502
An index in an argument is too large.

(1205) MSK_RES_ERR_PARAM_NAME . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 531
The parameter name is not correct.

(1206) MSK_RES_ERR_PARAM_NAME_DOU . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 532
The parameter name is not correct for a double parameter.

(1207) MSK_RES_ERR_PARAM_NAME_INT . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 532
The parameter name is not correct for a integer parameter.

(1208) MSK_RES_ERR_PARAM_NAME_STR . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 532
The parameter name is not correct for a string parameter.

(1210) MSK_RES_ERR_PARAM_INDEX . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 531
Parameter index is out of range.

(1215) MSK_RES_ERR_PARAM_IS_TOO_LARGE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 531
The parameter value is too large.

(1216) MSK_RES_ERR_PARAM_IS_TOO_SMALL . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 531
The parameter value is too small.

(1217) MSK_RES_ERR_PARAM_VALUE_STR . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 532
The parameter value string is incorrect.

(1218) MSK_RES_ERR_PARAM_TYPE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 532
The parameter type is invalid.

(1219) MSK_RES_ERR_INF_DOU_INDEX . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 502
A double information index is out of range for the specified type.

(1220) MSK_RES_ERR_INF_INT_INDEX . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 502
An integer information index is out of range for the specified type.

(1221) MSK_RES_ERR_INDEX_ARR_IS_TOO_SMALL ........ 502
An index in an array argument is too small.

(1222) MSK_RES_ERR_INDEX_ARR_IS_TOO_LARGE ........ 501
An index in an array argument is too large.

(1230) MSK_RES_ERR_INF_DOU_NAME ........ 502
A double information name is invalid.

(1231) MSK_RES_ERR_INF_INT_NAME ........ 503
A integer information name is invalid.

(1232) MSK_RES_ERR_INF_TYPE ........ 503
The information type is invalid.

(1235) MSK_RES_ERR_INDEX ........ 501
An index is out of range.

(1236) MSK_RES_ERR_WHICHSOL ........ 542
The solution number `whichsol` does not exists.

(1237) MSK_RES_ERR_SOLITEM ........ 538
The solution item number `solitem` is invalid. Note for example MSK_SOL_ITEM_SNX is invalid for the basis solution.

(1238) MSK_RES_ERR_WHICHITEM_NOT_ALLOWED ........ 542
`whichitem` is unacceptable.

(1240) MSK_RES_ERR_MAXNUMCON ........ 517
The maximum number of constraints specified is smaller than the number of constants in the task.

(1241) MSK_RES_ERR_MAXNUMVAR ........ 518
The maximum number of variables specified is smaller than the number of variables in the task.

(1242) MSK_RES_ERR_MAXNUMANZ ........ 517
The maximum number of nonzeros specified for $A$ is smaller than the number of nonzeros in the current $A$.

(1243) MSK_RES_ERR_MAXNUMQNZ ........ 517
The maximum number of nonzeros specified for the $Q$ matrices is smaller than the number of nonzeros in the current $Q$ matrices.

(1250) MSK_RES_ERR_NUMCONLIM ........ 529
Maximum number of constraints limit is exceeded.

(1251) MSK_RES_ERR_NUMVARLIM ........ 529
Maximum number of variables limit is exceeded.

(1252) MSK_RES_ERR_TOO_SMALL_MAXNUMANZ ........ 539
Maximum number of non zeros allowed in $A$ is too small.

(1271) MSK_RES_ERR_INV_CONE_TYPE_STR ........ 505
Invalid cone type string encountered.

(1272) MSK_RES_ERR_INV_CONE_TYPE ........ 504
Invalid cone type code is encountered.

(1274) MSK_RES_ERR_INV_SKN ........ 508
Invalid value in `skn`.

(1280) MSK_RES_ERR_INV_NAME_ITEM ........ 505
An invalid name item code is used.

(1281) MSK_RES_ERR_PRO_ITEM ........ 533
An invalid problem is used.

(1283) MSK_RES_ERR_INVALID_FORMAT_TYPE ........ 510
Invalid format type.

(1285) MSK_RES_ERR_FIRSTI ........ 500
Invalid `firsti`.

(1286) MSK_RES_ERR_LASTI ........ 512
Invalid `lasti`.

(1287) MSK_RES_ERR_FIRSTJ ........ 500
Invalid `firstj`.

(1288) MSK_RES_ERR_LASTJ ........ 512
Invalid `lastj`.

(1290) MSK_RES_ERR_NONLINEAR_EQUALITY ........ 527
The model contains a nonlinear equality which defines a nonconvex set.

(1291) MSK_RES_ERR_NONCONVEX ........ 527
The optimization problem is nonconvex.

(1292) MSK_RES_ERR_NONLINEAR_RANGED ........ 527
The model contains a nonlinear ranged constraint which by definition defines a nonconvex set.

(1293) MSK_RES_ERR_CON_Q_NOT_PSD ........ 496
The quadratic constraint matrix is not positive semi-definite as expected for a constraint with finite upper bound. This results in a non-convex problem.

(1294) MSK_RES_ERR_CON_Q_NOT_NSD ........ 496
The quadratic constraint matrix is not negative semi-definite as expected for a constraint with finite upper bound. This results in a non-convex problem.

(1295) MSK_RES_ERR_OBJ_Q_NOT_PSD ........ 529
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 ........ 529
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 ........ 494
An invalid permutation array is specified.

(1300) MSK_RES_ERR_CONE_INDEX ........ 497
An index of a non existing cone has been specified.

(1301) MSK_RES_ERR_CONE_SIZE ........ 497
A cone with too few members are specified.

(1302) MSK_RES_ERR_CONE_OVERLAP ........ 497
A new cone which variables overlap with an existing cone has been specified.

(1303) MSK_RES_ERR_CONE_REP_VAR ........ 497
A variable is included multiple times in the cone.

(1304) MSK_RES_ERR_MAXNUMCONE ........ 517
The value specified for maxnumcone is too small.

(1305) MSK_RES_ERR_CONE_TYPE ........ 497
Invalid cone type specified.

(1306) MSK_RES_ERR_CONE_TYPE_STR ........ 497
Invalid cone type specified.

(1310) MSK_RES_ERR_REMOVE_CONE_VARIABLE ........ 535
A variable cannot be removed because it will make a cone invalid.

(1350) MSK_RES_ERR_SOL_FILE_NUMBER ........ 537
An invalid number is specified in a solution file.

(1375) MSK_RES_ERR_HUGE_C ........ 501
A huge value in absolute size is specified for one $c_j$.

(1400) MSK_RES_ERR_INFINITE_BOUND ........ 503
A finite bound value is too large in absolute value.

(1401) MSK_RES_ERR_INV_QOBJ_SUBI ........ 507
Invalid value in qosubi encountered.

(1402) MSK_RES_ERR_INV_QOBJ_SUBJ ........ 507
Invalid value in qosubj.

(1403) MSK_RES_ERR_INV_QOBJ_VAL ........ 507
Invalid value in qoval.

(1404) MSK_RES_ERR_INV_QCON_SUBK ........ 507
Invalid value in qcsubk.

(1461) MSK_RES_ERR_NAN_IN_BLC ........ 524
$l^c$ contains an invalid floating point value i.e. a NaN.

(1462) MSK_RES_ERR_NAN_IN_BUC ........ 524
$u^c$ contains an invalid floating point value i.e. a NaN.

(1470) MSK_RES_ERR_NAN_IN_C ........ 525
$c$ contains an invalid floating point value i.e. a NaN.

(1471) MSK_RES_ERR_NAN_IN_BLX ........ 524
$l^x$ contains an invalid floating point value i.e. a NaN.

(1472) MSK_RES_ERR_NAN_IN_BUX ........ 525
$u^x$ contains an invalid floating point value i.e. a NaN.

(1500) MSK_RES_ERR_INV_PROBLEM ........ 506
Invalid problem type. Properly a non-convex problem has been specified.

(1501) MSK_RES_ERR_MIXED_PROBLEM ........ 519
The problem contains both conic and nonlinear constraints.

(1550) MSK_RES_ERR_INV_OPTIMIZER ........ 506
An invalid optimizer has been chosen for the problem. This happens if the simplex or conic optimizer is chosen to optimize a nonlinear problem.

(1551) MSK_RES_ERR_MIO_NO_OPTIMIZER ........ 518
No optimizer is available for the current class of integer optimization problems.

(1552) MSK_RES_ERR_NO_OPTIMIZER_VAR_TYPE ........ 527
No optimizer is available for this class of optimization problems.

(1553) MSK_RES_ERR_MIO_NOT_LOADED ........ 518
The mixed-integer optimizer is not loaded.

(1580) MSK_RES_ERR_POSTSOLVE ........ 533
An error occurred during the postsolve. Please contact MOSEK support.

(1600) MSK_RES_ERR_NO_BASIS_SOL ........ 526
No basis solution is defined as expected.

(1610) MSK_RES_ERR_BASIS_FACTOR ........ 495
The factorization of the basis is invalid.

(1615) MSK_RES_ERR_BASIS_SINGULAR ........ 495
The basis is singular and hence cannot be factored.

(1650) MSK_RES_ERR_FACTOR ........ 498
An error occurred while factorizing a matrix.

(1700) MSK_RES_ERR_FEASREPAIR_CANNOT_RELAX ........ 498
An optimization problem cannot be relaxed. This is for instance the case for general nonlinear optimization problems.

(1701) MSK_RES_ERR_FEASREPAIR_SOLVING_RELAXED ........ 499
The relaxed problem could not be solved to optimality. Consult the log file for further details.

(1702) MSK_RES_ERR_FEASREPAIR_INCONSISTENT_BOUND ........ 499
A ranged constraint on a bound or variable has a lower bound that is larger than its upper bound. Pleas fix this before running feasibility repair.

(1800) MSK_RES_ERR_INVALID_COMPRESSION ........ 509
Invalid compression type.

(1801) MSK_RES_ERR_INVALID_IOMODE ........ 510
Invalid io mode.

(2000) MSK_RES_ERR_NO_PRIMAL_INFEAS_CER ........ 527
A primal infeasibility certificate is not available.

(2001) MSK_RES_ERR_NO_DUAL_INFEAS_CER ........ 526
A dual infeasibility certificate is not available.

(2500) MSK_RES_ERR_NO_SOLUTION_IN_CALLBACK ........ 527
The required solution is not available.

(2501) MSK_RES_ERR_INV_MARKI ........ 505
Invalid value in marki.

(2502) MSK_RES_ERR_INV_MARKJ ........ 505
Invalid value in markj.

(2503) MSK_RES_ERR_INV_NUMI ........ 505
Invalid numi.

(2504) MSK_RES_ERR_INV_NUMJ ........ 506
Invalid numj.

(2505) MSK_RES_ERR_CANNOT_CLONE_NL ........ 495
A task that has a nonlinear function callback cannot be cloned.

(2506) MSK_RES_ERR_CANNOT_HANDLE_NL ........ 496
A function cannot handle a task with nonlinear function callbacks.

(2520) MSK_RES_ERR_INVALID_ACCMODE ........ 509
An invalid access mode is specified.

(2550) MSK_RES_ERR_MBT_INCOMPATIBLE ........ 518
The MBT file is incompatible withe 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 ........ 516
Could not compute the LU factors of matrix within the maximum number of allowed tries.

(2900) MSK_RES_ERR_INVALID_UTF8 ........ 512
An invalid UTF8 string is encountered.

(2901) MSK_RES_ERR_INVALID_WCHAR ........ 512
An invalid wchar string is encountered.

(2950) MSK_RES_ERR_NO_DUAL_FOR_ITG_SOL ........ 526
No dual information is available for the integer solution.

(3000) MSK_RES_ERR_INTERNAL ........ 503
An internal error occurred. Please report this problem.

(3001) MSK_RES_ERR_API_ARRAY_TOO_SMALL ........ 493
An input array was too short.

(3002) MSK_RES_ERR_API_CB_CONNECT ........ 493

(3003) MSK_RES_ERR_API_NL_DATA ........ 494

(3004) MSK_RES_ERR_API_CALLBACK ........ 493

(3005) MSK_RES_ERR_API_FATAL_ERROR ........ 494
An internal error occurred in the API. Please report this problem.

(3050) MSK_RES_ERR_SEN_FORMAT ........ 535
Syntax error in sensitivity analysis file.

(3051) MSK_RES_ERR_SEN_UNDEF_NAME ........ 536
An undefined name was encountered in the sensitivity analysis file.

(3052) MSK_RES_ERR_SEN_INDEX_RANGE ........ 536
Index out of range in the sensitivity analysis file.

(3053) MSK_RES_ERR_SEN_BOUND_INVALID_UP ........ 535
Analysis of upper bound requested for an index, where no upper bound exists.

(3054) MSK_RES_ERR_SEN_BOUND_INVALID_LO ........ 535
Analysis of lower bound requested for an index, where no upper bound exists.

(3055) MSK_RES_ERR_SEN_INDEX_INVALID ........ 535
Invalid range given in sensitivity file.

(3056) MSK_RES_ERR_SEN_INVALID_REGEXP ........ 536
Syntax error in regexp or regexp longer than 1024

(3057) MSK_RES_ERR_SEN_SOLUTION_STATUS ........ 536
No optimal solution found to the original problem given for sensitivity analysis.

(3058) MSK_RES_ERR_SEN_NUMERICAL ........ 536
Numerical difficulties encountered doing sensitivity analysis.

(3059) MSK_RES_ERR_CONCURRENT_OPTIMIZER ........ 496
An unsupported optimizer was chosen for use with the concurrent optimizer.

(3100) MSK_RES_ERR_UNB_STEP_SIZE 540
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 this is an error occurs. Normally this will only happen if the problem is badly formulated. In any case it is suggested to contact MOSEK support in case this error occurs.

(3101) MSK_RES_ERR_IDENTICAL_TASKS 501
Some tasks related to this function call was identical. Unique tasks were expected.

(3200) MSK_RES_ERR_INVALID_BRANCH_DIRECTION 509
An invalid branching direction is specified.

(3201) MSK_RES_ERR_INVALID_BRANCH_PRIORITY 509
An invalid branching priority is specified. It should nonnegative.

(3500) MSK_RES_ERR_INTERNAL_TEST_FAILED 503
An internal unit test function failed.

(3600) MSK_RES_ERR_XML_INVALID_PROBLEM_TYPE 543
The problem type is not supported by the XML format.

(3700) MSK_RES_ERR_INVALID_AMPL_STUB 509
Invalid AMPL stub.

(3999) MSK_RES_ERR_API_INTERNAL 494

(4000) MSK_RES_TRM_MAX_ITERATIONS 544
The optimizer was terminated on maximum number of iterations.

(4001) MSK_RES_TRM_MAX_TIME 544
The optimizer was terminated on maximum amount of time.

(4002) MSK_RES_TRM_OBJECTIVE_RANGE 546
The optimizer was terminated on the objective range.

(4003) MSK_RES_TRM_MIO_NEAR_REL_GAP 545
The mixed-integer optimizer was terminated because the near optimal relative gap tolerance was satisfied.

(4004) MSK_RES_TRM_MIO_NEAR_ABS_GAP 544
The mixed-integer optimizer was terminated because the near optimal absolute gap tolerance was satisfied.

(4005) MSK_RES_TRM_USER_BREAK 546
The optimizer was terminated on a user break.

(4006) MSK_RES_TRM_STALL 546
The optimizer terminated due to it makes so slow progress that it is not worthwhile to continue. Normally there can be two reasons why this happen. Either there is a bug in MOSEK or the problem is badly formulated. In general we recommend that MOSEK support is contacted if this happens.

(4007) MSK_RES_TRM_USER_CALLBACK .......... 546
The optimizer terminated due to the return of the user defined call-back function.

(4008) MSK_RES_TRM_MIO_NUM_RELAXS .......... 545
The mixed-integer optimizer was terminated due to the maximum number relaxations was reached.

(4009) MSK_RES_TRM_MIO_NUM_BRANCHES .......... 545
The mixed-integer optimizer was terminated due to the maximum number branches was reached.

(4015) MSK_RES_TRM_NUM_MAX_NUM_INT_SOLUTIONS .......... 545
The mixed-integer optimizer was terminated due to the maximum number feasible solutions was reached.

(4020) MSK_RES_TRM_MAX_NUM_SETBACKS .......... 544
The optimizer terminated due to the maximum number of setbacks is reached. This indicates serious numerical problems and a possibly badly formulated problem.

(4025) MSK_RES_TRM_NUMERICAL_PROBLEM .......... 546
The optimizer terminated due to a numerical problem. This indicates serious numerical problems and a possibly badly formulated problem.

(4030) MSK_RES_TRM_INTERNAL .......... 543
The optimizer terminated due to some internal reason.

(4031) MSK_RES_TRM_INTERNAL_STOP .......... 544
The optimizer terminated due to some internal reason.

- `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:**
  Wrong length of arguments.

  **Response message string:**
  "Wrong 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:**
Wrong argument type.

**Response message string:**
"Wrong 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 that has a nonlinear function callback cannot be cloned.

**Response message string:**
"A task that has a nonlinear function callback 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 callbacks.

**Response message string:**
"A function cannot handle a task with nonlinear function callbacks."

- 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 upper bound. This results in a non-convex 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 non-convex 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:**
"An unsupported optimizer was chosen for use with 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 are specified.

  **Response message string:**
  "`A cone with too few member are 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 a 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 for instance the case 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:**
A ranged constraint on a bound or variable has a lower bound that is larger than its upper bound. Pleas fix this before running 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. 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 was identical. Unique tasks were expected.

**Response message string:**
"Some tasks related to this function call was 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 happens if the simplex or 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. Properly a non-convex 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 encountered.

**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 in 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 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 pointer.

**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:**
"License error."

- err_license_cannot_allocate

**Corresponding constant:**
MSK_RES_ERR_LICENSE_CANNOT_ALLOCATE

**Description:**
The license system cannot allocate the memory it requires.

**Response message string:**
"The license system cannot allocate the memory it requires."

- 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:**
"License has expired."

- err_license_feature

**Corresponding constant:**
MSK_RES_ERR_LICENSE_FEATURE

**Description:**
A feature is not available in the license file(s). This is mosek likely due to an incorrect setup of the license system.

**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 hostid specified in the license file does not match the hostid of the computer.

  **Response message string:**
  "The hostid specified in the license file does not match the hostid of the computer."

- err_license_max

  **Corresponding constant:**
  MSK_RES_ERR_LICENSE_MAX

  **Description:**
  Maximum number of licenses are reached.

  **Response message string:**
  "Maximum number of licenses are 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_version

  **Corresponding constant:**
  MSK_RES_ERR_LICENSE_VERSION

  **Description:**
  The license is valid for another version of MOSEK.

  **Response message string:**
  "Feature %s version %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 LP file.

**Response message string:**
"Syntax error in LP file at (%d:%d)."

- err_lp_format

**Corresponding constant:**
MSK_RES_ERR_LP_FORMAT

**Description:**
Syntax error in LP file.

**Response message string:**
"Syntax error in 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 a LP formatted file.

  **Response message string:**
  "A problem of type '%s' contains cones that cannot be written to a 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 a LP formatted file.

  **Response message string:**
  "A problem of type '%s' contains general convex terms that cannot be written to a 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 matrix within the maximum number of allowed tries.

**Response message string:**
"Could not compute the LU factors of matrix within the maximum number of allowed tries."

- err_maxnumanz

**Corresponding constant:**
MSK_RES_ERR_MAXNUMANZ

**Description:**
The maximum number of nonzeros specified for $A$ is smaller than the number of nonzeros in the current $A$.

**Response message string:**
"Too small maximum number of nonzeros in A specified."

- err_maxnumcon

**Corresponding constant:**
MSK_RES_ERR_MAXNUMCON

**Description:**
The maximum number of constraints specified is smaller than the number of constants 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 nonzeros specified for the $Q$ matrices is smaller than the number of nonzeros in the current $Q$ matrices.

**Response message string:**
"Too small maximum number of nonzeros 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 withe 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 appropriately setup. 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 member of several cones.

  **Response message string:**
  "`Variable '%s' is specified to be 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 a MPS file.

**Response message string:**
"`An error occurred while reading a MPS file.`"

- `err_mps_inv_bound_key`

**Corresponding constant:**
`MSK_RES_ERR_MPS_INV_BOUND_KEY`

**Description:**
An invalid bound key occurred in a 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 a 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 a 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 is 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 a 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 a MPS file.

  **Response message string:**
  "An empty constraint name is used in a MPS file."

- err_mps_null_var_name

  **Corresponding constant:**
  MSK_RES_ERR_MPS_NULL_VAR_NAME

  **Description:**
  An empty variable name is used in a MPS file.

  **Response message string:**
  "An empty variable name is used in a MPS file."

- err_mps_splitted_var

**Corresponding constant:**
MSK_RES_ERR_MPS_SPLITTED_VAR

**Description:**
A variable is split in a 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 a 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 a 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:**
A 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 basis solution is defined as expected.

**Response message string:**
"No basis solution is defined as expected."

- 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 dual infeasibility certificate is not available.

**Response message string:**
"A dual infeasibility certificate 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 primal infeasibility certificate is not available.

  **Response message string:**
  "A primal infeasibility certificate 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:**
  Nr. of function arguments are wrong.

  **Response message string:**
  "Wrong 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 OPF file

  **Response message string:**
  "Syntax error in 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:**
An invalid branch ordering file has an invalid content.

**Response message string:**
"An invalid branch ordering file has an invalid content."

- 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 a 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 license feature is not available for the required platform is not licensed a feature.

**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 upper 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 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 doing sensitivity analysis.

  **Response message string:**
  "Numerical difficulties encountered doing 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:**
"The 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. Note for example MSK_SOL_ITEM_SNX is invalid for the basis 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 either be due to 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 information is available about the space usage.

  **Response message string:**
  "No information is available 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 happen 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 this is an error occurs. Normally this will only happen if the problem is badly formulated. In any case it is suggested to contact MOSEK support in case 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 nonzero 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 Hessian an invalid subscript.

  **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 number whichsol 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. This will usually 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.

**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 due to some internal reason.

  **Response message string:**
  "`The optimizer terminated due to some internal reason.`"

- `trm_max_iterations`

  **Corresponding constant:**
  `MSK_RES_TRM_MAX_ITERATIONS`

  **Description:**
  The optimizer was terminated on 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 due to the maximum number of setbacks is reached. This indicates serious numerical problems and a possibly badly formulated problem.

  **Response message string:**
  "`The optimizer terminated due to the maximum number of setbacks is reached.`"

- `trm_max_time`

  **Corresponding constant:**
  `MSK_RES_TRM_MAX_TIME`

  **Description:**
  The optimizer was terminated on 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 was terminated because the near optimal absolute gap tolerance was satisfied.

**Response message string:**
"The mixed-integer optimizer was 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 was terminated because the near optimal relative gap tolerance was satisfied.

**Response message string:**
"The mixed-integer optimizer was 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 was terminated due to the maximum number branches was reached.

**Response message string:**
"The mixed-integer optimizer was terminated due to the maximum number branches was reached."

- trm_mio_num_relaxs

**Corresponding constant:**
MSK_RES_TRM_MIO_NUM_RELAXS

**Description:**
The mixed-integer optimizer was terminated due to the maximum number relaxations was reached.

**Response message string:**
"The mixed-integer optimizer was terminated due to the maximum number relaxations was reached."

- trm_num_max_num_int_solutions

**Corresponding constant:**
MSK_RES_TRM_NUM_MAX_NUM_INT_SOLUTIONS

**Description:**
The mixed-integer optimizer was terminated due to the maximum number feasible solutions was reached.

**Response message string:**
"The mixed-integer optimizer was terminated due to the maximum number feasible solutions was reached."

- trm_numerical_problem

  **Corresponding constant:**
  MSK_RES_TRM_NUMERICAL_PROBLEM

  **Description:**
  The optimizer terminated due to a numerical problem. This indicates serious numerical problems and a possibly badly formulated problem.

  **Response message string:**
  "The optimizer terminated due to a numerical problem."

- trm_objective_range

  **Corresponding constant:**
  MSK_RES_TRM_OBJECTIVE_RANGE

  **Description:**
  The optimizer was terminated on 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 it makes so slow progress that it is not worthwhile to continue. Normally there can be two reasons why this happen. Either there is a bug in MOSEK or the problem is badly formulated. In general we recommend that MOSEK support is contacted 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 was terminated on 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 nonzero elements are dropped from the Q matrix in the objective.

**Response message string:**
"'%d' nonzero element(s) are dropped from 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 large value in absolute size 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 large 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)."

- 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:**
Ignore a variable because the variable has not been previously defined. Usually this implies a variable has been deigned in the bound section but not in the objective or the constraints.

**Response message string:**
"The variable '%s' is ignored because the variable has not been 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:**
When the MOSEK mixed integer optimizer reoptimizes a mixed integer problem with all the integer variables fixed at their "optimal value" the then problem becomes infeasible. Sometimes the problem can be resolved by reducing the tolerances MSK_DPAR_MIO_TOL_ABS_RELAX_INT and MSK_DPAR_MIO_TOL_REL_RELAX_INT.

**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 a 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 a 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:**
A RHS vector is split into several nonadjacent parts in a 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:**
Nonzero elements is specified in the upper triangle of a matrix which is ignored by the code.

**Response message string:**
"Nonzero 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 in too low precision.

**Response message string:**
"The presolve estimates that the model is specified in too low 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 string parameter is longer than the buffer that is supposed to hold it.

  **Response message string:**
  "A value for 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. Too few basis variables are specified.

**Response message string:**
"%d number of 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 19

# Constants

## 19.1 Constraint or variable access modes

(1) MSK_ACC_CON
Access constraints or equivalently rows

(0) MSK_ACC_VAR
Access variables or equivalently columns

## 19.2 Basis identification

(1) MSK_BI_ALWAYS
Basis identification is always performed even though 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 that says 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.

## 19.3 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 a infinite lower bound and an finite upper bound.

## 19.4 Specifies the branching direction.

(2) `MSK_BRANCH_DIR_DOWN`
The mixed integer optimizer always chooses the up 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 down branch first.

## 19.5 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

(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 has been terminated.

(26) MSK_CALLBACK_END_CONCURRENT
Concurrent optimizer is terminated.

(27) MSK_CALLBACK_END_CONIC
The call-back function is called when 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 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 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 phase 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 callback does not indicate a new phase in the optimization, but is simply a time-triggered callback.

(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 is started. The frequency of the call back is controlled by 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 at after a new integer solution has been located by 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 back is controlled by MSK_IPAR_LOG_SIM_FREQ parameter.

## 19.6 Types of convexity checks.

(0) MSK_CHECK_CONVEXITY_NONE
No convexity check

(1) MSK_CHECK_CONVEXITY_SIMPLE
Perform simple and fast convexity check

## 19.7 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.

## 19.8 Cone types

(0) MSK_CT_QUAD
The cone is a quadratic cone.

(1) MSK_CT_RQUAD
The cone is a rotated quadratic cone.

## 19.9 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
A 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.

## 19.10 Data format types

(0) MSK_DATA_FORMAT_EXTENSION
The extension of the file name 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 a optimization problem formatted file.

(5) MSK_DATA_FORMAT_XML
The data file is a XML formatted file.

## 19.11 Double information items

(0) MSK_DINF_BI_CLEAN_CPUTIME
Time (in CPU seconds) spend within clean-up phase basis identification procedure since its invocation.

(1) MSK_DINF_BI_CPUTIME
Time (in CPU seconds) spend within basis identification procedure since its invocation.

(2) MSK_DINF_BI_DUAL_CPUTIME
Time (in CPU seconds) spend within dual phase basis identification procedure since its invocation.

(3) MSK_DINF_BI_PRIMAL_CPUTIME
Time (in CPU seconds) spend within primal phase of the basis identification procedure since its invocation.

(4) MSK_DINF_CONCURRENT_CPUTIME
Time (in CPU seconds) spend 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) spend 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 an primal-dual optimal solution. Whereas it should converge to infinity when the problem is (strictly) primal or dual infeasible. In the case the measure is converging towards a positive but bounded constant then 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 optimizer. (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 end within the interior-point optimizer since its invocation.

(15) MSK_DINF_MIO_CONSTRUCT_SOLUTION_OBJ
If MOSEK successfully constructed a integer feasible solution, then this item contains the optimal objective value corresponding to feasible solution.

(16) MSK_DINF_MIO_CPUTIME
Time spend 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 is defined by

$$|(\text{objective value of feasible solution}) - (\text{objective bound})|.$$

Otherwise it has the value -1.0.

The best bound objective value corresponding to the best integer feasible solution located. Note at least one integer feasible solution must have located i.e. check MSK_IINF_MIO_NUM_INT_SOLUTIONS.

(18) MSK_DINF_MIO_OBJ_BOUND
The best bound objective value corresponding to the best integer feasible solution located. Note at least one integer feasible solution must have 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 located. Note 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 the mixed integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the relative gap is 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) spend in the optimizer since it was invoked.

(23) MSK_DINF_OPTIMIZER_REALTIME
Total time (in wall-clock seconds) spend in the optimizer since it was invoked.

(24) MSK_DINF_PRESOLVE_CPUTIME
Total time (in CPU second) spend in the presolve since it was invoked.

(25) MSK_DINF_PRESOLVE_ELI_CPUTIME
Total time (in CPU second) spend in the eliminator since the presolve was invoked.

(26) MSK_DINF_PRESOLVE_LINDEP_CPUTIME
Total time (in CPU second) spend in the linear dependency checker since the presolve was invoked.

(27) MSK_DINF_RD_CPUTIME
Time (in CPU seconds) spend reading the data file.

(28) MSK_DINF_SIM_CPUTIME
Time (in CPU seconds) spend 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 basis solution. Updated at the end of the optimization.

(32) MSK_DINF_SOL_BAS_MAX_DBI
Maximal dual bound infeasibility in the basis solution. Updated at the end of the optimization.

(33) MSK_DINF_SOL_BAS_MAX_DEQI
Maximal dual equality infeasibility in the basis solution. Updated at the end of the optimization.

(34) MSK_DINF_SOL_BAS_MAX_PBI
Maximal primal bound infeasibility in the basis solution. Updated at the end of the optimization.

(35) `MSK_DINF_SOL_BAS_MAX_PEQI`
Maximal primal equality infeasibility in the basis solution. Updated at the end of the optimization.

(36) `MSK_DINF_SOL_BAS_MAX_PINTI`
Maximal primal integer infeasibility in the basis solution. Updated at the end of the optimization.

(37) `MSK_DINF_SOL_BAS_PRIMAL_OBJ`
Primal objective value of the basis 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 basis 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 interior point solution. Updated at the end of the optimization.

(43) `MSK_DINF_SOL_ITR_MAX_DBI`
Maximal dual bound infeasibility in interior point solution. Updated at the end of the optimization.

(44) `MSK_DINF_SOL_ITR_MAX_DCNI`
Maximal dual cone infeasibility in interior point solution. Updated at the end of the optimization.

(45) `MSK_DINF_SOL_ITR_MAX_DEQI`
Maximal dual equality infeasibility in interior point solution. Updated at the end of the optimization.

(46) `MSK_DINF_SOL_ITR_MAX_PBI`
Maximal primal bound infeasibility in interior point solution. Updated at the end of the optimization.

(47) `MSK_DINF_SOL_ITR_MAX_PCNI`
Maximal primal cone infeasibility in interior point solution. Updated at the end of the optimization.

(48) `MSK_DINF_SOL_ITR_MAX_PEQI`
Maximal primal equality infeasibility in interior point solution. Updated at the end of the optimization.

(49) MSK_DINF_SOL_ITR_MAX_PINTI
Maximal primal integer infeasibility in interior point solution. Updated at the end of the optimization.

(50) MSK_DINF_SOL_ITR_PRIMAL_OBJ
Primal objective value of interior point solution. Updated at the end of the optimization.

## 19.12 Double values

(1.0e30) MSK_INFINITY
Definition of infinity.

## 19.13 Feasibility repair types

(2) MSK_FEASREPAIR_OPTIMIZE_COMBINED

(0) MSK_FEASREPAIR_OPTIMIZE_NONE

(1) MSK_FEASREPAIR_OPTIMIZE_PENALTY

## 19.14 Integer information items.

(0) MSK_IINF_BI_ITER
Number simplex pivots performed since invoking 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 that in 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 the interior-point optimizer is using.

(9) MSK_IINF_INTPNT_SOLVE_DUAL
Nonzero if 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. whereas if it has a positive value, the MOSEK successfully constructed an initial integer feasible solution.

(11) MSK_IINF_MIO_INITIAL_SOLUTION
Is nonzero 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 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 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 nonzero, then the objective cut is used.

(41) MSK_IINF_OPT_NUMCON
Number of constraints in the problem solved when optimize is called.

(42) MSK_IINF_OPT_NUMVAR
Number of variables in the problem solved when optimize is called

(43) MSK_IINF_OPTIMIZE_RESPONSE
The reponse code returned by optimize.

(44) MSK_IINF_RD_NUMANZ
Number of nonzeros in A 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 Q nonzeros.

(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 nonzero is dual problem is solved.

(65) `MSK_IINF_SOL_BAS_PROSTA`
Problem status of the basis solution. Updated after each optimization.

(66) `MSK_IINF_SOL_BAS_SOLSTA`
Solution status of the basis 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 solution. Updated after each optimization.

(71) `MSK_IINF_STO_NUM_A_CACHE_FLUSHES`
Number of times the cache of $A$ elements are flushed. A large number implies `maxnumanz` is too small and 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 can occur.

(73) `MSK_IINF_STO_NUM_A_TRANSPOSES`
Number of times the $A$ matrix is transposed. A large number implies `maxnumanz` is too small or an inefficient usage of MOSEK. This will in particular occur if the code alternate between accessing rows and columns of $A$.

## 19.15 Information item types

(0) MSK_INF_DOU_TYPE
Is a double information type.

(1) MSK_INF_INT_TYPE
Is an integer.

## 19.16 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.

## 19.17 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.

## 19.18 Continuous mixed integer solution type

(2) MSK_MIO_CONT_SOL_ITG
The reported interior-point and basis 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 the case the problem has a primal feasible solution.

(3) MSK_MIO_CONT_SOL_ITG_REL
In the case the problem is primal feasible then the reported interior-point and basis 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 or basis solutions are reported when the mixed integer optimizer is used.

(1) MSK_MIO_CONT_SOL_ROOT
The reported interior-point and basis solutions are a solution to the root node problem when mixed integer optimizer is used.

## 19.19 Integer restrictions

(0) MSK_MIO_MODE_IGNORED
The integer constraints are ignored and the problem is solved as 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.

## 19.20 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.

## 19.21 MPS file format type

(2) MSK_MPS_FORMAT_FREE
It is assumed the input file satisfies the free MPS format. This implies spaces are not allowed names. On the other hand 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.

## 19.22 Message keys

(1100) MSK_MSG_MPS_SELECTED

(1000) MSK_MSG_READING_FILE

(1001) MSK_MSG_WRITING_FILE

## 19.23 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

## 19.24 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.

## 19.25 On/off

(0) MSK_OFF
Switch the option off.

(1) MSK_ON
Switch the option on.

## 19.26 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 choice of optimizer is made 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.

## 19.27 Ordering strategy

(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 automatically chosen.

(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.

## 19.28 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.

## 19.29 Presolve method.

(2) `MSK_PRESOLVE_MODE_FREE`
It is automatically decided whether the presolved 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.

## 19.30 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.

## 19.31 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

(2) MSK_PROBTYPE_QCQO
The problem is a quadratically constrained optimization.

(1) MSK_PROBTYPE_QO
The problem is a quadratic optimization problem.

## 19.32 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 might 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.

## 19.33 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.

## 19.34 Response code type

(3) MSK_RESPONSE_ERR
The response code is an error .

(0) MSK_RESPONSE_OK
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.

## 19.35 Scaling type

(3) MSK_SCALING_AGGRESSIVE
A very aggressive scaling is performed.

(0) MSK_SCALING_FREE
The optimizer choose the scaling heuristic.

(2) MSK_SCALING_MODERATE
A conservative scaling is performed.

(1) MSK_SCALING_NONE
No scaling is performed.

## 19.36 Sensitivity types

(0) MSK_SENSITIVITY_TYPE_BASIS

(1) MSK_SENSITIVITY_TYPE_OPTIMAL_PARTITION

## 19.37 Degeneracy strategies

(2) MSK_SIM_DEGEN_AGGRESSIVE
The simplex optimize should use a aggressive degeneration strategy.

(1) MSK_SIM_DEGEN_FREE
The simplex optimize chooses the degeneration strategy.

(4) MSK_SIM_DEGEN_MINIMUM
The simplex optimize should use minimum degeneration strategy.

(3) MSK_SIM_DEGEN_MODERATE
The simplex optimize should use a moderate degeneration strategy.

(0) MSK_SIM_DEGEN_NONE
The simplex optimize should use no degeneration strategy.

## 19.38 Hotstart type employed by the simplex optimizer.

(1) MSK_SIM_HOTSTART_FREE
The simplex optimize chooses the hotstart 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 hotstart.

## 19.39 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 choose the pricing strategy.

(1) MSK_SIM_SELECTION_FULL
The optimizer uses full pricing.

(5) MSK_SIM_SELECTION_PARTIAL
The optimizer uses an 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).

## 19.40 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.

## 19.41 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 a 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.

## 19.42 Solution types

(1) MSK_SOL_BAS
The basic solution.

(2) MSK_SOL_ITG
The integer solution.

(0) MSK_SOL_ITR
The interior solution.

## 19.43 Solve primal or dual

(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.

## 19.44 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.

## 19.45 Starting point types

(1) `MSK_STARTING_POINT_CONSTANT`
The starting point is chosen to constant. This is more reliable than a non-constant starting point.

(0) `MSK_STARTING_POINT_FREE`
The starting point is chosen automatically.

## 19.46 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.

## 19.47 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.

## 19.48 Variable types

(0) MSK_VAR_TYPE_CONT
Is a continuous variable.

(1) MSK_VAR_TYPE_INT
Is an integer variable.

## 19.49 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 of Nazareth [19] 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

$$\begin{array}{rcccl} l^c & \leq & Ax + q(x) & \leq & u^c, \\ l^x & \leq & x & \leq & u^x, \\ & & x \in \mathcal{C}, & & \\ & & x_{\mathcal{J}} \text{ integer}, & & \end{array} \tag{B.1}$$

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

  $$Q^i = (Q^i)^T. \tag{B.2}$$

  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 capital are keywords of the MPS format, and names appearing brackets are user 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 capital names indicates 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 at most one line which can be one of the following lines

```
    MIN
    MINIMIZE
    MAX
    MAXIMIZE
```

It should be obvious what the implication is of each those 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 at most one line 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]`. 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` which have the interpretation:

| Constraint type | $l_i^c$ | $u_i^c$ |
|---|---|---|
| `E` | finite | $l_i^c$ |
| `G` | finite | $\infty$ |
| `L` | $-\infty$ | finite |
| `N` | $-\infty$ | $\infty$ |

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]` specifies $j$ and $i$ of $a_{ij}$ respectively. Moreover, `[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 should not be specified. However, 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 this name has been assigned to the $i$th constraint and $v_1$ denotes the value specified by `[value1]`, then the interpretation of $v_1$ is:

| Constraint 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] is interpretated in the same way. Note it is not necessary to specify zero elements, because elements by assumption 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. [cname1] is a valid constraint name. Assume this name is assigned to the $i$th constraint and let $v_1$ be the value specified by [value1], then a record has the interpretation:

| Constraint type | Sign of $v_1$ | $l_i^c$ | $u_i^c$ |
|---|---|---|---|
| E | - | $u_i^c + v_1$ | |
| E | + | | $l_i^c + v_1$ |
| G | - or + | | $l_i^c + \|v_1\|$ |
| L | - or + | $u_i^c - \|v_1\|$ | |
| N | | | |

### 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 the quadratic term belongs to. 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 $k$th and $j$th variable, then $Q^i_{kj}$ is assigned the value given by `[value1]` An optional second element is specified in the same way by the fields `[vname1]`, `[vname3]`, and `[value2]`.

The example

$$\begin{array}{lcl} \text{minimize} & -x_2 + 0.5(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & \\ \text{subject to} & x_1 + x_2 + x_3 & \geq \quad 1, \\ & x \geq 0 & \end{array}$$

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 `QSECTION`s it should be noted that:

- Only one `QSECTION` is allowed for each constraint.
- The `QSECTION`s can appear in an arbitrary order after the `COLUMNS` section.
- All variable names occurring in the `QSECTION` must have been 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 | Required | Description |
|---|---|---|---|---|
| `??` | 2 | 2 | Yes | Bound key |
| `[name]` | 5 | 8 | Yes | Name of the `BOUNDS` vector |
| `[vname1]` | 15 | 8 | Yes | Variable name |
| `[value1]` | 25 | 12 | No | Variable name |

Hence, a record in the `BOUNDS` section has the following interpretation. `[name]` is the name of the bound vector. `[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` | $-\infty$ | $\infty$ | No |
| `FX` | $v_1$ | $v_1$ | No |
| `LO` | $v_1$ | unchanged | No |
| `MI` | $-\infty$ | unchanged | No |
| `PL` | unchanged | $\infty$ | No |
| `UP` | unchanged | $v_1$ | No |
| `BV` | 0 | 1 | Yes |
| `LI` | $\lceil v_1 \rceil$ | $\infty$ | 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 assumed that $\mathcal{C}$ satisfy the following requirements. Let

$$x^t \in R^{n^t}, \; t = 1, \ldots, k$$

be vectors comprised of parts of the decision variables $x$ such that each decision variable is a member of exactly **one** vector $x^t$. For example we could have

$$x^1 = \begin{bmatrix} x_1 \\ x_4 \\ x_7 \end{bmatrix} \text{ and } x^2 = \begin{bmatrix} x_6 \\ x_5 \\ x_3 \\ x_2 \end{bmatrix}.$$

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 but membership of the $R$ set is not specified. The reason is if a variable is not a member of any other cone then it is a member of a $R$ cone.

Next let us study an example. Assume 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
*23456789012345678901234567890123456789012345678901234567890
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 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 specfies the rotated quadratic cone (B.6). Note the keyword RQUAD in the CSECTION which is used 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 | $\geq 1$ | Quadratic cone i.e. (B.3). |
| RQUAD | $\geq 2$ | Rotated quadratic cone i.e. (B.4). |

Observe 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 requirement for each field is

| 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 variable must only occur in 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 member of $\mathcal{J}$. However, an alternative method is available.

This method is only available 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.
```

Note special markers lines are used to indicate the start and the end of the integer variables. It should be observed that

- 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 that is not intended, then the correct bounds should defined in the BOUNDS section of the MPS formatted file.
- MOSEK ignores field 1 i.e. MARK0001 and MARK001. However, other optimization systems requires 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 is 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 it is less restrictive e.g. it allows longer names. However, there are three main limitations, which are:

- 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` then an arbitrary large line width can be accepted.
- A name must not contain any blanks.

If the free MPS format should be used instead of the default MPS format, then 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

  $$q^o(x) = x^T Q^o x$$

  where 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.
- $q : R^n \rightarrow R$ 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, the 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 between 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
```

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 then if the same variables multiplied or squared occurs 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 `<=` means the same), and the bound may be any number.

In the standart 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 "`:`" after the constraint name, ie.

$$-5 \leq x_1 + x_2 \leq 5 \tag{C.3}$$

may be written as

```
con:: -5 < x_1 + x_2 < 5
```

By default MOSEK automaticly writes ranged constraints this way.

If the files must adhere to the LP standard, ranged constraints must be either be split into upper bounded and a lower bounded constrants or it must 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 lower bound is $-\infty$ and upper bound is $+\infty$. A variable may also be assigned a finite lower and upper bound. The bound definitions may be written in one or two lines, and bounds may 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 two last and optional sections 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 a 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 may not be a number, a period or the letter 'e' or 'E' (to distinguish between numbers and names; a variable `e12` with coefficient `4.5` could be written as `4.5e12` - indistinguishable from a number). Keywords may 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

As strict definition of the LP format employed by some optimization software packages does not

- allow quadratic terms in the constraints,
- allow names to contain more than 16 characters
- and lines may not exceed 255 characters in length.
- Finally, names must obey the rules stated in Section C.2.2.2.

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 be 1024 characters and names may have any length (shorter than the 1024 characters).

Internally in MOSEK names may contain any (printable) characters, many of which cannot be written as 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 formated 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 make 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 interpretation of the LP format. To make the LP format written by MOSEK more compatible with other vendors definition of the LP format then 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 systematically renamed in the output file.

### C.2.4 Formatting of an LP file

There are a few parameters controlling the visual formatting of LP files written by MOSEKin order to make it easier to read them. 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. The third parameter means the same, only for constraints instead.

#### 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 files is read (but unnamed constraints in MOSEK are written without name).

# 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 also be written as an OPF file; furthermore it naturally accommodates ranged constraints and variables as well as arbitrary characters in names, fixed expression 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 it 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 might 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 quoted [/tag]
```

Unnamed arguments are identified by their order, while named arguments may appear in any order, but never before a unnamed argument. The `value` can be an 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 may 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 (in the above example '`min`') is either `min` or `max` (regardless of case) defines the objective sense, and the second (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]`) that 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 on the number of variables, constraints and non-zeros. Placed before all other sections containing data this may reduce the time used for reading the file.

  In the `hints` section, any subsection which is not recognized by MOSEK is simply ignored. In this section a hint is defined in a subsection 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 nonzeros in constraints) and `numqnz` (number of quadratic nonzeros in constraints).

- `[solutions]` This section can contain a number a full or a 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 warm-start or initial solution for an mixed integer problem, the safe thing is always to let it have 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 variable associated with its lower bound.
- `su` Defines the level of variable associated with its upper bound.
- `sn` Defines the level of 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 next line-break any text may be written, including markup characters.

### D.2.2 Numbers

Numbers, as used for parameter values or coefficients, are written as usually done by the printf function. That is, it 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, but in some cases they must be enclosed by quotes (single or double) and quoting characters must be preceded by a backslash. Unquoted names must begin with a letter (a-z or A-Z) and otherwise contain only 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 something 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`. 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`, in include all parameter settings.

## D.5 Examples

This section contains a set of small examples as written in OPF, 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 looks 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:

$$\begin{array}{lccl} \text{minimize} & 1x_1 + 2x_2 & & \\ \text{subject to} & 2x_3 + 4x_4 & = & 5, \\ & x_5^2 & \leq & 2x_1x_3, \\ & x_6^2 & \leq & 2x_2x_4, \\ & x_5 & = & 1, \\ & x_6 & = & 1, \\ & x \geq 0. & & \end{array} \tag{D.4}$$

Note that the type of the cones is defined by the parameter to `[cone ...]`; the content of the `cone`-section is the names off variables which 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 pleas see `http://www.optimizationservices.org/`. Only linear constraints (possibly with integer variables) are supported. Output files with the extension `.xml` is by default 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 in which order the mixed integer optimizer branch on variables. The format of a ORD file is shown in Figure F.1. In the figure capital names are keywords of the ORD

```
*        1         2         3         4         5         6
*23456789012345678901234567890123456789012345678901234567890
NAME          [name]
 ?? [vname1]              [value1]
ENDATA
```

Figure F.1: The standard ORD format.

format, whereas names appearing in brackets are user defined names or values. The ?? is an optional key specifing the preferred branch direction. The possible keys are `DN` and `UP` which implies that down and up are the preferred the branching direction respectively. The branching direction key is optional and if it is left blank then the mixed integer optimizer will decide whether it is best to branch up or down.

## 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 a priority value assigned to 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 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 of 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 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 which presents the optimal basis solution. Finally, if the problem contain 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 then a solution report consists of three section i.e.

`HEADER` In this section the name of the problem is first listed. Next the problem and solution status 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 value is displayed.

`CONSTRAINTS` Subsequently in the constraint section is following information listed for each constraint:

- `INDEX` A sequential index assigned to the constraint by MOSEK.
- `NAME` The name of constraint assigned by the user.

| 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.

`AT` The status of the constraint. In Table G.1 are the possible values of the status key shown and their interpretation.

`ACTIVITY` Given the $i$th constraint has 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 values 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 is 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 basic and interior solution files except 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] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear programming: Theory and algorithms.* John Wiley and Sons, New York, 2 edition, 1993.

[11] C. Beightler and D. T. Phillips. *Applied geometric programming.* John Wiley and Sons, New York, 1976.

[12] A. Ben-Tal and A Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications.* MPS/SIAM Series on Optimization. SIAM, 2001.

[13] 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.

[14] V. Chvátal. *Linear programming.* W.H. Freeman and Company, 1983.

[15] N. Gould and P. L. Toint. Preprocessing for quadratic programming. *Math. Programming*, 100(1):95–132, 2004.

[16] J. L. Kenningon and K. R. Lewis. Generalized networks: The theory of preprocessing and an emperical analysis. *INFORMS Journal on Computing*, 16(2):162–173, 2004.

[17] M. S. Lobo, L. Vanderberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra Appl.*, 284:193–228, November 1998.

[18] 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.

[19] J. L. Nazareth. *Computer Solution of Linear Programs.* Oxford University Press, New York, 1987.

[20] G. Pataki. Teaching integer programming formulations using the traveling salesman problem. *SIAM Rev.*, 45(1):116–123, 2003.

[21] 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.

[22] Bernd Scherer. *Portfolio construction and risk budgeting.* Risk Books, 2 edition, 2004.

[23] R. J. Vanderbei. *Linear Programming. Foundations and Extensions.* Kluwer Academic Publishers, Boston/London/Dordrect, 1997.

[24] S. W. Wallace. Decision making under uncertainty: Is sensitivity of any use. *Oper. Res.*, 48(1):20–25, January 2000.

[25] H. P. Williams. *Model building in mathematical programming.* John Wiley and Sons, 3 edition, 1993.

[26] L. A. Wolsey. *Integer programming.* John Wiley and Sons, 1998.

# Index