The MOSEK Python optimizer API manual
Version 7.1 (Revision 32)

support@mosek.com

www.mosek.com

- Published by MOSEK ApS, Denmark.

Contents

Contact information

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

License agreement

Before using the MOSEK software, please read the license agreement available in the distribution at `mosek\7\license.pdf`

Chapter 1

Changes and new features in MOSEK

The section presents improvements and new features added to MOSEK in version 7.

1.1 Platform support

In Table 1.1 the supported platform and compiler used to build MOSEK shown. Although RedHat is explicitly mentioned as the supported Linux distribution then MOSEK will work on most other variants of Linux. However, the license manager tools requires Linux Standard Base 3 or newer is installed.

1.2 General changes

- The interior-point optimizer has been extended to semi-definite optimization problems. Hence, MOSEK can optimize over the positive semi-definite cone.
- The network detection has been completely redesigned. MOSEK no longer try detect partial networks. The problem must be a pure primal network for the network optimizer to be used.
- The parameter iparam.objective_sense has been removed.
- The parameter iparam.intpnt_num_threads has been removed. Use the parameter iparam.num_threads instead.
- MOSEK now automatically exploit multiple CPUs i.e. the parameter iparam.num_threads is set to 0 be default. Note the amount memory that MOSEK uses grows with the number of threads employed.

Platform	OS version	C compiler
linux32x86	Redhat 5 or newer (LSB 3+)	Intel C 13.0 (gcc 4.3, glibc 2.3.4)
linux64x86	RedHat 5 or newer (LSB 3+)	Intel C 13.0 (gcc 4.3, glibc 2.3.4)
osx64x86	OSX 10.7 Lion or newer	Intel C 13.0 (llvm-gcc-4.2)
win32x86	Windows Vista, Server 2003 or newer	Intel C 13.0 (VS 2008)
win64x86	Windows Vista, Server 2003 or newer	Intel C 13.0 (VS 2008)

Interface	Supported versions
Java	Sun Java 1.6+
Microsoft.NET	2.1+
Python 2	2.6+
Python 3	3.1+

Table 1.1: Supported platforms

- The MBT file format has been replaced by a new task format. The new format supports semi-definite optimization.

- the HTML version of the documentation is no longer included in the downloads to save space. It is still available online.

- MOSEK is more restrictive about the allowed names on variables etc. This is in particular the case when writing LP files.

- MOSEK no longer tries to detect the cache sizes and is in general less sensitive to the hardware.

- The parameter is set iparam.auto_update_sol_info is default off. In previous version it was by default on.

- The function relaxprimal has been deprecated and replaced by the function primalrepair.

1.3 Optimizers

1.3.1 Interior point optimizer

- The factorization routines employd by the interior-point optimizer for linear and conic optimization problems has been completely rewritten. In particular the dense column detection and handling is improved. The factorization routine will also exploit vendor tuned BLAS routines.

1.3.2 The simplex optimizers

- No major changes.

1.3.3 Mixed-integer optimizer

- A new mixed-integer for linear and conic problems has been introduced. It is from run-to-run determinitic and is parallelized. It is particular suitable for conic problems.

1.4 API changes

- Added support for semidefinite optimization.
- Some clean up has been performed implying some functions have been renamed.

1.5 Optimization toolbox for MATLAB

- A MOSEK equivalent of `bintprog` has been introduced.
- The functionality of the MOSEK version of `linprog` has been improved. It is now possible to employ the simplex optimizer in `linprog`.
- `mosekopt` now accepts a dense A matrix.
- An new method for specification of cones that is more efficient when the problem has many cones has introduced. The old method is still allowed but is deprecated.
- Support for semidefinite optimization problems has been added to the toolbox.

1.6 License system

- Flexlm has been upgraded to version 11.11.

1.7 Other changes

- The documentation has been improved.

1.8 Interfaces

- Semi-definite optimization capabilities have been add to the optimizer APIs.
- A major clean up have occured in the optimizer APIs. This should have little effect for most users.
- A new object orientated interface called Fusion has been added. Fusion is available Java, MATLAB, .NET and Python.
- The AMPL command line tool has been updated to the latest version.

1.9 Platform changes

- 32 bit MAC OSX on Intel x86 (osx32x86) is no longer supported.
- 32 and 64 bit Solaris on Intel x86 (solaris32x86,solaris64x86) is no longer supported.

1.10 Summary of API changes

1.10.1 Parameters

- `dparam.callback_freq` removed.
- `dparam.mio_tol_max_cut_frac_rhs` added.
- `dparam.mio_tol_min_cut_frac_rhs` added.
- `dparam.mio_tol_rel_dual_bound_improvement` added.
- `dparam.presolve_tol_abs_lindep` added.
- `dparam.presolve_tol_lin_dep` removed.
- `dparam.presolve_tol_rel_lindep` added.
- `iparam.bi_clean_optimizer` Valid parameter values changed.
- `iparam.cache_size_l1` removed.
- `iparam.cache_size_l2` removed.
- `iparam.check_task_data` removed.
- `iparam.cpu_type` removed.
- `iparam.data_check` removed.
- `iparam.intpnt_basis` Valid parameter values changed.
- `iparam.intpnt_num_threads` removed.
- `iparam.intpnt_order_method` Valid parameter values changed.
- `iparam.license_allow_overuse` removed.
- `iparam.license_cache_time` removed.
- `iparam.license_check_time` removed.
- `iparam.log_expand` added.
- `iparam.log_feasrepair` removed.

- iparam.log_feas_repair added.
- iparam.lp_write_ignore_incompatible_items removed.
- iparam.mio_cut_cg added.
- iparam.mio_cut_cmir added.
- iparam.mio_node_optimizer Valid parameter values changed.
- iparam.mio_probing_level added.
- iparam.mio_rins_max_nodes added.
- iparam.mio_root_optimizer Valid parameter values changed.
- iparam.mio_use_multithreaded_optimizer added.
- iparam.num_threads added.
- iparam.objective_sense removed.
- iparam.optimizer Valid parameter values changed.
- iparam.presolve_lindep_abs_work_trh added.
- iparam.presolve_lindep_rel_work_trh added.
- iparam.presolve_lindep_work_lim removed.
- iparam.presolve_max_num_reductions added.
- iparam.read_add_anz removed.
- iparam.read_add_con removed.
- iparam.read_add_cone removed.
- iparam.read_add_qnz removed.
- iparam.read_add_var removed.
- iparam.read_mps_quoted_names removed.
- iparam.read_q_mode removed.
- iparam.sim_network_detect removed.
- iparam.sim_network_detect_hotstart removed.
- iparam.sim_network_detect_method removed.
- iparam.sol_quoted_names removed.
- iparam.write_mps_obj_sense removed.
- iparam.write_mps_quoted_names removed.
- iparam.write_mps_strict removed.
- sparam.mio_debug_string added.

1.10.2 Functions

- `Env.axpy` added.
- `Env.dot` added.
- `Env.gemm` added.
- `Env.gemv` added.
- `Env.initenv` removed.
- `Env.licensecleanup` added.
- `Env.potrf` added.
- `Env.putcpudefaults` removed.
- `Env.putlicensedebug` added.
- `Env.putlicensedefaults` removed.
- `Env.putlicensepath` added.
- `Env.putlicensewait` added.
- `Env.syeig` added.
- `Env.syevd` added.
- `Env.syrk` added.
- `Task.append` removed.
- `Task.appendbarvars` added.
- `Task.appendcons` changed.
- `Task.appendsparsesymmat` added.
- `Task.appendvars` changed.
- `Task.checkdata` removed.
- `Task.core_append` removed.
- `Task.core_appendcones` removed.
- `Task.core_removecones` removed.
- `Task.getaslicetrip` removed.
- `Task.getavec` removed.
- `Task.getavecnumnz` removed.

- Task.getbarsj added.
- Task.getbarxj added.
- Task.getconname64 removed.
- Task.getdviolbarvar added.
- Task.getdviolcon added.
- Task.getdviolcones added.
- Task.getdviolvar added.
- Task.getintpntnumthreads removed.
- Task.getmemusagetask64 removed.
- Task.getname64 removed.
- Task.getnameapi64 removed.
- Task.getnameindex removed.
- Task.getnamelen64 removed.
- Task.getnumqobjnz removed.
- Task.getobjname64 removed.
- Task.getprosta added.
- Task.getpviolbarvar added.
- Task.getpviolcon added.
- Task.getpviolcones added.
- Task.getpviolvar added.
- Task.getqconk removed.
- Task.getskcslice added.
- Task.getskxslice added.
- Task.getslcslice added.
- Task.getslxslice added.
- Task.getsnxslice added.
- Task.getsolsta added.
- Task.getsolutioninfo added.

- `Task.getsolutionstatus` removed.
- `Task.getsolutionstatuskeyslice` removed.
- `Task.getsucslice` added.
- `Task.getsuxslice` added.
- `Task.gettaskname64` removed.
- `Task.getvarname64` removed.
- `Task.getxcslice` added.
- `Task.getxxslice` added.
- `Task.getyslice` added.
- `Task.makesolutionstatusunknown` removed.
- `Task.netextraction` removed.
- `Task.netoptimize` removed.
- `Task.primalrepair` added.
- `Task.putacol` added.
- `Task.putaijlist` removed.
- `Task.putarow` added.
- `Task.putavec` removed.
- `Task.putaveclist64` removed.
- `Task.putbaraij` added.
- `Task.putbarcj` added.
- `Task.putbarsj` added.
- `Task.putbarvarname` added.
- `Task.putbarxj` added.
- `Task.putconbound` added.
- `Task.putconboundlist` added.
- `Task.putconename` added.
- `Task.putconname` added.
- `Task.putmaxnumanz64` removed.

- `Task.putmaxnumqnz64` removed.
- `Task.putname` removed.
- `Task.putskcslice` added.
- `Task.putskxslice` added.
- `Task.putslcslice` added.
- `Task.putslxslice` added.
- `Task.putsnxslice` added.
- `Task.putsucslice` added.
- `Task.putsuxslice` added.
- `Task.putvarbound` added.
- `Task.putvarboundlist` added.
- `Task.putvarname` added.
- `Task.putxcslice` added.
- `Task.putxxslice` added.
- `Task.putyslice` added.
- `Task.readdata` removed.
- `Task.readtask` added.
- `Task.remove` removed.
- `Task.removecone` removed.
- `Task.removecones` added.
- `Task.removecons` added.
- `Task.removevars` added.
- `Task.toconic` added.
- `Task.undefsolution` removed.
- `Task.updatesolutioninfo` added.
- `Task.writetask` added.

Chapter 2

About this manual

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

The MOSEK Python Application Programming Interface makes it possible to access the MOSEK optimizer from any Python application. The whole functionality of the native C API is available through a thin class-based interface using native Python types and exceptions. All methods in the interface are thin wrappers around functions in the native C API, keeping the overhead induced by the API to a minimum.

The API can be used in Python scripts as well as from the interactive Python command-line. The Python interface is particularly well-suited for fast prototyping of models and for debugging and displaying portions of a problem loaded from a file.

The Python interface consists of a `mosek` module that defines objects, functions and constants.

New users of the MOSEK Python 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 9 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 10 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 10 is useful when trying to find a good formulation of a specific model.

More advanced examples of modeling and model debugging are located in

- Chapter 14 which deals with analysis of infeasible problems,

- Chapter 15 about the sensitivity analysis interface, and

Finally, the Python API reference material is located in

- Chapter A which lists all types and functions,
- Chapter B which lists all available parameters,
- Chapter C which lists all response codes, and
- Chapter D 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/7/docs/
```

in the distribution.

3.2 Bug reporting

If you think MOSEK is solving your problem incorrectly, please contact MOSEK support at

support@mosek.com

providing a detailed description of the problem. MOSEK support may ask for the task file which is produced as follows

```
task.writedata("data.task.gz")
task.optimize()
```

The task data will then be written to a binary file named `data.task.gz` which is useful when reproducing a problem.

3.3 Additional reading

In this manual it is assumed that the reader is familiar with mathematics and in particular mathematical optimization. Some introduction to linear programming is found in books such as "Linear programming" by Chvátal [1] or "Computer Solution of Linear Programs" by Nazareth [2]. For more theoretical aspects see e.g. "Nonlinear programming: Theory and algorithms" by Bazaraa, Shetty, and Sherali [3]. Finally, the book ”Model building in mathematical programming” by Williams [4] provides an excellent introduction to modeling issues in optimization.

Another useful resource is ”Mathematical Programming Glossary” available at

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

Chapter 4

Testing installation and compiling examples

This chapter describes how to verify that the MOSEK Python API has been installed and works, and how to run the Python examples distributed with MOSEK.

To use the MOSEK Python API, a working MOSEK installation must be present — see the MOSEK Installation manual for instructions. Part of this installation are two versions of the Python interface; one for Python 2.5 and later, and one for the Python 3 series. Note that MOSEK can use one-dimensional arrays from the NumPy package from `scipy.org`. In case NumPy is not installed, the MOSEK/Python interface includes a minimal array implementation, `mosek.array`, that supports the basic array functionality for a limited number of native types.

A Python installer can be obtained from the official site:

http://www.python.org/

The NumPy package providing arrays and mathematical functionality can be obtained from:

http://numpy.scipy.org/

Note that the architecture of the Python binary and the MOSEK DLL must match: A 32 bit MOSEK can only be used from a 32 bit Python, and a 64 bit MOSEK can only be used from a 64 bit Python. The architecture of the Python binary can be checked using the following command:

```
python -c "print(__import__('platform').architecture())"
```

4.1 Microsoft Windows platform

MOSEK includes a binary Python that can be used interactively or for running scripts, but if you have a Python installed on your system you may prefer to use that instead. The MOSEK installer does *not* set up global paths to the included Python: To use it, either execute Python with full path, or set up the `PATH` environment variable to include the relevant `bin` directory.

The MOSEK/Python module structure is located under platform directory (`mosek\7\tools\platform`):

64 bit MOSEK, Python 2.5+	`win64x86\python\2`
32 bit MOSEK, Python 2.5+	`win32x86\python\2`
64 bit MOSEK, Python 3	`win64x86\python\3`
32 bit MOSEK, Python 3	`win32x86\python\3`

Examples using the MOSEK/Python interface are found in

```
mosek\7\tools\examples\python
```

4.1.1 Running a Python example

To run one of the distributed examples, open a DOS box and type

```
C:
cd "C:\Program Files\mosek\7\tools\examples\python"
```

then to execute example `lo1` type

```
python lo1.py
```

4.2 Implementation details

The MOSEK/Python module is implemented as a pure Python module using `ctypes` to call native DLLs. The CTypes module is included in the Python standard library from 2.5 on.

CTypes is considered an *unsafe* module, so it may be disabled under some circumstances, e.g. for web server scripts.

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, conic, semidefinite and quadratic problems as well as mixed integer problems.

Please note that the section on linear optimization also describes most of the basic functionality needed to specify optimization problems. Hence, it is recommended to read Section 5.2 before reading about other optimization problems.

5.1 The basics

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

- Create an environment object (`Env`).
- Set up some environment specific data and initialize the environment object.
- Create a task object (`Task`).
- Load a problem into the task object.
- Optimize the problem.
- Fetch the result.
- Dispose of the environment and task.

5.1.1 The environment and the task

The first MOSEK related step in any program that employs MOSEK is to create an environment object. The environment contains environment specific data such as information about the license file, streams for environment messages etc. When this is done one or more task 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.

When done, tasks and environments may be disposed explicitly by calling the `__del__` method. This is not strictly necessary, but it will free up allocated resources and checked-out licenses immediately instead of when the garbage collector runs.

In Python, the creation of an environment and a task would look something like this:

```
# Create an environment
env = mosek.Env()

# You may connect streams and other callbacks to env here.

# Create a task
task = env.Task()

# Load a problem into the task, optimize etc.
```

From Python 2.6 and later the `with` construction can be used to dispose objects automatically when they drop of out of the `with`-scope:

```
# Create an environment
with mosek.Env() as env:
    # You may connect streams and other callbacks to env here.

    # Create a task
    with env.Task() as task:
        # Load a problem into the task, optimize etc.
```

Please note that multiple tasks should, if possible, share the same environment.

5.1.2 Example: Simple working example

The following simple example shows a working Python program which

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

[simple.py]

```
#
# Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
# File:    simple.py
```

```python
#
# Purpose: Demonstrates a very simple example using MOSEK by
# reading a problem file, solving the problem and
# writing the solution to a file.
#

import mosek
import sys

def streamprinter(msg):
    sys.stdout.write (msg)
    sys.stdout.flush ()

if len(sys.argv) <= 1:
    print ("Missing argument, syntax is:")
    print ("  simple inputfile [ solutionfile ]")
else:
    # Create the mosek environment.
    env  = mosek.Env ()

    # Create a task object linked with the environment env.
    # We create it with 0 variables and 0 constraints initially,
    # since we do not know the size of the problem.
    task = env.Task (0, 0)
    task.set_Stream (mosek.streamtype.log, streamprinter)

    # We assume that a problem file was given as the first command
    # line argument (received in `argv')
    task.readdata (sys.argv[1])

    # Solve the problem
    task.optimize ()

    # Print a summary of the solution
    task.solutionsummary (mosek.streamtype.log)

    # If an output file was specified, write a solution
    if len(sys.argv) >= 3:
        # We define the output format to be OPF, and tell MOSEK to
        # leave out parameters and problem data from the output file.
        task.putintparam (mosek.iparam.write_data_format,    mosek.dataformat.op)
        task.putintparam (mosek.iparam.opf_write_solutions,  mosek.onoffkey.on)
        task.putintparam (mosek.iparam.opf_write_hints,      mosek.onoffkey.off)
        task.putintparam (mosek.iparam.opf_write_parameters, mosek.onoffkey.off)
        task.putintparam (mosek.iparam.opf_write_problem,    mosek.onoffkey.off)

        task.writedata (sys.argv[2])
```

5.1.2.1 Reading and writing problems

Use the Task.writedata function to write a problem to a file. By default, when not choosing any specific file format for the parameter iparam.write_data_format, MOSEK will determine the output file format by the extension of the file name:

[simple.py]
```
task.writedata (sys.argv[2])
```

Similarly, controlled by iparam.read_data_format, the function Task.readdata can read a problem from a file:

[simple.py]
```
task.readdata (sys.argv[1])
```

5.1.2.2 Working with the problem data

An optimization problem consists of several components; objective, objective sense, constraints, variable bounds etc. Therefore, the interface provides a number of methods to operate on the task specific data, all of which are listed under the Task class-specification.

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 iparam.optimizer parameter defines which optimizer to use. There are three kinds of parameters in MOSEK

- Integer parameters that can be set with Task.putintparam,
- Double parameters that can be set with Task.putdouparam, and
- string parameters that can be set with Task.putstrparam,

The values for integer parameters are either simple integer values or enum values.

A complete list of all parameters is found in Chapter B.

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

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

c^f

which is a constant,

A

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

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

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 Python arrays which are indexed from 0.

5.2.1 Example: Linear optimization

The following is an example of a linear optimization problem:

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

having the bounds

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

5.2.1.1 Solving the problem

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

- Create an environment.
- Create an optimization task.
- Load a problem into the task object.
- Optimization.
- Extracting the solution.

Below we explain each of these steps. For the complete source code see section 5.2.1.2.

Create an environment.

Before setting up the optimization problem, a MOSEK environment must be created. All tasks in the program should share the same environment.

[lo1.py]

```
# Make mosek environment
with mosek.Env() as env:
```

Create an optimization task.

Next, an empty task object is created:

[lo1.py]

```
# Create a task object
with env.Task(0,0) as task:
  # Attach a log stream printer to the task
  task.set_Stream (mosek.streamtype.log, streamprinter)
```

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

Load a problem into the task object.

Before any problem data can be set, variables and constraints must be added to the problem via calls to the functions Task.appendcons and Task.appendvars.

[lo1.py]
```
# Append 'numcon' empty constraints.
# The constraints will initially have no bounds.
task.appendcons(numcon)

# Append 'numvar' variables.
# The variables will initially be fixed at zero (x=0).
task.appendvars(numvar)
```

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

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

[lo1.py]
```
task.putcj(j,c[j])
```

The bounds on variables are stored in the arrays

[lo1.py]
```
# Bound keys for variables
bkx = [mosek.boundkey.lo,
       mosek.boundkey.ra,
       mosek.boundkey.lo,
       mosek.boundkey.lo]

# Bound values for variables
blx = [ 0.0,  0.0,  0.0,  0.0]
bux = [+inf, 10.0, +inf, +inf]
```

and are set with calls to Task.putvarbound.

[lo1.py]
```
# Set the bounds on variable j
# blx[j] <= x_j <= bux[j]
task.putvarbound(j,bkx[j],blx[j],bux[j])
```

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

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

and

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

Bound key	Type of bound	Lower bound	Upper bound
boundkey.fx	$\cdots = l_j$	Finite	Identical to the lower bound
boundkey.fr	Free	Minus infinity	Plus infinity
boundkey.lo	$l_j \leq \cdots$	Finite	Plus infinity
boundkey.ra	$l_j \leq \cdots \leq u_j$	Finite	Finite
boundkey.up	$\cdots \leq u_j$	Minus infinity	Finite

Table 5.1: Interpretation of the bound keys.

Recall that in our example the A matrix is given by

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

This matrix is stored in sparse format in the arrays:

[lo1.py]

```
asub = [ array([0, 1]),
         array([0, 1, 2]),
         array([0, 1]),
         array([1, 2])]
aval = [ array([3.0, 2.0]),
         array([1.0, 1.0, 2.0]),
         array([2.0, 3.0]),
         array([1.0, 3.0]) ]
```

The array `aval[j]` contains the non-zero values of column j and `asub[j]` contains the row index of these non-zeros.

Using the function Task.putacol we set column j of A

[lo1.py]

```
task.putacol(j,                  # Variable (column) index.
             asub[j],            # Row index of non-zeros in column j.
             aval[j])            # Non-zero Values of column j.
```

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

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

[lo1.py]

```
 # Set the bounds on constraints.
# blc[i] <= constraint_i <= buc[i]
 for i in range(numcon):
     task.putconbound(i,bkc[i],blc[i],buc[i])
 task.putconboundslice(0,numcon, bkc,blc,buc);
```

Optimization:

After the problem is set-up the task can be optimized by calling the function Task.optimize.

[lo1.py]
```
task.optimize()
```

Extracting the solution.

After optimizing the status of the solution is examined with a call to Task.getsolsta. If the solution status is reported as solsta.optimal or solsta.near_optimal the solution is extracted in the lines below:

[lo1.py]
```
xx = zeros(numvar, float)
task.getxx(mosek.soltype.bas, # Request the basic solution.
           xx)
```

The Task.getxx function obtains the solution. MOSEK may compute several solutions depending on the optimizer employed. In this example the *basic solution* is requested by setting the first argument to soltype.bas.

5.2.1.2 Source code for lo1

[lo1.py]
```
#
#  Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#  File:    lo1.py
#
#  Purpose: Demonstrates how to solve small linear
#           optimization problem using the MOSEK Python API.
##
from __future__ import with_statement

import sys
import mosek

# If numpy is installed, use that, otherwise use the
# Mosek's array module.
try:
    from numpy import array,zeros,ones
except ImportError:
    from mosek.array import array, zeros, ones

# Since the value of infinity is ignored, we define it solely
# for symbolic purposes
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

```

```
def main ():
  # Make mosek environment
  with mosek.Env() as env:
    # Create a task object
    with env.Task(0,0) as task:
      # Attach a log stream printer to the task
      task.set_Stream (mosek.streamtype.log, streamprinter)

      # Bound keys for constraints
      bkc = [mosek.boundkey.fx,
             mosek.boundkey.lo,
             mosek.boundkey.up]

      # Bound values for constraints
      blc = [30.0, 15.0, -inf]
      buc = [30.0, +inf, 25.0]

      # Bound keys for variables
      bkx = [mosek.boundkey.lo,
             mosek.boundkey.ra,
             mosek.boundkey.lo,
             mosek.boundkey.lo]

      # Bound values for variables
      blx = [ 0.0,  0.0,  0.0,  0.0]
      bux = [+inf, 10.0, +inf, +inf]

      # Objective coefficients
      c = [ 3.0, 1.0, 5.0, 1.0 ]

      # Below is the sparse representation of the A
      # matrix stored by column.
      asub = [ array([0, 1]),
               array([0, 1, 2]),
               array([0, 1]),
               array([1, 2])]
      aval = [ array([3.0, 2.0]),
               array([1.0, 1.0, 2.0]),
               array([2.0, 3.0]),
               array([1.0, 3.0]) ]

      numvar = len(bkx)
      numcon = len(bkc)

      # Append 'numcon' empty constraints.
      # The constraints will initially have no bounds.
      task.appendcons(numcon)

      # Append 'numvar' variables.
      # The variables will initially be fixed at zero (x=0).
      task.appendvars(numvar)

      for j in range(numvar):
        # Set the linear term c_j in the objective.
        task.putcj(j,c[j])

        # Set the bounds on variable j
        # blx[j] <= x_j <= bux[j]
```

```
    task.putvarbound(j,bkx[j],blx[j],bux[j])

    # Input column j of A
    task.putacol(j,                    # Variable (column) index.
          asub[j],             # Row index of non-zeros in column j.
          aval[j])             # Non-zero Values of column j.

   # Set the bounds on constraints.
  # blc[i] <= constraint_i <= buc[i]
   for i in range(numcon):
     task.putconbound(i,bkc[i],blc[i],buc[i])
   task.putconboundslice(0,numcon, bkc,blc,buc);

   # Input the objective sense (minimize/maximize)
   task.putobjsense(mosek.objsense.maximize)

   # Solve the problem
   task.optimize()

   # Print a summary containing information
   # about the solution for debugging purposes
   task.solutionsummary(mosek.streamtype.msg)

   # Get status information about the solution
   solsta = task.getsolsta(mosek.soltype.bas)

   if (solsta == mosek.solsta.optimal or
     solsta == mosek.solsta.near_optimal):
    xx = zeros(numvar, float)
    task.getxx(mosek.soltype.bas, # Request the basic solution.
         xx)
    print ("Optimal solution: ")
    for i in range(numvar):
      print ("x["+str(i)+"]="+str(xx[i]))
   elif (solsta == mosek.solsta.dual_infeas_cer or
      solsta == mosek.solsta.prim_infeas_cer or
      solsta == mosek.solsta.near_dual_infeas_cer or
      solsta == mosek.solsta.near_prim_infeas_cer):
    print("Primal or dual infeasibility certificate found.\n")
   elif solsta == mosek.solsta.unknown:
    print("Unknown solution status")
   else:
    print("Other solution status")

# call the main function
try:
  main ()
except mosek.Exception as e:
  print ("ERROR: %s" % str(e.errno))
  if e.msg is not None:
    print ("\t%s" % e.msg)
    sys.exit(1)
except:
  import traceback
  traceback.print_exc()
  sys.exit(1)
sys.exit(0)
```

5.2.2 Row-wise input

In the previous example the A matrix is set one column at a time. Alternatively the same matrix can be set one row at a time or the two methods can be mixed as in the example in section 5.10. The following example show how to set the A matrix by rows.

[lo2.py]

```
#
#  Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#  File:    lo2.py
#
#  Purpose: Demonstrates how to solve small linear
#           optimization problem using the MOSEK Python API.
##

import sys

import mosek
# If numpy is installed, use that, otherwise use the
# Mosek's array module.
try:
    from numpy import array,zeros,ones
except ImportError:
    from mosek.array import array, zeros, ones

# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main ():
  # Make a MOSEK environment
  env = mosek.Env ()
  # Attach a printer to the environment
  env.set_Stream (mosek.streamtype.log, streamprinter)

  # Create a task
  task = env.Task(0,0)
  # Attach a printer to the task
  task.set_Stream (mosek.streamtype.log, streamprinter)

  # Bound keys for constraints
  bkc = [mosek.boundkey.fx,
         mosek.boundkey.lo,
         mosek.boundkey.up]
  # Bound values for constraints
  blc = [30.0, 15.0, -inf]
  buc = [30.0, +inf, 25.0]
  # Bound keys for variables
  bkx = [mosek.boundkey.lo,
```

```
            mosek.boundkey.ra,
            mosek.boundkey.lo,
            mosek.boundkey.lo]
    # Bound values for variables
    blx = [ 0.0,  0.0,  0.0,  0.0]
    bux = [+inf, 10.0, +inf, +inf]
    # Objective coefficients

    c = [ 3.0, 1.0, 5.0, 1.0 ]

    # We input the A matrix column-wise
    # asub contains row indexes
    asub = [ array([0, 1, 2]),
             array([0, 1, 2, 3]),
             array([0, 3])]
      # acof contains coefficients
    aval = [ array([3.0, 1.0, 2.0]),
             array([2.0, 1.0, 3.0, 1.0]),
             array([2.0, 3.0])]
    numvar = len(bkx)
    numcon = len(bkc)
    # Append 'numcon' empty constraints.
    # The constraints will initially have no bounds.
    task.appendcons(numcon)

    #Append 'numvar' variables.
    # The variables will initially be fixed at zero (x=0).
    task.appendvars(numvar)

    for j in range(numvar):
      # Set the linear term c_j in the objective.
      task.putcj(j,c[j])
      # Set the bounds on variable j
      # blx[j] <= x_j <= bux[j]
      task.putbound(mosek.accmode.var,j,bkx[j],blx[j],bux[j])

    for i in range(numcon):
      task.putbound(mosek.accmode.con,i,bkc[i],blc[i],buc[i])
      # Input row i of A
      task.putarow(i,                     # Row index.
                   asub[i],               # Column indexes of non-zeros in row i.
                   aval[i]);              # Non-zero Values of row i.


    # Input the objective sense (minimize/maximize)
    task.putobjsense(mosek.objsense.maximize)

    # Optimize the task
    task.optimize()

    # Print a summary containing information
    # about the solution for debugging purposes
    task.solutionsummary(mosek.streamtype.msg)

    prosta = task.getprosta(mosek.soltype.bas)
    solsta = task.getsolsta(mosek.soltype.bas)

    # Output a solution
```

```
    xx = zeros(numvar, float)
    task.getxx(mosek.soltype.bas,
                          xx)

    if solsta == mosek.solsta.optimal or solsta == mosek.solsta.near_optimal:
        print("Optimal solution: %s" % xx)
    elif solsta == mosek.solsta.dual_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif solsta == mosek.solsta.prim_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif solsta == mosek.solsta.near_dual_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif  solsta == mosek.solsta.near_prim_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif mosek.solsta.unknown:
      print("Unknown solution status")
    else:
      print("Other solution status")

  # call the main function
  try:
      main ()
  except mosek.Exception as e:
      print ("ERROR: %s" % str(e.errno))
      if e.msg is not None:
          print ("\t%s" % e.msg)
          sys.exit(1)
  except:
      import traceback
      traceback.print_exc()
      sys.exit(1)
  sys.exit(0)
```

5.3 Conic quadratic optimization

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

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

where x^t is a subset of the problem variables and $\mathcal{C}_t$ is a convex cone. Actually, since the set $\mathbb{R}^n$ of real numbers is also a convex cone, all variables can in fact be partitioned into subsets belonging to separate convex cones, simply stated $x \in \mathcal{C}$.

MOSEK can solve conic quadratic optimization problems of the form

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x \in \mathcal{C}, & & \end{array} \tag{5.4}$$

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

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

For convenience, the user only specify subsets of variables x^t belonging to cones $\mathcal{C}_t$ different from the set $\mathbb{R}^{n_t}$ of real numbers. These cones can be a:

- Quadratic cone:

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

- Rotated quadratic cone:

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

From these definition it follows that

$$(x_4, x_0, x_2) \in \mathcal{Q}_3,$$

is equivalent to

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

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

5.3.1 Example: Conic quadratic optimization

The problem

$$\begin{array}{lccc} \text{minimize} & x_3 + x_4 + x_5 & & \\ \text{subject to} & x_0 + x_1 + 2x_2 & = & 1, \\ & x_0, x_1, x_2 & \geq & 0, \\ & x_3 \geq \sqrt{x_0^2 + x_1^2}, & & \\ & 2x_4 x_5 \geq x_2^2. & & \end{array} \tag{5.5}$$

is an example of a conic quadratic optimization problem. The problem includes a set of linear constraints, a quadratic cone and a rotated quadratic cone.

5.3.1.1 Source code

[cqo1.py]

```python
#
#  Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#  File:    cqo1.py
#
#  Purpose: Demonstrates how to solve small linear
#           optimization problem using the MOSEK Python API.
##

import sys

import mosek
# If numpy is installed, use that, otherwise use the
# Mosek's array module.
try:
    from numpy import array,zeros,ones
except ImportError:
    from mosek.array import array, zeros, ones

# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main ():
  # Make a MOSEK environment
  env = mosek.Env ()
  # Attach a printer to the environment
  env.set_Stream (mosek.streamtype.log, streamprinter)

  # Create a task
  task = env.Task(0,0)
  # Attach a printer to the task
  task.set_Stream (mosek.streamtype.log, streamprinter)

  bkc = [ mosek.boundkey.fx ]
  blc = [ 1.0 ]
  buc = [ 1.0 ]

  c   = [               0.0,              0.0,              0.0,
                        1.0,              1.0,              1.0 ]
  bkx = [ mosek.boundkey.lo,mosek.boundkey.lo,mosek.boundkey.lo,
          mosek.boundkey.fr,mosek.boundkey.fr,mosek.boundkey.fr ]
  blx = [               0.0,              0.0,              0.0,
                       -inf,             -inf,             -inf ]
  bux = [               inf,              inf,              inf,
                        inf,              inf,              inf ]

  asub = [ array([0]),   array([0]),   array([0])   ]
```

```
    aval = [ array([1.0]), array([1.0]), array([2.0]) ]


    numvar = len(bkx)
    numcon = len(bkc)
    NUMANZ = 4
    # Append 'numcon' empty constraints.
    # The constraints will initially have no bounds.
    task.appendcons(numcon)

    #Append 'numvar' variables.
    # The variables will initially be fixed at zero (x=0).
    task.appendvars(numvar)

    for j in range(numvar):
      # Set the linear term c_j in the objective.
      task.putcj(j,c[j])
      # Set the bounds on variable j
      # blx[j] <= x_j <= bux[j]
      task.putbound(mosek.accmode.var,j,bkx[j],blx[j],bux[j])

    for j in range(len(aval)):
      # Input column j of A
      task.putacol(j,                  # Variable (column) index.
                   asub[j],            # Row index of non-zeros in column j.
                   aval[j])            # Non-zero Values of column j.
    for i in range(numcon):
      task.putbound(mosek.accmode.con,i,bkc[i],blc[i],buc[i])

    # Input the cones
    task.appendcone(mosek.conetype.quad,
                    0.0,
                    [ 3, 0, 1 ])
    task.appendcone(mosek.conetype.rquad,
                    0.0,
                    [ 4, 5, 2 ])

    # Input the objective sense (minimize/maximize)
    task.putobjsense(mosek.objsense.minimize)

    # Optimize the task
    task.optimize()
    # Print a summary containing information
    # about the solution for debugging purposes
    task.solutionsummary(mosek.streamtype.msg)
    prosta = task.getprosta(mosek.soltype.itr)
    solsta = task.getsolsta(mosek.soltype.itr)

    # Output a solution
    xx = zeros(numvar, float)
    task.getxx(mosek.soltype.itr,
                          xx)

    if solsta == mosek.solsta.optimal or solsta == mosek.solsta.near_optimal:
        print("Optimal solution: %s" % xx)
    elif solsta == mosek.solsta.dual_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif solsta == mosek.solsta.prim_infeas_cer:
```

```
        print("Primal or dual infeasibility.\n")
    elif solsta == mosek.solsta.near_dual_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif  solsta == mosek.solsta.near_prim_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif mosek.solsta.unknown:
      print("Unknown solution status")
    else:
      print("Other solution status")


# call the main function
try:
    main ()
except mosek.Exception as e:
    print ("ERROR: %s" % str(e.code))
    if msg is not None:
        print ("\t%s" % e.msg)
        sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
sys.exit(0)
```

5.3.1.2 Source code comments

The only new function introduced in the example is `Task.appendcone`, which is called here:

[cqo1.py]

```
task.appendcone(mosek.conetype.quad,
                0.0,
                [ 3, 0, 1 ])
```

The first argument selects the type of quadratic cone. Either `conetype.quad` for a *quadratic cone* or `conetype.rquad` for a *rotated quadratic cone*. The cone parameter `0.0` is currently not used by MOSEK — simply passing 0.0 will work.

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

5.4 Semidefinite optimization

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

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

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

MOSEK can solve semidefinite optimization problems of the form

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

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

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

Since all $\overline{C}_j, \overline{A}_{ij}$ are assummed to be symmetric, only their lower triangular parts are specified.

Some attention must be paid when formulating linear constraints involving semidefinite matrices. A common mistake is not to consider that, being all $\overline{C}_j, \overline{A}_{ij}$ symmetric, their off-diagonal entries are counted twice. Indeed in that case we can write

$$
\left\langle \overline{A}, \overline{X} \right\rangle := \sum_{i=0}^{p-1} \overline{A}_{ii} \overline{X}_{ii} + 2 \sum_{i=0}^{p-1} \sum_{j=i+1}^{p-1} \overline{A}_{ij} \overline{X}_{ij},
$$

and hence the contribution of each off-diagonal element to the linear constraint is double.

For instance, let's consider $\overline{X} \in \mathcal{S}_3^+$ and a constraint of the form $\overline{X}_{01} = 1$. Introducing a symmetric matrix

$$
A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},
$$

we write the constraint as

$$
\left\langle A, \overline{X} \right\rangle = \overline{X}_{01} + \overline{X}_{10} = 2\overline{X}_{01} = 2.
$$

Otherwise, we could use

$$
A = \begin{bmatrix} 0 & 0.5 & 0 \\ 0.5 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},
$$

and rewrite the constraint as

$$
\left\langle A, \overline{X} \right\rangle = 0.5(\overline{X}_{10} + \overline{X}_{01}) = \overline{X}_{01} = 1.
$$

5.4.1 Example: Semidefinite optimization

The problem

$$
\begin{array}{lrcl}
\text{minimize} & \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, \overline{X} \right\rangle + x_0 & & \\
\text{subject to} & \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \overline{X} \right\rangle + x_0 & = & 1, \\
 & \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \overline{X} \right\rangle + x_1 + x_2 & = & 1/2, \\
 & x_0 \geq \sqrt{x_1^2 + x_2^2}, & & \\
 & \overline{X} \succeq 0, & &
\end{array}
\tag{5.6}
$$

is a mixed semidefinite and conic quadratic programming problem with a 3-dimensional semidefinite variable

$$
\overline{X} = \begin{bmatrix} \overline{x}_{00} & \overline{x}_{10} & \overline{x}_{20} \\ \overline{x}_{10} & \overline{x}_{11} & \overline{x}_{21} \\ \overline{x}_{20} & \overline{x}_{21} & \overline{x}_{22} \end{bmatrix} \in \mathcal{S}_3^+,
$$

and a conic quadratic variable $(x_0, x_1, x_2) \in \mathcal{Q}_3$. The objective is to minimize

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

subject to the two linear constraints

$$
\overline{x}_{00} + \overline{x}_{11} + \overline{x}_{22} + x_0 = 1,
$$

and

$$
\overline{x}_{00} + \overline{x}_{11} + \overline{x}_{22} + 2(\overline{x}_{10} + \overline{x}_{20} + \overline{x}_{21}) + x_1 + x_2 = 1/2.
$$

5.4.1.1 Source code

[sdo1.py]

```python
##
#   Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#   File:      sdo1.py
#
#   Purpose:   Demonstrates how to solve a small mixed semidefinite and conic quadratic
#              optimization problem using the MOSEK Python API.
##

import sys
```

```
import mosek

# If numpy is installed, use that, otherwise use the
# Mosek's array module.
try:
    from numpy import array,zeros,ones
except ImportError:
    from mosek.array import array, zeros, ones

# Since the value of infinity is ignored, we define it solely
# for symbolic purposes
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main ():
  # Make mosek environment
  env = mosek.Env()

  # Create a task object and attach log stream printer
  task = env.Task(0,0)
  task.set_Stream(mosek.streamtype.log, streamprinter)

  # Bound keys for constraints
  bkc = [mosek.boundkey.fx,
         mosek.boundkey.fx]

  # Bound values for constraints
  blc = [1.0, 0.5]
  buc = [1.0, 0.5]

  # Below is the sparse representation of the A
  # matrix stored by row.
  asub = [ array([0]),
           array([1, 2])]
  aval = [ array([1.0]),
           array([1.0, 1.0]) ]

  conesub = [0, 1, 2]

  barci = [0, 1, 1, 2, 2]
  barcj = [0, 0, 1, 1, 2]
  barcval = [2.0, 1.0, 2.0, 1.0, 2.0]

  barai   = [array([0, 1, 2]),
             array([0, 1, 2, 1, 2, 2])]
  baraj   = [array([0, 1, 2]),
             array([0, 0, 0, 1, 1, 2])]
  baraval = [array([1.0, 1.0, 1.0]),
             array([1.0, 1.0, 1.0, 1.0, 1.0, 1.0])]

  numvar = 3
  numcon = len(bkc)
  BARVARDIM = [3]

```

```
    # Append 'numvar' variables.
    # The variables will initially be fixed at zero (x=0).
    task.appendvars(numvar)

    # Append 'numcon' empty constraints.
    # The constraints will initially have no bounds.
    task.appendcons(numcon)

    # Append matrix variables of sizes in 'BARVARDIM'.
    # The variables will initially be fixed at zero.
    task.appendbarvars(BARVARDIM)

    # Set the linear term c_0 in the objective.
    task.putcj(0, 1.0)

    for j in range(numvar):
      # Set the bounds on variable j
      # blx[j] <= x_j <= bux[j]
      task.putvarbound(j, mosek.boundkey.fr, -inf, +inf)

    for i in range(numcon):
      # Set the bounds on constraints.
      # blc[i] <= constraint_i <= buc[i]
      task.putconbound(i, bkc[i], blc[i], buc[i])

      # Input row i of A
      task.putarow(i,                  # Constraint (row) index.
                   asub[i],            # Column index of non-zeros in constraint j.
                   aval[i])            # Non-zero values of row j.

    task.appendcone(mosek.conetype.quad,
                    0.0,
                    conesub)

    symc = \
      task.appendsparsesymmat(BARVARDIM[0],
                              barci,
                              barcj,
                              barcval)

    syma0 = \
      task.appendsparsesymmat(BARVARDIM[0],
                              barai[0],
                              baraj[0],
                              baraval[0])

    syma1 = \
      task.appendsparsesymmat(BARVARDIM[0],
                              barai[1],
                              baraj[1],
                              baraval[1])

    task.putbarcj(0, [symc], [1.0])

    task.putbaraij(0, 0, [syma0], [1.0])
    task.putbaraij(1, 0, [syma1], [1.0])

    # Input the objective sense (minimize/maximize)
```

```
  task.putobjsense(mosek.objsense.minimize)

  task.writedata("sdo1.task")

  # Solve the problem and print summary
  task.optimize()
  task.solutionsummary(mosek.streamtype.msg)

  # Get status information about the solution
  prosta = task.getprosta(mosek.soltype.itr)
  solsta = task.getsolsta(mosek.soltype.itr)

  if (solsta == mosek.solsta.optimal or
      solsta == mosek.solsta.near_optimal):
    xx = zeros(numvar, float)
    task.getxx(mosek.soltype.itr, xx)

    lenbarvar = BARVARDIM[0] * (BARVARDIM[0]+1) / 2
    barx = zeros(int(lenbarvar), float)
    task.getbarxj(mosek.soltype.itr, 0, barx)

    print("Optimal solution:\nx=%s\nbarx=%s" % (xx,barx))
  elif (solsta == mosek.solsta.dual_infeas_cer or
        solsta == mosek.solsta.prim_infeas_cer or
        solsta == mosek.solsta.near_dual_infeas_cer or
        solsta == mosek.solsta.near_prim_infeas_cer):
    print("Primal or dual infeasibility certificate found.\n")
  elif solsta == mosek.solsta.unknown:
    print("Unknown solution status")
  else:
    print("Other solution status")

# call the main function
try:
    main ()
except mosek.Exception as e:
    print ("ERROR: %s" % str(e.errno))
    if e.msg is not None:
        print ("\t%s" % e.msg)
        sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
sys.exit(0)
```

5.4.1.2 Source code comments

This example introduces several new functions. The first new function Task.appendbarvars is used to append the semidefinite variable:

[sdo1.py]

```
task.appendbarvars(BARVARDIM)
```

Symmetric matrices are created using the function Task.appendsparsesymmat:

[sdo1.py]
```
symc = \
  task.appendsparsesymmat(BARVARDIM[0],
                          barci,
                          barcj,
                          barcval)

syma0 = \
  task.appendsparsesymmat(BARVARDIM[0],
                          barai[0],
                          baraj[0],
                          baraval[0])

syma1 = \
  task.appendsparsesymmat(BARVARDIM[0],
                          barai[1],
                          baraj[1],
                          baraval[1])
```

The second argument specifies the dimension of the symmetric variable and the third argument gives the number of non-zeros in the lower triangular part of the matrix. The next three arguments specify the non-zeros in the lower-triangle in triplet format, and the last argument will be updated with a unique index of the created symmetric matrix.

After one or more symmetric matrices have been created using Task.appendsparsesymmat, we can combine them to setup a objective matrix coefficient $\bar{c}_j$ using Task.putbarcj, which forms a linear combination of one more symmetric matrices:

[sdo1.py]
```
task.putbarcj(0, [symc], [1.0])
```

The second argument specify the semidefinite variable index j; in this example there is only a single variable, so the index is 0. The next three arguments give the number of matrices used in the linear combination, their indices (as returned by Task.appendsparsesymmat), and the weights for the individual matrices, respectively. In this example, we form the objective matrix coefficient directly from a single symmetric matrix.

Similary, a constraint matrix coefficient $\overline{A}_{ij}$ is setup by the function Task.putbaraij:

[sdo1.py]
```
task.putbaraij(0, 0, [syma0], [1.0])
task.putbaraij(1, 0, [syma1], [1.0])
```

where the second argument specifies the constraint number (the corresponding row of $\overline{A}$), and the third argument specifies the semidefinite variable index (the corresponding column of $\overline{A}$). The next three arguments specify a weighted combination of symmetric matrices used to form the constraint matrix coefficient.

After the problem is solved, we read the solution using Task.getbarxj:

[sdo1.py]
```
task.getbarxj(mosek.soltype.itr, 0, barx)
```

The function returns the half-vectorization of $\overline{x}_j$ (the lower triangular part stacked as a column vector), where the semidefinite variable index j is given in the second argument, and the third argument is a pointer to an array for storing the numerical values.

5.5 Quadratic optimization

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

$$\begin{array}{lccccl} \text{minimize} & & & \frac{1}{2}x^T Q^o x + c^T x + c^f & & \\ \text{subject to} & l_k^c & \leq & \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j & \leq & u_k^c, \quad k = 0, \ldots, m-1, \\ & l_j^x & \leq & x_j & \leq & u_j^x, \quad j = 0, \ldots, n-1. \end{array} \tag{5.7}$$

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

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

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

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

$$l_k^c \leq \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j \tag{5.8}$$

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

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

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

5.5.1 Example: Quadratic objective

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

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

This can be written equivalently as

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

where

$$Q^o = \left[\begin{array}{ccc} 2 & 0 & -1 \\ 0 & 0.2 & 0 \\ -1 & 0 & 2 \end{array}\right], c = \left[\begin{array}{c} 0 \\ -1 \\ 0 \end{array}\right], A = \left[\begin{array}{ccc} 1 & 1 & 1 \end{array}\right], \text{ and } b = 1.$$

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

5.5.1.1 Source code

[qo1.py]

```python
##
#   Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#   File:    qo1.py
#
#   Purpose: Demonstrate how to solve a quadratic
#            optimization problem using the MOSEK Python API.
##

import sys
import os

import mosek

# If numpy is installed, use that, otherwise use the
# Mosek's array module.
try:
    from numpy import array,zeros,ones
except ImportError:
    from mosek.array import array, zeros, ones

# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

```

```
# We might write everything directly as a script, but it looks nicer
# to create a function.
def main ():
  # Open MOSEK and create an environment and task
  # Make a MOSEK environment
  env = mosek.Env ()
  # Attach a printer to the environment
  env.set_Stream (mosek.streamtype.log, streamprinter)
  # Create a task
  task = env.Task()
  task.set_Stream (mosek.streamtype.log, streamprinter)
  # Set up and input bounds and linear coefficients
  bkc   = [ mosek.boundkey.lo ]
  blc   = [ 1.0 ]
  buc   = [ inf ]

  bkx   = [ mosek.boundkey.lo,
            mosek.boundkey.lo,
            mosek.boundkey.lo ]
  blx   = [ 0.0,  0.0, 0.0 ]
  bux   = [ inf,  inf, inf ]
  c     = [ 0.0, -1.0, 0.0 ]
  asub  = [ array([0]),   array([0]),   array([0])  ]
  aval  = [ array([1.0]), array([1.0]), array([1.0])]

  numvar = len(bkx)
  numcon = len(bkc)

  # Append 'numcon' empty constraints.
  # The constraints will initially have no bounds.
  task.appendcons(numcon)

  # Append 'numvar' variables.
  # The variables will initially be fixed at zero (x=0).
  task.appendvars(numvar)

  for j in range(numvar):
  # Set the linear term c_j in the objective.
    task.putcj(j,c[j])
    # Set the bounds on variable j
    # blx[j] <= x_j <= bux[j]
    task.putbound(mosek.accmode.var,j,bkx[j],blx[j],bux[j])
    # Input column j of A
    task.putacol( j,                  # Variable (column) index.
                  asub[j],            # Row index of non-zeros in column j.
                  aval[j])            # Non-zero Values of column j.
  for i in range(numcon):
    task.putbound(mosek.accmode.con,i,bkc[i],blc[i],buc[i])

  # Input the objective sense (minimize/maximize)
  task.putobjsense(mosek.objsense.maximize)

  # Set up and input quadratic objective
  qsubi = [ 0,   1,    2,   2   ]
  qsubj = [ 0,   1,    0,   2   ]
  qval  = [ 2.0, 0.2, -1.0, 2.0 ]

  task.putqobj(qsubi,qsubj,qval)
```

```
    task.putobjsense(mosek.objsense.minimize)

    # Optimize
    task.optimize()
    # Print a summary containing information
    # about the solution for debugging purposes
    task.solutionsummary(mosek.streamtype.msg)

    prosta = task.getprosta(mosek.soltype.itr)
    solsta = task.getsolsta(mosek.soltype.itr)

    # Output a solution
    xx = zeros(numvar, float)
    task.getxx(mosek.soltype.itr,
               xx)

    if solsta == mosek.solsta.optimal or solsta == mosek.solsta.near_optimal:
        print("Optimal solution: %s" % xx)
    elif solsta == mosek.solsta.dual_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif solsta == mosek.solsta.prim_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif solsta == mosek.solsta.near_dual_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif  solsta == mosek.solsta.near_prim_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif mosek.solsta.unknown:
      print("Unknown solution status")
    else:
      print("Other solution status")

  # call the main function
  try:
      main()
  except mosek.Exception as e:
      print ("ERROR: %s" % str(e.errno))
      if e.msg is not None:
          import traceback
          traceback.print_exc()
          print ("\t%s" % e.msg)
      sys.exit(1)
  except:
      import traceback
      traceback.print_exc()
      sys.exit(1)
  print ("Finished OK")
  sys.exit(0)
```

5.5.1.2 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, `Task.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 5.13.3):

[qo1.py]

```
qsubi = [ 0,   1,    2,   2   ]
qsubj = [ 0,   1,    0,   2   ]
qval  = [ 2.0, 0.2, -1.0, 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:

[qo1.py]

```
task.putqobj(qsubi,qsubj,qval)
```

5.5.2 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 (5.8).

Consider the problem:

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

This is equivalent to

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

where

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

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

5.5.2.1 Source code

[qcqo1.py]

```
#
#  Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#  File:    qcqo1.py
#
#  Purpose: Demonstrates how to solve small linear
#           optimization problem using the MOSEK Python API.
##

import sys

import mosek
# If numpy is installed, use that, otherwise use the
# Mosek's array module.
try:
    from numpy import array,zeros,ones
except ImportError:
    from mosek.array import array, zeros, ones

# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main ():
  # Make a MOSEK environment
  env = mosek.Env ()
  # Attach a printer to the environment
  env.set_Stream (mosek.streamtype.log, streamprinter)

  # Create a task
  task = env.Task(0,0)
  # Attach a printer to the task
  task.set_Stream (mosek.streamtype.log, streamprinter)


  # Set up and input bounds and linear coefficients
```

```
    bkc   = [ mosek.boundkey.lo ]
    blc   = [ 1.0 ]
    buc   = [ inf ]

    bkx   = [ mosek.boundkey.lo,
              mosek.boundkey.lo,
              mosek.boundkey.lo ]
    blx   = [ 0.0,  0.0, 0.0 ]
    bux   = [ inf,  inf, inf ]

    c     = [ 0.0, -1.0, 0.0 ]

    asub  = [   array([0]),   array([0]),   array([0]) ]
    aval  = [  array([1.0]),  array([1.0]),  array([1.0]) ]

    numvar = len(bkx)
    numcon = len(bkc)
    NUMANZ = 3
    # Append 'numcon' empty constraints.
    # The constraints will initially have no bounds.
    task.appendcons(numcon)

    #Append 'numvar' variables.
    # The variables will initially be fixed at zero (x=0).
    task.appendvars(numvar)

    #Optionally add a constant term to the objective.
    task.putcfix(0.0)

    for j in range(numvar):
      # Set the linear term c_j in the objective.
      task.putcj(j,c[j])
      # Set the bounds on variable j
      # blx[j] <= x_j <= bux[j]
      task.putbound(mosek.accmode.var,j,bkx[j],blx[j],bux[j])
      # Input column j of A
      task.putacol(j,                  # Variable (column) index.
                   asub[j],            # Row index of non-zeros in column j.
                   aval[j])            # Non-zero Values of column j.

    for i in range(numcon):
      task.putbound(mosek.accmode.con,i,bkc[i],blc[i],buc[i])

    # Set up and input quadratic objective

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

    task.putqobj(qsubi,qsubj,qval)

    # The lower triangular part of the Q^0
    # matrix in the first constraint is specified.
    # This corresponds to adding the term
    # - x0^2 - x1^2 - 0.1 x2^2 + 0.2 x0 x2

    qsubi = [  0,    1,    2,   2   ]
    qsubj = [  0,    1,    2,   0   ]
```

```
    qval  = [ -2.0, -2.0, -0.2, 0.2 ]

    # put Q^0 in constraint with index 0.

    task.putqconk (0, qsubi,qsubj, qval);

    # Input the objective sense (minimize/maximize)
    task.putobjsense(mosek.objsense.minimize)

    # Optimize the task
    task.optimize()

    # Print a summary containing information
    # about the solution for debugging purposes
    task.solutionsummary(mosek.streamtype.msg)

    prosta = task.getprosta(mosek.soltype.itr)
    solsta = task.getsolsta(mosek.soltype.itr)

    # Output a solution
    xx = zeros(numvar, float)
    task.getxx(mosek.soltype.itr,
                          xx)

    if solsta == mosek.solsta.optimal or solsta == mosek.solsta.near_optimal:
        print("Optimal solution: %s" % xx)
    elif solsta == mosek.solsta.dual_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif solsta == mosek.solsta.prim_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif solsta == mosek.solsta.near_dual_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif  solsta == mosek.solsta.near_prim_infeas_cer:
        print("Primal or dual infeasibility.\n")
    elif mosek.solsta.unknown:
      print("Unknown solution status")
    else:
      print("Other solution status")

  # call the main function
  try:
      main ()
  except mosek.Exception as e:
      print ("ERROR: %s" % str(code))
      if msg is not None:
          print ("\t%s" % e.msg)
          sys.exit(1)
  except:
      import traceback
      traceback.print_exc()
      sys.exit(1)
  sys.exit(0)
```

The only new function introduced in this example is `Task.putqconk`, which is used to add quadratic terms to the constraints. While `Task.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 `Task.putqcon` function.

5.6 The solution summary

All computations inside MOSEK are performed using finite precision floating point numbers. This implies the reported solution isonly be an approximate optimal solution. Therefore after solving an optimization problem it is important to investigate how good an approximation the solution is. This can easily be done using the function `Task.solutionsummary` which reports how much the solution violate the primal and dual constraints and the primal and dual objective values. Recall for a convex optimization problem the optimality conditions are:

- The primal solution must satisfy all the primal constraints.
- The dual solution much satisfy all the dual constraints.
- The primal and dual objective values must be identical.

Thus the solution summary reports information that makes it possible to evaluate the quality of the solution obtained.

In case of a linear optimization problem the solution summary may look like

```
Basic solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -4.6475314286e+002  Viol.  con: 2e-014   var: 0e+000
  Dual.    obj: -4.6475316001e+002  Viol.  con: 7e-009   var: 4e-016
```

The summary reports information for the basic solution. In this case we see:

- The problem status is primal and dual feasible which means the problem has an optimal solution. The problem status can be obtained using `Task.getprosta`.
- The solution status is optimal. The solution status can be obtained using `Task.getsolsta`.
- Next information about the primal solution is reported. The information consists of the objective value and violation meassures for the primal solution. In this case violations for the constraints and variables are small meaning the solution is very close to being an exact feasible solution. The violation meassure for the variables is the worst violation of the solution in any of the bounds on the variables.

  The constraint and variable violations are computed with `Task.getpviolcon` and `Task.getpviolvar`.
- Similarly for the dual solution the violations are small and hence the dual solution is feasible. The constraint and variable violations are computed with `Task.getdviolcon` and `Task.getdviolvar` respectively.
- Finally, it can be seen that the primal and dual objective values are almost identical. Using `Task.getprimalobj` and `Task.getdualobj` the primal and dual objective values can be obtained.

To summarize in this case a primal and a dual solution with small feasiblity violations are available. Moreover, the primal and dual objective values are almost identical and hence it can be concluded that the reported solution is a good approximation to the optimal solution.

Now what happens if the problem does not have an optimal solution e.g. it is primal infeasible. In that case the solution summary may look like

```
Basic solution summary
  Problem status  : PRIMAL_INFEASIBLE
  Solution status : PRIMAL_INFEASIBLE_CER
  Dual.    obj: 3.5894503823e+004   Viol.  con: 0e+000   var: 2e-008
```

i.e. MOSEK reports that the solution is a certificate of primal infeasibility. Since the problem is primal infeasible it does not make sense to report any information about the primal solution. However, the dual solution should be a certificate of the primal infeasibility. If the problem is a minimization problem then the dual objective value should be positive and in the case of a maximization problem it should be negative. The quality of the certificate can be evaluated by comparing the dual objective value to the violations. Indeed if the objective value is large compared to the largest violation then the certificate highly accurate. Here is an example

```
Basic solution summary
  Problem status  : PRIMAL_INFEASIBLE
  Solution status : PRIMAL_INFEASIBLE_CER
  Dual.    obj: 3.0056574100e-005   Viol.  con: 9e-013   var: 2e-011
```

of a not so strong infeasibility certificate because the dual objective value is small compared to largest violation.

In the case a problem is dual infeasible then the solution summary may look like

```
Basic solution summary
  Problem status  : DUAL_INFEASIBLE
  Solution status : DUAL_INFEASIBLE_CER
  Primal.  obj: -1.4500853392e+001  Viol.  con: 0e+000   var: 0e+000
```

Observe when a solution is a certificate of dual infeasibility then the primal solution contains the certificate. Moreoever, given the problem is a minimization problem the objective value should negative and the objective should be large compared to the worst violation if the certificate is strong.

5.7 Integer optimization

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

5.7.1 Example: Mixed integer linear optimization

In this section the example

$$\begin{array}{lccl} \text{maximize} & x_0 + 0.64x_1 & & \\ \text{subject to} & 50x_0 + 31x_1 & \leq & 250, \\ & 3x_0 - 2x_1 & \geq & -4, \\ & x_0, x_1 \geq 0 & & \text{and integer} \end{array} \tag{5.10}$$

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

5.7.1.1 Source code

The example (5.10) 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 `Task.putvartype` as shown in the code:

[milo1.py]

```
task.putvartypelist([ 0, 1 ],
                    [ mosek.variabletype.type_int,
                      mosek.variabletype.type_int ])
```

The complete source for the example is listed below.

[milo1.py]

```
##
#    Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#    File:    milo1.py
#
#    Purpose:  Demonstrates how to solve a small mixed
#              integer linear optimization problem using the MOSEK Python API.
##

import sys

import mosek
# If numpy is installed, use that, otherwise use the
# Mosek's array module.
try:
    from numpy import array,zeros,ones
except ImportError:
    from mosek.array import array, zeros, ones

# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main ():
  # Make a MOSEK environment
  env = mosek.Env ()
  # Attach a printer to the environment
  env.set_Stream (mosek.streamtype.log, streamprinter)

  # Create a task
  task = env.Task(0,0)
  # Attach a printer to the task
  task.set_Stream (mosek.streamtype.log, streamprinter)

```

```
    bkc = [ mosek.boundkey.up, mosek.boundkey.lo  ]
    blc = [              -inf,              -4.0  ]
    buc = [             250.0,               inf  ]

    bkx = [ mosek.boundkey.lo, mosek.boundkey.lo  ]
    blx = [               0.0,               0.0  ]
    bux = [               inf,               inf  ]

    c   = [               1.0,               0.64 ]

    asub = [  array([0,   1]),    array([0,    1])   ]
    aval = [ array([50.0, 3.0]), array([31.0, -2.0]) ]

    numvar = len(bkx)
    numcon = len(bkc)

    # Append 'numcon' empty constraints.
    # The constraints will initially have no bounds.
    task.appendcons(numcon)

    #Append 'numvar' variables.
    # The variables will initially be fixed at zero (x=0).
    task.appendvars(numvar)

    for j in range(numvar):
      # Set the linear term c_j in the objective.
      task.putcj(j,c[j])
      # Set the bounds on variable j
      # blx[j] <= x_j <= bux[j]
      task.putvarbound(j,bkx[j],blx[j],bux[j])
      # Input column j of A
      task.putacol(j,                  # Variable (column) index.
                   asub[j],            # Row index of non-zeros in column j.
                   aval[j])            # Non-zero Values of column j.

    task.putconboundlist(range(numcon),bkc,blc,buc)

    # Input the objective sense (minimize/maximize)
    task.putobjsense(mosek.objsense.maximize)

    # Define variables to be integers
    task.putvartypelist([ 0, 1 ],
                        [ mosek.variabletype.type_int,
                          mosek.variabletype.type_int ])

    # Optimize the task
    task.optimize()

    # Print a summary containing information
    # about the solution for debugging purposes
    task.solutionsummary(mosek.streamtype.msg)

    prosta = task.getprosta(mosek.soltype.itg)
    solsta = task.getsolsta(mosek.soltype.itg)

    # X

    # Output a solution
```

```
  xx = zeros(numvar, float)
  task.getxx(mosek.soltype.itg,xx)

  if solsta in [ mosek.solsta.integer_optimal, mosek.solsta.near_integer_optimal ]:
      print("Optimal solution: %s" % xx)
  elif solsta == mosek.solsta.dual_infeas_cer:
      print("Primal or dual infeasibility.\n")
  elif solsta == mosek.solsta.prim_infeas_cer:
      print("Primal or dual infeasibility.\n")
  elif solsta == mosek.solsta.near_dual_infeas_cer:
      print("Primal or dual infeasibility.\n")
  elif  solsta == mosek.solsta.near_prim_infeas_cer:
      print("Primal or dual infeasibility.\n")
  elif mosek.solsta.unknown:
    if prosta == mosek.prosta.prim_infeas_or_unbounded:
        print("Problem status Infeasible or unbounded.\n")
    elif prosta == mosek.prosta.prim_infeas:
        print("Problem status Infeasible.\n")
    elif prosta == mosek.prosta.unkown:
        print("Problem status unkown.\n")
    else:
        print("Other problem status.\n")
  else:
      print("Other solution status")

# call the main function
try:
    main ()
except mosek.Exception as msg:
    #print "ERROR: %s" % str(code)
    if msg is not None:
        print ("\t%s" % msg)
        sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
sys.exit(0)
```

5.7.1.2 Code comments

Please note that when Task.getsolutionslice is called, the integer solution is requested by using soltype.itg. No dual solution is defined for integer optimization problems.

5.7.2 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 Task.putsolution (for inputting a whole solution) or Task.putsolutioni (for inputting solution values related to a single variable or constraint).

It is not necessary to specify the whole solution. By setting the iparam.mio_construct_sol parameter to `onoffkey.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.

5.7.3 Example: Specifying an integer solution

Consider the problem

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

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

[mioinitsol.py]

```
##
#    Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#    File:    mioinitsol.py
#
#    Purpose:  Demonstrates how to solve a small mixed
#              integer linear optimization problem using the MOSEK Python API.
##

import sys

import mosek

# If numpy is installed, use that, otherwise use the
# Mosek's array module.
try:
    from numpy import array,zeros,ones
except ImportError:
    from mosek.array import array, zeros, ones

# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0


# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()


# We might write everything directly as a script, but it looks nicer
# to create a function.
def main ():
    # Make a MOSEK environment
    env = mosek.Env ()
    # Attach a printer to the environment
    env.set_Stream (mosek.streamtype.log, streamprinter)
```

```
    # Create a task
    task = env.Task(0,0)
    # Attach a printer to the task
    task.set_Stream (mosek.streamtype.log, streamprinter)


    bkc  = [ mosek.boundkey.up ]
    blc  = [ -inf,             ]
    buc  = [ 2.5               ]

    bkx  = [ mosek.boundkey.lo,
             mosek.boundkey.lo,
             mosek.boundkey.lo,
             mosek.boundkey.lo ]

    blx  = [0.0, 0.0, 0.0, 0.0 ]
    bux  = [ inf, inf,  inf,  inf ]

    c    = [ 7.0, 10.0, 1.0, 5.0 ]

    asub = [  0,   0,   0,   0   ]
    acof = [  1.0, 1.0, 1.0, 1.0]

    ptrb = [ 0, 1, 2, 3 ]
    ptre = [ 1, 2, 3, 4 ]

    numvar = len(bkx)
    numcon = len(bkc)

    # Input linear data
    task.inputdata(numcon,numvar,
                   c,0.0,
                   ptrb, ptre, asub, acof,
                   bkc,  blc,  buc,
                   bkx,  blx,  bux)

    # Input objective sense
    task.putobjsense(mosek.objsense.maximize)

    # Define variables to be integers
    task.putvartypelist([ 0, 1, 2 ],
                        [ mosek.variabletype.type_int,
                          mosek.variabletype.type_int,
                          mosek.variabletype.type_int])

    # Construct an initial feasible solution from the
    #     values of the integer valuse specified
    task.putintparam(mosek.iparam.mio_construct_sol,
                     mosek.onoffkey.on);

    # Assign values 0,2,0 to integer variables. Important to
    # assign a value to all integer constrained variables.
    task.putxxslice(mosek.soltype.itg,0,3,[0.0, 2.0, 0.0])

    # Optimize
    task.optimize()

```

```
    # Did mosek construct a feasible initial solution ?
    if task.getintinf(mosek.iinfitem.mio_construct_solution) > 0:
        print("Objective value of constructed integer solution: %-24.12e" % task.getdouinf(mosek.dinfitem.mio_construct_solu
    else:
        print("Intial integer solution construction failed.");

    if task.solutiondef(mosek.soltype.itg):

        # Output a solution
        xx = zeros(numvar, float)
        task.getxx(mosek.soltype.itg, xx)
        print("Integer optimal solution")
        for j in range(0,numvar) :
            print("\tx[%d] = %e" % (j,xx[j]))
    else:
        print("No integer solution is available.")

# call the main function
try:
    main ()
except mosek.Exception as e:
    print ("ERROR: %s" % str(e.errno))
    if e.msg is not None:
        print ("\t%s" % e.msg)
    sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

5.8 The solution summary for mixed integer problems

The solution summary for a mixed-integer problem may look like

```
Integer solution solution summary
  Problem status  : PRIMAL_FEASIBLE
  Solution status : INTEGER_OPTIMAL
  Primal.  obj: 4.0593518000e+005   Viol.  con: 4e-015   var: 3e-014   itg: 3e-014
```

The main diffrence compared to continous case covered previously is that no information about the dual solution is provided. Simply because there is no dual solution available for a mixed integer problem. In this case it can be seen that the solution is higly feasible because the violations are small. Moreoever, the solution is denoted integer optimal. Observe `itg: 3e-014` implies that all the integer constrained variables are at most $3e-014$ from being an exact integer.

5.9 Response handling

After solving an optimization problem with MOSEK an approriate action must be taken depending on the outcome. Usually, the expected outcome is an optimal solution, but there may be several situations where this is not the result. E.g., if the problem is infeasible or nearly so or if the solver ran out of memory or stalled while optimizing, the result may not be as expected.

This section discusses what should be considered when an optimization has ended unsuccessfully.

Before continuing, let us consider the four status codes available in MOSEK that is relevant for the error handing:

The termination code:

: The termination provides information about why the optimizer terminated. For instance if a time limit has been specfied (this is common for mixed integer problems), the termination code will tell if this termination limit was the cause of the termination. Note that reaching a prespecfied time limit is not considered an exceptional case. It must be expected that this occurs occasionally.

Note that if we want to report, e.g., that the optimizer terminated due to a time limit or because it stalled but with a feasible solution, we have to consider *both* the termination code, *and* the solution status.

The following pseudo code demonstrates a best practice way of dealing with the status codes.

```
if ( the solution status is as expected )
{
  The normal case:
    Do whatever that was planned. Note the response code is
    ignored because the solution has the expected status.
    Of course we may check the response anyway if we like.
}
else
{
  Exceptional case:
    Based on solution status, response and termination codes take
    appropriate action.
}
```

In the following example the pseudo code has implemented. The idea of the example is to read an optimization problem from a file, e.g., an MPS file and optimize it. Based on status codes an appropriate action is taken, which in this case is to print a suitable message.

[response.py]

```
#
# Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
# File:      response.py
#
# Purpose:   This examples demonstrates proper response handling.
#

import mosek
import sys

def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main(args):
  if len(args) < 1:
    print ("No input file specified")
```

```
      return
    else:
      print ("Inputfile: %s" % args[0])

    with mosek.Env() as env:
      with env.Task(0,0) as task:
        task.set_Stream (mosek.streamtype.log, streamprinter)

        task.readdata(args[0])
        e = None
        trmcode = None
        try:
          trmcode = task.optimize()
        except mosek.MosekException as err:
          e = err

        solsta = task.getsolsta(mosek.soltype.itr)

        if   solsta in [ mosek.solsta.optimal,
                         mosek.solsta.near_optimal ]:
          print ("An optimal basic solution is located.")
          task.solutionsummary(mosek.streamtype.log)
        elif solsta in [ mosek.solsta.dual_infeas_cer,
                         mosek.solsta.near_dual_infeas_cer ]:
          print ("Dual infeasibility certificate found.")
        elif solsta in [ mosek.solsta.prim_infeas_cerl,
                         mosek.solsta.near_prim_infeas_cer ]:
          printf("Primal infeasibility certificate found.\n");
        elif solsta == mosek.solsta.sta_unknown:
          # The solutions status is unknown. The termination code
          # indicating why the optimizer terminated prematurely.
          print ("The solution status is unknown.")
          if trmcode is not None:
            print ("Termination code: %s" % str(trmcode))
            #print mosek.getcodedesc(trmcode)
          elif e is not None:
            print ("Error:")
            print (e)
        else:
          print ("An unexpected solution status is obtained.")

  if __name__ == '__main__':
    import sys
    main(sys.argv[1:])
```

5.10 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 re-optimize an existing problem. The example we study is a simple production planning model.

5.10.1 Example: Production planning

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:

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

and

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

The following code loads this problem into the optimization task.

[production.py]

```
# Create a MOSEK environment
env = mosek.Env ()

# Create a task
task = env.Task(0,0)
# Attach a printer to the task
task.set_Stream (mosek.streamtype.log, streamprinter)

# Bound keys for constraints
bkc = [mosek.boundkey.up,
              mosek.boundkey.up,
              mosek.boundkey.up]
# Bound values for constraints
blc = array ([-inf, -inf, -inf])
buc = array ([100000.0 , 50000.0, 60000.0])

# Bound keys for variables
bkx = [mosek.boundkey.lo,
              mosek.boundkey.lo,
              mosek.boundkey.lo]
# Bound values for variables
```

```
blx = array ([ 0.0,  0.0,  0.0])
bux = array ([+inf, +inf, +inf])

# Objective coefficients
csub = array([   0,    1,    2 ])
cval = array([ 1.5, 2.5, 3.0 ])

# We input the A matrix column-wise
# asub contains row indexes
asub = array([ 0, 1, 2,
               0, 1, 2,
               0, 1, 2])
# acof contains coefficients
acof = array([ 2.0, 3.0, 2.0,
               4.0, 2.0, 3.0,
               3.0, 3.0, 2.0 ])
# aptrb and aptre contains the offsets into asub and acof where
# columns start and end respectively
aptrb = array([ 0, 3, 6 ])
aptre = array([ 3, 6, 9 ])

numvar = len(bkx)
numcon = len(bkc)

# Append the constraints
task.appendcons(numcon)

# Append the variables.
task.appendvars(numvar)

# Input objective
task.putcfix(0.0)
task.putclist(csub,cval)

# Put constraint bounds
task.putconboundslice(0, numcon, bkc, blc, buc)

# Put variable bounds
task.putvarboundslice(0, numvar,bkx, blx, bux)

# Input A non-zeros by columns
for j in range(numvar):
    ptrb,ptre = aptrb[j],aptre[j]
    task.putacol(j,
                 asub[ptrb:ptre],
                 acof[ptrb:ptre])

# Input the objective sense (minimize/maximize)
task.putobjsense(mosek.objsense.maximize)


# Optimize the task
task.optimize()

# Output a solution
xx = zeros(numvar, float)
task.getsolutionslice(mosek.soltype.bas,
                      mosek.solitem.xx,
```

```
                     0,numvar,
                     xx)
print ("xx =", [i for i in xx])
```

5.10.2 Changing the A matrix

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

[production.py]

```
task.putaij(0, 0, 3.0)
```

The problem now has the form:

$$\begin{array}{lccccccc} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & & \\ \text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & \leq & 100000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & \leq & 50000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & \leq & 60000, \end{array} \tag{5.11}$$

and

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

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

Task.optimize again.

5.10.3 Appending variables

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

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

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

[production.py]

```
# Append a new varaible x_3 to the problem */
task.appendvars(1)

# Set bounds on new varaible
task.putbound(mosek.accmode.var,
              task.getnumvar()-1,
              mosek.boundkey.lo,
              0,
              +inf)

```

```
# Change objective
task.putcj(task.getnumvar()-1,1.0)

# Put new values in the A matrix
acolsub =   array([0,   2])
acolval =   array([4.0, 1.0])

task.putacol(task.getnumvar()-1, # column index
             acolsub,
             acolval)
```

After this operation the problem looks this way:

$$\begin{array}{lccccccccc} \text{maximize} & 1.5x_0 & + & 2.5x_1 & + & 3.0x_2 & + & 1.0x_3 & & \\ \text{subject to} & 3x_0 & + & 4x_1 & + & 3x_2 & + & 4x_3 & \leq & 100000, \\ & 3x_0 & + & 2x_1 & + & 3x_2 & & & \leq & 50000, \\ & 2x_0 & + & 3x_1 & + & 2x_2 & + & 1x_3 & \leq & 60000, \end{array} \tag{5.12}$$

and

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

5.10.4 Reoptimization

When

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

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

In this case an optimal solution to problem (5.11) was found and then added a column was added to get (5.12). The simplex optimizer is well suited for exploiting an existing primal or dual feasible solution. Hence, the subsequent code instructs MOSEK to choose the simplex optimizer freely when optimizing.

[production.py]

```
# Change optimizer to simplex free and reoptimize
task.putintparam(mosek.iparam.optimizer,mosek.optimizertype.free_simplex)
task.optimize()
```

5.10.5 Appending constraints

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

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

This corresponds to adding the constraint

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

to the problem which is done in the following code:

[production.py]

```
# Append a new constraint
task.appendcons(1)

# Set bounds on new constraint
task.putconbound( task.getnumcon()-1, mosek.boundkey.up,-inf, 30000)

# Put new values in the A matrix

arowsub = array([0,   1,   2,   3  ])
arowval = array([1.0, 2.0, 1.0, 1.0])

task.putarow(task.getnumcon()-1, # row index
             arowsub,
             arowval)
```

5.11 Solution analysis

5.11.1 Retrieving solution quality information with the API

Information about the solution quality may be retrieved in the API with the help of the following functions:

- `Task.getsolutioninfo`: Obtains information about objective values and the solution violations of the constraints.
- `Task.analyzesolution`: Print additional information about the solution, e.g basis condition number and optionally a list of violated constraints.
- `Task.getpviolcon`, `Task.getpviolvar`, `Task.getpviolbarvar`,`Task.getpviolcones`, `Task.getdviolcon`, `Task.getdviolvar`, `Task.getdviolbarvar`,`Task.getdviolcones`. Obtains violation of the individual constraints.

5.12 Efficiency considerations

Although MOSEK is implemented to handle memory efficiently, the user may have valuable knowledge about a problem, which could be used to improve the performance of MOSEK This section discusses some tricks and general advice that hopefully make MOSEK process your problem faster.

Avoiding memory fragmentation:

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

- `Task.putmaxnumvar`. Estimate for the number of variables.
- `Task.putmaxnumcon`. Estimate for the number of constraints.
- `Task.putmaxnumcone`. Estimate for the number of cones.
- `Task.putmaxnumbarvar`. Estimate for the number of semidefinite matrix variables.
- `Task.putmaxnumanz`. Estimate for the number of non-zeros in A.
- `Task.putmaxnumqnz`. Estimate for the number of non-zeros in the quadratic terms.

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

Do not mix `put-` and `get-` functions:

For instance, the functions `Task.putacol` and `Task.getacol`. MOSEK will queue `put-` commands internally until a `get-` function is called. If every `put-` function call is followed by a `get-` function call, the queue will have to be flushed often, decreasing efficiency.

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

Use the LIFO principle when removing constraints and variables:

MOSEK can more efficiently remove constraints and variables with a high index than a small index.

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

Add more constraints and variables than you need (now):

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

Use one environment (env) only:

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

Do not remove basic variables:

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

5.12.1 API overhead

The Python interface is a thin wrapper around a native MOSEK library. The layer between the Python application and the native MOSEK library is made as thin as possible to minimize the overhead from function calls.

The methods in `mosek.Env` and `mosek.Task` are all written in C and resides in the module `pymosek`. Each method converts the call parameter data structures (i.e. creates a complete copy of the data), calls a MOSEK function and converts the returned values back into Python structures.

The following rules will often improve the performance of the MOSEK/Python API:

Reuse Env and Task whenever possible

There may be some overhead involved in creating and deleting task and environment objects, so if possible reuse these.

Make sure to delete task and environment when not in use anymore

Using the `with`-construction (available in python 2.6 and later) will allow automatic deletion of the environment and task. If this is not an option, use `Env.__del__()` and `Task.__del__()` to destroy the objects. Failing to do this may cause memory leaks in some cases.

Avoid input loops

Whenever possible imput data in large chunks or vectors instead of using loops. For small `put-` and `get-` methods there is a significant overhead, so for example inputting one row of the A-matrix at the time may be much slower than inputting the whole matrix.

For example, a loop with `Task.putarow` may be replaced with one `Task.putarowlist`, or a loop of `Task.putqobjij` may be replaced with `Task.putqobj`.

5.13 Conventions employed in the API

5.13.1 Naming conventions for arguments

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

In Table 5.2 the variable names used to specify the problem parameters are listed. The relation between the variable names and the problem parameters is as follows:

Python name	Python 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			
c	float[]	numvar	c_j
cfix	float		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
aptrb	int[]	numvar	a_{ij}
aptre	int[]	numvar	a_{ij}
asub	int[]	aptre[numvar-1]	a_{ij}
aval	float[]	aptre[numvar-1]	a_{ij}
blc	float[]	numcon	l_k^c
buc	float[]	numcon	u_k^c
blx	float[]	numvar	l_k^x
bux	float[]	numvar	u_k^x

Table 5.2: Naming convensions used in the MOSEK Python API.

Symbolic constant	Lower bound	Upper bound
boundkey.fx	finite	identical to the lower bound
boundkey.fr	minus infinity	plus infinity
boundkey.lo	finite	plus infinity
boundkey.ra	finite	finite
boundkey.up	minus infinity	finite

Table 5.3: Interpretation of the bound keys.

- The quadratic terms in the objective:

$$q^o_{\mathtt{qosubi[t]},\mathtt{qosubj[t]}} = \mathtt{qoval[t]},\ t = 0, \ldots, \mathtt{numqonz} - 1. \tag{5.13}$$

- The linear terms in the objective:

$$c_j = \mathtt{c[j]},\ j = 0, \ldots, \mathtt{numvar} - 1 \tag{5.14}$$

- The fixed term in the objective:

$$c^f = \mathtt{cfix}.$$

- The quadratic terms in the constraints:

$$q^{\mathtt{qcsubk[t]}}_{\mathtt{qcsubi[t]},\mathtt{qcsubj[t]}} = \mathtt{qcval[t]},\ t = 0, \ldots, \mathtt{numqcnz} - 1. \tag{5.15}$$

- The linear terms in the constraints:

$$a_{\mathtt{asub[t]},\mathtt{j}} = \mathtt{aval[t]}, \quad \begin{array}{l} t = \mathtt{ptrb[j]}, \ldots, \mathtt{ptre[j]} - 1, \\ j = 0, \ldots, \mathtt{numvar} - 1. \end{array} \tag{5.16}$$

- The bounds on the constraints are specified using the variables `bkc`, `blc`, and `buc`. The components of the integer array `bkc` specify the bound type according to Table 5.3. For instance `bkc[2]=boundkey.lo` means that $-\infty < l_2^c$ and $u_2^c = \infty$. Finally, the numerical values of the bounds are given by

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

and

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

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

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

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

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

5.13.1.1 Bounds

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

5.13.2 Vector formats

Three different vector formats are used in the MOSEK API:

Full vector:

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

```
c = zeros(numvar,float)
task.getc(c)
```

where numvar is the number of variables in the problem.

Vector slice:

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

```
upper_bound = zeros(8,float)
lower_bound = zeros(8,float)
bound_key   = array([None] * 8)

task.getboundslice(accmode.con, 2, 10,
                   bound_key,lower_bound,upper_bound)
```

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

Sparse vector:

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

```
bound_index = [          1,          6,          3,          9]
bound_key   = [boundkey.fr,boundkey.lo,boundkey.up,boundkey.fx]
lower_bound = [     0.0,         -10.0,        0.0,        5.0]
upper_bound = [     0.0,           0.0,        6.0,        5.0]
task.putboundlist(accmode.con, bound_index,
                  bound_key,lower_bound,upper_bound)
```

Note that the list of indexes need not be ordered.

5.13.3 Matrix formats

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

5.13.3.1 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:

```
subi = array([   1,   3,   5 ])
subj = array([   2,   3,   4 ])
cof  = array([ 1.1, 4.3, 0.2 ])
task.putaijlist(subi,subj,cof)
```

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

5.13.3.2 Row or column ordered sparse matrix

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

A sparse matrix in column ordered format consists of:

`asub`:

List of row indexes.

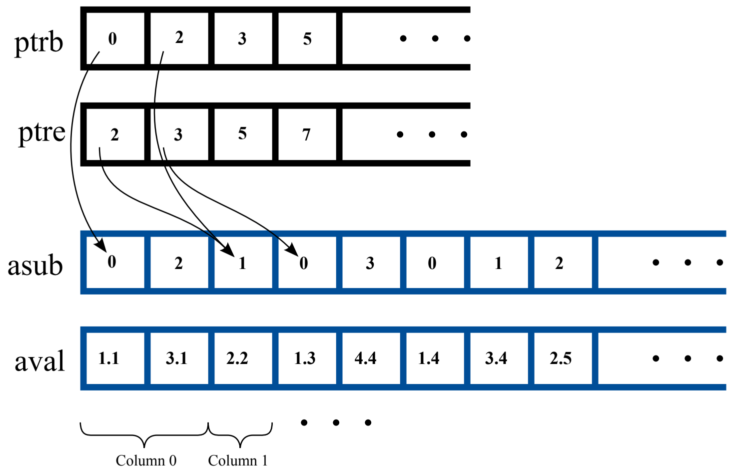

Figure 5.1: The matrix A (5.17) represented in column ordered packed sparse matrix format.

`aval`:

List of non-zero entries of A ordered by columns.

`ptrb`:

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

`ptre`:

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

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

$$j = 0, \ldots, numcol - 1.$$

We define

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

As an example consider the matrix

$$A = \begin{bmatrix} 1.1 & & 1.3 & 1.4 & \\ & 2.2 & & & 2.5 \\ 3.1 & & & 3.4 & \\ & & 4.4 & & \end{bmatrix}. \tag{5.17}$$

which can be represented in the column ordered sparse matrix 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 (5.17) is represented in column ordered sparse matrix format.

5.13.3.3 Row ordered sparse matrix

The matrix A (5.17) can also be represented in the row ordered sparse matrix format as:

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

5.13.4 Array objects

The MOSEK Python API provides a simple array object in the module `mosekarr`. This includes a one-dimensional dense array which can be of type `Float`, `Int` or `Object`, and a few operators and functions to create and modify array objects.

Arrays can be constructed in several ways:

```
# Create an array of integers
a0 = array([1,2,3],int)
# Create an array of floats
a1 = array([1,2,3],float)
# Create an integer array of ones
a2 = ones(10)
# Create an float array of ones
a3 = ones(10,float)
# Create a range of integers 5,6,...,9
a4 = range(5,10)
# Create and array of objects
a5 = array(['a string', 'b string', 10, 2.2])
```

A limited set of operations on arrays are available - these should work more or less like the equivalent `Numeric` operations:

```
a = ones(10,float)
b = 1.0 * arange(10)


# element-wise multiplication, addition and subtraction
c0 = a * b
c1 = a + b
c2 = a - b
```

```
# multiplly each element by 2.1
c4 = a * 2.1

# add 2 to each element
c5 = a + 2
```

If more advanced array operations is needed, it is necessary to install the Python `Numeric` package.

5.13.5 Typical problems using the Python API

Since all all type-information in Python is implicit, type-checking is performed only when required, and in certain cases it is necessary to explicitly write type information.

The MOSEK API currently *only* supports its own array object (`mosek.array.array`) and Python`numpy arrays`. Other array or list compatible objects will are accepted but are converted.

Typically type errors occur in two situations:

- An array argument did not have the right type and could not be converted.
- An array was expected, but the argument was not an array and not a list-compatible object.

Furthermore, please note that `mosek.array` module only supports a limited set of array types: `int32`, `int64`, `float64` and `bool`. The numerical types support normal simple mathematical operation (addition, subtraction, multiplication etc.)

5.14 The license system

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

To change the license systems behavior to returning the license token after each call to `Task.optimize` set the parameter `iparam.cache_license` to `onoffkey.off`. Please note that there is a small overhead associated with setting this parameter, since checking out a license token from the license server can take a small amount of time.

Additionally license checkout and checkin can be controlled manually with the functions `Env.checkinlicense` and `Env.checkoutlicense`.

5.14.1 Waiting for a free license

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

Chapter 6

Nonlinear API tutorial

This chapter provides information about how to solve general convex nonlinear optimization problems using MOSEK. By general nonlinear problems it is meant problems that cannot be formulated as a conic quadratic optimization or a convex quadratically constrained optimization problem.

In general it is recommended not to use nonlinear optimizer unless needed. The reasons are

- MOSEK has no way of checking whether the formulated problem is convex and if this assumption is not satisfied the optimizer will not work.
- The nonlinear optimizer requires 1st and 2nd order derivative information which is hard to provide correctly i.e. it is nontrivial to program the code that computes the derivative information.
- The specification of nonlinear problems requires C function callbacks. Such C function callbacks cannot be dump to disk and that makes it hard to report issues to MOSEK support.
- The algorithm employed for nonlinear optimization problems is not as good as the one employed for conic problems i.e. conic problems has special that can be exploited to make the optimizer faster and more robust.

This leads to following advices in decreasing order of importance.

- Consider reformulating the problem to a conic quadratic optimization problem if at all possible. In particular many problems involving polynomial terms can easily be reformulated to conic quadratic form.
- Consider reformulating the problem to a separable optimization problem because that simplifies the issue with verifying convexity and computing 1st and 2nd order derivatives significantly. In most cases problems on separable form also solves faster because of the simpler structure of the functions. In Section 6.1 some utility code that makes it easy to solve separable problems is discussed.
- Finally, if the problem cannot be reformulated to separable form then use a modelling language like AMPL or GAMS. The reason is the modeling language will do all the computing of function

values and derivatives. This eliminates an important source of errors. Therefore, it is strongly recommended to use a modelling language at the protype stage.

6.1 Separable convex (SCopt) interface

The MOSEK Python API provides a way to add simple non-linear functions composed from a limited set of non-linear terms. Non-linear terms can be mixed with quadratic terms in objective and constraints.

We consider a normal linear problem with additional non-linear terms z:

$$\begin{array}{lccccl}
\text{minimize} & & & z_0(x) + c^T x & & \\
\text{subject to} & l_i^c & \leq & z_i(x) + a_i^T x & \leq & u_i^c, \ i = 1 \ldots m \\
 & l^x & \leq & x & \leq & u^x, \\
 & x \in \mathbb{R}^n & & & & \\
 & z : \mathbb{R}^n \rightarrow \mathbb{R}^{(m+1)} & & & &
\end{array}$$

Using the separable non-linear interface it is possible to add non-linear functions of the form

$$z_i(x) = \sum_{k=1}^{K_i} w_k^i(x_{p_{ik}}), \ w_k^i : \mathbb{R} \rightarrow \mathbb{R}$$

In other words, each non-linear function z_i is a sum of separable functions w_k^i of one variable each. A limited set of functions are supported; each w_k^i can be one of the separable functions:

Separable function	Operator name	
$fx\mathtt{ln}(x)$	`scopr.ent`	Entropy function
fe^{gx+h}	`scopr.exp`	Exponential function
$f\mathtt{ln}(gx+h)$	`scopr.log`	Logarithm
$f(x+h)^g$	`scopr.pow`	Power function

where f, g and h are constants.

This formulation does not guarantee convexity. For MOSEK to be able to solve the problem, following requirements must be met:

- If the objective is minimized, the sum of non-linear terms must be convex, otherwise it must be concave.
- Any constraint bounded below must be concave, and any constraint bounded above must be convex.
- Each separable term must be twice differentiable within the bounds of the variable it is applied to.

If these are not satisfied MOSEK may not be able to solve the problem or produce a meaningful status report. For details see section 6.1.3.

6.1.1 Adding separable terms

Separable terms — both objective and constraint terms — are added in one chunk and replaces any previously added non-linear terms. Each individual term can be describes by a set of values:

- `opr`, an indicator of which of the basic functions is applied,
- i, the constraint index for terms in constraints,
- j, the index of the variable the functions is applied to,
- f, the constant f in the basic function,
- g, the constant g in the basic function, and
- h, the constant h in the basic function.

For example:

Term	opr	j	f	g	h
$0.1x_1\mathtt{ln}(x_1)$	`scopr.ent`	1	0.1	0.0	0.0
$e^{x_2+1.1}$	`scopr.exp`	2	1.0	1.0	1.1
$2.1x_1^{1.75}$	`scopr.pow`	1	2.1	1.75	0.0
$\sqrt{x_1}$	`scopr.pow`	1	1.0	0.5	0.0
$\mathtt{ln}(x_2+1.2)$	`scopr.log`	2	1.0	1.0	1.2

The separable terms of the objective can now be defined by a set of arrays

```
mosek.scopr array opro  # which method
int array         oprjo # variable index
float array       oprfo # f constant
float array       oprgo # g constant
float array       oprho # h constant
```

and the separable constraint terms can be defined the same way, only using an additional array indicating which constraint each term belongs in

```
mosek.scopr array oprc  # which method
int array         opric # constraint index
int array         oprjc # variable index
float array       oprfc # f constant
float array       oprgc # g constant
float array       oprhc # h constant
```

We can now input the separable terms using the `Task.putSCeval` function:

```
task.putSCeval(opro, oprjo, oprfo, oprgo, oprho,
               oprc, opric, oprjc, oprfc, oprgc, oprhc);
```

If we wish to input no objective terms, all `opr*o` arguments may be `None`, and similarly, if we have no constraint terms, we may let all `opr*c` be `None`.

This will replace all existing non-linear separable terms. To remove all non-linear separable terms, we can call `Task.clearSCeval`.

6.1.2 Example: Simple separable problem

We consider the convex separable problem

$$
\begin{array}{lllllllll}
\text{minimize} & & & e^{x_2} & +x^{x_3} & & & & \\
\text{such that} & & & & & e^{x_4} & +e^{x_5} & < & 1 \\
& x_0 & +x_1 & -x_2 & & & & = & 0 \\
& -x_0 & -x_1 & & -x_3 & & & = & 0 \\
& 0.5x_0 & & & & -x_4 & & = & 1.3862944 \\
& & x_1 & & & & -x_5 & = & 0
\end{array}
$$

with 4 separable terms, two in the objective and two in the constraints.

The separable terms of the objective can be described by the arrays

[demb781.py]

```
opro  = [ mosek.scopr.exp, mosek.scopr.exp ]
oprjo = [ 2, 3 ]
oprfo = [ 1.0, 1.0 ]
oprgo = [ 1.0, 1.0 ]
oprho = [ 0.0, 0.0 ]
```

and constraint terms by

[demb781.py]

```
oprc  = [ mosek.scopr.exp, mosek.scopr.exp ]
opric = [ 0, 0 ]
oprjc = [ 4, 5 ]
oprfc = [ 1.0, 1.0 ]
oprgc = [ 1.0, 1.0 ]
oprhc = [ 0.0, 0.0 ]
```

6.1.2.1 Source code: demb781

[demb781.py]

```
#
#  Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#  File:    demb781.py
#
#  Purpose: Demonstrates how to solve a simple non-liner separable problem
#  using the SCopt interface for Python. Then problem is this:
#    Minimize   e^x2 + e^x3
#    Such that  e^x4 + e^x5                       <= 1
#                   x0 + x1 - x2                   = 0
#                 - x0 - x1      - x3              = 0e+00
#               0.5 x0                - x4         = 1.3862944
#                        x1                  - x5  = 0
#              x0 ... x5 are unrestricted
#
##
from __future__ import with_statement
```

```

import sys

import mosek

def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main ():
  with mosek.Env() as env:
    env.set_Stream (mosek.streamtype.log, streamprinter)
    with env.Task(0,0) as task:
      task.set_Stream (mosek.streamtype.log, streamprinter)

      numvar = 6
      numcon = 5

      bkc = [ mosek.boundkey.up,
              mosek.boundkey.fx,
              mosek.boundkey.fx,
              mosek.boundkey.fx,
              mosek.boundkey.fx ]
      blc = [ 0.0, 0.0, 0.0, 1.3862944, 0.0 ]
      buc = [ 1.0, 0.0, 0.0, 1.3862944, 0.0 ]

      bkx = [ mosek.boundkey.fr ] * numvar
      blx = [ 0.0 ] * numvar
      bux = [ 0.0 ] * numvar

      aptrb = [ 0, 0, 3, 6, 8 ]
      aptre = [ 0, 3, 6, 8, 10 ]
      asubi = [ 0, 1, 2, 3, 4 ]
      asubj = [ 0, 1, 2,
                0, 1, 3,
                0, 4,
                1, 5 ]
      aval  = [  1.0,  1.0, -1.0,
                -1.0, -1.0, -1.0,
                 0.5, -1.0,
                 1.0, -1.0 ]

      task.appendvars(numvar)
      task.appendcons(numcon)

      task.putobjsense(mosek.objsense.minimize)

      task.putvarboundslice(0, numvar, bkx, blx, bux)
      task.putconboundslice(0, numcon, bkc, blc, buc)

      task.putarowlist(asubi, aptrb, aptre, asubj, aval )

      opro  = [ mosek.scopr.exp, mosek.scopr.exp ]
      oprjo = [ 2, 3 ]
      oprfo = [ 1.0, 1.0 ]
      oprgo = [ 1.0, 1.0 ]
      oprho = [ 0.0, 0.0 ]

```

```

        oprc  = [ mosek.scopr.exp, mosek.scopr.exp ]
        opric = [ 0, 0 ]
        oprjc = [ 4, 5 ]
        oprfc = [ 1.0, 1.0 ]
        oprgc = [ 1.0, 1.0 ]
        oprhc = [ 0.0, 0.0 ]

        task.putSCeval(opro, oprjo, oprfo, oprgo, oprho,
                       oprc, opric, oprjc, oprfc, oprgc, oprhc)

        task.optimize()

        res = [ 0.0 ] * numvar
        task.getsolutionslice(
          mosek.soltype.itr,
          mosek.solitem.xx,
          0, numvar,
          res)

        print ( "Solution is: %s" % res )

main()
```

6.1.3 Ensuring convexity and differentiability

Some simple rules can be set up to ensure that the problem satisfies MOSEK's convexity and differentiability requirements. First of all, for any variable x_i used in a separable term, the variable bounds must define a range within which the function is twice differentiable.

We can define these bounds as follows:

Separable function	Operator name	Safe x bounds
$fx\mathtt{ln}(x)$	scopr.ent	$0 < x$.
fe^{gx+h}	scopr.exp	$-\infty < x < \infty$.
$f\mathtt{ln}(gx+h)$	scopr.log	If $g > 0$: $-h/g < x$.
		If $g < 0$: $x < -h/g$.
$f(x+h)^g$	scopr.pow	If $g > 0$ and integer: $-\infty < x < \infty$.
		If $g < 0$ and integer: either $-h < x$ or $x < -h$.
		Otherwise: $-h < x$.

To ensure convexity, we require that each $z_i(x)$ is either a sum of convex terms or a sum of concave terms. The following table lists convexity for the relevant ranges for $f > 0$ — changing the sign of f switches concavity/convexity.

Separable function	Operator name	
$f x \mathtt{ln}(x)$	scopr.ent	Convex within safe bounds.
$f e^{gx+h}$	scopr.exp	Convex for all x.
$f \mathtt{ln}(gx+h)$	scopr.log	Concave within safe bounds.
$f(x+h)^g$	scopr.pow	If g is even integer: convex within safe bounds.
		If g is odd integer: concave $(-\infty, -h)$, convex $(-h, \infty)$.
		If $0 < g < 1$: concave within safe bounds.
		Otherwise: convex within safe bounds.

6.1.4 SCopt Reference

Functions used to manipulate separable terms:

`(opro, oprjo, oprfo, oprgo, oprho, oprc, opric, oprjc, oprfc, oprgc, oprhc)`

Replace all current non-linear separable terms with a new set.

`opro`
List of function indicators defining the objective terms; see `scopr`.

`oprjo`
List of variable indexes for the objective terms.

`oprfo`
List of f values for the objective terms

`oprgo`
List of g values for the objective terms

`oprho`
List of h values for the objective terms

`oprc`
List of function indicators defining the constraint terms; see `scopr`.

`opric`
List of variable indexes for the constraint terms.

`oprjc`
List of constraint indexes for the constraint terms.

`oprfc`
List of f values for the constraint terms

`oprgc`
List of g values for the constraint terms

`oprhc`
List of h values for the constraint terms

`()`

Remove all non-linear separable terms from the task.

Constants used to define :

Entropy function, $fx\mathtt{ln}(x)$

Exponential function,fe^{gx+h}

Logarithm, $f\mathtt{ln}(gx+h)$

Power function, $f(x+h)^g$

Chapter 7

Advanced API tutorial

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

7.1 The progress call-back

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

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

7.1.1 Source code example

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

[callback.py]

```
##
#   Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#   File:      callback.py
#
#   Purpose:   To demonstrate how to use the progress
#              callback.
#
#              Use this script as follows:
#              callback.py psim  25fv47.mps
#              callback.py dsim  25fv47.mps
#              callback.py intpnt 25fv47.mps
```

```
#
#              The first argument tells which optimizer to use
#              i.e. psim is primal simplex, dsim is dual simplex
#              and intpnt is interior-point.
##
from __future__ import with_statement

import sys

import mosek
from mosek import *


def makeUserCallback(maxtime):
    def userCallback(caller,
                     douinf,
                     intinf,
                     lintinf):
        opttime = 0.0

        if   caller == callbackcode.begin_intpnt:
            print ("Starting interior-point optimizer")
        elif caller == callbackcode.intpnt:
            itrn    = intinf[iinfitem.intpnt_iter      ]
            pobj    = douinf[dinfitem.intpnt_primal_obj]
            dobj    = douinf[dinfitem.intpnt_dual_obj  ]
            stime   = douinf[dinfitem.intpnt_time      ]
            opttime = douinf[dinfitem.optimizer_time   ]

            print ("Iterations: %-3d" % itrn)
            print ("  Elapsed time: %6.2f(%.2f) " % (opttime,stime))
            print ("  Primal obj.: %-18.6e  Dual obj.: %-18.6e" % (pobj,dobj))
        elif caller == callbackcode.end_intpnt:
            print ("Interior-point optimizer finished.")
        elif caller == callbackcode.begin_primal_simplex:
            print ("Primal simplex optimizer started.")
        elif caller == callbackcode.update_primal_simplex:
            itrn    = intinf[iinfitem.sim_primal_iter  ]
            pobj    = douinf[dinfitem.sim_obj          ]
            stime   = douinf[dinfitem.sim_time         ]
            opttime = douinf[dinfitem.optimizer_time   ]

            print ("Iterations: %-3d" % itrn)
            print ("  Elapsed time: %6.2f(%.2f)" % (opttime,stime))
            print ("  Obj.: %-18.6e" % pobj )
        elif caller == callbackcode.end_primal_simplex:
            print ("Primal simplex optimizer finished.")
        elif caller == callbackcode.begin_dual_simplex:
            print ("Dual simplex optimizer started.")
        elif caller == callbackcode.update_dual_simplex:
            itrn    = intinf[iinfitem.sim_dual_iter    ]
            pobj    = douinf[dinfitem.sim_obj          ]
            stime   = douinf[dinfitem.sim_time         ]
            opttime = douinf[dinfitem.optimizer_time   ]
            print ("Iterations: %-3d" % itrn)
            print ("  Elapsed time: %6.2f(%.2f)" % (opttime,stime))
            print ("  Obj.: %-18.6e" % pobj)
        elif caller == callbackcode.end_dual_simplex:
```

```
            print ("Dual simplex optimizer finished.")
        elif caller == callbackcode.begin_bi:
            print ("Basis identification started.")
        elif caller == callbackcode.end_bi:
            print ("Basis identification finished.")
        else:
            pass

        if opttime >= maxtime:
            # mosek is spending too much time. Terminate it.
            return 1

        return 0
    return userCallback

def msgPrinter(msg):
    sys.stdout.write(msg)
    sys.stdout.flush()

def main(args):

  if len(args) < 3:
      print ("Too few input arguments. Syntax:")
      print ("\tcallback.py psim inputfile")
      print ("\tcallback.py dsim inputfile")
      print ("\tcallback.py intpnt inputfile")
      return

  with mosek.Env() as env:
      with mosek.Task(env) as task:
          filename = args[2]
          task.readdata(filename)

          task.set_Stream(streamtype.log, msgPrinter)

          if   args[1] == 'psim':
              task.putintparam(iparam.optimizer,optimizertype.primal_simplex)
          elif args[1] == "dsim":
              task.putintparam(iparam.optimizer,optimizertype.dual_simplex)
          elif args[1] == "intpnt":
              task.putintparam(iparam.optimizer,optimizertype.intpnt)

          # Turn all MOSEK logging off (note that errors and other messages
          # are still sent through the log stream)
          task.putintparam(iparam.log, 0)

          usercallback = makeUserCallback(maxtime = 3600)
          task.set_Progress(usercallback)

          task.optimize()

          task.solutionsummary(streamtype.msg)

if __name__ == '__main__':
    main(sys.argv)
```

7.2 Solving linear systems involving the basis matrix

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

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

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

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

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

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

$$B\bar{x} = w \tag{7.1}$$

and

$$B^T\bar{x} = w \tag{7.2}$$

each has a unique solution for all w .

MOSEK provides functions for solving the linear systems (7.1) and (7.2) for an arbitrary w-¿.

7.2.1 Identifying the basis

To use the solutions to (7.1) and (7.2) it is important to know how the basis matrix B is constructed.

Internally MOSEK employs the linear optimization problem

$$\begin{array}{lccccc} \text{maximize} & & & c^T x & & \\ \text{subject to} & & & Ax - x^c & = & 0, \\ & l^x & \leq & x & \leq & u^x, \\ & l^c & \leq & x^c & \leq & u^c. \end{array} \tag{7.3}$$

where

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

The basis matrix is constructed of m columns taken from

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

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

```
task.initbasissolve(basis)
```

where `basis` is an array of variable indexes.

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

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

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

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

then the i 'th basis variable is variable

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

and the i 'th column of B is the column

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

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

7.2.2 An example

Consider the linear optimization problem:

$$\begin{array}{lccc} \text{minimize} & x_0 + x_1 & & \\ \text{subject to} & x_0 + 2x_1 & \leq & 2, \\ & x_0 + x_1 & \leq & 6, \\ & x_0, x_1 \geq 0. & & \end{array} \tag{7.4}$$

Suppose a call to Task.initbasissolve returns an array `basis` so that

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

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

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

Please note the ordering of the columns in B .

The following program demonstrates the use of Task.solvewithbasis.

[solvebasis.py]

```
#
#  Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#  File     : solvebasis.py
#
#  Purpose  :  To demonstrate the usage of
#              MSK_solvewithbasis on the problem:
#
#              maximize  x0 + x1
#              st.
#                      x0 + 2.0 x1 <= 2
#                      x0  +    x1 <= 6
#                      x0 >= 0, x1>= 0
#
#               The problem has the slack variables
#               xc0, xc1 on the constraints
#               and the 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


import mosek

def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main():
    numcon = 2
    numvar = 2

    # Since the value infinity is never used, we define
    # 'infinity' symbolic purposes only
    infinity = 0

    c    = [1.0, 1.0]
    ptrb = [0, 2]
    ptre = [2, 3]
    asub = [0, 1,
            0, 1]
    aval = [1.0, 1.0,
            2.0, 1.0]
    bkc  = [mosek.boundkey.up,
            mosek.boundkey.up]

    blc  = [-infinity,
            -infinity]
    buc  = [2.0,
```

```
            6.0]

    bkx  = [mosek.boundkey.lo,
            mosek.boundkey.lo]
    blx  = [0.0,
            0.0]

    bux  = [+infinity,
            +infinity]
    w1 = [2.0, 6.0]
    w2 = [1.0, 0.0]
    try:
      with mosek.Env() as env:
        with env.Task(0,0) as task:
          task.set_Stream (mosek.streamtype.log, streamprinter)
          task.inputdata(numcon,numvar,
                         c,
                         0.0,
                         ptrb,
                         ptre,
                         asub,
                         aval,
                         bkc,
                         blc,
                         buc,
                         bkx,
                         blx,
                         bux)
          task.putobjsense(mosek.objsense.maximize)
          r = task.optimize()
          if r != mosek.rescode.ok:
            print ("Mosek warning:",r)

          basis = [0] * numcon
          task.initbasissolve(basis)

          #List basis variables corresponding to columns of B
          varsub = [0,1]

          for i in range(numcon):
            if basis[varsub[i]] < numcon:
              print ("Basis variable no %d is xc%d" % (i,basis[i]))
            else:
              print ("Basis variable no %d is x%d" % (i,basis[i] - numcon))

          # solve Bx = w1
          # varsub contains index of non-zeros in b.
          #  On return b contains the solution x and
          # varsub the index of the non-zeros in x.
          nz = 2

          nz = task.solvewithbasis(0, nz, varsub, w1)
          print ("nz = %s" % nz)
          print ("Solution to Bx = w1:")

          for i in range(nz):
            if basis[varsub[i]] < numcon:
              print ("xc %s = %s" % (basis[varsub[i]],w1[varsub[i]]))
```

```
            else:
              print ("x%s = %s" % (basis[varsub[i]] - numcon, w1[varsub[i]]))

          # Solve B^Tx = w2
          nz = 1
          varsub[0] = 0

          nz = task.solvewithbasis(1, nz, varsub, w2)

          print ("Solution to B^Tx = w2:")

          for i in range(nz):
            if basis[varsub[i]] < numcon:
              print ("xc %s = %s" % (basis[varsub[i]], w2[varsub[i]]))
            else:
              print ("x %s = %s" % (basis[varsub[i]] - numcon, w2[varsub[i]]) )
    except Exception as e:
      print (e)

if __name__ == '__main__':
  main()
```

In the example above the linear system is solved using the optimal basis for (7.4) and the original right-hand side of the problem. Thus the solution to the linear system is the optimal solution to the problem. When running the example program the following output is produced.

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

Solution to Bx = b:

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

Solution to B^Tx = c:

x1 = -1.000000e+00
x0 = 1.000000e+00
```

Please note that the ordering of the basis variables is

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

and thus the basis is given by:

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

It can be verified that

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

is a solution to

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

7.2.3 Solving arbitrary linear systems

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

$$Ax = b$$

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

Below we demonstrate how to solve the linear system

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{7.5}$$

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

[solvelinear.py]

```
from __future__ import with_statement
##
#  Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#  File     :  solvelinear.py
#
#  Purpose  :  To demonstrate the usage of MSK_solvewithbasis
#              when solving 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)
##

import mosek

def put_a( task,
           aval,
           asub,
           ptrb,
           ptre,
           numvar,
           basis):
```

```
    # Since the value infinity is never used, we define
    # 'infinity' symbolic purposes only
    infinity = 0

    skx = [mosek.stakey.bas] * numvar
    skc = [mosek.stakey.fix] * numvar

    task.appendvars(numvar)
    task.appendcons(numvar)


    for i in range(len(asub)):
      task.putacol(i,asub[i],aval[i])

    for i in range(numvar):
      task.putconbound(i,mosek.boundkey.fx,0.0,0.0)

    for i in range(numvar):
      task.putvarbound(i,
                       mosek.boundkey.fr,
                       -infinity,
                       infinity)

    # Define a basic solution by specifying
    # status keys for variables & constraints.

    for i in range(numvar):
      task.putsolutioni (mosek.accmode.var,
                         i,
                         mosek.soltype.bas,
                         skx[i],
                         0.0,
                         0.0,
                         0.0,
                         0.0)

    for i in range(numvar):
      task.putsolutioni (mosek.accmode.con,
                         i,
                         mosek.soltype.bas,
                         skc[i],
                         0.0,
                         0.0,
                         0.0,
                         0.0)

    task.initbasissolve(basis)



def main():
    numcon = 2
    numvar = 2

    aval = [ [ -1.0 ],
             [ 1.0, 1.0 ] ]
    asub = [ [ 1 ],
             [ 0,   1   ] ]
```

```
    ptrb = [ 0,1 ]
    ptre = [ 1,3 ]

    #int[]      bsub  = new int[numvar];
    #double[]   b     = new double[numvar];
    #int[]      basis = new int[numvar];


    with mosek.Env() as env:
      with mosek.Task(env) as task:
        # Directs the log task stream to the user specified
        # method task_msg_obj.streamCB
        task.set_Stream(mosek.streamtype.log,
                        lambda msg : sys.stdout.write(msg))
        # Put A matrix and factor A.
        # Call this function only once for a given task.

        basis = [0] * numvar
        b     = [ 0.0, -2.0 ]
        bsub  = [ 0,    1 ]

        put_a(task,
              aval,
              asub,
              ptrb,
              ptre,
              numvar,
              basis)

        # now solve rhs
        b    = [ 1, -2]
        bsub = [ 0,  1 ]
        nz = task.solvewithbasis(0,2,bsub,b)
        print("\nSolution to Bx = b:\n")

        # Print solution and show correspondents
        # to original variables in the problem
        for i in range(nz):
          if basis[bsub[i]] < numcon:
            print("This should never happen")
          else:
            print("x%d = %d" % (basis[bsub[i]] - numcon, b[bsub[i]]))

        b[0]    = 7
        bsub[0] = 0

        nz = task.solvewithbasis(0,1,bsub,b);

        print("\nSolution to Bx = b:\n")
        # Print solution and show correspondents
        # to original variables in the problem
        for i in range(nz):
          if basis[bsub[i]] < numcon:
            print ("This should never happen")
          else:
            print ("x%d = %d" % (basis[bsub[i]] - numcon, b[bsub[i]] ))
```

```
if __name__ == "__main__":
  try:
    main()
  except:
    import traceback
    traceback.print_exc()
```

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

The program produces the output:

```
Solution to Bx = b:

x1 = 1
x0 = 3

Solution to Bx = b:

x1 = 7
x0 = 7
```

and we can verify that $x_0 = 2, x_1 = -4$ is indeed a solution to (7.5).

7.3 Calling BLAS/LAPACK routines from MOSEK

Sometimes users need to perform linear algebra operations that involve dense matrices and vectors. Also MOSEK uses extensively high-performance linear algebra routines from the BLAS and LAPACK packages and some of this routine are included in the package shipped to the users.

MOSEK makes available to the user some BLAS and LAPACK routines by MOSEK functions that

- use MOSEK data types and response code;
- keep BLAS/LAPACK naming convention.

Therefore the user can leverage on efficient linear algebra routines, with a simplified interface, with no need for additional packages. In the following table we list BLAS functions:

Name	MOSEK name	Expression
AXPY	Env.axpy	$y = \alpha x + y$
DOT	Env.dot	$x^T y$
GEMV	Env.gemv	$y = \alpha A x + \beta y$
GEMM	Env.gemm	$C = \alpha AB + \beta C$
SYRK	Env.syrk	$C = \alpha AA^T + \beta C$

Function from LAPACK are listed below:

Name	MOSEK name	Description
POTRF	Env.potrf	Cholesky factorization
SYEVD	Env.syevd	Eigen-values of a symmetric matrix
SYEIG	Env.syeig	Eigen-values and eigen-vectors of a symmetric matrix

A detailed list of the available routines follows. All code snippets are taken from the example `blas-lapack` distributed with MOSEK and listed below. All code snippets assume a valid MOSEK environment named `env` is available. For more details please refer to Section 7.3.1.

Scaled Vectors Addiction (AXPY)

It computes the sum of a scaled vector x with a second vector y, i.e.

$$y = \alpha x + y, \tag{7.6}$$

where α are two scalars and $x, y \in \mathbb{R}^n$. It is available through the `Env.axpy`. This routine may use optimized loop unrolling. Note that the results overwrites y. For example, we may use the following code:

[blas_lapack.py]
```
env.axpy(n,alpha,x,y)
```

Inner Product (DOT)

Given two vectors $x, y \in \mathbb{R}^n$, it computes the inner product (or dot product) defined as

$$x^T \cdot y = \sum_{i=0}^{n-1} x_i y_i = y^T \cdot x. \tag{7.7}$$

The inner product is a special case of the generalized matrix-vector multiplication. MOSEK provide access to BLAS implementation by the `Env.dot` function.

For example we may want to perform the dot product among two arrays x, y of the same dimension we can write

[blas_lapack.py]
```
env.dot(n,x,y)
```

Generalized Matrix-Vector Multiplication (GEMV)

This function performs matrix-vector operations of the form

$$y = \alpha A x + \beta y, \tag{7.8}$$

or

$$y = \alpha A^T x + \beta y. \tag{7.9}$$

where α, β are two scalars and $A \in \mathbb{R}^{m \times n}$, Dimension of x and y must be compatible with those of A depending whether it is transpose or not. MOSEK provides access to GEMV by the

Env.gemv function. Please note that the result overwrites the vector y. Expression (7.8) can be calculated as

[blas_lapack.py]
```
env.gemv(mosek.transpose.no, m, n, alpha, A, x, beta,z)
```

Generalized Matrix-Matrix Multiplication (GEMM)

This function perform a matrix-matrix multiplication followed by an addition. Given matrices A, B and C of compatible dimensions, and two scalars α, β it performs the following

$$C = \alpha AB + \beta C. \tag{7.10}$$

Matrices A and B can be considered transposed or not, and their dimensions must be compatible accordingly.

MOSEK provides access to GEMM by the Env.gemm function. Please note that the result overwrites the matrix C.

[blas_lapack.py]
```
env.gemm(mosek.transpose.no,mosek.transpose.no,m,n,k,alpha,A,B,beta,C)
```

Symmetric rank-k update (SYRK)

Given a symmetric matrix $\in \mathbb{R}^{n\times n}$, two scalars α, β and a matrix A of rank k, this function computes either

$$C = \beta C + \alpha A^T A, \tag{7.11}$$

withfor $A \in \mathbb{R}^{k\times n}$, or

$$C = \beta C + \alpha A A^T, \tag{7.12}$$

for $A \in \mathbb{R}^{k\times n}$. The corresponding routine provided by MOSEK is Env.syrk. The matrix C only needs to be specified as triangular. Note also that the result ovewrites C in the relevant upper or lower triangular part, accordingly with the way it has been input.

[blas_lapack.py]
```
env.syrk(mosek.uplo.lo, mosek.transpose.no, n,k,alpha, A, beta,D)
```

Eigenvalue Computation (SYEIG)

This function returns the eigenvalues of a given square matrix A. MOSEK provides access to SYEIG by the Env.syeig function.

[blas_lapack.py]
```
env.syeig(mosek.uplo.lo,m,Q,v)
```

Eigenvalue Decomposition (SYEVD)

Given a symmetric matrix A, this function returns its eigenvalue decomposition, i.e. a diagonal matrix V and a lower triangular matrix U, of the same dimension as A, such that

$$A = UVU^T.$$

The diagonal of V contains the eigenvalues of A, while U is formed by the orthonormal eigenvectors of A stored column-wise. Note that U will ovewrites A. MOSEK provides access to SYEVD by the Env.syevd function.

[blas_lapack.py]

```
env.syevd(mosek.uplo.lo,m,Q,v)
```

Cholesky Factorization (POTRF)

This function computes the Cholesky factorization of a symmetric positive-definite matrix $A \in \mathbb{R}^{n\times n}$, i.e. it return a lower triangular matrix U such that

$$A = U^T U.$$

It is available through the function Env.potrf. Note that The result will overwrite the lower triangle of A.

[blas_lapack.py]

```
env.potrf(mosek.uplo.lo,m,Q)
```

7.3.1 A working example

The following code shows how to call the BLAS/LAPACK routines provided by MOSEK. The code has no practical purpose and it is only meant to show which kind of input the routines accept.

[blas_lapack.py]

```
#
# Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
# File:      blas_lapack.py
#
# Purpose: To demonstrate how to call /LAPACK routines for whose MOSEK provides simplified interfaces.
#

import mosek

with mosek.Env() as env:

    n=3
    m=2
    k=3

    alpha=2.0
    beta=0.5

```

```
    x=[1.0,1.0,1.0]
    y=[1.0,2.0,3.0]
    z=[1.0,1.0]
    v=[0.0,0.0]
    #A has m=2 rows and k=3 cols
    A=[ 1.0,1.0,2.0,2.0, 3.,3.]
    #B has k=3 rows and n=3 cols
    B=[ 1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
    C=[ 0.0 for i in range(n*m)]
    D=[ 1.0,1.0,1.0,1.0]
    Q=[ 1.0,0.0,0.0,2.0]

    xy=[]

#  routines

    env.dot(n,x,y)

    env.axpy(n,alpha,x,y)

    env.gemv(mosek.transpose.no, m, n, alpha, A, x, beta,z)

    env.gemm(mosek.transpose.no,mosek.transpose.no,m,n,k,alpha,A,B,beta,C)

    env.syrk(mosek.uplo.lo, mosek.transpose.no, n,k,alpha, A, beta,D)

# LAPACK routines

    env.potrf(mosek.uplo.lo,m,Q)

    env.syeig(mosek.uplo.lo,m,Q,v)

    env.syevd(mosek.uplo.lo,m,Q,v)
```

7.4 Automatic reformulation of QCQP problems in conic form

Despite that MOSEK can solve quadratic and quadratically constrained convex problems, as detailed in Section 5.5, it often performs better when the problems are reformulated in conic form. Moreover, the conic formulation can rely on a more sound duality theory. For this reason MOSEK provides a tool to reformulate automatically QCQP problem as Conic Quadratic problems.

We recall that QCQP problems that MOSEK can solve are of the form:

$$
\begin{array}{lllllll}
\text{minimize} & & & \frac{1}{2}x^T Q^o x + c^T x + c^f & & & \\
\text{subject to} & l_k^c & \leq & \frac{1}{2}x^T Q^k x + \sum_{j=0}^{n-1} a_{k,j} x_j & \leq & u_k^c, & k = 0, \ldots, m-1, \\
\text{subject to} & l_k^{cl} & \leq & \sum_{j=0}^{n-1} a_{k,j}^l x_j & \leq & u_k^{cl}, & k = 0, \ldots, m_l - 1, \\
& l_j^x & \leq & x_j & \leq & u_j^x, & j = 0, \ldots, n-1.
\end{array}
\tag{7.13}
$$

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

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

The reformulation is not in general unique. The approach followed in `Task.toconic` is to introduce additional variables, linear constraints and second order cones to obtain a larger but equivalent problem in which the original variables are preserved.

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

The reformulated model will contain:

- one second-order cone for each quadratic constraint,
- one secod-order cone if the objective function is quadratic,
- each quadratic constraint will contain no coefficients and upper/lower bounds will be set to $\infty, -\infty$ respectively.

It is important to notice that `Task.toconic` modified the input task in-place: this means that if the reformulation is not possible, i.e. the problem is not conic representable, the state of the task is in general undefined. The user should consider cloning the task.

7.4.1 Quadratic constraint reformulation

Let assume that the $k-$th constraint has some quadratic terms, i.e. it can be written in the form

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

First we note that either $l_k^c = -\infty$ or $l_k^c = -\infty$ must hold, otherwise either the constraint can be dropped, or the constraint is not convex. Thus

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

can be considered without loss of generality. Introducing an additional variable y_k we obtain the equivalent form

$$\begin{array}{lcl} x^T Q^k x & \leq & 2y_k, \\ \sum_{j=0}^{n-1} a_{k,j}x_j - u_k^c & = & y_k. \end{array}$$

If Q^k is positive semidefinite, we can compute its Cholesky factorization F^k and write

$$
\begin{array}{lcl}
\|F^k x\|_2 & \leq & 2y_k, \\
\sum_{j=0}^{n-1} a_{k,j}x_j - u_k^c & = & y_k.
\end{array}
$$

The first constraint defines a second-order cone of dimension, i.e.

$$
\|F^k x\|_2 \leq 2y_k \quad \Leftrightarrow (y_k, Fx) \in \mathcal{Q}_r^{2+n}.
$$

Thus, the constraint can be cast as

$$
\begin{array}{lcl}
\sum_{j=0}^{n-1} a_{k,j}x_j - u_k^c & = & y_k, \\
z & = & Fx, \\
(1, y_k, z) & \in & \mathcal{Q}_r^{2+n}.
\end{array}
$$

A similar approach is followed to deal with the case in which Q^k has exactly one negative eigenvalue. Moreover, some special cases, as such Q^k being diagonal, are taken into account.

7.4.2 Objective function reformulation

Let us assume that the objective function of problem (7.13) contains a quadratic terms, i.e. the matrix Q_o is not null.

From a logical point of view, we can introduce an additional free variable t and remove the quadratic term from the objective function, which reads

$$
x_n + a^T x + c. \tag{7.14}
$$

The next step is to introduce a quadratic constraint of the form

$$
\frac{1}{2}x^T Q_o x \leq t. \tag{7.15}
$$

where $Q_m = Q_o$. The problem has now a linear objective function, as required for any COP. The quadratic constraint can be converted as in Section 7.4.1.

In practice the transformation will not introduce any additional quadratic constraint, but a second order cone will be included along with the additional linear constraints.

Chapter 8

A case study

8.1 Portfolio optimization

8.1.1 Introduction

In this section the Markowitz portfolio optimization problem and variants are implemented using the MOSEK optimizer API.

An alternative to using the optimizer API is the Fusion API which is much simpler to use because it makes it possible to implement the model almost as stated on paper. It is not uncommon that an optimization problem can be implemented using the Fusion API in 1/10th of the time implementing it using the optimizer API. On the other hand, a well implemented model in the optimizer API will usually run faster than the same Fusion model.

Since it so fast to implement a model in Fusion it can be attractive to implement a model in Fusion first because that way the results from the Fusion based code can be used to validate the results of the optimizer API implementation.

Subsequently the following MATLAB inspired notation will be employed. The : operator is used as follows

$$i : j = \{i, i+1, \ldots, j\}$$

and hence

$$x_{2:4} = \begin{bmatrix} x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

If x and y are two column vectors, then

$$[x; y] = \begin{bmatrix} x \\ y \end{bmatrix}$$

Furthermore, if $f \in R^{m \times n}$ then

$$f(:) = \begin{bmatrix} f_{1,1} \\ f_{2,1} \\ \\ f_{m-1,n} \\ \\ f_{m,n} \end{bmatrix}$$

i.e. $f(:)$ stacks the columns of the matrix f.

8.1.2 A basic portfolio optimization model

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

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

and covariance

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

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

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

and variance (or risk)

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

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

$$\begin{array}{lllll} \text{maximize} & \mu^T x & & & \\ \text{subject to} & e^T x & = & w + e^T x^0, & \\ & x^T \Sigma x & \leq & \gamma^2, & \\ & x & \geq 0. & & \end{array} \tag{8.1}$$

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

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

Since e is the vector of all ones then

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

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

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

This leads to the first constraint

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

The second constraint

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

ensures that the variance, or the risk, is bounded by γ^2. Therefore, γ specifies an upper bound of the standard deviation the investor is willing to undertake. Finally, the constraint

$$x_j \geq 0$$

excludes the possibility of short-selling. This constraint can of course be excluded if short-selling is allowed.

The covariance matrix Σ is positive semidefinite by definition and therefore there exist a matrix G such that

$$\Sigma = GG^T. \tag{8.2}$$

In general the choice of G is **not** unique and one possible choice of G is the Cholesky factorization of Σ. However, in many cases another choice is better for efficiency reasons as discussed in Section 8.1.4.

For a given G we have that

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

Hence, we may write the risk constraint as

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

or equivalently

$$[\gamma; G^T x] \in Q^{n+1}.$$

where Q^{n+1} is the $n+1$ dimensional quadratic cone. Therefore, problem (8.1) can be written as

$$
\begin{array}{lllll}
\text{maximize} & \mu^T x & & & \\
\text{subject to} & e^T x & = & w + e^T x^0, & \\
& [\gamma; G^T x] & \in & Q^{n+1}, & \\
& x & \geq & 0, &
\end{array} \tag{8.3}
$$

which is a conic quadratic optimization problem that can easily be solved using MOSEK.

Subsequently we will use the example data

$$
\mu = \begin{bmatrix} 0.1073 \\ 0.0737 \\ 0.0627 \end{bmatrix}
$$

and

$$
\Sigma = 0.1 \begin{bmatrix} 0.2778 & 0.0387 & 0.0021 \\ 0.0387 & 0.1112 & -0.0020 \\ 0.0021 & -0.0020 & 0.0115 \end{bmatrix}
$$

This implies

$$
G^T = \sqrt{0.1} \begin{bmatrix} 0.5271 & 0.0734 & 0.0040 \\ 0 & 0.3253 & -0.0070 \\ 0 & 0 & 0.1069 \end{bmatrix}
$$

using 5 figures of accuracy. Moreover, let

$$
x^0 = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \end{bmatrix}
$$

and

$$
w = 1.0.
$$

The data has been taken from [5].

8.1.2.1 Why a conic formulation?

The problem (8.1) is a convex quadratically constrained optimization problems that can be solved directly using MOSEK, then why reformulate it as a conic quadratic optimization problem? The main reason for choosing a conic model is that it is more robust and usually leads to a shorter solution times. For instance it is not always easy to determine whether the Q matrix in (8.1) is positive semidefinite due to the presence of rounding errors. It is also very easy to make a mistake so Q becomes indefinite. These causes of problems are completely eliminated in the conic formulation.

Moreover, observe the constraint

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

is nicer than

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

for small and values of γ. For instance assume a γ of 10000 then γ^2 would 1.0e8 which introduces a scaling issue in the model. Hence, using conic formulation it is possible to work with the standard deviation instead of the variance, which usually gives rise to a better scaled model.

8.1.2.2 Implementing the portfolio model

The model (8.3) can not be implemented as stated using the MOSEK optimizer API because the API requires the problem to be on the form

$$\begin{array}{lllll} \text{maximize} & & c^T \hat{x} & & \\ \text{subject to} & l^c \leq & A\hat{x} & \leq & u^c, \\ & l^x \leq & \hat{x} & \leq & u^x, \\ & & \hat{x} \in K. & & \end{array} \tag{8.4}$$

where $\hat{x}$ is referred to as the API variable.

The first step in bringing (8.3) to the form (8.4) is the reformulation

$$\begin{array}{llll} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & G^T x - t & = & 0 \\ & [s; t] & \in & Q^{n+1}, \\ & x & \geq & 0, \\ & s & 0. & \end{array} \tag{8.5}$$

where s is an additional scalar variable and t is a n dimensional vector variable. The next step is to define a mapping of the variables

$$\hat{x} = [x; s; t] = \begin{bmatrix} x \\ s \\ t \end{bmatrix}. \tag{8.6}$$

Hence, the API variable $\hat{x}$ is concatenation of model variables x, s and t. In Table (8.1) the details of the concatenation are specified. For instance it can be seen that

$$\hat{x}_{n+2} = t_1.$$

because the offset of the t variable is $n + 2$.

Given the ordering of the variables specified by (8.6) the data should be defined as follows

Variable	Length	Offset
x	n	1
s	1	n+1
t	n	n+2

Figure 8.1: Storage layout of the $\hat{x}$ variable.

$$
\begin{array}{lcl}
c & = & \left[\begin{array}{ccc} \mu^T & 0 & 0_{n,1} \end{array} \right]^T, \\
A & = & \left[\begin{array}{ccc} e^T & 0 & 0_{n,1} \\ G^T & 0_{n,1} & -I_n \end{array} \right], \\
l^c & = & \left[\begin{array}{cc} w + e^T x^0 & 0_{1,n} \end{array} \right]^T, \\
u^c & = & \left[\begin{array}{cc} w + e^T x^0 & 0_{1,n} \end{array} \right]^T, \\
l^x & = & \left[\begin{array}{ccc} 0_{1,n} & \gamma & -\infty_{n,1} \end{array} \right]^T, \\
u^x & = & \left[\begin{array}{ccc} \infty_{n,1} & \gamma & \infty_{n,1} \end{array} \right]^T.
\end{array}
$$

The next step is to consider how the columns of A is defined. The following pseudo code

$$
\begin{array}{ll}
for & j = 1:n \\
 & \hat{x}_j = x_j \\
 & A_{1,j} = 1.0 \\
 & A_{2:(n+1),j} = G^T_{j,1:n} \\
 & \\
\hat{x}_{n+1} = s & \\
 & \\
for & j = 1:n \\
 & \hat{x}_{n+1+j} = t_j \\
 & A_{n+1+j,n+1+j} = -1.0
\end{array}
$$

show how to construct each column of A.

In the above discussion index origin 1 is employed, i.e., the first position in a vector is 1. The Python programming language employs 0 as index origin and that should be kept in mind when reading the example code.

[case_portfolio_1.py]

```
"""
  File : case_portfolio_1.py

  Copyright : Copyright (c) MOSEK ApS, Denmark. All rights reserved.

  Description :  Implements a basic portfolio optimization model.
"""

import mosek

try:
    from numpy import zeros
except ImportError:
    from mosek.array import zeros
```

```

def streamprinter(text):
    print("%s" % text),

if __name__ == '__main__':

  n     = 3
  gamma = 0.05
  mu    = [0.1073,  0.0737,  0.0627]
  GT    = [[0.1667,  0.0232,  0.0013],
           [0.0000,  0.1033, -0.0022],
           [0.0000,  0.0000,  0.0338]]
  x0    = [0.0, 0.0, 0.0]
  w     = 1.0

  inf   = 0.0 # This value has no significance

  with mosek.Env() as env:
      with env.Task(0,0) as task:
          task.set_Stream(mosek.streamtype.log,streamprinter)

          rtemp = w
          for j in range(0,n):
              rtemp += x0[j]

          # Constraints.
          task.appendcons(1+n)
          task.putconbound(0,mosek.boundkey.fx,rtemp,rtemp)
          task.putconname(0,"budget")

          task.putconboundlist(range(1+0,1+n),n*[mosek.boundkey.fx],n*[0.0],n*[0.0])
          for j in range(1,1+n) :
              task.putconname(j,"GT[%d]" % j)

          # Variables.
          task.appendvars(1+2*n)

          # Offset of variables into the API variable.
          offsetx = 0
          offsets = n
          offsett = n+1

          # x variables.
          task.putclist(range(offsetx+0,offsetx+n),mu)
          task.putaijlist(n*[0],range(offsetx+0,offsetx+n),n*[1.0])
          for j in range(0,n):
              task.putaijlist(n*[1+j],range(offsetx+0,offsetx+n),GT[j])

          task.putvarboundlist(range(offsetx+0,offsetx+n),n*[mosek.boundkey.lo],n*[0.0],n*[inf])
          for j in range(0,n):
              task.putvarname(offsetx+j,"x[%d]" % (1+j))

          # s variable.
          task.putvarbound(offsets+0,mosek.boundkey.fx,gamma,gamma)
          task.putvarname(offsets+0,"s")

          # t variables.
          task.putaijlist(range(1,n+1),range(offsett+0,offsett+n),n*[-1.0])
```

```
        task.putvarboundlist(range(offsett+0,offsett+n),n*[mosek.boundkey.fr],n*[-inf],n*[inf])
        for j in range(0,n):
            task.putvarname(offsett+j,"t[%d]" % (1+j))

        task.appendcone(mosek.conetype.quad,0.0,[offsets] + range(offsett,offsett+n))
        task.putconename(0,"stddev")

        task.putobjsense(mosek.objsense.maximize)

        # Turn all log output off.
        task.putintparam(mosek.iparam.log,1)

        # Dump the problem to a human readable OPF file.
        #task.writedata("dump.opf")

        task.optimize()

        # Display the solution summary for quick inspection of results.
        task.solutionsummary(mosek.streamtype.msg)

        expret = 0.0
        x      = zeros(n,float)
        task.getxxslice(mosek.soltype.itr,offsetx+0,offsetx+n,x)
        for j in range(0,n):
            expret += mu[j]*x[j]

        stddev = zeros(1,float)
        task.getxxslice(mosek.soltype.itr,offsets+0,offsets+1,stddev)

        print("\nExpected return %e for gamma %e\n" % (expret,stddev[0]))
```

The above code produce the result

```
Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 7.4766497707e-002   Viol.  con: 2e-008   var: 0e+000   cones: 3e-009
  Dual.    obj: 7.4766522618e-002   Viol.  con: 0e+000   var: 4e-008   cones: 0e+000

Expected return 7.476650e-02 for gamma 5.000000e-02
```

The source code should be self-explanatory but a few comments are nevertheless in place. In the lines

[case_portfolio_1.py]

```
# Offset of variables into the API variable.
offsetx = 0
offsets = n
offsett = n+1
```

offsets into the MOSEK API variables are stored and those offsets are used later. The code

[case_portfolio_1.py]

```
task.putclist(range(offsetx+0,offsetx+n),mu)
task.putaijlist(n*[0],range(offsetx+0,offsetx+n),n*[1.0])
for j in range(0,n):
    task.putaijlist(n*[1+j],range(offsetx+0,offsetx+n),GT[j])

```

```
task.putvarboundlist(range(offsetx+0,offsetx+n),n*[mosek.boundkey.lo],n*[0.0],n*[inf])
for j in range(0,n):
    task.putvarname(offsetx+j,"x[%d]" % (1+j))
```

sets up the data for x variables. For instance

[case_portfolio_1.py]

```
task.putclist(range(offsetx+0,offsetx+n),mu)
```

inputs the objective coefficients for the x variables. Moreover, the code

[case_portfolio_1.py]

```
for j in range(0,n):
    task.putvarname(offsetx+j,"x[%d]" % (1+j))
```

assigns meaningful names to the API variables. This is not needed but it makes debugging easier.

8.1.2.3 Debugging tips

Implementing an optimization model in optimizer can be cumbersome and error-prone and it is very easy to make mistakes. In order to check the implemented code for mistakes it is very useful to dump the problem to a file in a human readable form for visual inspection. The line

[case_portfolio_1.py]

```
#task.writedata("dump.opf")
```

does that and this will produce a file with the content

```
[comment]
   Written by MOSEK version 7.0.0.86
   Date 01-10-13
   Time 08:15:47
[/comment]

[hints]
  [hint NUMVAR] 7 [/hint]
  [hint NUMCON] 4 [/hint]
  [hint NUMANZ] 12 [/hint]
  [hint NUMQNZ] 0 [/hint]
  [hint NUMCONE] 1 [/hint]
[/hints]

[variables disallow_new_variables]
  'x[1]' 'x[2]' 'x[3]' s 't[1]'
  't[2]' 't[3]'
[/variables]

[objective maximize]
   1.073e-001 'x[1]' + 7.37e-002 'x[2]' + 6.270000000000001e-002 'x[3]'
[/objective]

[constraints]
  [con 'budget']  'x[1]' + 'x[2]' + 'x[3]' = 1e+000 [/con]
```

```
  [con 'GT[1]'] 1.667e-001 'x[1]' + 2.32e-002 'x[2]' + 1.3e-003 'x[3]' - 't[1]' = 0e+000 [/con]
  [con 'GT[2]'] 1.033e-001 'x[2]' - 2.2e-003 'x[3]' - 't[2]' = 0e+000 [/con]
  [con 'GT[3]'] 3.38e-002 'x[3]' - 't[3]' = 0e+000 [/con]
[/constraints]

[bounds]
  [b]         0 <= * [/b]
  [b]              s =  5e-002 [/b]
  [b]              't[1]','t[2]','t[3]' free [/b]
  [cone quad 'stddev'] s, 't[1]', 't[2]', 't[3]' [/cone]
[/bounds]
```

Observe that since the API variables have been given meaningful names it is easy to see the model is correct.

8.1.3 The efficient frontier

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

Given a nonnegative α then the problem

$$\begin{array}{lllll} \text{maximize} & \mu^T x - \alpha s & & \\ \text{subject to} & e^T x & = & w + e^T x^0, \\ & [s; G^T x] & \in & Q^{n+1}, \\ & x & \geq & 0. \end{array} \tag{8.7}$$

computes efficient portfolios. Note that the objective maximizes the expected return while maximizing $-\alpha$ times the standard deviation. Hence, the standard deviation is minimized while α specifies the tradeoff between expected return and risk.

Ideally the problem 8.7 should be solved for all values $\alpha \geq 0$ but in practice that is computationally too costly.

Using the example data from Section 8.1.2, the optimal values of return and risk for several αs are listed below:

```
Expected return 1.073000e-01 for gamma 7.261311e-01
Expected return 1.032557e-01 for gamma 1.499440e-01
Expected return 6.975524e-02 for gamma 3.735435e-02
Expected return 6.766068e-02 for gamma 3.382809e-02
Expected return 6.679238e-02 for gamma 3.281319e-02
Expected return 6.598822e-02 for gamma 3.214199e-02
Expected return 6.560055e-02 for gamma 3.191601e-02
Expected return 6.537354e-02 for gamma 3.181398e-02
Expected return 6.522238e-02 for gamma 3.175861e-02
Expected return 6.511552e-02 for gamma 3.172556e-02
Expected return 6.503462e-02 for gamma 3.170391e-02
Expected return 6.497237e-02 for gamma 3.168923e-02
```

8.1.3.1 Example code

The following example code demonstrates how to compute the efficient portfolios for several values of α.

[case_portfolio_2.py]

```
"""
  File : case_portfolio_2.py

  Copyright : Copyright (c) MOSEK ApS, Denmark. All rights reserved.

  Description :  Implements a basic portfolio optimization model.
"""

import mosek

try:
    from numpy import zeros
except ImportError:
    from mosek.array import zeros

def streamprinter(text):
    print("%s" % text),

if __name__ == '__main__':

  n      = 3
  gamma  = 0.05
  mu     = [0.1073,  0.0737,  0.0627]
  GT     = [[0.1667,  0.0232,  0.0013],
            [0.0000,  0.1033, -0.0022],
            [0.0000,  0.0000,  0.0338]]
  x0     = [0.0, 0.0, 0.0]
  w      = 1.0
  alphas = [0.0, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5]

  inf    = 0.0 # This value has no significance

  with mosek.Env() as env:
      with env.Task(0,0) as task:
          task.set_Stream(mosek.streamtype.log,streamprinter)

          rtemp = w
          for j in range(0,n):
              rtemp += x0[j]

          # Constraints.
          task.appendcons(1+n)
          task.putconbound(0,mosek.boundkey.fx,rtemp,rtemp)
          task.putconname(0,"budget")

          task.putconboundlist(range(1+0,1+n),n*[mosek.boundkey.fx],n*[0.0],n*[0.0])
          for j in range(1,1+n) :
              task.putconname(j,"GT[%d]" % j)

          # Variables.
          task.appendvars(1+2*n)

```

```
        offsetx = 0   # Offset of variable x into the API variable.
        offsets = n   # Offset of variable x into the API variable.
        offsett = n+1 # Offset of variable t into the API variable.

        # x variables.
        task.putclist(range(offsetx+0,offsetx+n),mu)
        task.putaijlist(n*[0],range(offsetx+0,offsetx+n),n*[1.0])
        for j in range(0,n):
            task.putaijlist(n*[1+j],range(offsetx+0,offsetx+n),GT[j])

        task.putvarboundlist(range(offsetx+0,offsetx+n),n*[mosek.boundkey.lo],n*[0.0],n*[inf])
        for j in range(0,n):
            task.putvarname(offsetx+j,"x[%d]" % (1+j))

        # s variable.
        task.putvarbound(offsets+0,mosek.boundkey.fr,gamma,gamma)
        task.putvarname(offsets+0,"s")

        # t variables.
        task.putaijlist(range(1,n+1),range(offsett+0,offsett+n),n*[-1.0])
        task.putvarboundlist(range(offsett+0,offsett+n),n*[mosek.boundkey.fr],n*[-inf],n*[inf])
        for j in range(0,n):
            task.putvarname(offsett+j,"t[%d]" % (1+j))

        task.appendcone(mosek.conetype.quad,0.0,[offsets] + range(offsett,offsett+n))
        task.putconename(0,"stddev")

        task.putobjsense(mosek.objsense.maximize)

        # Turn all log output off.
        task.putintparam(mosek.iparam.log,0)

        for alpha in alphas:
            # Dump the problem to a human readable OPF file.
            #task.writedata("dump.opf")

            task.putcj(offsets+0,-alpha);

            task.optimize()

            # Display the solution summary for quick inspection of results.
            # task.solutionsummary(mosek.streamtype.msg)

            solsta = task.getsolsta(mosek.soltype.itr)

            if solsta in [mosek.solsta.optimal, mosek.solsta.near_optimal]:
                expret = 0.0
                x      = zeros(n,float)
                task.getxxslice(mosek.soltype.itr,offsetx+0,offsetx+n,x)
                for j in range(0,n):
                    expret += mu[j]*x[j]

                stddev = zeros(1,float)
                task.getxxslice(mosek.soltype.itr,offsets+0,offsets+1,stddev)

                print("\nExpected return %e for gamma %e" % (expret,stddev[0])),
            else:
```

```
                print("An error occurred when solving for alpha=%e\n" % alpha)
```

8.1.4 Improving the computational efficiency

In practice it is often important to solve the portfolio problem in a short amount of time; this section it is discusses what can be done at the modelling stage to improve the computational efficiency.

The computational cost is of course to some extent dependent on the number of constraints and variables in the optimization problem. However, in practice a more important factor is the number nonzeros used to represent the problem. Indeed it is often better to focus at the number of nonzeros in G (see (8.2)) and try to reduce that number by for instance changing the choice of G.

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

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

where D is positive definite diagonal matrix. Moreover, V is a matrix with n rows and p columns. Such a model for the covariance matrix is called a factor model and usually p is much smaller than n. In practice p tends be a small number say less than 100 independent of n.

One possible choice for G is the Cholesky factorization of Σ which requires storage proportional to $n(n+1)/2$. However, another choice is

$$G^T = \begin{bmatrix} D^{1/2} \\ V^T \end{bmatrix}$$

because then

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

This choice requires storage proportional to $n + pn$ which is much less than for the Cholesky choice of G. Indeed assuming p is a constant then the difference in storage requirements is a factor of n.

The example above exploits the so-called factor structure and demonstrates that an alternative choice of G may lead to a significant reduction in the amount of storage used to represent the problem. This will in most cases also lead to a significant reduction in the solution time.

The lesson to be learned is that it is important to investigate how the covariance is formed. Given this knowledge it might be possible to make a special choice for G that helps reducing the storage requirements and enhance the computational efficiency.

8.1.5 Slippage cost

The basic Markowitz portfolio model assumes that there are no costs associated with trading the assets and that the returns of the assets is independent of the amount traded. None of those assumptions are usually valid in practice. Therefore, a more realistic model is

$$
\begin{array}{lllll}
\text{maximize} & \mu^T x & & & \\
\text{subject to} & e^T x + \sum_{j=1}^{n} C_j(x_j - x_j^0) & = & w + e^T x^0, & \\
& x^T \Sigma x & \leq & \gamma^2, & \\
& x & \geq & 0,
\end{array}
\tag{8.8}
$$

where the function

$$
C_j(x_j - x_j^0)
$$

specifies the transaction costs when the holding of asset j is changed from its initial value.

8.1.5.1 Market impact costs

If the initial wealth is fairly small and short selling is not allowed, then the holdings will be small. Therefore, the amount traded of each asset must also be small. Hence, it is reasonable to assume that the prices of the assets is independent of the amount traded. However, if a large volume of an assert is sold or purchased it can be expected that the price change and hence the expected return also change. This effect is called market impact costs. It is common to assume that market impact costs for asset j can be modelled by

$$
m_j \sqrt{|x_j - x_j^0|}
$$

where m_j is a constant that is estimated in some way. See [6][p. 452] for details. To summarize then

$$
C_j(x_j - x_j^0) = m_j |x_j - x_j^0| \sqrt{|x_j - x_j^0|} = m_j |x_j - x_j^0|^{3/2}.
$$

From [7] it is known

$$
\{(c, z) : c \geq z^{3/2}, z \geq 0\} = \{(c, z) : [v; c; z], [z; 1/8; v] \in Q_r^3\}
$$

where Q_r^3 is the 3 dimensional rotated quadratic cone implying

$$
\begin{array}{lll}
z_j & = & |x_j - x_j^0|, \\
{[v_j; c_j; z_j], [z_j; 1/8; v_j]} & \in & Q_r^3, \\
\sum_{j=1}^{n} C_j(x_j - x_j^0) & = & \sum_{j=1}^{n} c_j.
\end{array}
$$

Unfortunately this set of constraints is nonconvex due to the constraint

$$
z_j = |x_j - x_j^0| \tag{8.9}
$$

but in many cases that constraint can safely be replaced by the relaxed constraint

$$z_j \geq |x_j - x_j^0| \tag{8.10}$$

which is convex. If for instance the universe of assets contains a risk free asset with a positive return then

$$z_j > |x_j - x_j^0| \tag{8.11}$$

cannot hold for an optimal solution because that would imply the solution is not optimal.

Now assume that the optimal solution has the property that (8.11) holds then the market impact cost within the model is larger than the true market impact cost and hence money are essentially considered garbage and removed by generating transaction costs. This may happen if a portfolio with very small risk is requested because then the only way to obtain a small risk is to get rid of some of the assets by generating transaction costs. Here it is assumed this is not the case and hence the models (8.9) and (8.10) are equivalent.

Formula (8.10) is replaced by constraints

$$\begin{array}{rcl} z_j & \geq & x_j - x_j^0, \\ z_j & \geq & -(x_j - x_j^0). \end{array} \tag{8.12}$$

Now we have

$$\begin{array}{lllll} \text{maximize} & \mu^T x & & & \\ \text{subject to} & e^T x + m^T c & = & w + e^T x^0, & \\ & z_j & \geq & x_j - x_j^0, & j = 1, \ldots, n, \\ & z_j & \geq & x_j^0 - x_j, & j = 1, \ldots, n, \\ & [\gamma; G^T x] & \in & Q^{n+1}, & \\ & [v_j; c_j; z_j] & \in & Q_r^3, & j = 1, \ldots, n, \\ & [z_j; 1/8; v_j] & \in & Q_r^3, & j = 1, \ldots, n, \\ & x & \geq & 0. & \end{array} \tag{8.13}$$

The revised budget constraint

$$e^T x = w + e^T x^0 - m^T c$$

specifies that the total investment must be equal to the initial wealth minus the transaction costs. Moreover, observe the variables v and z are some auxiliary variables that model the market impact cost. Indeed it holds

$$z_j \geq |x_j - x_j^0|$$

and

$$c_j \geq z_j^{3/2}.$$

Before proceeding it should be mentioned that transaction costs of the form

$$c_j \geq z_j^{p/q}$$

where p and q are both integers and $p \geq q$ can be modelled using quadratic cones. See [7] for details.

One more reformulation of (8.13) is needed,

$$\begin{array}{lllll}
\text{maximize} & \mu^T x & & & \\
\text{subject to} & e^T x + m^T c & = & w + e^T x^0, & \\
 & G^T x - t & = & 0, & \\
 & z_j - x_j & \geq & -x_j^0, & j = 1, \ldots, n, \\
 & z_j + x_j & \geq & x_j^0, & j = 1, \ldots, n, \\
 & [v_j; c_j; z_j] - f_{j,1:3} & = & 0, & j = 1, \ldots, n, \\
 & [z_j; 0; v_j] - g_{j,1:3} & = & [0; -1/8; 0], & j = 1, \ldots, n, \\
 & [s; t] & \in & Q^{n+1}, & \\
 & f_{j,1:3}^T & \in & Q_r^3, & j = 1, \ldots, n, \\
 & g_{j,1:3}^T & \in & Q_r^3, & j = 1, \ldots, n, \\
 & x & \geq & 0, & \\
 & s & = & \gamma, &
\end{array} \tag{8.14}$$

where $f, g \in R^{n \times 3}$. These additional variables f and g are only introduced to bring the problem on the API standard form.

The formulation (8.14) is not the most compact possible. However, the MOSEK presolve will automatically make it more compact and since it is easier to implement (8.14) than a more compact form then the form (8.14) is preferred.

The first step in developing the optimizer API implementation is to chose an ordering of the variables. In this case the ordering

$$\hat{x} = \begin{bmatrix} x \\ s \\ t \\ c \\ v \\ z \\ f^T(:) \\ g^T(:) \end{bmatrix}$$

will be used. Note $f^T(:)$ means the rows of f are transposed and stacked on top of each other to form a long column vector. The Table 8.2 shows the mapping between the $\hat{x}$ and the model variables.

The next step is to consider how the columns of A is defined. Reusing the idea in Section 8.1.2 then the following pseudo code describes the setup of A.

Variable	Length	Offset
x	n	1
s	1	n+1
t	n	n+2
c	n	2n+2
v	n	3n+2
z	n	4n+2
$f(:)^T$	3n	7n+2
$g(:)^T$	3n	10n+2

Figure 8.2: Storage layout for the $\hat{x}$

$$
\begin{array}{ll}
for & j = 1:n \\
 & \hat{x}_j = x_j \\
 & A_{1,j} = 1.0 \\
 & A_{2:n+1,j} = G^T_{j,1:n} \\
 & A_{n+1+j,j} = -1.0 \\
 & A_{2n+1+j,j} = 1.0
\end{array}
$$

$$\hat{x}_{n+1} = s$$

$$
\begin{array}{ll}
for & j = 1:n \\
 & \hat{x}_{n+1+j} = t_j \\
 & A_{1+j,n+1+j} = -1.0
\end{array}
$$

$$
\begin{array}{ll}
for & j = 1:n \\
 & \hat{x}_{2n+1+j} = c_j \\
 & A_{1,2n+1+j} = m_j \\
 & A_{3n+1+3(j-1)+2,2n+1+j} = 1.0
\end{array}
$$

$$
\begin{array}{ll}
for & j = 1:n \\
 & \hat{x}_{3n+1+j} = v_j \\
 & A_{3n+1+3(j-1)+1,3n+1+j} = 1.0 \\
 & A_{6n+1+3(j-1)+3,3n+1+j} = 1.0
\end{array}
$$

$$
\begin{array}{ll}
for & j = 1:n \\
 & \hat{x}_{4n+1+j} = z_j \\
 & A_{1+n+j,4n+1+j} = 1.0 \\
 & A_{1+2n+j,4n+1+j} = 1.0 \\
 & A_{3n+1+3(j-1)+3,4n+1+j} = 1.0 \\
 & A_{6n+1+3(j-1)+1,4n+1+j} = 1.0
\end{array}
$$

$$
\begin{array}{ll}
for & j = 1:n \\
 & \hat{x}_{7n+1+3(j-1)+1} = f_{j,1} \\
 & A_{3n+1+3(j-1)+1,7n+(3(j-1)+1} = -1.0 \\
 & \hat{x}_{7n+1+3(j-1)+2} = f_{j,2} \\
 & A_{3n+1+3(j-1)+2,7n+(3(j-1)+2} = -1.0 \\
 & \hat{x}_{7n+1+3(j-1)+3} = f_{j,3} \\
 & A_{3n+1+3(j-1)+3,7n+(3(j-1)+3} = -1.0
\end{array}
$$

$$
\begin{array}{ll}
for & j = 1:n \\
 & \hat{x}_{10n+1+3(j-1)+1} = g_{j,1} \\
 & A_{6n+1+3(j-1)+1,7n+(3(j-1)+1} = -1.0 \\
 & \hat{x}_{10n+1+3(j-1)+2} = g_{j,2} \\
 & A_{6n+1+3(j-1)+2,7n+(3(j-1)+2} = -1.0 \\
 & \hat{x}_{10n+1+3(j-1)+3} = g_{j,3} \\
 & A_{6n+1+3(j-1)+3,7n+(3(j-1)+3} = -1.0
\end{array}
$$

The following example code demonstrates how to implement the model (8.14).

case_portfolio_3.py

```
"""
  File : case_portfolio_3.py

  Copyright : Copyright (c) MOSEK ApS, Denmark. All rights reserved.

  Description :  Implements a basic portfolio optimization model.
"""

import mosek

try:
    from numpy import zeros
except ImportError:
    from mosek.array import zeros

def streamprinter(text):
    print("%s" % text),

if __name__ == '__main__':

  n     = 3
  gamma = 0.05
  mu    = [0.1073,  0.0737,  0.0627]
  GT    = [[0.1667,  0.0232,  0.0013],
           [0.0000,  0.1033, -0.0022],
           [0.0000,  0.0000,  0.0338]]
  x0    = [0.0, 0.0, 0.0]
  w     = 1.0
  m     = [0.01, 0.01, 0.01]

  # This value has no significance.
  inf   = 0.0

  with mosek.Env() as env:
      with env.Task(0,0) as task:
          task.set_Stream(mosek.streamtype.log,streamprinter)

          rtemp = w
          for j in range(0,n):
              rtemp += x0[j]

          # Constraints.
          task.appendcons(1+9*n)
          task.putconbound(0,mosek.boundkey.fx,rtemp,rtemp)
          task.putconname(0,"budget")

          task.putconboundlist(range(1+0,1+n),n*[mosek.boundkey.fx],n*[0.0],n*[0.0])
          for j in range(1,1+n) :
              task.putconname(j,"GT[%d]" % j)

          task.putconboundlist(range(1+n,1+2*n),n*[mosek.boundkey.lo],[-x0[j] for j in range(0,n)],n*[inf])
          for i in range(0,n):
              task.putconname(1+n+i,"zabs1[%d]" % (1+i))

          task.putconboundlist(range(1+2*n,1+3*n),n*[mosek.boundkey.lo],x0,n*[inf])
```

```
        for i in range(0,n):
            task.putconname(1+2*n+i,"zabs2[%d]" % (1+i))

        task.putconboundlist(range(1+3*n,1+3*n+3*n),3*n*[mosek.boundkey.fx],3*n*[0.],3*n*[0.0])
        for i in range(0,n):
            for k in range(0,n):
                task.putconname(1+3*n+3*i+k,"f[%d,%d]" % (1+i,1+k))

        task.putconboundlist(range(1+6*n,1+9*n),3*n*[mosek.boundkey.fx],
                             3*[0.0, -1.0/8.0, 0.0],3*[0.0, -1.0/8.0, 0.0])
        for i in range(0,n) :
            for k in range(0,n):
                task.putconname(1+6*n+3*i+k,"g[%d,%d]" % (1+i,1+k))

        # Offset of variables into the API variable.
        offsetx = 0
        offsets = n
        offsett = n+1
        offsetc = 2*n+1
        offsetv = 3*n+1
        offsetz = 4*n+1
        offsetf = 5*n+1
        offsetg = 8*n+1

        # Variables.
        task.appendvars(1+11*n)

        # x variables.
        task.putclist(range(offsetx+0,offsetx+n),mu)
        task.putaijlist(n*[0],range(offsetx+0,offsetx+n),n*[1.0])
        for j in range(0,n):
            task.putaijlist(n*[1+j],range(offsetx+0,offsetx+n),GT[j])
            task.putaij(1+n+j,offsetx+j,-1.0)
            task.putaij(1+2*n+j,offsetx+j,1.0)

        task.putvarboundlist(range(offsetx+0,offsetx+n),n*[mosek.boundkey.lo],n*[0.0],n*[inf])
        for j in range(0,n):
            task.putvarname(offsetx+j,"x[%d]" % (1+j))

        # s variable.
        task.putvarbound(offsets+0,mosek.boundkey.fx,gamma,gamma)
        task.putvarname(offsets+0,"s")

        # t variables.
        task.putaijlist(range(1,n+1),range(offsett+0,offsett+n),n*[-1.0])
        task.putvarboundlist(range(offsett+0,offsett+n),n*[mosek.boundkey.fr],n*[-inf],n*[inf])
        for j in range(0,n):
            task.putvarname(offsett+j,"t[%d]" % (1+j))

        # c variables.
        task.putaijlist(n*[0],range(offsetc,offsetc+n),m)
        task.putaijlist(range(1+3*n+1,1+6*n+1,3),range(offsetc,offsetc+n),n*[1.0])
        task.putvarboundlist(range(offsetc,offsetc+n),n*[mosek.boundkey.fr],n*[-inf],n*[inf])
        for j in range(0,n):
            task.putvarname(offsetc+j,"c[%d]" % (1+j))

        # v variables.
        task.putaijlist(range(1+3*n+0,1+6*n+0,3),range(offsetv,offsetv+n),n*[1.0])
```

```
        task.putaijlist(range(1+6*n+2,1+9*n+2,3),range(offsetv,offsetv+n),n*[1.0])
        task.putvarboundlist(range(offsetv,offsetv+n),n*[mosek.boundkey.fr],n*[-inf],n*[inf])
        for j in range(0,n):
            task.putvarname(offsetv+j,"v[%d]" % (1+j))

        # z variables.
        task.putaijlist(range(1+1*n,1+2*n),range(offsetz,offsetz+n),n*[1.0])
        task.putaijlist(range(1+2*n,1+3*n),range(offsetz,offsetz+n),n*[1.0])
        task.putaijlist(range(1+3*n+2,1+6*n+2,3),range(offsetz,offsetz+n),n*[1.0])
        task.putaijlist(range(1+6*n+0,1+9*n+0,3),range(offsetz,offsetz+n),n*[1.0])
        task.putvarboundlist(range(offsetz,offsetz+n),n*[mosek.boundkey.fr],n*[-inf],n*[inf])
        for j in range(0,n):
            task.putvarname(offsetz+j,"z[%d]" % (1+j))

        # f variables.
        for j in range(0,n):
            for k in range(0,n):
                task.putaij(1+3*n+3*j+k,offsetf+3*j+k,-1.0)
                task.putvarbound(offsetf+3*j+k,mosek.boundkey.fr,-inf,inf)
                task.putvarname(offsetf+3*j+k,"f[%d,%d]" % (1+j,1+k))

        # g variables.
        for j in range(0,n):
            for k in range(0,n):
                task.putaij(1+6*n+3*j+k,offsetg+3*j+k,-1.0)
                task.putvarbound(offsetg+3*j+k,mosek.boundkey.fr,-inf,inf)
                task.putvarname(offsetg+3*j+k,"g[%d,%d]" % (1+j,1+k))

        task.appendcone(mosek.conetype.quad,0.0,[offsets] + range(offsett,offsett+n))
        task.putconename(0,"stddev")

        for k in range(0,n):
            task.appendconeseq(mosek.conetype.rquad,0.0,3,offsetf+3*k)
            task.putconename(1+k,"f[%d]" % (1+k))

        for k in range(0,n):
            task.appendconeseq(mosek.conetype.rquad,0.0,3,offsetg+3*k)
            task.putconename(1+n+k,"g[%d]" % (1+k))

        task.putobjsense(mosek.objsense.maximize)

        # Turn all log output off.
        # task.putintparam(mosek.iparam.log,0)

        # Dump the problem to a human readable OPF file.
        #task.writedata("dump.opf")

        task.optimize()

        # Display the solution summary for quick inspection of results.
        task.solutionsummary(mosek.streamtype.msg)

        expret = 0.0
        x      = zeros(n,float)
        task.getxxslice(mosek.soltype.itr,offsetx+0,offsetx+n,x)
        for j in range(0,n):
            expret += mu[j]*x[j]
```

```
            stddev = zeros(1,float)
            task.getxxslice(mosek.soltype.itr,offsets+0,offsets+1,stddev)

            print("\nExpected return %e for gamma %e\n" % (expret,stddev[0]))
```

The example code above produces the result

```
Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 7.4390660228e-002   Viol.  con: 2e-007   var: 0e+000   cones: 1e-009
  Dual.    obj: 7.4390669047e-002   Viol.  con: 1e-008   var: 1e-008   cones: 0e+000

Expected return 7.439066e-02 for gamma 5.000000e-02
```

If the problem is dumped to an OPF formatted file, then it has the following content.

```
[comment]
   Written by MOSEK version 7.0.0.86
   Date 01-10-13
   Time 07:47:34
[/comment]

[hints]
  [hint NUMVAR] 34 [/hint]
  [hint NUMCON] 28 [/hint]
  [hint NUMANZ] 60 [/hint]
  [hint NUMQNZ] 0 [/hint]
  [hint NUMCONE] 7 [/hint]
[/hints]

[variables disallow_new_variables]
  'x[1]' 'x[2]' 'x[3]' s 't[1]'
  't[2]' 't[3]' 'c[1]' 'c[2]' 'c[3]'
  'v[1]' 'v[2]' 'v[3]' 'z[1]' 'z[2]'
  'z[3]' 'f[1,1]' 'f[1,2]' 'f[1,3]' 'f[2,1]'
  'f[2,2]' 'f[2,3]' 'f[3,1]' 'f[3,2]' 'f[3,3]'
  'g[1,1]' 'g[1,2]' 'g[1,3]' 'g[2,1]' 'g[2,2]'
  'g[2,3]' 'g[3,1]' 'g[3,2]' 'g[3,3]'
[/variables]

[objective maximize]
   1.073e-001 'x[1]' + 7.37e-002 'x[2]' + 6.270000000000001e-002 'x[3]'
[/objective]

[constraints]
  [con 'budget']  'x[1]' + 'x[2]' + 'x[3]' + 1e-002 'c[1]' + 1e-002 'c[2]'
     + 1e-002 'c[3]' = 1e+000 [/con]
  [con 'GT[1]']  1.667e-001 'x[1]' + 2.32e-002 'x[2]' + 1.3e-003 'x[3]' - 't[1]' = 0e+000 [/con]
  [con 'GT[2]']  1.033e-001 'x[2]' - 2.2e-003 'x[3]' - 't[2]' = 0e+000 [/con]
  [con 'GT[3]']  3.38e-002 'x[3]' - 't[3]' = 0e+000 [/con]
  [con 'zabs1[1]'] 0e+000 <= - 'x[1]' + 'z[1]' [/con]
  [con 'zabs1[2]'] 0e+000 <= - 'x[2]' + 'z[2]' [/con]
  [con 'zabs1[3]'] 0e+000 <= - 'x[3]' + 'z[3]' [/con]
  [con 'zabs2[1]'] 0e+000 <= 'x[1]' + 'z[1]' [/con]
  [con 'zabs2[2]'] 0e+000 <= 'x[2]' + 'z[2]' [/con]
  [con 'zabs2[3]'] 0e+000 <= 'x[3]' + 'z[3]' [/con]
  [con 'f[1,1]']  'v[1]' - 'f[1,1]' = 0e+000 [/con]
```

```
  [con 'f[1,2]']  'c[1]' - 'f[1,2]' = 0e+000 [/con]
  [con 'f[1,3]']  'z[1]' - 'f[1,3]' = 0e+000 [/con]
  [con 'f[2,1]']  'v[2]' - 'f[2,1]' = 0e+000 [/con]
  [con 'f[2,2]']  'c[2]' - 'f[2,2]' = 0e+000 [/con]
  [con 'f[2,3]']  'z[2]' - 'f[2,3]' = 0e+000 [/con]
  [con 'f[3,1]']  'v[3]' - 'f[3,1]' = 0e+000 [/con]
  [con 'f[3,2]']  'c[3]' - 'f[3,2]' = 0e+000 [/con]
  [con 'f[3,3]']  'z[3]' - 'f[3,3]' = 0e+000 [/con]
  [con 'g[1,1]']  'z[1]' - 'g[1,1]' = 0e+000 [/con]
  [con 'g[1,2]']  - 'g[1,2]' = -1.25e-001 [/con]
  [con 'g[1,3]']  'v[1]' - 'g[1,3]' = 0e+000 [/con]
  [con 'g[2,1]']  'z[2]' - 'g[2,1]' = 0e+000 [/con]
  [con 'g[2,2]']  - 'g[2,2]' = -1.25e-001 [/con]
  [con 'g[2,3]']  'v[2]' - 'g[2,3]' = 0e+000 [/con]
  [con 'g[3,1]']  'z[3]' - 'g[3,1]' = 0e+000 [/con]
  [con 'g[3,2]']  - 'g[3,2]' = -1.25e-001 [/con]
  [con 'g[3,3]']  'v[3]' - 'g[3,3]' = 0e+000 [/con]
 [/constraints]

 [bounds]
  [b]          0 <= * [/b]
  [b]               s =  5e-002 [/b]
  [b]               't[1]','t[2]','t[3]','c[1]','c[2]','c[3]' free [/b]
  [b]               'v[1]','v[2]','v[3]','z[1]','z[2]','z[3]' free [/b]
  [b]               'f[1,1]','f[1,2]','f[1,3]','f[2,1]','f[2,2]','f[2,3]' free [/b]
  [b]               'f[3,1]','f[3,2]','f[3,3]','g[1,1]','g[1,2]','g[1,3]' free [/b]
  [b]               'g[2,1]','g[2,2]','g[2,3]','g[3,1]','g[3,2]','g[3,3]' free [/b]
  [cone quad 'stddev'] s, 't[1]', 't[2]', 't[3]' [/cone]
  [cone rquad 'f[1]'] 'f[1,1]', 'f[1,2]', 'f[1,3]' [/cone]
  [cone rquad 'f[2]'] 'f[2,1]', 'f[2,2]', 'f[2,3]' [/cone]
  [cone rquad 'f[3]'] 'f[3,1]', 'f[3,2]', 'f[3,3]' [/cone]
  [cone rquad 'g[1]'] 'g[1,1]', 'g[1,2]', 'g[1,3]' [/cone]
  [cone rquad 'g[2]'] 'g[2,1]', 'g[2,2]', 'g[2,3]' [/cone]
  [cone rquad 'g[3]'] 'g[3,1]', 'g[3,2]', 'g[3,3]' [/cone]
 [/bounds]
```

The file verifies that the correct problem has been setup.

Chapter 9

Usage guidelines

The purpose of this chapter is to present some general guidelines to follow when using MOSEK.

9.1 Verifying the results

The main purpose of MOSEK is to solve optimization problems and therefore the most fundamental question to be asked is whether the solution reported by MOSEK is a solution to the desired optimization problem.

There can be several reasons why it might be not case. The most prominent reasons are:

- A wrong problem. The problem inputted to MOSEK is simply not the right problem, i.e. some of the data may have been corrupted or the model has been incorrectly built.
- Numerical issues. The problem is badly scaled or otherwise badly posed.
- Other reasons. E.g. not enough memory or an explicit user request to stop.

The first step in verifying that MOSEK reports the expected solution is to inspect the solution summary generated by MOSEK. The solution summary provides information about

- the problem and solution statuses,
- objective value and infeasibility measures for the primal solution, and
- objective value and infeasibility measures for the dual solution, where applicable.

By inspecting the solution summary it can be verified that MOSEK produces a feasible solution, and, in the continuous case, the optimality can be checked using the dual solution. Furthermore, the problem itself ca be inspected using the problem analyzer discussed in section 13.1.

If the summary reports conflicting information (e.g. a solution status that does not match the actual solution), or the cause for terminating the solver before a solution was found cannot be traced back to the reasons stated above, it may be caused by a bug in the solver; in this case, please contact MOSEK support.

9.1.1 Verifying primal feasibility

If it has been verified that MOSEK solves the problem correctly but the solution is still not as expected, next step is to verify that the primal solution satisfies all the constraints. Hence, using the original problem it must be determined whether the solution satisfies all the required constraints in the model. For instance assume that the problem has the constraints

$$\begin{array}{rcl} x_1 + 2x_2 + x_3 & \leq & 1, \\ x_1, x_2, x_3 \geq 0 & & \end{array}$$

and MOSEK reports the optimal solution

$$x_1 = x_2 = x_3 = 1.$$

Then clearly the solution violates the constraints. The most likely explanation is that the model does not match the problem entered into MOSEK, for instance

$$x_1 - 2x_2 + x_3 \leq 1$$

may have been inputted instead of

$$x_1 + 2x_2 + x_3 \leq 1.$$

A good way to debug such an issue is to dump the problem to OPF file and check whether the violated constraint has been specified correctly.

9.1.2 Verifying optimality

Verifying that a feasible solution is optimal can be harder. However, for continuous problems optimality can verified using a dual solution. Normally, MOSEK will report a dual solution; if that is feasible and has the same objective value as the primal solution, then the primal solution must be optimal.

An alternative method is to find another primal solution that has better objective value than the one reported to MOSEK. If that is possible then either the problem is badly posed or there is bug in MOSEK.

9.2 Turn on logging

While developing a new application it is recommended to turn on logging, so that error and diagnostics messages are displayed. See example in section 5.2 for instructions on turning log output on. You should also always cache and handle any exceptions thrown by MOSEK.

More log information can be obtained by modifying one or more of the parameters:

- iparam.log,
- iparam.log_intpnt,
- iparam.log_mio,
- iparam.log_cut_second_opt,
- iparam.log_sim, and
- iparam.log_sim_minor.

By default MOSEK will reduce the amount of log information after the first optimization on a given task. To get full log output on subsequent optimizations set:

```
iparam.log_cut_second_opt 0
```

9.3 Writing task data to a file

If something is wrong with a problem or a solution, one option is to output the problem to an OPF file and inspect it by hand. Use the Task.writedata function to write a task to a file immediately before optimizing, for example as follows:

```
task.writedata("taskdump.opf")
task.optimizetrm()
```

This will write the problem in `task` to the file `taskdump.opf`. Inspecting the text file `taskdump.opf` may reveal what is wrong in the problem setup.

9.4 Important API limitations

9.4.1 Thread safety

The MOSEK API is thread safe in the sense that any number of threads may use it simultaneously. However, the individual tasks and environments may *only* be accessed from at most one thread at a time.

Chapter 10

Problem formulation and solutions

In this chapter we will discuss the following issues:

- The formal definitions of the problem types that MOSEK can solve.
- The solution information produced by MOSEK.
- The information produced by MOSEK if the problem is infeasible.

10.1 Linear optimization

A linear optimization problem can be written as

$$\begin{array}{lccccc} \mbox{minimize} & & & c^T x + c^f & & \\ \mbox{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \end{array} \tag{10.1}$$

where

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

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

A primal solution (x) is *(primal) feasible* if it satisfies all constraints in (10.1). If (10.1) has at least one primal feasible solution, then (10.1) is said to be (primal) feasible.

In case (10.1) does not have a feasible solution, the problem is said to be *(primal) infeasible* .

10.1.1 Duality for linear optimization

Corresponding to the primal problem (10.1), there is a dual problem

$$\begin{array}{lrcl} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\ & -y + s_l^c - s_u^c & = & 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0. \end{array} \tag{10.2}$$

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

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

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

A solution

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

to the dual problem is feasible if it satisfies all the constraints in (10.2). If (10.2) has at least one feasible solution, then (10.2) is *(dual) feasible*, otherwise the problem is *(dual) infeasible*.

10.1.1.1 A primal-dual feasible solution

A solution

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

is denoted a *primal-dual feasible solution*, if (x) is a solution to the primal problem (10.1) and $(y, s_l^c, s_u^c, s_l^x, s_u^x)$ is a solution to the corresponding dual problem (10.2).

10.1.1.2 The duality gap

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 *duality gap* as the difference between the primal and the dual objective value,

$$\begin{aligned}
& c^T x^* + c^f - \left((l^c)^T (s_l^c)^* - (u^c)^T (s_u^c)^* + (l^x)^T (s_l^x)^* - (u^x)^T (s_u^x)^* + c^f\right) \\
= & \sum_{i=0}^{m-1} [(s_l^c)_i^*((x_i^c)^* - l_i^c) + (s_u^c)_i^*(u_i^c - (x_i^c)^*)] + \sum_{j=0}^{n-1} \left[(s_l^x)_j^*(x_j - l_j^x) + (s_u^x)_j^*(u_j^x - x_j^*)\right] \\
\geq & \ 0
\end{aligned} \tag{10.3}$$

where the first relation can be obtained by transposing and multiplying the dual constraints (10.2) by x^* and $(x^c)^*$ respectively, and the second relation comes from the fact that each term in each sum is nonnegative. It follows that the primal objective will always be greater than or equal to the dual objective.

10.1.1.3 When the objective is to be maximized

When the objective sense of problem (10.1) is maximization, i.e.

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

the objective sense of the dual problem changes to minimization, and the domain of all dual variables changes sign in comparison to (10.2). The dual problem thus takes the form

$$\begin{array}{lrcl}
\text{minimize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\
\text{subject to} & A^T y + s_l^x - s_u^x & = & c, \\
 & -y + s_l^c - s_u^c & = & 0, \\
 & s_l^c, s_u^c, s_l^x, s_u^x & \leq & 0.
\end{array}$$

This means that the duality gap, defined in (10.3) as the primal minus the dual objective value, becomes nonpositive. It follows that the dual objective will always be greater than or equal to the primal objective.

10.1.1.4 An optimal solution

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

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

are satisfied.

If (10.1) has an optimal solution and MOSEK solves the problem successfully, both the primal and dual solution are reported, including a status indicating the exact state of the solution.

10.1.2 Infeasibility for linear optimization

10.1.2.1 Primal infeasible problems

If the problem (10.1) is infeasible (has no feasible solution), MOSEK will report a certificate of primal infeasibility: The dual solution reported is the certificate of infeasibility, and the primal solution is undefined.

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

$$
\begin{array}{ll}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\
\text{subject to} & \begin{array}{rcl} A^T y + s_l^x - s_u^x & = & 0, \\ -y + s_l^c - s_u^c & = & 0, \\ s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, \end{array}
\end{array}
\tag{10.4}
$$

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

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

to (10.4) so that

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

Such a solution implies that (10.4) is unbounded, and that its dual is infeasible. As the constraints to the dual of (10.4) is identical to the constraints of problem (10.1), we thus have that problem (10.1) is also infeasible.

10.1.2.2 Dual infeasible problems

If the problem (10.2) is infeasible (has no feasible solution), MOSEK will report a certificate of dual infeasibility: The primal solution reported is the certificate of infeasibility, and the dual solution is undefined.

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

$$
\begin{array}{lccccc}
\text{minimize} & & & c^T x & & \\
\text{subject to} & \hat{l}^c & \leq & Ax & \leq & \hat{u}^c, \\
 & \hat{l}^x & \leq & x & \leq & \hat{u}^x,
\end{array}
\tag{10.5}
$$

where

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

and

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

such that the objective value $c^T x$ is strictly negative.

Such a solution implies that (10.5) is unbounded, and that its dual is infeasible. As the constraints to the dual of (10.5) is identical to the constraints of problem (10.2), we thus have that problem (10.2) is also infeasible.

10.1.2.3 Primal and dual infeasible case

In case that both the primal problem (10.1) and the dual problem (10.2) are infeasible, MOSEK will report only one of the two possible certificates — which one is not defined (MOSEK returns the first certificate found).

10.2 Conic quadratic optimization

Conic quadratic optimization is an extensions of linear optimization (see Section 10.1) allowing conic domains to be specified for subsets of the problem variables. A conic quadratic optimization problem can be written as

$$\begin{array}{lccccc} \text{minimize} & & & c^T x + c^f & & \\ \text{subject to} & l^c & \leq & Ax & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x \in \mathcal{C}, & & \end{array} \tag{10.6}$$

where set $\mathcal{C}$ is a Cartesian product of convex cones, namely $\mathcal{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_p$. Having the domain restriction, $x \in \mathcal{C}$, is thus equivalent to

$$x^t \in \mathcal{C}_t \subseteq \mathbb{R}^{n_t},$$

where $x = (x^1, \ldots, x^p)$ is a partition of the problem variables. Please note that the n-dimensional Euclidean space $\mathbb{R}^n$ is a cone itself, so simple linear variables are still allowed.

MOSEK supports only a limited number of cones, specifically:

- The $\mathbb{R}^n$ set.

- The quadratic cone:

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

- The rotated quadratic cone:

$$\mathcal{Q}_n^r = \left\{ x \in \mathbb{R}^n : 2x_1 x_2 \geq \sum_{j=3}^{n} x_j^2, \;\; x_1 \geq 0, \;\; x_2 \geq 0 \right\}.$$

Although these cones may seem to provide only limited expressive power they can be used to model a wide range of problems as demonstrated in [7].

10.2.1 Duality for conic quadratic optimization

The dual problem corresponding to the conic quadratic optimization problem (10.6) is given by

$$\begin{array}{lccl}
\text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f & & \\
\text{subject to} & A^T y + s_l^x - s_u^x + s_n^x & = & c, \\
 & -y + s_l^c - s_u^c & = & 0, \\
 & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, \\
 & s_n^x \in \mathcal{C}^*, & &
\end{array} \tag{10.7}$$

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

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

where each $\mathcal{C}_t^*$ is the dual cone of $\mathcal{C}_t$. For the cone types MOSEK can handle, the relation between the primal and dual cone is given as follows:

- The $\mathbb{R}^n$ set:

$$\mathcal{C}_t = \mathbb{R}^{n_t} \;\Leftrightarrow\; \mathcal{C}_t^* = \{ s \in \mathbb{R}^{n_t} : \; s = 0 \}.$$

- The quadratic cone:

$$\mathcal{C}_t = \mathcal{Q}_{n_t} \;\Leftrightarrow\; \mathcal{C}_t^* = \mathcal{Q}_{n_t} = \left\{ s \in \mathbb{R}^{n_t} : s_1 \geq \sqrt{\sum_{j=2}^{n_t} s_j^2} \right\}.$$

- The rotated quadratic cone:

$$\mathcal{C}_t = \mathcal{Q}_{n_t}^r \;\Leftrightarrow\; \mathcal{C}_t^* = \mathcal{Q}_{n_t}^r = \left\{ s \in \mathbb{R}^{n_t} : 2 s_1 s_2 \geq \sum_{j=3}^{n_t} s_j^2, \;\; s_1 \geq 0, \;\; s_2 \geq 0 \right\}.$$

Please note that the dual problem of the dual problem is identical to the original primal problem.

10.2.2 Infeasibility for conic quadratic optimization

In case MOSEK finds a problem to be infeasible it reports a certificate of the infeasibility. This works exactly as for linear problems (see Section 10.1.2).

10.2.2.1 Primal infeasible problems

If the problem (10.6) is infeasible, MOSEK will report a certificate of primal infeasibility: The dual solution reported is the certificate of infeasibility, and the primal solution is undefined.

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

$$\begin{array}{ll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x \\ \text{subject to} & \begin{array}{rcl} A^T y + s_l^x - s_u^x + s_n^x & = & 0, \\ -y + s_l^c - s_u^c & = & 0, \\ s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, \\ s_n^x \in \mathcal{C}^*, & & \end{array} \end{array} \tag{10.8}$$

such that the objective value is strictly positive.

10.2.2.2 Dual infeasible problems

If the problem (10.7) is infeasible, MOSEK will report a certificate of dual infeasibility: The primal solution reported is the certificate of infeasibility, and the dual solution is undefined.

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

$$\begin{array}{lccccc} \text{minimize} & & & c^T x & & \\ \text{subject to} & \hat{l}^c & \leq & Ax & \leq & \hat{u}^c, \\ & \hat{l}^x & \leq & x & \leq & \hat{u}^x, \\ & & & x \in \mathcal{C}, & & \end{array} \tag{10.9}$$

where

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

and

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

such that the objective value is strictly negative.

10.3 Semidefinite optimization

Semidefinite optimization is an extension of conic quadratic optimization (see Section 10.2) allowing positive semidefinite matrix variables to be used in addition to the usual scalar variables. A semidefinite optimization problem can be written as

$$\begin{array}{lccccll}
\text{minimize} & & & \displaystyle\sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \left\langle \overline{C}_j, \overline{X}_j \right\rangle + c^f & & & \\
\text{subject to} & l_i^c & \leq & \displaystyle\sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} \left\langle \overline{A}_{ij}, \overline{X}_j \right\rangle & \leq & u_i^c, & i = 0, \ldots, m-1 \\
 & l_j^x & \leq & x_j & \leq & u_j^x, & j = 0, \ldots, n-1 \\
 & & & x \in \mathcal{C}, \overline{X}_j \in \mathcal{S}_{r_j}^+, & & & j = 0, \ldots, p-1
\end{array} \tag{10.10}$$

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

$$\langle U, V \rangle := \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} U_{ij} V_{ij}.$$

With semidefinite optimization we can model a wide range of problems as demonstrated in [7].

10.3.1 Duality for semidefinite optimization

The dual problem corresponding to the semidefinite optimization problem (10.10) is given by

$$\begin{array}{lccll}
\text{maximize} & \multicolumn{4}{l}{(l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f} \\
\text{subject to} & c - A^T y + s_u^x - s_l^x & = & s_n^x, & \\
 & \overline{C}_j - \displaystyle\sum_{i=0}^{m} y_i \overline{A}_{ij} & = & \overline{S}_j, & j = 0, \ldots, p-1 \\
 & s_l^c - s_u^c & = & y, & \\
 & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, & \\
 & s_n^x \in \mathcal{C}^*, \ \overline{S}_j \in \mathcal{S}_{r_j}^+, & & & j = 0, \ldots, p-1
\end{array} \tag{10.11}$$

where $A \in \mathbb{R}^{m \times n}$, $A_{ij} = a_{ij}$, which is similar to the dual problem for conic quadratic optimization (see Section 10.7), except for the addition of dual constraints

$$(\overline{C}_j - \sum_{i=0}^{m} y_i \overline{A}_{ij}) \in \mathcal{S}_{r_j}^+.$$

Note that the dual of the dual problem is identical to the original primal problem.

10.3.2 Infeasibility for semidefinite optimization

In case MOSEK finds a problem to be infeasible it reports a certificate of the infeasibility. This works exactly as for linear problems (see Section 10.1.2).

10.3.2.1 Primal infeasible problems

If the problem (10.10) is infeasible, MOSEK will report a certificate of primal infeasibility: The dual solution reported is a certificate of infeasibility, and the primal solution is undefined.

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

$$
\begin{array}{lrcll}
\text{maximize} & \multicolumn{3}{c}{(l^c)^T s_l^c - (u^c)^T s_u^c + (l^x)^T s_l^x - (u^x)^T s_u^x} & \\
\text{subject to} & A^T y + s_l^x - s_u^x + s_n^x & = & 0, & \\
 & \sum_{i=0}^{m-1} y_i \overline{A}_{ij} + \overline{S}_j & = & 0, & j = 0, \ldots, p-1 \\
 & -y + s_l^c - s_u^c & = & 0, & \\
 & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, & \\
 & s_n^x \in \mathcal{C}^*, \ \overline{S}_j \in \mathcal{S}_{r_j}^+, & & & j = 0, \ldots, p-1
\end{array}
\tag{10.12}
$$

such that the objective value is strictly positive.

10.3.2.2 Dual infeasible problems

If the problem (10.11) is infeasible, MOSEK will report a certificate of dual infeasibility: The primal solution reported is the certificate of infeasibility, and the dual solution is undefined.

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

$$
\begin{array}{lccccll}
\text{minimize} & & & \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \left\langle \overline{C}_j, \overline{X}_j \right\rangle & & & \\
\text{subject to} & \hat{l}_i^c & \leq & \sum_{j=1} a_{ij} x_j + \sum_{j=0}^{p-1} \left\langle \overline{A}_{ij}, \overline{X}_j \right\rangle & \leq & \hat{u}_i^c, & i = 0, \ldots, m-1 \\
 & \hat{l}^x & \leq & x & \leq & \hat{u}^x, & \\
 & & & x \in \mathcal{C}, \ \overline{X}_j \in \mathcal{S}_{r_j}^+, & & & j = 0, \ldots, p-1
\end{array}
\tag{10.13}
$$

where

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

and

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

such that the objective value is strictly negative.

10.4 Quadratic and quadratically constrained optimization

A convex quadratic and quadratically constrained optimization problem is an optimization problem of the form

$$
\begin{array}{lrclcll}
\text{minimize} & & & \frac{1}{2}x^TQ^ox + c^Tx + c^f & & & \\
\text{subject to} & l_k^c & \leq & \frac{1}{2}x^TQ^kx + \sum_{j=0}^{n-1} a_{kj}x_j & \leq & u_k^c, & k = 0, \ldots, m-1, \\
 & l_j^x & \leq & x_j & \leq & u_j^x, & j = 0, \ldots, n-1,
\end{array}
\tag{10.14}
$$

where Q^o and all Q^k are symmetric matrices. Moreover for convexity, Q^o must be a positive semidefinite matrix and Q^k must satisfy

$$
\begin{array}{rcl}
-\infty < l_k^c & \Rightarrow & Q^k \text{ is negative semidefinite,} \\
u_k^c < \infty & \Rightarrow & Q^k \text{ is positive semidefinite,} \\
-\infty < l_k^c \leq u_k^c < \infty & \Rightarrow & Q^k = 0.
\end{array}
$$

The convexity requirement is very important and it is strongly recommended that MOSEK is applied to convex problems only.

Note that any convex quadratic and quadratically constrained optimization problem can be reformulated as a conic optimization problem. It is our experience that for the majority of practical applications it is better to cast them as conic problems because

- the resulting problem is convex by construction, and
- the conic optimizer is more efficient than the optimizer for general quadratic problems.

See [7] for further details.

10.4.1 Duality for quadratic and quadratically constrained optimization

The dual problem corresponding to the quadratic and quadratically constrained optimization problem (10.14) is given by

$$
\begin{array}{lrcl}
\text{maximize} & (l^c)^Ts_l^c - (u^c)^Ts_u^c + (l^x)^Ts_l^x - (u^x)^Ts_u^x + \frac{1}{2}x^T\left(\sum_{k=0}^{m-1} y_kQ^k - Q^o\right)x + c^f & & \\
\text{subject to} & A^Ty + s_l^x - s_u^x + \left(\sum_{k=0}^{m-1} y_kQ^k - Q^o\right)x & = & c, \\
 & -y + s_l^c - s_u^c & = & 0, \\
 & s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0.
\end{array}
\tag{10.15}
$$

The dual problem is related to the dual problem for linear optimization (see Section 10.2), but depend on variable x which in general can not be eliminated. In the solutions reported by MOSEK, the value of x is the same for the primal problem (10.14) and the dual problem (10.15).

10.4.2 Infeasibility for quadratic and quadratically constrained optimization

In case MOSEK finds a problem to be infeasible it reports a certificate of the infeasibility. This works exactly as for linear problems (see Section 10.1.2).

10.4.2.1 Primal infeasible problems

If the problem (10.14) with all $Q^k = 0$ is infeasible, MOSEK will report a certificate of primal infeasibility. As the constraints is the same as for a linear problem, the certificate of infeasibility is the same as for linear optimization (see Section 10.1.2.1).

10.4.2.2 Dual infeasible problems

If the problem (10.15) with all $Q^k = 0$ is infeasible, MOSEK will report a certificate of dual infeasibility: The primal solution reported is the certificate of infeasibility, and the dual solution is undefined.

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

$$\begin{array}{lccccc} \text{minimize} & & & c^T x & & \\ \text{subject to} & \hat{l}^c & \leq & Ax & \leq & \hat{u}^c, \\ & 0 & \leq & Q^o x & \leq & 0, \\ & \hat{l}^x & \leq & x & \leq & \hat{u}^x, \end{array} \tag{10.16}$$

where

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

and

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

such that the objective value is strictly negative.

Chapter 11

The optimizers for continuous problems

The most essential part of MOSEK is the optimizers. Each optimizer is designed to solve a particular class of problems i.e. linear, conic, or general nonlinear problems. The purpose of the present chapter is to discuss which optimizers are available for the continuous problem classes and how the performance of an optimizer can be tuned, if needed.

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

11.1 How an optimizer works

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

Presolve:

Preprocessing to reduce the size of the problem.

Dualizer:

Choosing whether to solve the primal or the dual form of the problem.

Scaling:

Scaling the problem for better numerical stability.

Optimize:

Solve the problem using selected method.

The first three preprocessing steps are transparent to the user, but useful to know about for tuning purposes. In general, the purpose of the preprocessing steps is to make the actual optimization more efficient and robust.

11.1.1 Presolve

Before an optimizer actually performs the optimization the problem is preprocessed using the so-called presolve. The purpose of the presolve is to

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

After the presolved problem has been optimized the solution is automatically postsolved so that the returned solution is valid for the original problem. Hence, the presolve is completely transparent. For further details about the presolve phase, please see [8], [9].

It is possible to fine-tune the behavior of the presolve or to turn it off entirely. If presolve consumes too much time or memory compared to the reduction in problem size gained it may be disabled. This is done by setting the parameter `iparam.presolve_use` to `presolvemode.off`.

The two most time-consuming steps of the presolve are

- the eliminator, and
- the linear dependency check.

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

11.1.1.1 Numerical issues in the presolve

During the presolve the problem is reformulated so that it hopefully solves faster. However, in rare cases the presolved problem may be harder to solve then the original problem. The presolve may also be infeasible although the orinal problem is not.

If it is suspected that presolved problem is much harder to solve than the original then it is suggested to first turn the eliminator off by setting the parameter `iparam.presolve_eliminator_use`. If that does not help, then trying to turn presolve off may help.

Since all computations are done in finite prescision then the presolve employs some tolerances when concluding a variable is fixed or constraint is redundant. If it happens that MOSEK incorrectly concludes a problem is primal or dual infeasible, then it is worthwhile to try to reduce the parameters `dparam.presolve_tol_x` and `dparam.presolve_tol_s`. However, if actually help reducing the parameters then this should be taken as an indication of the problem is badly formulated.

11.1.1.2 Eliminator

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

$$\begin{aligned} y &= \sum x_j, \\ y, x &\geq 0, \end{aligned}$$

y is an implied free variable that can be substituted out of the problem, if deemed worthwhile.

If the eliminator consumes too much time or memory compared to the reduction in problem size gained it may be disabled. This can be done with the parameter `iparam.presolve_eliminator_use` to `onoffkey.off`.

In rare cases the eliminator may cause that the problem becomes much hard to solve.

11.1.1.3 Linear dependency checker

The purpose of the linear dependency check is to remove linear dependencies among the linear equalities. For instance, the three linear equalities

$$\begin{aligned} x_1 + x_2 + x_3 &= 1, \\ x_1 + 0.5x_2 &= 0.5, \\ 0.5x_2 + x_3 &= 0.5 \end{aligned}$$

contain exactly one linear dependency. This implies that one of the constraints can be dropped without changing the set of feasible solutions. Removing linear dependencies is in general a good idea since it reduces the size of the problem. Moreover, the linear dependencies are likely to introduce numerical problems in the optimization phase.

It is best practise to build models without linear dependencies. If the linear dependencies are removed at the modeling stage, the linear dependency check can safely be disabled by setting the parameter `iparam.presolve_lindep_use` to `onoffkey.off`.

11.1.2 Dualizer

All linear, conic, and convex optimization problems have an equivalent dual problem associated with them. MOSEK has built-in heuristics to determine if it is most efficient to solve the primal or dual problem. The form (primal or dual) solved is displayed in the MOSEK log. Should the internal heuristics not choose the most efficient form of the problem it may be worthwhile to set the dualizer manually by setting the parameters:

- `iparam.intpnt_solve_form`: In case of the interior-point optimizer.
- `iparam.sim_solve_form`: In case of the simplex optimizer.

Note that currently only linear problems may be dualized.

11.1.3 Scaling

Problems containing data with large and/or small coefficients, say $1.0e+9$ or $1.0e-7$, are often hard to solve. Significant digits may be truncated in calculations with finite precision, which can result in the optimizer relying on inaccurate calculations. Since computers work in finite precision, extreme coefficients should be avoided. In general, data around the same "order of magnitude" is preferred, and we will refer to a problem, satisfying this loose property, as being *well-scaled*. If the problem is not well scaled, MOSEK will try to scale (multiply) constraints and variables by suitable constants. MOSEK solves the scaled problem to improve the numerical properties.

The scaling process is transparent, i.e. the solution to the original problem is reported. It is important to be aware that the optimizer terminates when the termination criterion is met on the scaled problem, therefore significant primal or dual infeasibilities may occur after unscaling for badly scaled problems. The best solution to this problem is to reformulate it, making it better scaled.

By default MOSEK heuristically chooses a suitable scaling. The scaling for interior-point and simplex optimizers can be controlled with the parameters `iparam.intpnt_scaling` and `iparam.sim_scaling` respectively.

11.1.4 Using multiple threads

The interior-point optimizers in MOSEK have been parallelized. This means that if you solve linear, quadratic, conic, or general convex optimization problem using the interior-point optimizer, you can take advantage of multiple CPU's.

By default MOSEK will automatically select the number of threads to be employed when solving the problem. However, the number of threads employed can be changed by setting the parameter `iparam.num_threads`. This should never exceed the number of cores on the computer.

The speed-up obtained when using multiple threads is highly problem and hardware dependent, and consequently, it is advisable to compare single threaded and multi threaded performance for the given problem type to determine the optimal settings.

For small problems, using multiple threads is not be worthwhile and may even be counter productive.

11.2 Linear optimization

11.2.1 Optimizer selection

Two different types of optimizers are available for linear problems: The default is an interior-point method, and the alternatives are simplex methods. The optimizer can be selected using the parameter `iparam.optimizer`.

11.2.2 The interior-point optimizer

The purpose of this section is to provide information about the algorithm employed in MOSEK interior-point optimizer.

In order to keep the discussion simple it is assumed that MOSEK solves linear optimization problems on standard form

$$\begin{array}{lccc} \text{minimize} & c^T x & & \\ \text{subject to} & Ax & = & b, \\ & x \geq 0. & & \end{array} \tag{11.1}$$

This is in fact what happens inside MOSEK; for efficiency reasons MOSEK converts the problem to standard form before solving, then convert it back to the input form when reporting the solution.

Since it is not known beforehand whether problem (11.1) has an optimal solution, is primal infeasible or is dual infeasible, the optimization algorithm must deal with all three situations. This is the reason that MOSEK solves the so-called homogeneous model

$$\begin{array}{rcl} Ax - b\tau & = & 0, \\ A^T y + s - c\tau & = & 0, \\ -c^T x + b^T y - \kappa & = & 0, \\ x, s, \tau, \kappa & \geq & 0, \end{array} \tag{11.2}$$

where y and s correspond to the dual variables in (11.1), and τ and κ are two additional scalar variables. Note that the homogeneous model (11.2) always has solution since

$$(x, y, s, \tau, \kappa) = (0, 0, 0, 0, 0)$$

is a solution, although not a very interesting one.

Any solution

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

to the homogeneous model (11.2) satisfies

$$x_j^* s_j^* = 0 \text{ and } \tau^* \kappa^* = 0.$$

Moreover, there is always a solution that has the property

$$\tau^* + \kappa^* > 0.$$

First, assume that $\tau^* > 0$. It follows that

$$
\begin{aligned}
A \frac{x^*}{\tau^*} &= b, \\
A^T \frac{y^*}{\tau^*} + \frac{s^*}{\tau *} &= c, \\
- c^T \frac{x^*}{\tau^*} + b^T \frac{y^*}{\tau^*} &= 0, \\
x^*, s^*, \tau^*, \kappa^* &\geq 0.
\end{aligned}
$$

This shows that $\frac{x^*}{\tau^*}$ is a primal optimal solution and $(\frac{y^*}{\tau^*}, \frac{s^*}{\tau *})$ is a dual optimal solution; this is reported as the optimal interior-point solution since

$$
(x, y, s) = \left(\frac{x^*}{\tau^*}, \frac{y^*}{\tau^*}, \frac{s^*}{\tau *} \right)
$$

is a primal-dual optimal solution.

On other hand, if $\kappa^* > 0$ then

$$
\begin{aligned}
Ax^* &= 0, \\
A^T y^* + s^* &= 0, \\
- c^T x^* + b^T y^* &= \kappa^*, \\
x^*, s^*, \tau^*, \kappa^* &\geq 0.
\end{aligned}
$$

This implies that at least one of

$$
- c^T x^* > 0 \tag{11.3}
$$

or

$$
b^T y^* > 0 \tag{11.4}
$$

is satisfied. If (11.3) is satisfied then x^* is a certificate of dual infeasibility, whereas if (11.4) is satisfied then y^* is a certificate of dual infeasibility.

In summary, by computing an appropriate solution to the homogeneous model, all information required for a solution to the original problem is obtained. A solution to the homogeneous model can be computed using a primal-dual interior-point algorithm [10].

11.2.2.1 Interior-point termination criterion

For efficiency reasons it is not practical to solve the homogeneous model exactly. Hence, an exact optimal solution or an exact infeasibility certificate cannot be computed and a reasonable termination criterion has to be employed.

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

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

to homogeneous model is generated where

$$x^k, s^k, \tau^k, \kappa^k > 0.$$

Whenever the trial solution satisfies the criterion

$$\begin{aligned} \left\| A\frac{x^k}{\tau^k} - b \right\|_\infty &\leq \epsilon_p(1 + \|b\|_\infty), \\ \left\| A^T\frac{y^k}{\tau^k} + \frac{s^k}{\tau^k} - c \right\|_\infty &\leq \epsilon_d(1 + \|c\|_\infty), \text{ and} \\ \min\left(\frac{(x^k)^T s^k}{(\tau^k)^2}, \left| \frac{c^T x^k}{\tau^k} - \frac{b^T y^k}{\tau^k} \right| \right) &\leq \epsilon_g \max\left(1, \frac{\min(\left|c^T x^k\right|, \left|b^T y^k\right|)}{\tau^k}\right), \end{aligned} \tag{11.5}$$

the interior-point optimizer is terminated and

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

is reported as the primal-dual optimal solution. The interpretation of (11.5) is that the optimizer is terminated if

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

On the other hand, if the trial solution satisfies

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

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

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

and define

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

It is easy to verify that

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

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

Tolerance	Parameter name
ϵ_p	dparam.intpnt_tol_pfeas
ϵ_d	dparam.intpnt_tol_dfeas
ϵ_g	dparam.intpnt_tol_rel_gap
ϵ_i	dparam.intpnt_tol_infeas

Table 11.1: Parameters employed in termination criterion.

Finally, if

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

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

It is possible to adjust the tolerances ϵ_p, ϵ_d, ϵ_g and ϵ_i using parameters; see table 11.1 for details. The default values of the termination tolerances are chosen such that for a majority of problems appearing in practice it is not possible to achieve much better accuracy. Therefore, tightening the tolerances usually is not worthwhile. However, an inspection of (11.5) reveals that quality of the solution is dependent on $\|b\|_\infty$ and $\|c\|_\infty$; the smaller the norms are, the better the solution accuracy.

The interior-point method as implemented by MOSEK will converge toward optimality and primal and dual feasibility at the same rate [10]. This means that if the optimizer is stopped prematurely then it is very unlikely that either the primal or dual solution is feasible. Another consequence is that in most cases all the tolerances, ϵ_p, ϵ_d and ϵ_g, has to be relaxed together to achieve an effect.

In some cases the interior-point method terminates having found a solution not too far from meeting the optimality condition (11.5). A solution is defined as *near optimal* if scaling ϵ_p, ϵ_d and ϵ_g by any number $\epsilon_n \in [1.0, +\infty]$ conditions (11.5) are satisfied.

A near optimal solution is therefore of lower quality but still potentially valuable. If for instance the solver stalls, i.e. it can make no more significant progress towards the optimal solution, a near optimal solution could be available and be good enough for the user.

The basis identification discussed in section 11.2.2.2 requires an optimal solution to work well; hence basis identification should turned off if the termination criterion is relaxed.

To conclude the discussion in this section, relaxing the termination criterion is usually is not worthwhile.

11.2.2.2 Basis identification

An interior-point optimizer does not return an optimal basic solution unless the problem has a unique primal and dual optimal solution. Therefore, the interior-point optimizer has an optional post-processing step that computes an optimal basic solution starting from the optimal interior-point solution. More information about the basis identification procedure may be found in [11].

Please note that a basic solution is often more accurate than an interior-point solution.

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

- iparam.intpnt_basis,
- iparam.bi_ignore_max_iter, and
- iparam.bi_ignore_num_error

controls when basis identification is performed.

11.2.2.3 The interior-point log

Below is a typical log output from the interior-point optimizer presented:

```
Optimizer  - threads                : 1
Optimizer  - solved problem         : the dual
Optimizer  - Constraints            : 2
Optimizer  - Cones                  : 0
Optimizer  - Scalar variables       : 6                 conic                  : 0
Optimizer  - Semi-definite variables: 0                 scalarized             : 0
Factor     - setup time             : 0.00              dense det. time        : 0.00
Factor     - ML order time          : 0.00              GP order time          : 0.00
Factor     - nonzeros before factor : 3                 after factor           : 3
Factor     - dense dim.             : 0                 flops                  : 7.00e+001
ITE PFEAS    DFEAS    GFEAS    PRSTATUS   POBJ              DOBJ              MU       TIME
0   1.0e+000 8.6e+000 6.1e+000 1.00e+000  0.000000000e+000  -2.208000000e+003 1.0e+000 0.00
1   1.1e+000 2.5e+000 1.6e-001 0.00e+000  -7.901380925e+003 -7.394611417e+003 2.5e+000 0.00
2   1.4e-001 3.4e-001 2.1e-002 8.36e-001  -8.113031650e+003 -8.055866001e+003 3.3e-001 0.00
3   2.4e-002 5.8e-002 3.6e-003 1.27e+000  -7.777530698e+003 -7.766471080e+003 5.7e-002 0.01
4   1.3e-004 3.2e-004 2.0e-005 1.08e+000  -7.668323435e+003 -7.668207177e+003 3.2e-004 0.01
5   1.3e-008 3.2e-008 2.0e-009 1.00e+000  -7.668000027e+003 -7.668000015e+003 3.2e-008 0.01
6   1.3e-012 3.2e-012 2.0e-013 1.00e+000  -7.667999994e+003 -7.667999994e+003 3.2e-012 0.01
```

The first line displays the number of threads used by the optimizer and second line tells that the optimizer choose to solve the dual problem rather than the primal problem. The next line displays the problem dimensions as seen by the optimizer, and the "`Factor...`" lines show various statistics. This is followed by the iteration log.

Using the same notation as in section 11.2.2 the columns of the iteration log has the following meaning:

- `ITE`: Iteration index.
- `PFEAS`: $\left\|Ax^k - b\tau^k\right\|_\infty$. The numbers in this column should converge monotonically towards to zero but may stall at low level due to rounding errors.
- `DFEAS`: $\left\|A^Ty^k + s^k - c\tau^k\right\|_\infty$. The numbers in this column should converge monotonically toward to zero but may stall at low level due to rounding errors.
- `GFEAS`: $\left\| - cx^k + b^Ty^k - \kappa^k\right\|_\infty$. The numbers in this column should converge monotonically toward to zero but may stall at low level due to rounding errors.
- `PRSTATUS`: This number converge to 1 if the problem has an optimal solution whereas it converge to -1 if that is not the case.

- POBJ: $c^T x^k / \tau^k$. An estimate for the primal objective value.
- DOBJ: $b^T y^k / \tau^k$. An estimate for the dual objective value.
- MU: $\frac{(x^k)^T s^k + \tau^k \kappa^k}{n+1}$. The numbers in this column should always converge monotonically to zero.
- TIME: Time spend since the optimization started.

11.2.3 The simplex based optimizer

An alternative to the interior-point optimizer is the simplex optimizer.

The simplex optimizer uses a different method that allows exploiting an initial guess for the optimal solution to reduce the solution time. Depending on the problem it may be faster or slower to use an initial guess; see section 11.2.4 for a discussion.

MOSEK provides both a primal and a dual variant of the simplex optimizer — we will return to this later.

11.2.3.1 Simplex termination criterion

The simplex optimizer terminates when it finds an optimal basic solution or an infeasibility certificate. A basic solution is optimal when it is primal and dual feasible; see (10.1) and (10.2) for a definition of the primal and dual problem. Due the fact that to computations are performed in finite precision MOSEK allows violation of primal and dual feasibility within certain tolerances. The user can control the allowed primal and dual infeasibility with the parameters dparam.basis_tol_x and dparam.basis_tol_s.

11.2.3.2 Starting from an existing solution

When using the simplex optimizer it may be possible to reuse an existing solution and thereby reduce the solution time significantly. When a simplex optimizer starts from an existing solution it is said to perform a *hot-start*. If the user is solving a sequence of optimization problems by solving the problem, making modifications, and solving again, MOSEK will hot-start automatically.

Setting the parameter iparam.optimizer to optimizertype.free_simplex instructs MOSEK to select automatically between the primal and the dual simplex optimizers. Hence, MOSEK tries to choose the best optimizer for the given problem and the available solution.

By default MOSEK uses presolve when performing a hot-start. If the optimizer only needs very few iterations to find the optimal solution it may be better to turn off the presolve.

11.2.3.3 Numerical difficulties in the simplex optimizers

Though MOSEK is designed to minimize numerical instability, completely avoiding it is impossible when working in finite precision. MOSEK counts a "numerical unexpected behavior" event inside the optimizer as a *set-back*. The user can define how many set-backs the optimizer accepts; if that number is exceeded, the optimization will be aborted. Set-backs are implemented to avoid long sequences where the optimizer tries to recover from an unstable situation.

Set-backs are, for example, repeated singularities when factorizing the basis matrix, repeated loss of feasibility, degeneracy problems (no progress in objective) and other events indicating numerical difficulties. If the simplex optimizer encounters a lot of set-backs the problem is usually badly scaled; in such a situation try to reformulate into a better scaled problem. Then, if a lot of set-backs still occur, trying one or more of the following suggestions may be worthwhile:

- Raise tolerances for allowed primal or dual feasibility: Hence, increase the value of
  - `dparam.basis_tol_x`, and
  - `dparam.basis_tol_s`.
- Raise or lower pivot tolerance: Change the `dparam.simplex_abs_tol_piv` parameter.
- Switch optimizer: Try another optimizer.
- Switch off crash: Set both `iparam.sim_primal_crash` and `iparam.sim_dual_crash` to 0.
- Experiment with other pricing strategies: Try different values for the parameters
  - `iparam.sim_primal_selection` and
  - `iparam.sim_dual_selection`.
- If you are using hot-starts, in rare cases switching off this feature may improve stability. This is controlled by the `iparam.sim_hotstart` parameter.
- Increase maximum set-backs allowed controlled by `iparam.sim_max_num_setbacks`.
- If the problem repeatedly becomes infeasible try switching off the special degeneracy handling. See the parameter `iparam.sim_degen` for details.

11.2.4 The interior-point or the simplex optimizer?

Given a linear optimization problem, which optimizer is the best: The primal simplex, the dual simplex or the interior-point optimizer?

It is impossible to provide a general answer to this question, however, the interior-point optimizer behaves more predictably — it tends to use between 20 and 100 iterations, almost independently of problem size — but cannot perform hot-start, while simplex can take advantage of an initial solution, but is less predictable for cold-start. The interior-point optimizer is used by default.

11.2.5 The primal or the dual simplex variant?

MOSEK provides both a primal and a dual simplex optimizer. Predicting which simplex optimizer is faster is impossible, however, in recent years the dual optimizer has seen several algorithmic and computational improvements, which, in our experience, makes it faster on average than the primal simplex optimizer. Still, it depends much on the problem structure and size.

Setting the `iparam.optimizer` parameter to `optimizertype.free_simplex` instructs MOSEK to choose which simplex optimizer to use automatically.

To summarize, if you want to know which optimizer is faster for a given problem type, you should try all the optimizers.

Alternatively, use the concurrent optimizer presented in Section 11.6.3.

11.3 Linear network optimization

11.3.1 Network flow problems

Linear optimization problems with network flow structure can often be solved significantly faster with a specialized version of the simplex method [12] than with the general solvers.

MOSEK includes a network simplex solver which frequently solves network problems significantly faster than the standard simplex optimizers.

To use the network simplex optimizer, do the following:

- Input the network flow problem as an ordinary linear optimization problem.
- Set the parameters
  - `iparam.optimizer` to `optimizertype.network_primal_simplex`.
- Optimize the problem using `Task.optimize`.

MOSEK will automatically detect the network structure and apply the specialized simplex optimizer.

11.4 Conic optimization

11.4.1 The interior-point optimizer

For conic optimization problems only an interior-point type optimizer is available. The interior-point optimizer is an implementation of the so-called homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see [13].

11.4.1.1 Interior-point termination criteria

The parameters controlling when the conic interior-point optimizer terminates are shown in Table 11.2.

Parameter name	Purpose
dparam.intpnt_co_tol_pfeas	Controls primal feasibility
dparam.intpnt_co_tol_dfeas	Controls dual feasibility
dparam.intpnt_co_tol_rel_gap	Controls relative gap
dparam.intpnt_tol_infeas	Controls when the problem is declared infeasible
dparam.intpnt_co_tol_mu_red	Controls when the complementarity is reduced enough

Table 11.2: Parameters employed in termination criterion.

11.5 Nonlinear convex optimization

11.5.1 The interior-point optimizer

For quadratic, quadratically constrained, and general convex optimization problems an interior-point type optimizer is available. The interior-point optimizer is an implementation of the homogeneous and self-dual algorithm. For a detailed description of the algorithm, please see [14], [15].

11.5.1.1 The convexity requirement

Continuous nonlinear problems are required to be convex. For quadratic problems MOSEK test this requirement before optimizing. Specifying a non-convex problem results in an error message.

The following parameters are available to control the convexity check:

- iparam.check_convexity: Turn convexity check on/off.
- dparam.check_convexity_rel_tol: Tolerance for convexity check.
- iparam.log_check_convexity: Turn on more log information for debugging.

11.5.1.2 The differentiabilty requirement

The nonlinear optimizer in MOSEK requires both first order and second order derivatives. This of course implies care should be taken when solving problems involving non-differentiable functions.

For instance, the function

$$f(x) = x^2$$

is differentiable everywhere whereas the function

$$f(x) = \sqrt{x}$$

is only differentiable for $x > 0$. In order to make sure that MOSEK evaluates the functions at points where they are differentiable, the function domains must be defined by setting appropriate variable bounds.

Parameter name	Purpose
dparam.intpnt_nl_tol_pfeas	Controls primal feasibility
dparam.intpnt_nl_tol_dfeas	Controls dual feasibility
dparam.intpnt_nl_tol_rel_gap	Controls relative gap
dparam.intpnt_tol_infeas	Controls when the problem is declared infeasible
dparam.intpnt_nl_tol_mu_red	Controls when the complementarity is reduced enough

Table 11.3: Parameters employed in termination criteria.

In general, if a variable is not ranged MOSEK will only evaluate that variable at points strictly within the bounds. Hence, imposing the bound

$$x \geq 0$$

in the case of $\sqrt{x}$ is sufficient to guarantee that the function will only be evaluated in points where it is differentiable.

However, if a function is differentiable on closed a range, specifying the variable bounds is not sufficient. Consider the function

$$f(x) = \frac{1}{x} + \frac{1}{1-x}. \tag{11.6}$$

In this case the bounds

$$0 \leq x \leq 1$$

will not guarantee that MOSEK only evaluates the function for x between 0 and 1 . To force MOSEK to strictly satisfy both bounds on ranged variables set the parameter iparam.intpnt_starting_point to startpointtype.satisfy_bounds.

For efficiency reasons it may be better to reformulate the problem than to force MOSEK to observe ranged bounds strictly. For instance, (11.6) can be reformulated as follows

$$\begin{array}{rcl} f(x) & = & \frac{1}{x} + \frac{1}{y} \\ 0 & = & 1 - x - y \\ 0 & \leq & x \\ 0 & \leq & y. \end{array}$$

11.5.1.3 Interior-point termination criteria

The parameters controlling when the general convex interior-point optimizer terminates are shown in Table 11.3.

11.6 Solving problems in parallel

If a computer has multiple CPUs, or has a CPU with multiple cores, it is possible for MOSEK to take advantage of this to speed up solution times.

11.6.1 Thread safety

The MOSEK API is thread-safe provided that a task is only modified or accessed from one thread at any given time — accessing two separate tasks from two separate threads at the same time is safe. Sharing an environment between threads is safe.

11.6.2 The parallelized interior-point optimizer

The interior-point optimizer is capable of using multiple CPUs or cores. This implies that whenever the MOSEK interior-point optimizer solves an optimization problem, it will try to divide the work so that each core gets a share of the work. The user decides how many coress MOSEK should exploit.

It is not always possible to divide the work equally, and often parts of the computations and the coordination of the work is processed sequentially, even if several cores are present. Therefore, the speed-up obtained when using multiple cores is highly problem dependent. However, as a rule of thumb, if the problem solves very quickly, i.e. in less than 60 seconds, it is not advantageous to use the parallel option.

The `iparam.num_threads` parameter sets the number of threads (and therefore the number of cores) that the interior point optimizer will use.

11.6.3 The concurrent optimizer

An alternative to the parallel interior-point optimizer is the *concurrent optimizer*. The idea of the concurrent optimizer is to run multiple optimizers on the same problem concurrently, for instance, it allows you to apply the interior-point and the dual simplex optimizers to a linear optimization problem concurrently. The concurrent optimizer terminates when the first of the applied optimizers has terminated successfully, and it reports the solution of the fastest optimizer. In that way a new optimizer has been created which essentially performs as the fastest of the interior-point and the dual simplex optimizers. Hence, the concurrent optimizer is the best one to use if there are multiple optimizers available in MOSEK for the problem and you cannot say beforehand which one will be faster.

Note in particular that any solution present in the task will also be used for hot-starting the simplex algorithms. One possible scenario would therefore be running a hot-start dual simplex in parallel with interior point, taking advantage of both the stability of the interior-point method and the ability of the simplex method to use an initial solution.

By setting the

```
iparam.optimizer
```

parameter to

Optimizer	Associated parameter	Default priority
optimizertype.intpnt	iparam.concurrent_priority_intpnt	4
optimizertype.free_simplex	iparam.concurrent_priority_free_simplex	3
optimizertype.primal_simplex	iparam.concurrent_priority_primal_simplex	2
optimizertype.dual_simplex	iparam.concurrent_priority_dual_simplex	1

Table 11.4: Default priorities for optimizer selection in concurrent optimization.

optimizertype.concurrent

the concurrent optimizer chosen.

The number of optimizers used in parallel is determined by the

iparam.concurrent_num_optimizers.

parameter. Moreover, the optimizers are selected according to a preassigned priority with optimizers having the highest priority being selected first. The default priority for each optimizer is shown in Table 11.6.3. For example, setting the iparam.concurrent_num_optimizers parameter to 2 tells the concurrent optimizer to the apply the two optimizers with highest priorities: In the default case that means the interior-point optimizer and one of the simplex optimizers.

11.6.3.1 Concurrent optimization through the API

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

```
##
#   Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#   File:    concurrent1.py
#
#   Purpose: To demonstrate how to optimize in parallel using the
#            concurrent optimizer.
##


import sys

import mosek

from mosek.array import array


# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()
```

```
# We might write everything directly as a script, but it looks nicer
# to create a function.
def main (args):
    # Open MOSEK and create an environment and task
    # Create a MOSEK environment
    env = mosek.Env ()
    # Attach a printer to the environment
    env.set_Stream (mosek.streamtype.log, streamprinter)

    # Create a task
    task = env.Task(0,0)
    # Attach a printer to the task
    task.set_Stream (mosek.streamtype.log, streamprinter)

    task.readdata(args[0])
    task.putintparam(mosek.iparam.optimizer,
                     mosek.optimizertype.concurrent)

    task.putintparam(mosek.iparam.concurrent_num_optimizers, 2)

    task.optimize()

    task.solutionsummary(mosek.streamtype.msg)

# call the main function
try:
    main (sys.argv[1:])
except mosek.Exception as e:
    print ("ERROR: %s" % str(e.code))
    if msg is not None:
        print ("\t%s" % e.msg)
    sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)

sys.exit(0)
```

11.6.4 A more flexible concurrent optimizer

MOSEK also provides a more flexible method of concurrent optimization by using the function `Task.optimizeconcurrent`. The main advantages of this function are that it allows the calling application to assign arbitrary values to the parameters of each tasks, and that call-back functions can be attached to each task. This may be useful in the following situation: Assume that you know the primal simplex optimizer to be the best optimizer for your problem, but that you do not know which of the available selection strategies (as defined by the `iparam.sim_primal_selection` parameter) is the best. In this case you can solve the problem with the primal simplex optimizer using several different selection strategies concurrently.

An example demonstrating the usage of the `Task.optimizeconcurrent` function is included below. The example solves a single problem using the interior-point and primal simplex optimizers in parallel.

```
##
```

```
#  Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#  File:      concurrent2.py
#
#  Purpose:   To demonstrate a more flexible interface for concurrent optimization.
##


import sys

import mosek

from mosek.array import array

# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
class streamprinter:
    def __init__(self,prefix):
        self.prefix = str(prefix)
    def __call__(self,text):
        #sys.stdout.write (self.prefix + text)
        sys.stdout.write (self.prefix + text)
        sys.stdout.flush()
        pass

# We might write everything directly as a script, but it looks nicer
# to create a function.
def main (args):
    # Open MOSEK and create an environment and task
    # Create a MOSEK environment
    env = mosek.Env ()
    # Attach a printer to the environment
    env.set_Stream (mosek.streamtype.log, streamprinter("[env]"))

    # Create a task
    task = env.Task(0,0)
    # Attach a printer to the task
    task.set_Stream (mosek.streamtype.log, streamprinter("simplex: "))

    # Create a task
    task_list = [env.Task(0,0)]
    # Attach a printer to the task
    task_list[0].set_Stream(mosek.streamtype.log, streamprinter("intpnt: "))

    task.readdata(args[0]);

    # Assign different parameter values to each task.
    # In this case different optimizers.
    task.putintparam(mosek.iparam.optimizer,
                       mosek.optimizertype.primal_simplex)

    task_list[0].putintparam(mosek.iparam.optimizer,
                               mosek.optimizertype.intpnt)
```

```
    # Optimize task and task_list[0] in parallel.
    # The problem data i.e. C, A, etc.
    # is copied from task to task_list[0].
    task.optimizeconcurrent(task_list)

    task.solutionsummary(mosek.streamtype.log)


# call the main function
try:
    main (sys.argv[1:])
except mosek.Exception as e:
    print ("ERROR: %s" % str(e.errno))
    if e.msg is not None:
        print ("\t%s" % e.msg)
    sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

Chapter 12

The optimizers for mixed-integer problems

A problem is a mixed-integer optimization problem when one or more of the variables are constrained to be integer valued. MOSEK contains two optimizers for mixed integer problems that is capable for solving mixed-integer

- linear,
- quadratic and quadratically constrained, and
- conic

problems.

Readers unfamiliar with integer optimization are recommended to consult some relevant literature, e.g. the book [16] by Wolsey.

12.1 Some concepts and facts related to mixed-integer optimization

It is important to understand that in a worst-case scenario, the time required to solve integer optimization problems grows exponentially with the size of the problem. For instance, assume that a problem contains n binary variables, then the time required to solve the problem in the worst case may be proportional to 2^n . The value of 2^n is huge even for moderate values of n .

In practice this implies that the focus should be on computing a near optimal solution quickly rather than at locating an optimal solution. Even if the problem is only solved approximately, it is important to know how far the approximate solution is from an optimal one. In order to say something about the goodness of an approximate solution then the concept of a relaxation is important.

Name	Run-to-run deterministic	Parallelized	Strength	Cost
Mixed-integer conic	Yes	Yes	Conic	Free add-on
Mixed-integer	No	Partial	Linear	Payed add-on

Table 12.1: Mixed-integer optimizers.

The mixed-integer optimization problem

$$
\begin{array}{rclccl}
z^* & = & \text{minimize} & c^T x & & \\
 & & \text{subject to} & Ax & = & b, \\
 & & & x \geq 0 & & \\
 & & & x_j \in \mathbb{Z}, & & \forall j \in \mathcal{J},
\end{array}
\tag{12.1}
$$

has the continuous relaxation

$$
\begin{array}{rclccl}
\underline{z} & = & \text{minimize} & c^T x & & \\
 & & \text{subject to} & Ax & = & b, \\
 & & & x \geq 0 & &
\end{array}
\tag{12.2}
$$

The continuos relaxation is identical to the mixed-integer problem with the restriction that some variables must be integer removed.

There are two important observations about the continuous relaxation. Firstly, the continuous relaxation is usually much faster to optimize than the mixed-integer problem. Secondly if $\hat{x}$ is any feasible solution to (12.1) and

$$\bar{z} := c^T \hat{x}$$

then

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

This is an important observation since if it is only possible to find a near optimal solution within a reasonable time frame then the quality of the solution can nevertheless be evaluated. The value $\underline{z}$ is a lower bound on the optimal objective value. This implies that the obtained solution is no further away from the optimum than $\bar{z} - \underline{z}$ in terms of the objective value.

Whenever a mixed-integer problem is solved MOSEK reports this lower bound so that the quality of the reported solution can be evaluated.

12.2 The mixed-integer optimizers

MOSEK includes two mixed-integer optimizers which are compared in Table 12.1. Both optimizers can handle problems with linear, quadratic objective and constraints and conic constraints. However, a problem must not contain both quadratic objective and constraints and conic constraints.

The mixed-integer conic optimizer is specialized for solving linear and conic optimization problems. It can also solve pure quadratic and quadratically constrained problems, these problems are automatically converted to conic problems before being solved. Whereas the mixed-integer optimizer deals with quadratic and quadratically constrained problems directly.

The mixed-integer conic optimizer is run-to-run deterministic. This means that if a problem is solved twice on the same computer with identical options then the obtained solution will be bit-for-bit identical for the two runs. However, if a time limit is set then this may not be case since the time taken to solve a problem is not deterministic. Moreover, the mixed-integer conic optimizer is parallelized i.e. it can exploit multiple cores during the optimization. Finally, the mixed-integer conic optimizer is a free add-on to the continuous optimizers. However, for some linear problems the mixed-integer optimizer may outperform the mixed-integer conic optimizer. On the other hand the mixed-integer conic optimizer is included with continuous optimizers free of charge and usually the fastest for conic problems.

None of the mixed-integer optimizers handles symmetric matrix variables i.e semi-definite optimization problems.

12.3 The mixed-integer conic optimizer

The mixed-integer conic optimizer is employed by setting the parameter `iparam.optimizer` to `optimizertype.mixed_int_`

The mixed-integer conic employs three phases:

Presolve:

In this phase the optimizer tries to reduce the size of the problem using preprocessing techniques. Moreover, it strengthens the continuous relaxation, if possible.

Heuristic:

Using heuristics the optimizer tries to guess a good feasible solution.

Optimization:

The optimal solution is located using a variant of the branch-and-cut method.

12.3.1 Presolve

In the preprocessing stage redundant variables and constraints are removed. The presolve stage can be turned off using the `iparam.mio_presolve_use` parameter.

12.3.2 Heuristic

Initially, the integer optimizer tries to guess a good feasible solution using a heuristic.

12.3.3 The optimization phase

This phase solves the problem using the branch and cut algorithm.

12.3.4 Caveats

The mixed-integer conic optimizer ignores the parameter

`iparam.mio_cont_sol`:

The user should fix all the integer variables at their optimal value and reoptimize instead of relying in this option.

12.4 The mixed-integer optimizer

The mixed-integer optimizer is employed by setting the parameter `iparam.optimizer` to `optimizertype.mixed_int`. In the following it is briefly described how the optimizer works.

The process of solving an integer optimization problem can be split in three phases:

Presolve:

In this phase the optimizer tries to reduce the size of the problem using preprocessing techniques. Moreover, it strengthens the continuous relaxation, if possible.

Heuristic:

Using heuristics the optimizer tries to guess a good feasible solution.

Optimization:

The optimal solution is located using a variant of the branch-and-cut method.

12.4.1 Presolve

In the preprocessing stage redundant variables and constraints are removed. The presolve stage can be turned off using the `iparam.mio_presolve_use` parameter.

12.4.2 Heuristic

Initially, the integer optimizer tries to guess a good feasible solution using different heuristics:

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

The following parameters can be used to control the effort made by the integer optimizer to find an initial feasible solution.

- iparam.mio_heuristic_level: Controls how sophisticated and computationally expensive a heuristic to employ.
- dparam.mio_heuristic_time: The minimum amount of time to spend in the heuristic search.
- iparam.mio_feaspump_level: Controls how aggressively the feasibility pump heuristic is used.

12.4.3 The optimization phase

This phase solves the problem using the branch and cut algorithm.

12.5 Termination criterion

In general, it is time consuming to find an exact feasible and optimal solution to an integer optimization problem, though in many practical cases it may be possible to find a sufficiently good solution. Therefore, the mixed-integer optimizer employs a relaxed feasibility and optimality criterion to determine when a satisfactory solution is located.

A candidate solution that is feasible to the continuous relaxation is said to be an integer feasible solution if the criterion

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

is satisfied.

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

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

is satisfied. If this is the case, the integer optimizer terminates and reports the integer feasible solution as an optimal solution. Please note that $\underline{z}$ is a valid lower bound determined by the integer optimizer during the solution process, i.e.

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

The lower bound $\underline{z}$ normally increases during the solution process.

12.5.1 Relaxed termination

If an optimal solution cannot be located within a reasonable time, it may be advantageous to employ a relaxed termination criterion after some time. Whenever the integer optimizer locates an integer feasible solution and has spent at least the number of seconds defined by the dparam.mio_disable_term_time parameter on solving the problem, it will check whether the criterion

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

Tolerance	Parameter name
δ_1	dparam.mio_tol_abs_relax_int
δ_2	dparam.mio_tol_rel_relax_int
δ_3	dparam.mio_tol_abs_gap
δ_4	dparam.mio_tol_rel_gap
δ_5	dparam.mio_near_tol_abs_gap
δ_6	dparam.mio_near_tol_rel_gap

Table 12.2: Integer optimizer tolerances.

Parameter name	Delayed	Explanation
iparam.mio_max_num_branches	Yes	Maximum number of branches allowed.
iparam.mio_max_num_relaxs	Yes	Maximum number of relaxations allowed.
iparam.mio_max_num_solutions	Yes	Maximum number of feasible integer solutions allowed.

Table 12.3: Parameters affecting the termination of the integer optimizer.

is satisfied. If it is satisfied, the optimizer will report that the candidate solution is **near optimal** and then terminate. Please note that since this criteria depends on timing, the optimizer will not be run to run deterministic.

12.5.2 Important parameters

All δ tolerances can be adjusted using suitable parameters — see Table 12.2. In Table 12.3 some other parameters affecting the integer optimizer termination criterion are shown. Please note that if the effect of a parameter is delayed, the associated termination criterion is applied only after some time, specified by the dparam.mio_disable_term_time parameter.

12.6 How to speed up the solution process

As mentioned previously, in many cases it is not possible to find an optimal solution to an integer optimization problem in a reasonable amount of time. Some suggestions to reduce the solution time are:

- Relax the termination criterion: In case the run time is not acceptable, the first thing to do is to relax the termination criterion — see Section 12.5 for details.
- Specify a good initial solution: In many cases a good feasible solution is either known or easily computed using problem specific knowledge. If a good feasible solution is known, it is usually worthwhile to use this as a starting point for the integer optimizer.
- Improve the formulation: A mixed-integer optimization problem may be impossible to solve in one form and quite easy in another form. However, it is beyond the scope of this manual

to discuss good formulations for mixed-integer problems. For discussions on this topic see for example [16].

12.7 Understanding solution quality

To determine the quality of the solution one should check the following:

- The solution status key returned by MOSEK.
- The *optimality gap*: A measure for how much the located solution can deviate from the optimal solution to the problem.
- Feasibility. How much the solution violates the constraints of the problem.

The *optimality gap* is a measure for how close the solution is to the optimal solution. The optimality gap is given by

$$\epsilon = |(\text{objective value of feasible solution}) - (\text{objective bound})|.$$

The objective value of the solution is guarantied to be within ϵ of the optimal solution.

The optimality gap can be retrieved through the solution item `dinfitem.mio_obj_abs_gap`. Often it is more meaningful to look at the optimality gap normalized with the magnitude of the solution. The relative optimality gap is available in `dinfitem.mio_obj_rel_gap`.

Chapter 13

The analyzers

13.1 The problem analyzer

The problem analyzer prints a detailed survey of the

- linear constraints and objective
- quadratic constraints
- conic constraints
- variables

of the model.

In the initial stages of model formulation the problem analyzer may be used as a quick way of verifying that the model has been built or imported correctly. In later stages it can help revealing special structures within the model that may be used to tune the optimizer's performance or to identify the causes of numerical difficulties.

The problem analyzer is run from the command line using the `-anapro` argument and produces something similar to the following (this is the problemanalyzer's survey of the `aflow30a` problem from the MIPLIB 2003 collection, see Appendix G for more examples):

```
Analyzing the problem

Constraints              Bounds                   Variables
 upper bd:       421      ranged  : all            cont:       421
 fixed   :        58                               bin :       421

-------------------------------------------------------------------------------

Objective, min cx
   range: min |c|: 0.00000   min |c|>0: 11.0000     max |c|: 500.000
 distrib:        |c|        vars
                  0          421
```

```
          [11, 100)        150
         [100, 500]        271

-------------------------------------------------------------------------------

Constraint matrix A has
       479 rows (constraints)
       842 columns (variables)
      2091 (0.518449%) nonzero entries (coefficients)

Row nonzeros, A_i
   range: min A_i: 2 (0.23753%)    max A_i: 34 (4.038%)
 distrib:        A_i        rows       rows%        acc%
                   2         421       87.89       87.89
             [8, 15]          20        4.18       92.07
            [16, 31]          30        6.26       98.33
            [32, 34]           8        1.67      100.00

Column nonzeros, A|j
   range: min A|j: 2 (0.417537%)    max A|j: 3 (0.626305%)
 distrib:        A|j        cols       cols%        acc%
                   2         435       51.66       51.66
                   3         407       48.34      100.00

A nonzeros, A(ij)
   range: min |A(ij)|: 1.00000     max |A(ij)|: 100.000
 distrib:      A(ij)      coeffs
             [1, 10)        1670
           [10, 100]         421

-------------------------------------------------------------------------------

Constraint bounds, lb <= Ax <= ub
 distrib:        |b|             lbs             ubs
                   0                             421
             [1, 10]              58              58

Variable bounds, lb <= x <= ub
 distrib:        |b|             lbs             ubs
                   0             842
             [1, 10)                             421
           [10, 100]                             421

-------------------------------------------------------------------------------
```

The survey is divided into six different sections, each described below. To keep the presentation short with focus on key elements the analyzer generally attempts to display information on issues relevant for the current model only: E.g., if the model does not have any conic constraints (this is the case in the example above) or any integer variables, those parts of the analysis will not appear.

13.1.1 General characteristics

The first part of the survey consists of a brief summary of the model's linear and quadratic constraints (indexed by i) and variables (indexed by j). The summary is divided into three subsections:

Constraints

upper bd:
: The number of upper bounded constraints, $\sum_{j=0}^{n-1} a_{ij}x_j \leq u_i^c$

lower bd:
: The number of lower bounded constraints, $l_i^c \leq \sum_{j=0}^{n-1} a_{ij}x_j$

ranged :
: The number of ranged constraints, $l_i^c \leq \sum_{j=0}^{n-1} a_{ij}x_j \leq u_i^c$

fixed :
: The number of fixed constraints, $l_i^c = \sum_{j=0}^{n-1} a_{ij}x_j = u_i^c$

free :
: The number of free constraints

Bounds

upper bd:
: The number of upper bounded variables, $x_j \leq u_j^x$

lower bd:
: The number of lower bounded variables, $l_k^x \leq x_j$

ranged :
: The number of ranged variables, $l_k^x \leq x_j \leq u_j^x$

fixed :
: The number of fixed variables, $l_k^x = x_j = u_j^x$

free :
: The number of free variables

Variables

cont:
: The number of continuous variables, $x_j \in \mathbb{R}$

bin :
: The number of binary variables, $x_j \in \{0, 1\}$

int :
: The number of general integer variables, $x_j \in \mathbb{Z}$

Only constraints, bounds and domains actually in the model will be reported on, cf. appendix G; if all entities in a section turn out to be of the same kind, the number will be replaced by `all` for brevity.

13.1.2 Objective

The second part of the survey focuses on (the linear part of) the objective, summarizing the optimization sense and the coefficients' absolute value range and distribution. The number of 0 (zero) coefficients is singled out (if any such variables are in the problem).

The range is displayed using three terms:

`min |c|:`
: The minimum absolute value among all coeffecients

`min |c|>0:`
: The minimum absolute value among the nonzero coefficients

`max |c|:`
: The maximum absolute value among the coefficients

If some of these extrema turn out to be equal, the display is shortened accordingly:

- If `min |c|` is greater than zero, the `min |c|?0` term is obsolete and will not be displayed
- If only one or two different coefficients occur this will be displayed using `all` and an explicit listing of the coefficients

The absolute value distribution is displayed as a table summarizing the numbers by orders of magnitude (with a ratio of 10). Again, the number of variables with a coefficient of 0 (if any) is singled out. Each line of the table is headed by an interval (half-open intervals including their lower bounds), and is followed by the number of variables with their objective coefficient in this interval. Intervals with no elements are skipped.

13.1.3 Linear constraints

The third part of the survey displays information on the nonzero coefficients of the linear constraint matrix.

Following a brief summary of the matrix dimensions and the number of nonzero coefficients in total, three sections provide further details on how the nonzero coefficients are distributed by row-wise count (`A_i`), by column-wise count (`A|j`), and by absolute value (`|A(ij)|`). Each section is headed by a brief display of the distribution's range (`min` and `max`), and for the row/column-wise counts the corresponding densities are displayed too (in parentheses).

The distribution tables single out three particularly interesting counts: zero, one, and two nonzeros per row/column; the remaining row/column nonzeros are displayed by orders of magnitude (ratio 2). For each interval the relative and accumulated relative counts are also displayed.

Note that constraints may have both linear and quadratic terms, but the empty rows and columns reported in this part of the survey relate to the linear terms only. If empty rows and/or columns are found in the linear constraint matrix, the problem is analyzed further in order to determine if the corresponding constraints have any quadratic terms or the corresponding variables are used in conic or quadratic constraints; cf. the last two examples of appendix G.

The distribution of the absolute values, `|A(ij)|`, is displayed just as for the objective coefficients described above.

13.1.4 Constraint and variable bounds

The fourth part of the survey displays distributions for the absolute values of the finite lower and upper bounds for both constraints and variables. The number of bounds at 0 is singled out and, otherwise, displayed by orders of magnitude (with a ratio of 10).

13.1.5 Quadratic constraints

The fifth part of the survey displays distributions for the nonzero elements in the gradient of the quadratic constraints, i.e. the nonzero row counts for the column vectors Qx . The table is similar to the tables for the linear constraints' nonzero row and column counts described in the survey's third part.

Note: Quadratic constraints may also have a linear part, but that will be included in the linear constraints survey; this means that if a problem has one or more pure quadratic constraints, part three of the survey will report an equal number of linear constraint rows with 0 (zero) nonzeros, cf. the last example in appendix G. Likewise, variables that appear in quadratic terms only will be reported as empty columns (0 nonzeros) in the linear constraint report.

13.1.6 Conic constraints

The last part of the survey summarizes the model's conic constraints. For each of the two types of cones, quadratic and rotated quadratic, the total number of cones are reported, and the distribution of the cones' dimensions are displayed using intervals. Cone dimensions of 2, 3, and 4 are singled out.

13.2 Analyzing infeasible problems

When developing and implementing a new optimization model, the first attempts will often be either infeasible, due to specification of inconsistent constraints, or unbounded, if important constraints have been left out.

In this chapter we will

- go over an example demonstrating how to locate infeasible constraints using the MOSEK infeasibility report tool,
- discuss in more general terms which properties that may cause infeasibilities, and
- present the more formal theory of infeasible and unbounded problems.

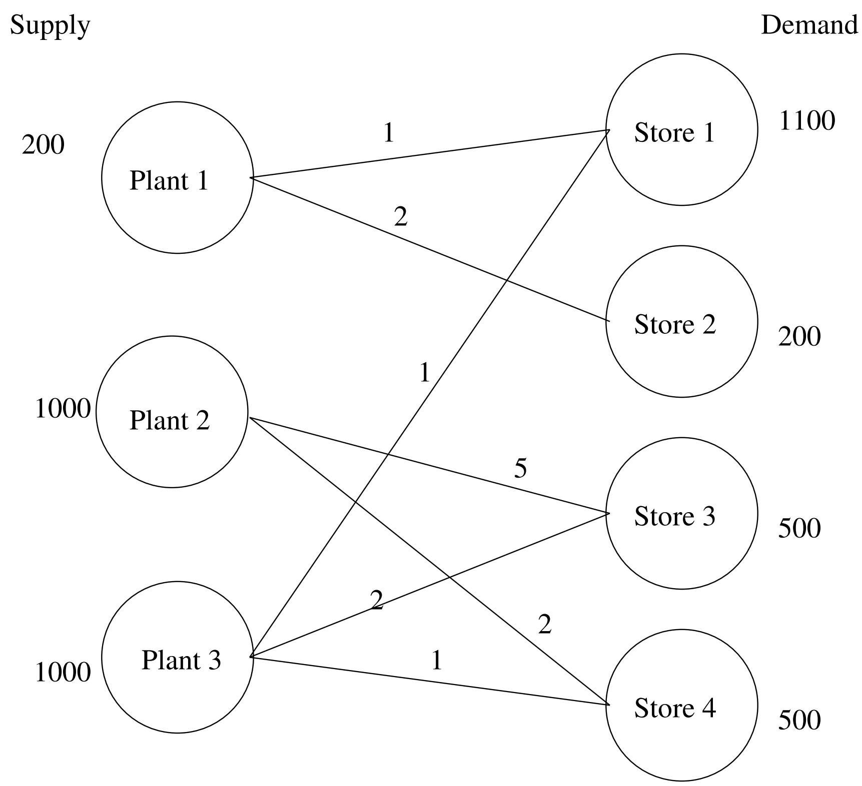

Figure 13.1: Supply, demand and cost of transportation.

Furthermore, chapter 14 contains a discussion on a specific method for repairing infeasibility problems where infeasibilities are caused by model parameters rather than errors in the model or the implementation.

13.2.1 Example: Primal infeasibility

A problem is said to be *primal infeasible* if no solution exists that satisfy all the constraints of the problem.

As an example of a primal infeasible problem consider the problem of minimizing the cost of transportation between a number of production plants and stores: Each plant produces a fixed number of goods, and each store has a fixed demand that must be met. Supply, demand and cost of transportation per unit are given in figure 13.1. The problem represented in figure 13.1 is infeasible, since the total demand

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

exceeds the total supply

$$2200 = 200 + 1000 + 1000$$

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

$$\begin{array}{lccccccccccccccccc}
\text{minimize} & x_{11} & + & 2x_{12} & + & 5x_{23} & + & 2x_{24} & + & x_{31} & + & 2x_{33} & + & x_{34} & & \\
\text{subject to} & x_{11} & + & x_{12} & & & & & & & & & & & \leq & 200, \\
& & & & & x_{23} & + & x_{24} & & & & & & & \leq & 1000, \\
& & & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\
& x_{11} & & & & & & & + & x_{31} & & & & & = & 1100, \\
& & & x_{12} & & & & & & & & & & & = & 200, \\
& & & & & x_{23} & + & & & & & x_{33} & & & = & 500, \\
& & & & & & & x_{24} & + & & & & & x_{34} & = & 500, \\
& x_{ij} \geq 0. & & & & & & & & & & & & & &
\end{array} \tag{13.1}$$

Solving the problem (13.1) using MOSEK will result in a solution, a solution status and a problem status. Among the log output from the execution of MOSEK on the above problem are the lines:

```
Basic solution
Problem status  : PRIMAL_INFEASIBLE
Solution status : PRIMAL_INFEASIBLE_CER
```

The first line indicates that the problem status is primal infeasible. The second line says that a *certificate of the infeasibility* was found. The certificate is returned in place of the solution to the problem.

13.2.2 Locating the cause of primal infeasibility

Usually a primal infeasible problem status is caused by a mistake in formulating the problem and therefore the question arises: "What is the cause of the infeasible status?" When trying to answer this question, it is often advantageous to follow these steps:

- Remove the objective function. This does not change the infeasible status but simplifies the problem, eliminating any possibility of problems related to the objective function.
- Consider whether your problem has some necessary conditions for feasibility and examine if these are satisfied, e.g. total supply should be greater than or equal to total demand.
- Verify that coefficients and bounds are reasonably sized in your problem.

If the problem is still primal infeasible, some of the constraints must be relaxed or removed completely. The MOSEK infeasibility report (Section 13.2.4) may assist you in finding the constraints causing the infeasibility.

Possible ways of relaxing your problem include:

- Increasing (decreasing) upper (lower) bounds on variables and constraints.

- Removing suspected constraints from the problem.

Returning to the transportation example, we discover that removing the fifth constraint

$$x_{12} = 200$$

makes the problem feasible.

13.2.3 Locating the cause of dual infeasibility

A problem may also be *dual infeasible*. In this case the primal problem is often unbounded, mening that feasbile solutions exists such that the objective tends towards infinity. An example of a dual infeasible and primal unbounded problem is:

$$\begin{array}{lc} \text{minimize} & x_1 \\ \text{subject to} & x_1 \leq 5. \end{array}$$

To resolve a dual infeasibility the primal problem must be made more restricted by

- Adding upper or lower bounds on variables or constraints.
- Removing variables.
- Changing the objective.

13.2.3.1 A cautious note

The problem

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

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

This illustrates the worst case scenario that all, or at least a significant portion, of the constraints are involved in the infeasibility. Hence, it may not always be easy or possible to pinpoint a few constraints which are causing the infeasibility.

13.2.4 The infeasibility report

MOSEK includes functionality for diagnosing the cause of a primal or a dual infeasibility. It can be turned on by setting the `iparam.infeas_report_auto` to `onoffkey.on`. This causes MOSEK to print a report on variables and constraints involved in the infeasibility.

The iparam.infeas_report_level parameter controls the amount of information presented in the infeasibility report. The default value is 1 .

13.2.4.1 Example: Primal infeasibility

We will reuse the example (13.1) located in `infeas.lp`:

```
\
\ An example of an infeasible linear problem.
\
minimize
 obj: + 1 x11 + 2 x12 + 1 x13
      + 4 x21 + 2 x22 + 5 x23
      + 4 x31 + 1 x32 + 2 x33
st
  s0: + x11 + x12      <= 200
  s1: + x23 + x24      <= 1000
  s2: + x31 +x33 + x34 <= 1000
  d1: + x11 + x31       = 1100
  d2: + x12             = 200
  d3: + x23 + x33       = 500
  d4: + x24 + x34       = 500
bounds
end
```

Using the command line (please remeber it accepts options following the C API format)

```
mosek -d iparam.infeas_report_auto onoffkey.on infeas.lp
```

MOSEK produces the following infeasibility report

```
MOSEK PRIMAL INFEASIBILITY REPORT.

Problem status: The problem is primal infeasible

The following constraints are involved in the primal infeasibility.

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

The following bound constraints are involved in the infeasibility.

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

The infeasibility report is divided into two sections where the first section shows which constraints that are important for the infeasibility. In this case the important constraints are the ones named `s0`, `s2`, `d1`, and `d2`. The values in the columns "`Dual lower`" and "`Dual upper`" are also useful, since a non-zero *dual lower* value for a constraint implies that the lower bound on the constraint is important for the infeasibility. Similarly, a non-zero *dual upper* value implies that the upper bound on the constraint is important for the infeasibility.

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

```
mosek -d iparam.infeas_report_auto onoffkey.on infeas.lp -info rinfeas.lp
```

produces the files `rinfeas.bas.inf.lp`. In this case the content of the file `rinfeas.bas.inf.lp` is

```
minimize
 Obj: + CFIXVAR
st
 s0: + x11 + x12 <= 200
 s2: + x31 + x33 + x34 <= 1e+003
 d1: + x11 + x31 = 1.1e+003
 d2: + x12 = 200
bounds
 x11 free
 x12 free
 x13 free
 x21 free
 x22 free
 x23 free
 x31 free
 x32 free
 x24 free
 CFIXVAR = 0e+000
end
```

which is an optimization problem. This problem is identical to (13.1), except that the objective and some of the constraints and bounds have been removed. Executing the command

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

demonstrates that the reduced problem is **primal infeasible**. Since the reduced problem is usually smaller than original problem, it should be easier to locate the cause of the infeasibility in this rather than in the original (13.1).

13.2.4.2 Example: Dual infeasibility

The example problem

```
maximize -  200 y1 - 1000 y2 - 1000 y3
         - 1100 y4 -  200 y5 -  500 y6
         -  500 y7
subject to
   x11: y1+y4 < 1
   x12: y1+y5 < 2
   x23: y2+y6 < 5
   x24: y2+y7 < 2
   x31: y3+y4 < 1
   x33: y3+y6 < 2
   x44: y3+y7 < 1
bounds
   y1 < 0
   y2 < 0
   y3 < 0
   y4 free
   y5 free
   y6 free
   y7 free
end
```

is dual infeasible. This can be verified by proving that

```
y1=-1, y2=-1, y3=0, y4=1, y5=1
```

is a certificate of dual infeasibility. In this example the following infeasibility report is produced (slightly edited):

```
The following constraints are involved in the infeasibility.

Index    Name             Activity         Objective        Lower bound      Upper bound
0        x11              -1.000000e+00                     NONE             1.000000e+00
4        x31              -1.000000e+00                     NONE             1.000000e+00

The following variables are involved in the infeasibility.

Index    Name             Activity         Objective        Lower bound      Upper bound
3        y4               -1.000000e+00    -1.100000e+03    NONE             NONE
Interior-point solution
Problem status  : DUAL_INFEASIBLE
Solution status : DUAL_INFEASIBLE_CER
Primal - objective: 1.1000000000e+03   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00 cone infeas.: 0.00e+00
Dual   - objective: 0.0000000000e+00   eq. infeas.: 0.00e+00 max bound infeas.: 0.00e+00 cone infeas.: 0.00e+00
```

Let x^* denote the reported primal solution. MOSEK states

- that the problem is *dual infeasible*,
- that the reported solution is a certificate of dual infeasibility, and
- that the infeasibility measure for x^* is approximately zero.

Since it was an maximization problem, this implies that

$$c^t x^* > 0. \tag{13.2}$$

For a minimization problem this inequality would have been reversed — see (13.5).

From the infeasibility report we see that the variable `y4`, and the constraints `x11` and `x33` are involved in the infeasibility since these appear with non-zero values in the "`Activity`" column.

One possible strategy to "fix" the infeasibility is to modify the problem so that the certificate of infeasibility becomes invalid. In this case we may do one the following things:

- Put a lower bound in `y3`. This will directly invalidate the certificate of dual infeasibility.
- Increase the object coefficient of `y3`. Changing the coefficients sufficiently will invalidate the inequality (13.2) and thus the certificate.
- Put lower bounds on `x11` or `x31`. This will directly invalidate the certificate of infeasibility.

Please note that modifying the problem to invalidate the reported certificate does *not* imply that the problem becomes dual feasible — the infeasibility may simply "move", resulting in a new infeasibility.

More often, the reported certificate can be used to give a hint about errors or inconsistencies in the model that produced the problem.

13.2.5 Theory concerning infeasible problems

This section discusses the theory of infeasibility certificates and how MOSEK uses a certificate to produce an infeasibility report. In general, MOSEK solves the problem

$$\begin{array}{lcccc} \text{minimize} & & c^T x + c^f & & \\ \text{subject to} & l^c \leq & Ax & \leq & u^c, \\ & l^x \leq & x & \leq & u^x \end{array} \tag{13.3}$$

where the corresponding dual problem is

$$\begin{array}{lll} \text{maximize} & (l^c)^T s_l^c - (u^c)^T s_u^c & \\ & + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f & \\ \text{subject to} & A^T y + s_l^x - s_u^x & = \; c, \\ & - y + s_l^c - s_u^c & = \; 0, \\ & s_l^c, s_u^c, s_l^x, s_u^x \geq 0. & \end{array} \tag{13.4}$$

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

$$l_j^x = -\infty \;\Rightarrow\; (s_l^x)_j = 0$$

13.2.6 The certificate of primal infeasibility

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

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

with a positive objective value. That is, $(s_l^{c*}, s_u^{c*}, s_l^{x*}, s_u^{x*})$ is a certificate of primal infeasibility if

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

and

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

The well-known Farkas Lemma tells us that (13.3) is infeasible if and only if a certificate of primal infeasibility exists.

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

$$(s_l^{c*})_i > 0((s_u^{c*})_i > 0)$$

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

$$(s_l^{x*})_j > 0((s_u^{x*})_i > 0)$$

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

13.2.7 The certificate of dual infeasibility

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

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

with negative objective value, where we use the definitions

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

and

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

Stated differently, a certificate of dual infeasibility is any x^* such that

$$\begin{array}{ccccc} & & c^T x^* & < & 0, \\ \bar{l}^c & \leq & Ax^* & \leq & \bar{u}^c, \\ \bar{l}^x & \leq & x^* & \leq & \bar{u}^x \end{array} \tag{13.5}$$

The well-known Farkas Lemma tells us that (13.4) is infeasible if and only if a certificate of dual infeasibility exists.

Note that if x^* is a certificate of dual infeasibility then for any j such that

$$x_j^* \neq 0,$$

variable j is involved in the dual infeasibility.

Chapter 14

Primal feasibility repair

Section 13.2.2 discusses how MOSEK treats infeasible problems. In particular, it is discussed which information MOSEK returns when a problem is infeasible and how this information can be used to pinpoint the cause of the infeasibility.

In this section we discuss how to repair a primal infeasible problem by relaxing the constraints in a controlled way. For the sake of simplicity we discuss the method in the context of linear optimization.

14.1 Manual repair

Subsequently we discuss an automatic method for repairing an infeasible optimization problem. However, it should be observed that the best way to repair an infeasible problem usually depends on what the optimization problem models. For instance in many optimization problem it does not make sense to relax the constraints $x \geq 0$ e.g. it is not possible to produce a negative quantity. Hence, whatever automatic method MOSEK provides it will never be as good as a method that exploits knowledge about what is being modelled. This implies that it is usually better to remove the underlying cause of infeasibility at the modelling stage.

Indeed consider the example

$$
\begin{array}{lrrrrrrrrrl}
\text{minimize} & & & & & & & & & & \\
\text{subject to} & & x_1 & + & x_2 & & & & & = & 1, \\
 & & & & & & x_3 & + & x_4 & = & 1, \\
 & - & x_1 & & & - & x_3 & & & = & -1+\epsilon \\
 & & & - & x_2 & & & - & x_4 & = & -1, \\
 & & x_1, & & x_2, & & x_3, & & x_4 & \geq & 0
\end{array}
\tag{14.1}
$$

then if we add the equalties together we obtain the implied equality

$$
0 = \epsilon
$$

which is infeasible for any $\epsilon \neq 0$. Here the infeasibility is caused by a linear dependency in the constraint matrix and that the right-hand side does not match if $\epsilon \neq 0$. Observe even if the problem is feasible then just a tiny perturbation to the right-hand side will make the problem infeasible. Therefore, even though the problem can be repaired then a much more robust solution is to avoid problems with linear dependent constraints. Indeed if a problem contains linear dependencies then the problem is either infeasible or contains redundant constraints. In the above case any of the equality constraints can be removed while not changing the set of feasible solutions.

To summarize linear dependencies in the constraints can give rise to infeasible problems and therefore it is better to avoid them. Note that most network flow models usually is formulated with one linear dependent constraint.

Next consider the problem

$$\begin{array}{lccc}
\text{minimize} & & & \\
\text{subject to} & x_1 - 0.01x_2 & = & 0 \\
 & x_2 - 0.01x_3 & = & 0 \\
 & x_3 - 0.01x_4 & = & 0 \\
 & x_1 & \geq & -1.0e-9 \\
 & x_1 & \leq & 1.0e-9 \\
 & x_4 & \leq & -1.0e-4
\end{array} \tag{14.2}$$

Now the MOSEK presolve for the sake of efficiency fix variables (and constraints) that has tight bounds where tightness is controlled by the parameter dparam.presolve_tol_x. Since, the bounds

$$-1.0e-9 \leq x_1 \leq 1.0e-9$$

are tight then the MOSEK presolve will fix variable x_1 at the mid point between the bounds i.e. at 0. It easy to see that this implies $x_4 = 0$ too which leads to the incorrect conclusion that the problem is infeasible. Observe tiny change of the size 1.0e-9 make the problem switch from feasible to infeasible. Such a problem is inherently unstable and is hard to solve. We normally call such a problem ill-posed. In general it is recommended to avoid ill-posed problems, but if that is not possible then one solution to this issue is is to reduce the parameter to say dparam.presolve_tol_x to say 1.0e-10. This will at least make sure that the presolve does not make the wrong conclusion.

14.2 Automatic repair

In this section we will describe the idea behind a method that automatically can repair an infeasible probem. The main idea can be described as follows.

Consider the linear optimization problem with m constraints and n variables

$$\begin{array}{lccccc}
\text{minimize} & & & c^T x + c^f & & \\
\text{subject to} & l^c & \leq & Ax & \leq & u^c, \\
 & l^x & \leq & x & \leq & u^x,
\end{array} \tag{14.3}$$

which is assumed to be infeasible.

One way of making the problem feasible is to reduce the lower bounds and increase the upper bounds. If the change is sufficiently large the problem becomes feasible. Now an obvious idea is to compute the optimal relaxation by solving an optimization problem. The problem

$$\begin{array}{lcccccl} \text{minimize} & & & p(v_l^c, v_u^c, v_l^x, v_u^x) & & & \\ \text{subject to} & l^c & \leq & Ax + v_l^c - v_u^c & \leq & u^c, & \\ & l^x & \leq & x + v_l^x - v_u^x & \leq & u^x, & \\ & & & v_l^c, v_u^c, v_l^x, v_u^x \geq 0 & & & \end{array} \tag{14.4}$$

does exactly that. The additional variables $(v_l^c)_i$, $(v_u^c)_i$, $(v_l^x)_j$ and $(v_u^c)_j$ are *elasticity* variables because they allow a constraint to be violated and hence add some elasticity to the problem. For instance, the elasticity variable $(v_l^c)_i$ controls how much the lower bound $(l^c)_i$ should be relaxed to make the problem feasible. Finally, the so-called penalty function

$$p(v_l^c, v_u^c, v_l^x, v_u^x)$$

is chosen so it penalize changes to bounds. Given the weights

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

then a natural choice is

$$p(v_l^c, v_u^c, v_l^x, v_u^x) = (w_l^c)^T v_l^c + (w_u^c)^T v_u^c + (w_l^x)^T v_l^x + (w_u^x)^T v_u^x. \tag{14.5}$$

Hence, the penalty function $p()$ is a weighted sum of the relaxation and therefore the problem (14.4) keeps the amount of relaxation at a minimum. Please observe that

- the problem (14.6) is always feasible.
- a negative weight implies problem (14.6) is unbounded. For this reason if the value of a weight is negative MOSEK fixes the associated elasticity variable to zero. Clearly, if one or more of the weights are negative may imply that it is not possible repair the problem.

A simple choice of weights is to let them all to be 1, but of course that does not take into account that constraints may have different importance.

14.2.1 Caveats

Observe if the infeasible problem

$$\begin{array}{lccc} \text{minimize} & x + z & & \\ \text{subject to} & x & = & -1, \\ & x & \geq & 0 \end{array} \tag{14.6}$$

is repaired then it will be unbounded. Hence, a repaired problem may not have an optimal solution.

Another and more important caveat is that only a minimial repair is perfomed i.e. the repair that just make the problem feasible. Hence, the repaired problem is barely feasible and that sometimes make the repaired problem hard to solve.

14.3 Feasibility repair in MOSEK

MOSEK includes a function that repair an infeasible problem using the idea described in the previous section simply by passing a set of weights to MOSEK. This can be used for linear and conic optimization problems, possibly having integer constrained variables.

14.3.1 An example using the command line tool

Consider the example linear optimization

$$\begin{array}{lccccc} \text{minimize} & -10x_1 & & -9x_2, & & \\ \text{subject to} & 7/10x_1 & + & 1x_2 & \leq & 630, \\ & 1/2x_1 & + & 5/6x_2 & \leq & 600, \\ & 1x_1 & + & 2/3x_2 & \leq & 708, \\ & 1/10x_1 & + & 1/4x_2 & \leq & 135, \\ & x_1, & & x_2 & \geq & 0, \\ & & x_2 \geq 650 & & & \end{array} \tag{14.7}$$

which is infeasible. Now suppose we wish to use MOSEK to suggest a modification to the bounds that makes the problem feasible.

Given the assumption that all weights are 1 then the command

```
mosek -primalrepair -d MSK_IPAR_LOG_FEAS_REPAIR 3 feasrepair.lp
```

will form the repaired problem and solve it. The parameter

```
MSK_IPAR_LOG_FEAS_REPAIR
```

controls the amount of log output from the repair. A value of 2 causes the optimal repair to printed out.

The output from running the above command is:

```
Copyright (c) 1998-2013 MOSEK ApS, Denmark. WWW: http://mosek.com

Open file 'feasrepair.lp'

Read summary
  Type            : LO (linear optimization problem)
  Objective sense : min
```

```
  Constraints      : 4
  Scalar variables : 2
  Matrix variables : 0
  Time             : 0.0

Computer
  Platform              : Windows/64-X86
  Cores                 : 4

Problem
  Name                  :
  Objective sense       : min
  Type                  : LO (linear optimization problem)
  Constraints           : 4
  Cones                 : 0
  Scalar variables      : 2
  Matrix variables      : 0
  Integer variables     : 0

Primal feasibility repair started.
Optimizer started.
Interior-point optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator started.
Total number of eliminations : 2
Eliminator terminated.
Eliminator - tries                  : 1                time                   : 0.00
Eliminator - elim's                 : 2
Lin. dep.  - tries                  : 1                time                   : 0.00
Lin. dep.  - number                 : 0
Presolve terminated. Time: 0.00
Optimizer  - threads                : 1
Optimizer  - solved problem         : the primal
Optimizer  - Constraints            : 2
Optimizer  - Cones                  : 0
Optimizer  - Scalar variables       : 6                conic                  : 0
Optimizer  - Semi-definite variables: 0                scalarized             : 0
Factor     - setup time             : 0.00             dense det. time        : 0.00
Factor     - ML order time          : 0.00             GP order time          : 0.00
Factor     - nonzeros before factor : 3                after factor           : 3
Factor     - dense dim.             : 0                flops                  : 5.40e+001
ITE PFEAS    DFEAS    GFEAS    PRSTATUS   POBJ              DOBJ              MU       TIME
0   2.7e+001 1.0e+000 4.8e+000 1.00e+000  4.195228609e+000  0.000000000e+000  1.0e+000 0.00
1   2.4e+001 8.6e-001 1.5e+000 0.00e+000  1.227497414e+001  1.504971820e+001  2.6e+000 0.00
2   2.6e+000 9.7e-002 1.7e-001 -6.19e-001 4.363064729e+001  4.648523094e+001  3.0e-001 0.00
3   4.7e-001 1.7e-002 3.1e-002 1.24e+000  4.256803136e+001  4.298540657e+001  5.2e-002 0.00
4   8.7e-004 3.2e-005 5.7e-005 1.08e+000  4.249989892e+001  4.250078747e+001  9.7e-005 0.00
5   8.7e-008 3.2e-009 5.7e-009 1.00e+000  4.249999999e+001  4.250000008e+001  9.7e-009 0.00
6   8.7e-012 3.2e-013 5.7e-013 1.00e+000  4.250000000e+001  4.250000000e+001  9.7e-013 0.00
Basis identification started.
Primal basis identification phase started.
ITER      TIME
0         0.00
Primal basis identification phase terminated. Time: 0.00
Dual basis identification phase started.
ITER      TIME
```

```
0        0.00
Dual basis identification phase terminated. Time: 0.00
Basis identification terminated. Time: 0.00
Interior-point optimizer terminated. Time: 0.00.

Optimizer terminated. Time: 0.03
Basic solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 4.2500000000e+001   Viol.  con: 1e-013   var: 0e+000
  Dual.    obj: 4.2500000000e+001   Viol.  con: 0e+000   var: 5e-013
Optimal objective value of the penalty problem: 4.250000000000e+001

Repairing bounds.
Increasing the upper bound -2.25e+001 on constraint 'c4' (3) with 1.35e+002.
Decreasing the lower bound 6.50e+002 on variable 'x2' (4) with 2.00e+001.
Primal feasibility repair terminated.
Optimizer started.
Interior-point optimizer started.
Presolve started.
Presolve terminated. Time: 0.00
Interior-point optimizer terminated. Time: 0.00.

Optimizer terminated. Time: 0.00

Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -5.6700000000e+003  Viol.  con: 0e+000   var: 0e+000
  Dual.    obj: -5.6700000000e+003  Viol.  con: 0e+000   var: 0e+000

Basic solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: -5.6700000000e+003  Viol.  con: 0e+000   var: 0e+000
  Dual.    obj: -5.6700000000e+003  Viol.  con: 0e+000   var: 0e+000

Optimizer summary
  Optimizer                 -                        time: 0.00
    Interior-point          - iterations : 0         time: 0.00
      Basis identification  -                        time: 0.00
        Primal              - iterations : 0         time: 0.00
        Dual                - iterations : 0         time: 0.00
        Clean primal        - iterations : 0         time: 0.00
        Clean dual          - iterations : 0         time: 0.00
        Clean primal-dual   - iterations : 0         time: 0.00
    Simplex                 -                        time: 0.00
      Primal simplex        - iterations : 0         time: 0.00
      Dual simplex          - iterations : 0         time: 0.00
      Primal-dual simplex   - iterations : 0         time: 0.00
    Mixed integer           - relaxations: 0         time: 0.00
```

reports the optimal repair. In this case it is to increase the upper bound on constraint `c4` by 1.35e2 and decrease the lower bound on variable `x2` by 20.

14.3.2 Feasibility repair using the API

The function Task.primalrepair can be used to repair an infeasible problem. Details about the function Task.primalrepair can be seen in the reference.

14.3.2.1 An example

Consider once again the example (14.7) then

[feasrepairex1.py]

```
#
#   Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#   File:      feasrepairex1.py
#
#   Purpose:    To demonstrate how to use the MSK_relaxprimal function to
#               locate the cause of an infeasibility.
#
#   Syntax: On command line
#           python feasrepairex1.py feasrepair.lp
#           feasrepair.lp is located in mosek\<version>\tools\examples.


import sys
import mosek

# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main (inputfile):
    # Make a MOSEK environment
    with mosek.Env () as env:
      with env.Task(0,0) as task:
        # Attach a printer to the task
        task.set_Stream (mosek.streamtype.log, streamprinter)

        # Read data
        task.readdata(inputfile)

        task.putintparam(mosek.iparam.log_feas_repair,3)

        task.primalrepair(None,None,None,None)

        sum_viol = task.getdouinf(mosek.dinfitem.primal_repair_penalty_obj)
        print ("Minimized sum of violations = %e" % sum_viol)

        task.optimize()

        task.solutionsummary(mosek.streamtype.msg)

```

```
# call the main function
try:
    main (sys.argv[1])
except Exception as e:
    print (e)
    raise
```

will produce the same output as the command line tool discussed in Section 14.3.1.

Chapter 15

Sensitivity analysis

15.1 Introduction

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

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

15.2 Restrictions

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

15.3 References

The book [1] discusses the classical sensitivity analysis in Chapter 10 whereas the book [17] presents a modern introduction to sensitivity analysis. Finally, it is recommended to read the short paper [18] to avoid some of the pitfalls associated with sensitivity analysis.

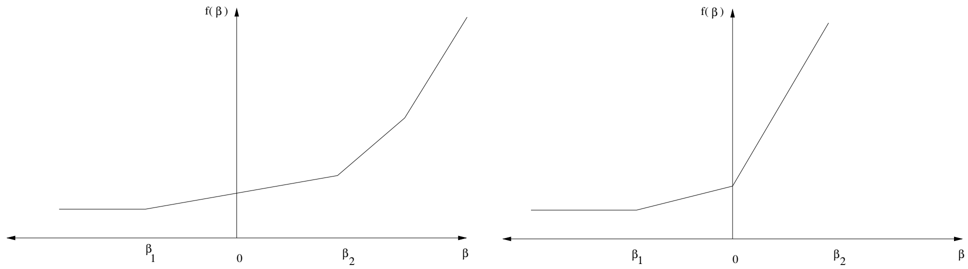

Figure 15.1: The optimal value function $f_{l_i^c}(\beta)$. Left: $\beta = 0$ is in the interior of linearity interval. Right: $\beta = 0$ is a breakpoint.

15.4 Sensitivity analysis for linear problems

15.4.1 The optimal objective value function

Assume that we are given the problem

$$\begin{array}{rclccc} z(l^c, u^c, l^x, u^x, c) & = & \text{minimize} & & c^T x & \\ & & \text{subject to} & l^c \leq & Ax & \leq u^c, \\ & & & & l^x \leq x \leq u^x, & \end{array} \tag{15.1}$$

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

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

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

$$f_{l_i^c}(\beta) \tag{15.2}$$

shows the optimal objective value as a function of β. Please note that a change in β corresponds to a perturbation in l_i^c and hence (15.2) shows the optimal objective value as a function of l_i^c.

It is possible to prove that the function (15.2) is a piecewise linear and convex function, i.e. the function may look like the illustration in Figure 15.1. Clearly, if the function $f_{l_i^c}(\beta)$ does not change much when β is changed, then we can conclude that the optimal objective value is insensitive to changes in l_i^c. Therefore, we are interested in the rate of change in $f_{l_i^c}(\beta)$ for small changes in β — specificly the gradient

$$f'_{l_i^c}(0),$$

which is called the *shadow price*related to l_i^c. The shadow price specifies how the objective value changes for small changes in β around zero. Moreover, we are interested in the *linearity interval*

$$\beta \in [\beta_1, \beta_2]$$

for which

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

Since $f_{l_i^c}$ is not a smooth function $f'_{l_i^c}$ may not be defined at 0, as illustrated by the right example in figure 15.1. In this case we can define a left and a right shadow price and a left and a right linearity interval.

The function $f_{l_i^c}$ considered only changes in l_i^c. We can define similar functions for the remaining parameters of the z defined in (15.1) as well:

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

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

15.4.1.1 Equality constraints

In MOSEK a constraint can be specified as either an equality constraint or a ranged constraint. If constraint i is an equality constraint, we define the optimal value function for this as

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

Thus for an equality constraint the upper and the lower bounds (which are equal) are perturbed simultaneously. Therefore, MOSEK will handle sensitivity analysis differently for a ranged constraint with $l_i^c = u_i^c$ and for an equality constraint.

15.4.2 The basis type sensitivity analysis

The classical sensitivity analysis discussed in most textbooks about linear optimization, e.g. [1], is based on an optimal basic solution or, equivalently, on an optimal basis. This method may produce misleading results [17] but is **computationally cheap**. Therefore, and for historical reasons this method is available in MOSEK We will now briefly discuss the basis type sensitivity analysis. Given an optimal basic solution which provides a partition of variables into basic and non-basic variables, the basis type sensitivity analysis computes the linearity interval $[\beta_1, \beta_2]$ so that the basis remains optimal for the perturbed problem. A shadow price associated with the linearity interval is also computed. However, it is well-known that an optimal basic solution may not be unique and therefore the result depends on the optimal basic solution employed in the sensitivity analysis. This implies that the computed interval is only a subset of the largest interval for which the shadow price is constant. Furthermore, the optimal objective value function might have a breakpoint for $\beta = 0$. In this case the basis type sensitivity method will only provide a subset of either the left or the right linearity interval.

In summary, the basis type sensitivity analysis is computationally cheap but does not provide complete information. Hence, the results of the basis type sensitivity analysis should be used with care.

15.4.3 The optimal partition type sensitivity analysis

Another method for computing the complete linearity interval is called the *optimal partition type sensitivity analysis*. The main drawback of the optimal partition type sensitivity analysis is that it is computationally expensive compared to the basis type analysts. This type of sensitivity analysis is currently provided as an experimental feature in MOSEK.

Given the optimal primal and dual solutions to (15.1), i.e. x^* and $((s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ the optimal objective value is given by

$$z^* := c^T x^*.$$

The left and right shadow prices σ_1 and σ_2 for l_i^c are given by this pair of optimization problems:

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

and

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

These two optimization problems make it easy to interpret the shadow price. Indeed, if $((s_l^c)^*, (s_u^c)^*, (s_l^x)^*, (s_u^x)^*)$ is an arbitrary optimal solution then

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

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

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

and

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

The linearity intervals and shadow prices for u_i^c, l_j^x, and u_j^x are computed similarly to l_i^c.

The left and right shadow prices for c_j denoted σ_1 and σ_2 respectively are computed as follows:

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

and

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

Once again the above two optimization problems make it easy to interpret the shadow prices. Indeed, if x^* is an arbitrary primal optimal solution, then

$$x_j^* \in [\sigma_1, \sigma_2].$$

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

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

and

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

15.4.4 Example: Sensitivity analysis

As an example we will use the following transportation problem. Consider the problem of minimizing the transportation cost between a number of production plants and stores. Each plant supplies a number of goods and each store has a given demand that must be met. Supply, demand and cost of transportation per unit are shown in Figure 15.2. If we denote the number of transported goods from location i to location j by x_{ij}, problem can be formulated as the linear optimization problem minimize

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

subject to

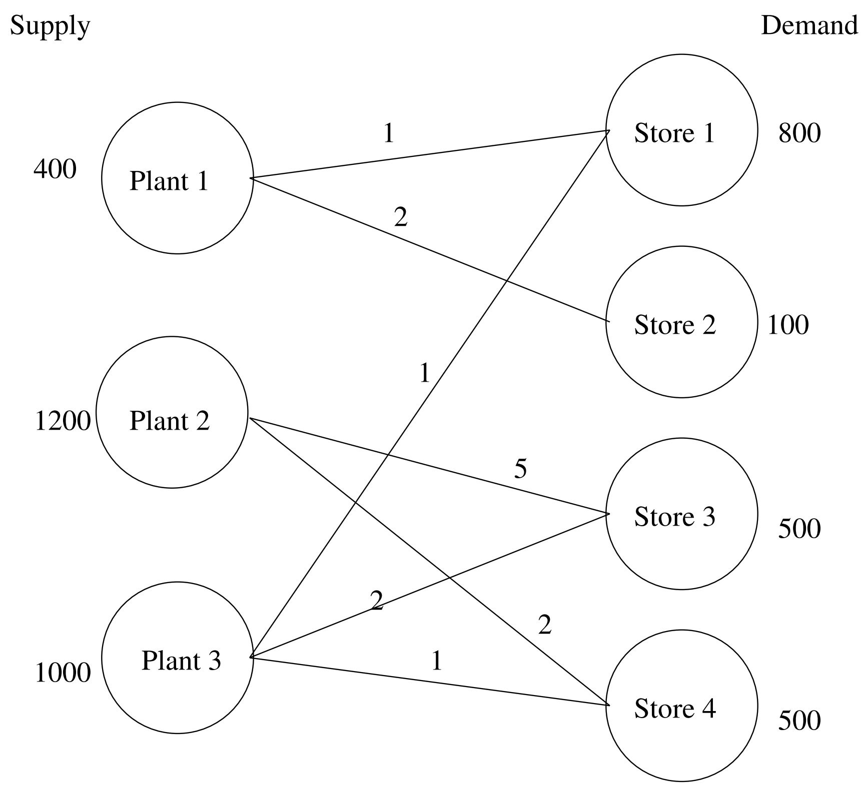

Figure 15.2: Supply, demand and cost of transportation.

Basis type

Con.	β_1	β_2	σ_1	σ_2
1	-300.00	0.00	3.00	3.00
2	-700.00	$+\infty$	0.00	0.00
3	-500.00	0.00	3.00	3.00
4	-0.00	500.00	4.00	4.00
5	-0.00	300.00	5.00	5.00
6	-0.00	700.00	5.00	5.00
7	-500.00	700.00	2.00	2.00
Var.	β_1	β_2	σ_1	σ_2
x_{11}	$-\infty$	300.00	0.00	0.00
x_{12}	$-\infty$	100.00	0.00	0.00
x_{23}	$-\infty$	0.00	0.00	0.00
x_{24}	$-\infty$	500.00	0.00	0.00
x_{31}	$-\infty$	500.00	0.00	0.00
x_{33}	$-\infty$	500.00	0.00	0.00
x_{34}	-0.000000	500.00	2.00	2.00

Optimal partition type

Con.	β_1	β_2	σ_1	σ_2
1	-300.00	500.00	3.00	1.00
2	-700.00	$+\infty$	-0.00	-0.00
3	-500.00	500.00	3.00	1.00
4	-500.00	500.00	2.00	4.00
5	-100.00	300.00	3.00	5.00
6	-500.00	700.00	3.00	5.00
7	-500.00	700.00	2.00	2.00
Var.	β_1	β_2	σ_1	σ_2
x_{11}	$-\infty$	300.00	0.00	0.00
x_{12}	$-\infty$	100.00	0.00	0.00
x_{23}	$-\infty$	500.00	0.00	2.00
x_{24}	$-\infty$	500.00	0.00	0.00
x_{31}	$-\infty$	500.00	0.00	0.00
x_{33}	$-\infty$	500.00	0.00	0.00
x_{34}	$-\infty$	500.00	0.00	2.00

Table 15.1: Ranges and shadow prices related to bounds on constraints and variables. Left: Results for the basis type sensitivity analysis. Right: Results for the optimal partition type sensitivity analysis.

$$
\begin{array}{ccccccccccccccc}
x_{11} & + & x_{12} & & & & & & & & & & & \leq & 400, \\
 & & & & x_{23} & + & x_{24} & & & & & & & \leq & 1200, \\
 & & & & & & & & x_{31} & + & x_{33} & + & x_{34} & \leq & 1000, \\
x_{11} & & & & & & & + & x_{31} & & & & & = & 800, \\
 & & x_{12} & & & & & & & & & & & = & 100, \\
 & & & & x_{23} & + & & & & & x_{33} & & & = & 500, \\
 & & & & & & x_{24} & + & & & & & x_{34} & = & 500, \\
x_{11}, & & x_{12}, & & x_{23}, & & x_{24}, & & x_{31}, & & x_{33}, & & x_{34} & \geq & 0.
\end{array} \tag{15.3}
$$

The basis type and the optimal partition type sensitivity results for the transportation problem are shown in Table 15.1 and 15.2 respectively. Examining the results from the optimal partition type sensitivity analysis we see that for constraint number 1 we have $\sigma_1 \neq \sigma_2$ and $\beta_1 \neq \beta_2$. Therefore, we have a left linearity interval of $[-300, 0]$ and a right interval of $[0, 500]$. The corresponding left and right shadow prices are 3 and 1 respectively. This implies that if the upper bound on constraint 1 increases by

$$\beta \in [0, \beta_1] = [0, 500]$$

then the optimal objective value will decrease by the value

$$\sigma_2 \beta = 1\beta.$$

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

Basis type

Var.	β_1	β_2	σ_1	σ_2
c_1	$-\infty$	3.00	300.00	300.00
c_2	$-\infty$	∞	100.00	100.00
c_3	-2.00	∞	0.00	0.00
c_4	$-\infty$	2.00	500.00	500.00
c_5	-3.00	∞	500.00	500.00
c_6	$-\infty$	2.00	500.00	500.00
c_7	-2.00	∞	0.00	0.00

Optimal partition type

Var.	β_1	β_2	σ_1	σ_2
c_1	$-\infty$	3.00	300.00	300.00
c_2	$-\infty$	∞	100.00	100.00
c_3	-2.00	∞	0.00	0.00
c_4	$-\infty$	2.00	500.00	500.00
c_5	-3.00	∞	500.00	500.00
c_6	$-\infty$	2.00	500.00	500.00
c_7	-2.00	∞	0.00	0.00

Table 15.2: Ranges and shadow prices related to the objective coefficients. Left: Results for the basis type sensitivity analysis. Right: Results for the optimal partition type sensitivity analysis.

$$\beta \in [0, 300]$$

then the optimal objective value will increase by the value

$$\sigma_1 \beta = 3\beta.$$

15.5 Sensitivity analysis from the MOSEK API

MOSEK provides the functions Task.primalsensitivity and Task.dualsensitivity for performing sensitivity analysis. The code below gives an example of its use.

[sensitivity.py]

```
##
#   Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
#
#   File:      sensitivity.py
#
#   Purpose:   To demonstrate how to perform sensitivity
#   analysis from the API on a small problem:
#
#   minimize
#
#   obj: +1 x11 + 2 x12 + 5 x23 + 2 x24 + 1 x31 + 2 x33 + 1 x34
#   st
#   c1:   +  x11 +   x12                                        <= 400
#   c2:                  +   x23 +   x24                        <= 1200
#   c3:                                  +   x31 +   x33 +   x34  <= 1000
#   c4:   +  x11                         +   x31                = 800
#   c5:          +   x12                                        = 100
#   c6:                  +   x23                 +   x33        = 500
#   c7:                          +   x24                 +   x34 = 500
#
#   The example uses basis type sensitivity analysis.
##

```

```python

import sys

import mosek
# If numpy is installed, use that, otherwise use the
# Mosek's array module.
try:
    from numpy import array,zeros,ones
except ImportError:
    from mosek.array import array, zeros, ones


# Since the actual value of Infinity is ignores, we define it solely
# for symbolic purposes:
inf = 0.0

# Define a stream printer to grab output from MOSEK
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()



# We might write everything directly as a script, but it looks nicer
# to create a function.
def main ():
    # Create a MOSEK environment
    env = mosek.Env ()
    # Attach a printer to the environment
    env.set_Stream (mosek.streamtype.log, streamprinter)

    # Create a task
    task = env.Task(0,0)
    # Attach a printer to the task
    task.set_Stream (mosek.streamtype.log, streamprinter)

    # Set up data

    bkc = [ mosek.boundkey.up,mosek.boundkey.up,
            mosek.boundkey.up,mosek.boundkey.fx,
            mosek.boundkey.fx,mosek.boundkey.fx,
            mosek.boundkey.fx ]
    blc = [ -inf,  -inf,  -inf,  800., 100., 500., 500. ]
    buc = [  400., 1200., 1000., 800., 100., 500., 500. ]

    bkx = [ mosek.boundkey.lo,mosek.boundkey.lo,
            mosek.boundkey.lo,mosek.boundkey.lo,
            mosek.boundkey.lo,mosek.boundkey.lo,
            mosek.boundkey.lo ]
    c   = [ 1.0,2.0,5.0,2.0,1.0,2.0,1.0 ]
    blx = [ 0.0,0.0,0.0,0.0,0.0,0.0,0.0 ]
    bux = [ inf,inf,inf,inf,inf,inf,inf ]

    ptrb = [ 0,2,4,6, 8,10,12 ]
    ptre = [ 2,4,6,8,10,12,14 ]
    sub  = [ 0,3,0,4,1,5,1,6,2,3,2,5,2,6 ]

    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 ]

    numcon = len(bkc)
    numvar = len(bkx)
    numanz = len(val)

    # Input linear data
    task.inputdata(numcon,numvar,
                   c,0.0,
                   ptrb, ptre,  sub,  val,
                   bkc, blc, buc,
                   bkx, blx, bux)
    # Set objective sense
    task.putobjsense(mosek.objsense.minimize)

    # Optimize
    task.optimize();

    # Analyze upper bound on c1 and the equality constraint on c4
    subi  = [ 0, 3 ]
    marki = [ mosek.mark.up, mosek.mark.up ]

    # Analyze lower bound on the variables x12 and x31
    subj  = [ 1, 4 ]
    markj = [ mosek.mark.lo, mosek.mark.lo ]

    leftpricei  = zeros(2,float)
    rightpricei = zeros(2,float)
    leftrangei  = zeros(2,float)
    rightrangei = zeros(2,float)
    leftpricej  = zeros(2,float)
    rightpricej = zeros(2,float)
    leftrangej  = zeros(2,float)
    rightrangej = zeros(2,float)


    task.primalsensitivity( subi,
                            marki,
                            subj,
                            markj,
                            leftpricei,
                            rightpricei,
                            leftrangei,
                            rightrangei,
                            leftpricej,
                            rightpricej,
                            leftrangej,
                            rightrangej)

    print ('Results from sensitivity analysis on bounds:')
    print ('\tleftprice  | rightprice | leftrange  | rightrange ' )
    print ('For constraints:')

    for i in range(2):
        print ('\t%10f   %10f   %10f   %10f' % (leftpricei[i],
                                               rightpricei[i],
                                               leftrangei[i],
                                               rightrangei[i]))
```

```

    print ('For variables:')
    for i in range(2):
        print ('\t%10f   %10f   %10f   %10f' % (leftpricej[i],
                                               rightpricej[i],
                                               leftrangej[i],
                                               rightrangej[i]))


    leftprice  = zeros(2,float)
    rightprice = zeros(2,float)
    leftrange  = zeros(2,float)
    rightrange = zeros(2,float)
    subc       = array([ 2, 5 ])

    task.dualsensitivity( subc,
                          leftprice,
                          rightprice,
                          leftrange,
                          rightrange)

    print ('Results from sensitivity analysis on objective coefficients:')

    for i in range(2):
        print ('\t%10f   %10f   %10f   %10f' % (leftprice[i],
                                               rightprice[i],
                                               leftrange[i],
                                               rightrange[i]))

    return None

# call the main function
try:
    main ()
except mosek.Exception as e:
    print ("ERROR: %s" % str(e.errno))
    if e.msg is not None:
        print ("\t%s" % e.msg)
    sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)
```

15.6 Sensitivity analysis with the command line tool

A sensitivity analysis can be performed with the MOSEK command line tool using the command

```
mosek myproblem.mps -sen sensitivity.ssp
```

where `sensitivity.ssp` is a file in the format described in the next section. The `ssp` file describes which parts of the problem the sensitivity analysis should be performed on.

By default results are written to a file named `myproblem.sen`. If necessary, this filename can be

```
* A comment
BOUNDS CONSTRAINTS
 U|L|LU [cname1]
 U|L|LU [cname2]-[cname3]
BOUNDS VARIABLES
 U|L|LU [vname1]
 U|L|LU [vname2]-[vname3]
OBJECTIVE VARIABLES
 [vname1]
 [vname2]-[vname3]
```

Figure 15.3: The sensitivity analysis file format.

changed by setting the

```
MSK_SPAR_SENSITIVITY_RES_FILE_NAME
```

parameter By default a basis type sensitivity analysis is performed. However, the type of sensitivity analysis (basis or optimal partition) can be changed by setting the parameter

```
MSK_IPAR_SENSITIVITY_TYPE
```

appropriately. Following values are accepted for this parameter:

- MSK_SENSITIVITY_TYPE_BASIS
- MSK_SENSITIVITY_TYPE_OPTIMAL_PARTITION

It is also possible to use the command line

```
mosek myproblem.mps -d MSK_IPAR_SENSITIVITY_ALL MSK_ON
```

in which case a sensitivity analysis on all the parameters is performed.

15.6.1 Sensitivity analysis specification file

MOSEK employs an MPS like file format to specify on which model parameters the sensitivity analysis should be performed. As the optimal partition type sensitivity analysis can be computationally expensive it is important to limit the sensitivity analysis. The format of the sensitivity specification file is shown in figure 15.3, where capitalized names are keywords, and names in brackets are names of the constraints and variables to be included in the analysis.

The sensitivity specification file has three sections, i.e.

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

```
* 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 15.4: Example of the sensitivity file format.

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

Recall from Section 15.4.1.1 that equality constraints are handled in a special way. Sensitivity analysis of an equality constraint can be specified with either `L`, `U`, or `LU`, all indicating the same, namely that upper and lower bounds (which are equal) are perturbed simultaneously.

As an example consider

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

which requests that sensitivity analysis is performed on the lower bound of the constraint named `cons1`, on the upper bound of the constraint named `cons2`, and on both lower and upper bound on the constraints named `cons3` to `cons6`.

It is allowed to use indexes instead of names, for instance

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

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

15.6.2 Example: Sensitivity analysis from command line

As an example consider the `sensitivity.ssp` file shown in Figure 15.4. The command

```
mosek transport.lp -sen sensitivity.ssp  -d iparam.sensitivity_type sensitivitytype.basis
```

produces the `transport.sen` file shown below.

```
BOUNDS CONSTRAINTS
INDEX    NAME             BOUND    LEFTRANGE        RIGHTRANGE       LEFTPRICE        RIGHTPRICE
0        c1               UP       -6.574875e-18    5.000000e+02     1.000000e+00     1.000000e+00
```

```
2        c3                UP     -6.574875e-18    5.000000e+02    1.000000e+00    1.000000e+00
3        c4                FIX    -5.000000e+02    6.574875e-18    2.000000e+00    2.000000e+00
4        c5                FIX    -1.000000e+02    6.574875e-18    3.000000e+00    3.000000e+00
5        c6                FIX    -5.000000e+02    6.574875e-18    3.000000e+00    3.000000e+00

BOUNDS VARIABLES
INDEX    NAME              BOUND  LEFTRANGE        RIGHTRANGE      LEFTPRICE       RIGHTPRICE
2        x23               LO     -6.574875e-18    5.000000e+02    2.000000e+00    2.000000e+00
3        x24               LO     -inf             5.000000e+02    0.000000e+00    0.000000e+00
4        x31               LO     -inf             5.000000e+02    0.000000e+00    0.000000e+00
0        x11               LO     -inf             3.000000e+02    0.000000e+00    0.000000e+00

OBJECTIVE VARIABLES
INDEX    NAME                     LEFTRANGE        RIGHTRANGE      LEFTPRICE       RIGHTPRICE
0        x11                      -inf             1.000000e+00    3.000000e+02    3.000000e+02
2        x23                      -2.000000e+00    +inf            0.000000e+00    0.000000e+00
```

15.6.3 Controlling log output

Setting the parameter

`MSK_IPAR_LOG_SENSITIVITY`

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

The parameter

`MSK_IPAR_LOG_SENSITIVITY_OPT`

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

Appendix A

API reference

This chapter lists all functionality in the MOSEK Python API.

Environment constructor and destructor

- Env()
- Env.__del__()

Task constructors and destructor

- Task(Env env)
- Task(Task task)
- Task(Env env, int maxnumcon, int maxnumvar)
- Task.__del__()

Exceptions

Callback functions

Bounds

- `Task.putconboundlist`
- `Task.putvarboundlist`

Operate on data associated with the conic constraints

- `Task.appendcone`
- `Task.putcone`
- `Task.removecones`

Operate on data associated with the constraints

- Task.appendcons
- Task.getnumcon
- Task.putconbound
- Task.removecons

Task diagnostics

- Task.checkconvexity
- Task.getprobtype
- Task.optimizersummary
- Task.printdata
- Task.printparam
- Task.solutionsummary
- Task.updatesolutioninfo

Reading and writing data files

- Task.readsolution
- Task.writedata
- Task.writesolution

Obtaining solution values

- Task.getbarsj
- Task.getbarxj
- Task.getskcslice
- Task.getskxslice
- Task.getslcslice
- Task.getslxslice
- Task.getsnxslice
- Task.getsucslice
- Task.getsuxslice
- Task.getxcslice
- Task.getxxslice
- Task.getyslice

Diagnosing infeasibility

- Task.getinfeasiblesubproblem
- Task.primalrepair

- Task.relaxprimal

Obtain information about the solutions.

- Task.getdualobj
- Task.getdviolbarvar
- Task.getdviolcon
- Task.getdviolcones
- Task.getdviolvar
- Task.getprimalobj
- Task.getprosta
- Task.getpviolbarvar
- Task.getpviolcon
- Task.getpviolcones
- Task.getpviolvar
- Task.getsolsta
- Task.getsolutioninfo
- Task.solutiondef

Linear algebra utility functions for performing linear algebra operations

- Env.axpy
- Env.dot
- Env.gemm
- Env.gemv
- Env.potrf
- Env.syeig
- Env.syevd
- Env.syrk

Management of the environment

- Env.licensecleanup
- Env.putlicensedebug
- Env.putlicensepath
- Env.putlicensewait

Naming

- Task.putbarvarname

- Task.putconename
- Task.putconname
- Task.putobjname
- Task.puttaskname
- Task.putvarname

Operate on data associated with objective.

- Task.putcfix
- Task.putobjsense

Optimization

- Task.optimizeconcurrent
- Task.optimize

Setting task parameter values

- Task.putdouparam
- Task.putintparam
- Task.putstrparam

Inputting solution values

- Task.putbarsj
- Task.putbarxj
- Task.putskcslice
- Task.putskxslice
- Task.putslcslice
- Task.putslxslice
- Task.putsnxslice
- Task.putsolution
- Task.putsolutioni
- Task.putsucslice
- Task.putsuxslice
- Task.putxcslice
- Task.putxxslice
- Task.putyslice

Operate on data associated with scalar variables

- Task.appendvars

- Task.getnumvar
- Task.putacol
- Task.putaij
- Task.putarow
- Task.putcj
- Task.putqcon
- Task.putqconk
- Task.putqobj
- Task.putqobjij
- Task.putvarbound
- Task.putvartype
- Task.removevars

Sensitivity analysis

- Task.dualsensitivity
- Task.primalsensitivity
- Task.sensitivityreport

Optimizer statistics

- Task.getdouinf
- Task.getintinf
- Task.getlintinf

Output stream functions

- Task.linkfiletostream

Operate on data associated with symmetric matrix variables

- Task.appendbarvars
- Task.appendsparsesymmat
- Task.putbaraij
- Task.putbarcj

Alphabetic list of functions

A.1 Exceptions

`mosek.Exception(Exception)`

Base exception class for all MOSEK exceptions.

`mosek.Error(mosek.Exception)`

Exception class used for all error response codes from MOSEK.

`mosek.Warning(mosek.Exception)`

Exception class used for all warning response codes from MOSEK.

A.2 Class Task

A.2.1 Task.analyzenames()

```
Task.analyzenames(
    whichstream,
    nametype)
```

Analyze the names and issue an error for the first invalid name.

Arguments

`nametype` : `nametype`
The type of names e.g. valid in MPS or LP files.

`whichstream` : `streamtype`
Index of the stream.

Description:

The function analyzes the names and issue an error if a name is invalid.

A.2.2 Task.analyzeproblem()

```
Task.analyzeproblem(whichstream)
```

Analyze the data of a task.

Arguments

`whichstream` : `streamtype`
Index of the stream.

Description:

The function analyzes the data of task and writes out a report.

A.2.3 Task.analyzesolution()

```
Task.analyzesolution(
    whichstream,
    whichsol)
```

Print information related to the quality of the solution.

Arguments

whichsol : soltype
: Selects a solution.

whichstream : streamtype
: Index of the stream.

Description:

Print information related to the quality of the solution and other solution statistics.

By default this function prints information about the largest infeasibilites in the solution, the primal (and possibly dual) objective value and the solution status.

Following parameters can be used to configure the printed statistics:

- iparam.ana_sol_basis. Enables or disables printing of statistics specific to the basis solution (condition number, number of basic variables etc.). Default is on.
- iparam.ana_sol_print_violated. Enables or disables listing names of all constraints (both primal and dual) which are violated by the solution. Default is off.
- dparam.ana_sol_infeas_tol. The tolerance defining when a constraint is considered violated. If a constraint is violated more than this, it will be listed in the summary.

See also

- Task.getpviolcon Computes the violation of a primal solution for a list of xc variables.
- Task.getpviolvar Computes the violation of a primal solution for a list of x variables.
- Task.getpviolbarvar Computes the violation of a primal solution for a list of barx variables.
- Task.getpviolcones Computes the violation of a solution for set of conic constraints.
- Task.getdviolcon Computes the violation of a dual solution associated with a set of constraints.
- Task.getdviolvar Computes the violation of a dual solution associated with a set of x variables.

- Task.getdviolbarvar Computes the violation of dual solution for a set of barx variables.
- Task.getdviolcones Computes the violation of a solution for set of dual conic constraints.
- iparam.ana_sol_basis Controls whether the basis matrix is analyzed in solaution analyzer.

A.2.4 Task.appendbarvars()

```
Task.appendbarvars(dim)
```

Appends a semidefinite variable of dimension dim to the problem.

Arguments

dim : int[]
Dimension of symmetric matrix variables to be added.

Description:

Appends a positive semidefinite matrix variable of dimension `dim` to the problem.

A.2.5 Task.appendcone()

```
Task.appendcone(
    conetype,
    conepar,
    submem)
```

Appends a new cone constraint to the problem.

Arguments

conepar : double
This argument is currently not used. Can be set to 0.0.

conetype : conetype
Specifies the type of the cone.

submem : int[]
Variable subscripts of the members in the cone.

Description:

Appends a new conic constraint to the problem. Hence, add a constraint

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

to the problem where $\mathcal{C}$ is a convex cone. $\hat{x}$ is a subset of the variables which will be specified by the argument `submem`.

Depending on the value of `conetype` this function appends a normal (`conetype.quad`) or rotated quadratic cone (`conetype.rquad`). Define

$$\hat{x} = x_{\texttt{submem}[0]}, \ldots, x_{\texttt{submem}[\texttt{nummem}-1]}$$

. Depending on the value of `conetype` this function appends one of the constraints:

- Quadratic cone (`conetype.quad`) :

$$\hat{x}_0 \geq \sqrt{\sum_{i=1}^{i<\texttt{nummem}} \hat{x}_i^2}$$

- Rotated quadratic cone (`conetype.rquad`) :

$$2\hat{x}_0\hat{x}_1 \geq \sum_{i=2}^{i<\texttt{nummem}} \hat{x}_i^2, \quad \hat{x}_0, \hat{x}_1 \geq 0$$

Please note that the sets of variables appearing in different conic constraints must be disjoint.

For an explained code example see Section 5.3.

See also

- `Task.appendconeseq` Appends a new conic constraint to the problem.
- `Task.appendconesseq` Appends multiple conic constraints to the problem.

A.2.6 `Task.appendconeseq()`

```
Task.appendconeseq(
    conetype,
    conepar,
    nummem,
    j)
```

Appends a new conic constraint to the problem.

Arguments

`conepar : double`
  This argument is currently not used. Can be set to 0.0.

`conetype : conetype`
  Specifies the type of the cone.

`j` : `int`
Index of the first variable in the conic constraint.

`nummem` : `int`
Dimension of the conic constraint to be appended.

Description:

Appends a new conic constraint to the problem. The function assumes the members of cone are sequential where the first emeber has index `j` and the last `j+nummem-1`.

See also

- `Task.appendcone` Appends a new cone constraint to the problem.
- `Task.appendconesseq` Appends multiple conic constraints to the problem.

A.2.7 `Task.appendconesseq()`

```
Task.appendconesseq(
    conetype,
    conepar,
    nummem,
    j)
```

Appends multiple conic constraints to the problem.

Arguments

`conepar` : `double[]`
This argument is currently not used. Can be set to 0.0.

`conetype` : `conetype`
Specifies the type of the cone.

`j` : `int`
Index of the first variable in the first cone to be appended.

`nummem` : `int[]`
Number of member variables in the cone.

Description:

Appends a number conic constraints to the problem. The kth cone is assumed to be of dimension `nummem[k]`. Moreover, is is asummed that the first variable of the first cone has index j and the index of the variable in each cone are sequential. Finally, it assumed in the second cone is the last index of first cone plus one and so forth.

See also

- `Task.appendcone` Appends a new cone constraint to the problem.
- `Task.appendconeseq` Appends a new conic constraint to the problem.

A.2.8 `Task.appendcons()`

```
Task.appendcons(num)
```

Appends a number of constraints to the optimization task.

Arguments

`num : int`
Number of constraints which should be appended.

Description:

Appends a number of constraints to the model. Appended constraints will be declared free. Please note that MOSEK will automatically expand the problem dimension to accommodate the additional constraints.

See also

- `Task.removecons` The function removes a number of constraints.

A.2.9 `Task.appendsparsesymmat()`

```
idx = Task.appendsparsesymmat(
    dim,
    subi,
    subj,
    valij)
```

Appends a general sparse symmetric matrix to the vector E of symmetric matrixes.

Arguments

`dim : int`
Dimension of the symmetric matrix that is appended.

`idx : long`
Each matrix that is appended to E is assigned a unique index i.e. `idx` that can be used for later reference.

`subi : int[]`
Row subscript in the triplets.

`subj : int[]`
Column subscripts in the triplets.

`valij : double[]`
Values of each triplet.

Description:

MOSEK maintains a storage of symmetric data matrixes that is used to build the $\bar{c}$ and $\bar{A}$. The storage can be thought of as a vector of symmetric matrixes denoted E. Hence, E_i is a symmetric matrix of certain dimension.

This functionsappends a general sparse symmetric matrix on triplet form to the vector E of symmetric matrixes. The vectors `subi`, `subj`, and `valij` contains the row subscripts, column subscripts and values of each element in the symmetric matrix to be appended. Since the matrix that is appended is symmetric then only the lower triangular part should be specified. Moreover, duplicates are not allowed.

Observe the function reports the index (position) of the appended matrix in E. This index should be used for later references to the appended matrix.

A.2.10 `Task.appendstat()`

```
Task.appendstat()
```

Appends a record the statistics file.

Description:

Appends a record to the statistics file.

A.2.11 `Task.appendvars()`

```
Task.appendvars(num)
```

Appends a number of variables to the optimization task.

Arguments

`num : int`
Number of variables which should be appended.

Description:

Appends a number of variables to the model. Appended variables will be fixed at zero. Please note that MOSEK will automatically expand the problem dimension to accommodate the additional variables.

See also

- `Task.removevars` The function removes a number of variables.

A.2.12 `Task.basiscond()`

```
nrmbasis,nrminvbasis = Task.basiscond()
```

Computes conditioning information for the basis matrix.

Arguments

`nrmbasis : double`
An estimate for the 1 norm of the basis.

`nrminvbasis : double`
An estimate for the 1 norm of the inverse of the basis.

Description:

If a basic solution is available and it defines a nonsingular basis, then this function computes the 1-norm estimate of the basis matrix and an 1-norm estimate for the inverse of the basis matrix. The 1-norm estimates are computed using the method outlined in [19].

By defintion the 1-norm condition number of a matrix B is defined as

$$\kappa_1(B) := \|B\|_1 \left\|B^{-1}\right\|.$$

Moreover, the larger the condition number is the harder it is to solve linear equation systems involving B. Given estimates for $\|B\|_1$ and $\left\|B^{-1}\right\|_1$ it is also possible to estimate $\kappa_1(B)$.

A.2.13 `Task.checkconvexity()`

```
Task.checkconvexity()
```

Checks if a quadratic optimization problem is convex.

Description:

This function checks if a quadratic optimization problem is convex. The amount of checking is controlled by `iparam.check_convexity`.

The function throws an exception if the problem is not convex.

See also

- `iparam.check_convexity` Specify the level of convexity check on quadratic problems

A.2.14 Task.checkmem()

```
Task.checkmem(
    file,
    line)
```

Checks the memory allocated by the task.

Arguments

`file` : `str`
: File from which the function is called.

`line` : `int`
: Line in the file from which the function is called.

Description:

Checks the memory allocated by the task.

A.2.15 Task.chgbound()

```
Task.chgbound(
    accmode,
    i,
    lower,
    finite,
    value)
```

Changes the bounds for one constraint or variable.

Arguments

`accmode` : `accmode`
: Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

`finite` : `int`
: If non-zero, then `value` is assumed to be finite.

`i` : `int`
: Index of the constraint or variable for which the bounds should be changed.

`lower` : `int`
: If non-zero, then the lower bound is changed, otherwise the upper bound is changed.

`value` : `double`
: New value for the bound.

Description:

Changes a bound for one constraint or variable. If `accmode` equals `accmode.con`, a constraint bound is changed, otherwise a variable bound is changed.

If `lower` is non-zero, then the lower bound is changed as follows:

$$\text{new lower bound} = \begin{cases} -\infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Otherwise if `lower` is zero, then

$$\text{new upper bound} = \begin{cases} \infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Please note that this function automatically updates the bound key for bound, in particular, if the lower and upper bounds are identical, the bound key is changed to `fixed`.

See also

- `Task.putbound` Changes the bound for either one constraint or one variable.
- `dparam.data_tol_bound_inf` Data tolerance threshold.
- `dparam.data_tol_bound_wrn` Data tolerance threshold.

A.2.16 `Task.commitchanges()`

```
Task.commitchanges()
```

Commits all cached problem changes.

Description:

Commits all cached problem changes to the task. It is usually not necessary explicitly to call this function since changes will be committed automatically when required.

A.2.17 `Task.deletesolution()`

```
Task.deletesolution(whichsol)
```

Undefines a solution and frees the memory it uses.

Arguments

`whichsol` : `soltype`
Selects a solution.

Description:

Undefines a solution and frees the memory it uses.

A.2.18 `Task.dualsensitivity()`

```
Task.dualsensitivity(
    subj,
    leftpricej,
    rightpricej,
    leftrangej,
    rightrangej)
```

Performs sensitivity analysis on objective coefficients.

Arguments

`leftpricej : double[]`
`leftpricej`$[j]$ is the left shadow price for the coefficients with index `subj`[j].

`leftrangej : double[]`
`leftrangej`$[j]$ is the left range β_1 for the coefficient with index `subj`[j].

`rightpricej : double[]`
`rightpricej`$[j]$ is the right shadow price for the coefficients with index `subj`[j].

`rightrangej : double[]`
`rightrangej`$[j]$ is the right range β_2 for the coefficient with index `subj`[j].

`subj : int[]`
Index of objective coefficients to analyze.

Description:

Calculates sensitivity information for objective coefficients. The indexes of the coefficients to analyze are

$$\{\texttt{subj}[i] | i \in 0, \ldots, \texttt{numj} - 1\}$$

The results are returned so that e.g `leftprice`$[j]$ is the left shadow price of the objective coefficient with index `subj`$[j]$.

The type of sensitivity analysis to perform (basis or optimal partition) is controlled by the parameter `iparam.sensitivity_type`.

For an example, please see Section 15.5.

See also

- `Task.primalsensitivity` Perform sensitivity analysis on bounds.
- `Task.sensitivityreport` Creates a sensitivity report.
- `iparam.sensitivity_type` Controls which type of sensitivity analysis is to be performed.
- `iparam.log_sensitivity` Control logging in sensitivity analyzer.
- `iparam.log_sensitivity_opt` Control logging in sensitivity analyzer.

A.2.19 Task.getacol()

```
nzj = Task.getacol(
    j,
    subj,
    valj)
```

Obtains one column of the linear constraint matrix.

Arguments

`j` : `int`
  Index of the column.

`nzj` : `int`
  Number of non-zeros in the column obtained.

`subj` : `int[]`
  Index of the non-zeros in the column obtained.

`valj` : `double[]`
  Numerical values of the column obtained.

Description:

Obtains one column of A in a sparse format.

A.2.20 Task.getacolnumnz()

```
nzj = Task.getacolnumnz(i)
```

Obtains the number of non-zero elements in one column of the linear constraint matrix

Arguments

`i` : `int`
  Index of the column.

`nzj` : `int`
  Number of non-zeros in the jth row or column of A.

Description:

Obtains the number of non-zero elements in one column of A.

A.2.21 Task.getacolslicetrip()

```
Task.getacolslicetrip(
    first,
    last,
    subi,
    subj,
    val)
```

Obtains a sequence of columns from the coefficient matrix in triplet format.

Arguments

`first : int`
Index of the first column in the sequence.

`last : int`
Index of the last column in the sequence **plus one**.

`subi : int[]`
Constraint subscripts.

`subj : int[]`
Column subscripts.

`val : double[]`
Values.

Description:

Obtains a sequence of columns from A in a sparse triplet format.

A.2.22 Task.getaij()

```
aij = Task.getaij(
    i,
    j)
```

Obtains a single coefficient in linear constraint matrix.

Arguments

`aij : double`
The required coefficient $a_{i,j}$.

`i : int`
Row index of the coefficient to be returned.

`j` : `int`

Column index of the coefficient to be returned.

Description:

Obtains a single coefficient in A.

A.2.23 `Task.getapiecenumnz()`

```
numnz = Task.getapiecenumnz(
    firsti,
    lasti,
    firstj,
    lastj)
```

Obtains the number non-zeros in a rectangular piece of the linear constraint matrix.

Arguments

`firsti` : `int`

Index of the first row in the rectangular piece.

`firstj` : `int`

Index of the first column in the rectangular piece.

`lasti` : `int`

Index of the last row plus one in the rectangular piece.

`lastj` : `int`

Index of the last column plus one in the rectangular piece.

`numnz` : `int`

Number of non-zero A elements in the rectangular piece.

Description:

Obtains the number non-zeros in a rectangular piece of A, i.e. the number

$$|\{(i,j):\ a_{i,j} \neq 0,\ \texttt{firsti} \leq i \leq \texttt{lasti}-1,\ \texttt{firstj} \leq j \leq \texttt{lastj}-1\}|$$

where $|\mathcal{I}|$ means the number of elements in the set $\mathcal{I}$.

This function is not an efficient way to obtain the number of non-zeros in one row or column. In that case use the function `Task.getarownumnz` or `Task.getacolnumnz`.

A.2.24 `Task.getarow()`

```
nzi = Task.getarow(
    i,
    subi,
    vali)
```

Obtains one row of the linear constraint matrix.

Arguments

`i : int`
: Index of the row or column.

`nzi : int`
: Number of non-zeros in the row obtained.

`subi : int[]`
: Index of the non-zeros in the row obtained.

`vali : double[]`
: Numerical values of the row obtained.

Description:

Obtains one row of A in a sparse format.

A.2.25 `Task.getarownumnz()`

```
nzi = Task.getarownumnz(i)
```

Obtains the number of non-zero elements in one row of the linear constraint matrix

Arguments

`i : int`
: Index of the row or column.

`nzi : int`
: Number of non-zeros in the ith row of A.

Description:

Obtains the number of non-zero elements in one row of A.

A.2.26 `Task.getarowslicetrip()`

```
Task.getarowslicetrip(
    first,
    last,
    subi,
    subj,
    val)
```

Obtains a sequence of rows from the coefficient matrix in triplet format.

Arguments

`first : int`
: Index of the first row or column in the sequence.

`last : int`
: Index of the last row or column in the sequence **plus one**.

`subi : int[]`
: Constraint subscripts.

`subj : int[]`
: Column subscripts.

`val : double[]`
: Values.

Description:

Obtains a sequence of rows from A in a sparse triplet format.

A.2.27 `Task.getaslice()`

```
Task.getaslice(
    accmode,
    first,
    last,
    ptrb,
    ptre,
    sub,
    val)
```

Obtains a sequence of rows or columns from the coefficient matrix.

Arguments

`accmode : accmode`
: Defines whether a column slice or a row slice is requested.

`first : int`
Index of the first row or column in the sequence.

`last : int`
Index of the last row or column in the sequence **plus one**.

`ptrb : long[]`
`ptrb[t]` is an index pointing to the first element in the tth row or column obtained.

`ptre : long[]`
`ptre[t]` is an index pointing to the last element plus one in the tth row or column obtained.

`sub : int[]`
Contains the row or column subscripts.

`val : double[]`
Contains the coefficient values.

Description:

Obtains a sequence of rows or columns from A in sparse format.

See also

- `Task.getaslicenumnz` Obtains the number of non-zeros in a slice of rows or columns of the coefficient matrix.

A.2.28 `Task.getaslicenumnz()`

```
numnz = Task.getaslicenumnz(
    accmode,
    first,
    last)
```

Obtains the number of non-zeros in a slice of rows or columns of the coefficient matrix.

Arguments

`accmode : accmode`
Defines whether non-zeros are counted in a column slice or a row slice.

`first : int`
Index of the first row or column in the sequence.

`last : int`
Index of the last row or column **plus one** in the sequence.

`numnz : long`
Number of non-zeros in the slice.

Description:

Obtains the number of non-zeros in a slice of rows or columns of A.

A.2.29 `Task.getbarablocktriplet()`

```
num = Task.getbarablocktriplet(
    subi,
    subj,
    subk,
    subl,
    valijkl)
```

Obtains barA in block triplet form.

Arguments

`num : long`
Number of elements in the block triplet form.

`subi : int[]`
Constraint index.

`subj : int[]`
Symmetric matrix variable index.

`subk : int[]`
Block row index.

`subl : int[]`
Block column index.

`valijkl : double[]`
A list indexes of the elements from symmetric matrix storage that appers in the weighted sum.

Description:

Obtains $\bar{A}$ in block triplet form.

A.2.30 `Task.getbaraidx()`

```
i,j,num = Task.getbaraidx(
    idx,
    sub,
    weights)
```

Obtains information about an element barA.

Arguments

`i : int`
Row index of the element at position `idx`.

`idx : long`
: Position of the element in the vectorized form.

`j : int`
: Column index of the element at position `idx`.

`num : long`
: Number of terms in weighted sum that forms the element.

`sub : long[]`
: A list indexes of the elements from symmetric matrix storage that appers in the weighted sum.

`weights : double[]`
: The weights associated with each term in the weighted sum.

Description:

Obtains information about an element in $\bar{A}$. Since $\bar{A}$ is a sparse matrix of symmetric matrixes then only the nonzero elements in $\bar{A}$ are stored in order to save space. Now $\bar{A}$ is stored vectorized form i.e. as one long vector. This function makes it possible to obtain information such as the row index and the column index of a particular element of the vectorized form of $\bar{A}$.

Please observe if one element of $\bar{A}$ is inputted multiple times then it may be stored several times in vectorized form. In that case the element with the highest index is the one that is used.

A.2.31 `Task.getbaraidxij()`

```
i,j = Task.getbaraidxij(idx)
```

Obtains information about an element barA.

Arguments

`i : int`
: Row index of the element at position `idx`.

`idx : long`
: Position of the element in the vectorized form.

`j : int`
: Column index of the element at position `idx`.

Description:

Obtains information about an element in $\bar{A}$. Since $\bar{A}$ is a sparse matrix of symmetric matrixes only the nonzero elements in $\bar{A}$ are stored in order to save space. Now $\bar{A}$ is stored vectorized form i.e. as one long vector. This function makes it possible to obtain information such as the row index and the column index of a particular element of the vectorized form of $\bar{A}$.

Please note that if one element of $\bar{A}$ is inputted multiple times then it may be stored several times in vectorized form. In that case the element with the highest index is the one that is used.

A.2.32 `Task.getbaraidxinfo()`

```
num = Task.getbaraidxinfo(idx)
```

Obtains the number terms in the weighted sum that forms a particular element in barA.

Arguments

`idx : long`
: The internal position of the element that should be obtained information for.

`num : long`
: Number of terms in the weighted sum that forms the specified element in $\bar{A}$.

Description:

Each nonzero element in $\bar{A}_{ij}$ is formed as a weighted sum of symmtric matrixes. Using this function the number terms in the weigthted sum can be obtained. See description of `Task.appendsparsesymmat` for details about the weighted sum.

A.2.33 `Task.getbarasparsity()`

```
numnz = Task.getbarasparsity(idxij)
```

Obtains the sparsity pattern of the barA matrix.

Arguments

`idxij : long[]`
: Position of each nonzero element in the vectorized form of $\bar{A}_{ij}$. Hence, `idxij[k]` is the vector position of the element in row `subi[k]` and column `subj[k]` of $\bar{A}_{ij}$.

`numnz : long`
: Number of nonzero elements in $\bar{A}$.

Description:

The matrix $\bar{A}$ is assumed to be a sparse matrix of symmetric matrixes. This implies that many of elements in $\bar{A}$ is likely to be zero matrixes. Therefore, in order to save space only nonzero elementsin $\bar{A}$ are stored on vectorized form. This function is used to obtain the sparsity pattern of $\bar{A}$ and the position of each nonzero element in the vectorized form of $\bar{A}$.

A.2.34 `Task.getbarcblocktriplet()`

```
num = Task.getbarcblocktriplet(
    subj,
    subk,
    subl,
    valijkl)
```

Obtains barc in block triplet form.

Arguments

`num : long`
Number of elements in the block triplet form.

`subj : int[]`
Symmetric matrix variable index.

`subk : int[]`
Block row index.

`subl : int[]`
Block column index.

`valijkl : double[]`
A list indexes of the elements from symmetric matrix storage that appers in the weighted sum.

Description:

Obtains $\bar{C}$ in block triplet form.

A.2.35 `Task.getbarcidx()`

```
j,num = Task.getbarcidx(
    idx,
    sub,
    weights)
```

Obtains information about an element in barc.

Arguments

`idx : long`
Index of the element that should be obtained information about.

`j : int`
Row index in $\bar{c}$.

`num : long`
: Number of terms in the weighted sum.

`sub : long[]`
: Elements appearing the weighted sum.

`weights : double[]`
: Weights of terms in the weighted sum.

Description:

Obtains information about an element in $\bar{c}$.

A.2.36 Task.getbarcidxinfo()

```
num = Task.getbarcidxinfo(idx)
```

Obtains information about an element in barc.

Arguments

`idx : long`
: Index of element that should be obtained information about. The value is an index of a symmetric sparse variable.

`num : long`
: Number of terms that appears in weighted that forms the requested element.

Description:

Obtains information about about the $\bar{c}_{ij}$.

A.2.37 Task.getbarcidxj()

```
j = Task.getbarcidxj(idx)
```

Obtains the row index of an element in barc.

Arguments

`idx : long`
: Index of the element that should be obtained information about.

`j : int`
: Row index in $\bar{c}$.

Description:

Obtains the row index of an element in $\bar{c}$.

A.2.38 Task.getbarcsparsity()

```
numnz = Task.getbarcsparsity(idxj)
```

Get the positions of the nonzero elements in barc.

Arguments

`idxj : long[]`
Internal positions of the nonzeros elements in $\bar{c}$.

`numnz : long`
Number of nonzero elements in $\bar{C}$.

Description:

Internally only the nonzero elements of $\bar{c}$ is stored

in a vector. This function returns which elements $\bar{c}$ that are nonzero (in `subj`) and their internal position (in `idx`). Using the position detailed information about each nonzero $\bar{C}_j$ can be obatined using `Task.getbarcidxinfo` and `Task.getbarcidx`.

A.2.39 Task.getbarsj()

```
Task.getbarsj(
    whichsol,
    j,
    barsj)
```

Obtains the dual solution for a semidefinite variable.

Arguments

`barsj : double[]`
Value of $\bar{s}_j$.

`j : int`
Index of the semidefinite variable.

`whichsol : soltype`
Selects a solution.

Description:

Obtains the dual solution for a semidefinite variable.

A.2.40 Task.getbarvarname()

```
name = Task.getbarvarname(i)
```

Obtains a name of a semidefinite variable.

Arguments

i : int
Index.

name : str
The requested name is copied to this buffer.

Description:

Obtains a name of a semidefinite variable.

See also

- Task.getbarvarnamelen Obtains the length of a name of a semidefinite variable.

A.2.41 Task.getbarvarnameindex()

```
asgn,index = Task.getbarvarnameindex(somename)
```

Obtains the index of name of semidefinite variable.

Arguments

asgn : int
Is non-zero if the name `somename` is assigned to a semidefinite variable.

index : int
If the name `somename` is assigned to a semidefinite variable, then `index` is the name of the constraint.

somename : str
The requested name is copied to this buffer.

Description:

Obtains the index of name of semidefinite variable.

See also

- Task.getbarvarname Obtains a name of a semidefinite variable.

A.2.42 `Task.getbarvarnamelen()`

```
len = Task.getbarvarnamelen(i)
```

Obtains the length of a name of a semidefinite variable.

Arguments

`i` : `int`
Index.

`len` : `int`
Returns the length of the indicated name.

Description:

Obtains the length of a name of a semidefinite variable.

See also

- `Task.getbarvarname` Obtains a name of a semidefinite variable.

A.2.43 `Task.getbarxj()`

```
Task.getbarxj(
    whichsol,
    j,
    barxj)
```

Obtains the primal solution for a semidefinite variable.

Arguments

`barxj` : `double[]`
Value of $\bar{X}_j$.

`j` : `int`
Index of the semidefinite variable.

`whichsol` : `soltype`
Selects a solution.

Description:

Obtains the primal solution for a semidefinite variable.

A.2.44 Task.getbound()

```
bk,bl,bu = Task.getbound(
    accmode,
    i)
```

Obtains bound information for one constraint or variable.

Arguments

accmode : accmode
: Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

bk : boundkey
: Bound keys.

bl : double
: Values for lower bounds.

bu : double
: Values for upper bounds.

i : int
: Index of the constraint or variable for which the bound information should be obtained.

Description:

Obtains bound information for one constraint or variable.

A.2.45 Task.getboundslice()

```
Task.getboundslice(
    accmode,
    first,
    last,
    bk,
    bl,
    bu)
```

Obtains bounds information for a sequence of variables or constraints.

Arguments

accmode : accmode
: Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

`bk : mosek.boundkey[]`
Bound keys.

`bl : double[]`
Values for lower bounds.

`bu : double[]`
Values for upper bounds.

`first : int`
First index in the sequence.

`last : int`
Last index plus 1 in the sequence.

Description:

Obtains bounds information for a sequence of variables or constraints.

A.2.46 Task.getc()

```
Task.getc(c)
```

Obtains all objective coefficients.

Arguments

`c : double[]`
Linear terms of the objective as a dense vector. The lengths is the number of variables.

Description:

Obtains all objective coefficients c.

A.2.47 Task.getcfix()

```
cfix = Task.getcfix()
```

Obtains the fixed term in the objective.

Arguments

`cfix : double`
Fixed term in the objective.

Description:

Obtains the fixed term in the objective.

A.2.48 Task.getcj()

```
cj = Task.getcj(j)
```

Obtains one coefficient of c.

Arguments

`cj` : `double`
The value of c_j.

`j` : `int`
Index of the variable for which c coefficient should be obtained.

Description:

Obtains one coefficient of c.

See also

- `Task.getcslice` Obtains a sequence of coefficients from the objective.

A.2.49 Task.getconbound()

```
bk,bl,bu = Task.getconbound(i)
```

Obtains bound information for one constraint.

Arguments

`bk` : `boundkey`
Bound keys.

`bl` : `double`
Values for lower bounds.

`bu` : `double`
Values for upper bounds.

`i` : `int`
Index of the constraint for which the bound information should be obtained.

Description:

Obtains bound information for one constraint.

A.2.50 `Task.getconboundslice()`

```
Task.getconboundslice(
    first,
    last,
    bk,
    bl,
    bu)
```

Obtains bounds information for a slice of the constraints.

Arguments

`bk : mosek.boundkey[]`
Bound keys.

`bl : double[]`
Values for lower bounds.

`bu : double[]`
Values for upper bounds.

`first : int`
First index in the sequence.

`last : int`
Last index plus 1 in the sequence.

Description:
Obtains bounds information for a slice of the constraints.

A.2.51 `Task.getcone()`

```
conetype,conepar,nummem = Task.getcone(
    k,
    submem)
```

Obtains a conic constraint.

Arguments

`conepar : double`
This argument is currently not used. Can be set to 0.0.

`conetype : conetype`
Specifies the type of the cone.

`k` : `int`
  Index of the cone constraint.

`nummem` : `int`
  Number of member variables in the cone.

`submem` : `int[]`
  Variable subscripts of the members in the cone.

Description:
  Obtains a conic constraint.

A.2.52 Task.getconeinfo()

```
conetype,conepar,nummem = Task.getconeinfo(k)
```

Obtains information about a conic constraint.

Arguments

`conepar` : `double`
  This argument is currently not used. Can be set to 0.0.

`conetype` : `conetype`
  Specifies the type of the cone.

`k` : `int`
  Index of the conic constraint.

`nummem` : `int`
  Number of member variables in the cone.

Description:
  Obtains information about a conic constraint.

A.2.53 Task.getconename()

```
name = Task.getconename(i)
```

Obtains a name of a cone.

Arguments

`i` : `int`
  Index.

`name : str`

Is assigned the required name.

Description:

Obtains a name of a cone.

See also

- `Task.getconnamelen` Obtains the length of a name of a constraint variable.

A.2.54 `Task.getconenameindex()`

```
asgn,index = Task.getconenameindex(somename)
```

Checks whether the name somename has been assigned to any cone.

Arguments

`asgn : int`

Is non-zero if the name `somename` is assigned to a cone.

`index : int`

If the name `somename` is assigned to a cone, then `index` is the name of the cone.

`somename : str`

The name which should be checked.

Description:

Checks whether the name `somename` has been assigned to any cone. If it has been assigned to cone, then index of the cone is reported.

A.2.55 `Task.getconenamelen()`

```
len = Task.getconenamelen(i)
```

Obtains the length of a name of a cone.

Arguments

`i : int`

Index.

`len : int`

Returns the length of the indicated name.

Description:

Obtains the length of a name of a cone.

See also

- Task.getbarvarname Obtains a name of a semidefinite variable.

A.2.56 Task.getconname()

```
name = Task.getconname(i)
```

Obtains a name of a constraint.

Arguments

`i : int`
Index.

`name : str`
Is assigned the required name.

Description:

Obtains a name of a constraint.

See also

- Task.getconnamelen Obtains the length of a name of a constraint variable.

A.2.57 Task.getconnameindex()

```
asgn,index = Task.getconnameindex(somename)
```

Checks whether the name somename has been assigned to any constraint.

Arguments

`asgn : int`
Is non-zero if the name `somename` is assigned to a constraint.

`index : int`
If the name `somename` is assigned to a constraint, then `index` is the name of the constraint.

`somename : str`
The name which should be checked.

Description:

Checks whether the name `somename` has been assigned to any constraint. If it has been assigned to constraint, then index of the constraint is reported.

A.2.58 Task.getconnamelen()

```
len = Task.getconnamelen(i)
```

Obtains the length of a name of a constraint variable.

Arguments

`i : int`
Index.

`len : int`
Returns the length of the indicated name.

Description:

Obtains the length of a name of a constraint variable.

See also

- `Task.getbarvarname` Obtains a name of a semidefinite variable.

A.2.59 Task.getcslice()

```
Task.getcslice(
    first,
    last,
    c)
```

Obtains a sequence of coefficients from the objective.

Arguments

`c : double[]`
Linear terms of the objective as a dense vector. The lengths is the number of variables.

`first : int`
First index in the sequence.

`last : int`
Last index plus 1 in the sequence.

Description:

Obtains a sequence of elements in c.

A.2.60 Task.getdbi()

```
Task.getdbi(
    whichsol,
    accmode,
    sub,
    dbi)
```

Deprecated.

Arguments

`accmode` : accmode
: If set to `accmode.con` then `sub` contains constraint indexes, otherwise variable indexes.

`dbi` : `double[]`
: Dual bound infeasibility. If `acmode` is `accmode.con` then

$$\texttt{dbi}[i] = \max(-(s_l^c)_{\texttt{sub}[i]}, -(s_u^c)_{\texttt{sub}[i]}, 0) \;\; \texttt{for} \;\; i = 0, \ldots, \texttt{len} - 1$$

else

$$\texttt{dbi}[i] = \max(-(s_l^x)_{\texttt{sub}[i]}, -(s_u^x)_{\texttt{sub}[i]}, 0) \;\; \texttt{for} \;\; i = 0, \ldots, \texttt{len} - 1.$$

`sub` : `int[]`
: Indexes of constraints or variables.

`whichsol` : soltype
: Selects a solution.

Description:

Deprecated.

Obtains the dual bound infeasibility.

A.2.61 Task.getdcni()

```
Task.getdcni(
    whichsol,
    sub,
    dcni)
```

Deprecated.

Arguments

`dcni : double[]`

`dcni[i]` contains dual cone infeasibility for the cone with index `sub[i]`.

`sub : int[]`

Constraint indexes to calculate equation infeasibility for.

`whichsol : soltype`

Selects a solution.

Description:

Deprecated.

Obtains the dual cone infeasibility.

A.2.62 `Task.getdeqi()`

```
Task.getdeqi(
    whichsol,
    accmode,
    sub,
    deqi,
    normalize)
```

Deprecated.

Arguments

`accmode : accmode`

If set to `accmode.con` the dual equation infeasibilities corresponding to constraints are retrieved. Otherwise for a variables.

`deqi : double[]`

Dual equation infeasibilities corresponding to constraints or variables.

`normalize : int`

If non-zero, normalize with largest absolute value of the input data used to compute the individual infeasibility.

`sub : int[]`

Indexes of constraints or variables.

`whichsol : soltype`

Selects a solution.

Description:

Deprecated.

Optains the dual equation infeasibility. If `acmode` is `accmode.con` then

$$\texttt{pbi}[i] = \left|(-y + s_l^c - s_u^c)_{\texttt{sub}[i]}\right| \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1$$

If `acmode` is `accmode.var` then

$$\texttt{pbi}[i] = \left|(A^T y + s_l^x - s_u^x - c)_{\texttt{sub}[i]}\right| \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1$$

A.2.63 `Task.getdimbarvarj()`

```
dimbarvarj = Task.getdimbarvarj(j)
```

Obtains the dimension of a symmetric matrix variable.

Arguments

`dimbarvarj` : `int`
The dimension of the j'th semidefinite variable.

`j` : `int`
Index of the semidefinite variable whose dimension is requested.

Description:

Obtains the dimension of a symmetric matrix variable.

A.2.64 `Task.getdouinf()`

```
dvalue = Task.getdouinf(whichdinf)
```

Obtains a double information item.

Arguments

`dvalue` : `double`
The value of the required double information item.

`whichdinf` : `dinfitem`
A double float information item.

Description:

Obtains a double information item from the task information database.

A.2.65 Task.getdouparam()

```
parvalue = Task.getdouparam(param)
```

Obtains a double parameter.

Arguments

`param` : `dparam`
Which parameter.

`parvalue` : `double`
Parameter value.

Description:

Obtains the value of a double parameter.

A.2.66 Task.getdualobj()

```
dualobj = Task.getdualobj(whichsol)
```

Computes the dual objective value associated with the solution.

Arguments

`dualobj` : `double`
Objective value corresponding to the dual solution.

`whichsol` : `soltype`
Selects a solution.

Description:

Computes the dual objective value associated with the solution. Note if the solution is a primal infeasibility certificate, then the fixed term in the objective value is not included.

A.2.67 Task.getdviolbarvar()

```
Task.getdviolbarvar(
    whichsol,
    sub,
    viol)
```

Computes the violation of dual solution for a set of barx variables.

Arguments

`sub : int[]`
An array of indexes of $\bar{X}$ variables.

`viol : double[]`
`viol[k]` is violation of the solution for the constraint $\bar{S}_{\texttt{sub}[k]} \in \mathcal{S}$.

`whichsol : soltype`
Selects a solution.

Description:

Let $(\bar{S}_j)^*$ be the value of variable $\bar{S}_j$ for the specified solution. Then the dual violation of the solution associated with variable $\bar{S}_j$ is given by

$$\max(-\lambda_{\min}(\bar{S}_j), 0.0).$$

Both when the solution is a certificate of primal infeasibility or when it is dual feasibible solution the violation should be small.

A.2.68 `Task.getdviolcon()`

```
Task.getdviolcon(
    whichsol,
    sub,
    viol)
```

Computes the violation of a dual solution associated with a set of constraints.

Arguments

`sub : int[]`
An array of indexes of constraints.

`viol : double[]`
`viol[k]` is the violation of dual solution associated with the constraint `sub[k]`.

`whichsol : soltype`
Selects a solution.

Description:

The violation of the dual solution associated with the i'th constraint is computed as follows

$$\max(\rho((s_l^c)_i^*, (b_l^c)_i), \rho((s_u^c)_i^*, -(b_u^c)_i), |-y_i + (s_l^c)_i^* - (s_u^c)_i^*|)$$

where

$$\rho(x,l) = \begin{cases} -x, & l > -\infty, \\ |x|, & \text{otherwise} \end{cases}$$

Both when the solution is a certificate of primal infeasibility or it is a dual feasibible solution the violation should be small.

A.2.69 `Task.getdviolcones()`

```
Task.getdviolcones(
    whichsol,
    sub,
    viol)
```

Computes the violation of a solution for set of dual conic constraints.

Arguments

`sub` : `int[]`
: An array of indexes of $\bar{X}$ variables.

`viol` : `double[]`
: `viol[k]` violation of the solution associated with `sub[k]`'th dual conic constraint.

`whichsol` : `soltype`
: Selects a solution.

Description:

Let $(s_n^x)^*$ be the value of variable (s_n^x) for the specified solution. For simplicity let us assume that s_n^x is a member of quadratic cone, then the violation is computed as follows

$$\begin{cases} \max(0, \left\|(s_n^x)_{2;n}\right\|^* - (s_n^x)_1^*)/\sqrt{2}, & (s_n^x)^* \geq -\left\|(s_n^x)_{2:n}^*\right\|, \\ \left\|(s_n^x)^*\right\|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of primal infeasibility or when it is a dual feasibible solution the violation should be small.

A.2.70 `Task.getdviolvar()`

```
Task.getdviolvar(
    whichsol,
    sub,
    viol)
```

Computes the violation of a dual solution associated with a set of x variables.

Arguments

`sub : int[]`
An array of indexes of x variables.

`viol : double[]`
`viol[k]` is the maximal violation of the solution for the constraints $(s_l^x)_{\mathtt{sub}[k]} \geq 0$ and $(s_u^x)_{\mathtt{sub}[k]} \geq 0$.

`whichsol : soltype`
Selects a solution.

Description:

The violation fo dual solution associated with the j'th variable is computed as follows

$$\max(\rho((s_l^x)_i^*, (b_l^x)_i), \rho((s_u^x)_i^*, -(b_u^x)_i), |\sum j = 0^{numcon-1} a_{ij} y_i + (s_l^x)_i^* - (s_u^x)_i^* - \tau c_j|)$$

where

$$\rho(x, l) = \begin{cases} -x, & l > -\infty, \\ |x|, & \text{otherwise} \end{cases}$$

$\tau = 0$ if the the solution is certificate of dual infeasibility and $\tau = 1$ otherwise. The formula for computing the violation is only shown for linear case but is generalized approriately for the more general problems.

A.2.71 `Task.getinfeasiblesubproblem()`

```
inftask = Task.getinfeasiblesubproblem(whichsol)
```

Obtains an infeasible sub problem.

Arguments

`inftask : Task`
A new task containing the infeasible subproblem.

`whichsol : soltype`
Which solution to use when determining the infeasible subproblem.

Description:

Given the solution is a certificate of primal or dual infeasibility then a primal or dual infeasible subproblem is obtained respectively. The subproblem tend to be much smaller than the original problem and hence it easier to locate the infeasibility inspecting the subproblem than the original problem.

For the procedure to be useful then it is important to assigning meaningful names to constraints, variables etc. in the original task because those names will be duplicated in the subproblem.

The function is only applicable to linear and conic quadrtic optimization problems.

For more information see Section 13.2.

See also

- `iparam.infeas_prefer_primal` Controls which certificate is used if both primal- and dual-certificate of infeasibility is available.
- `Task.relaxprimal` Deprecated.

A.2.72 `Task.getinti()`

```
Task.getinti(
    whichsol,
    sub,
    inti)
```

Deprecated.

Arguments

`inti : double[]`
: `inti[i]` contains integer infeasibility of variable `sub[i]`.

`sub : int[]`
: Variable indexes for which to calculate the integer infeasibility.

`whichsol : soltype`
: Selects a solution.

Description:

Deprecated.

Obtains the primal equation infeasibility.

$$\mathtt{peqi}[i] = \left|(Ax - x^c)_{\mathtt{sub}[i]}\right| \quad \mathtt{for} \quad i = 0, \ldots, \mathtt{len} - 1.$$

A.2.73 `Task.getintinf()`

```
ivalue = Task.getintinf(whichiinf)
```

Obtains an integer information item.

Arguments

`ivalue : int`
The value of the required integer information item.

`whichiinf : iinfitem`
Specifies an information item.

Description:

Obtains an integer information item from the task information database.

A.2.74 Task.getintparam()

```
parvalue = Task.getintparam(param)
```

Obtains an integer parameter.

Arguments

`param : iparam`
Which parameter.

`parvalue : int`
Parameter value.

Description:

Obtains the value of an integer parameter.

A.2.75 Task.getlenbarvarj()

```
lenbarvarj = Task.getlenbarvarj(j)
```

Obtains the length if the j'th semidefinite variables.

Arguments

`j : int`
Index of the semidefinite variable whose length if requested.

`lenbarvarj : long`
Number of scalar elements in the lower triangular part of the semidefinite variable.

Description:

Obtains the length of the jth semidefinite variable i.e. the number of elements in the triangular part.

A.2.76 Task.getlintinf()

```
ivalue = Task.getlintinf(whichliinf)
```

Obtains an integer information item.

Arguments

`ivalue : long`
The value of the required integer information item.

`whichliinf : liinfitem`
Specifies an information item.

Description:

Obtains an integer information item from the task information database.

A.2.77 Task.getmaxnumanz()

```
maxnumanz = Task.getmaxnumanz()
```

Obtains number of preallocated non-zeros in the linear constraint matrix.

Arguments

`maxnumanz : long`
Number of preallocated non-zero linear matrix elements.

Description:

Obtains number of preallocated non-zeros in A. When this number of non-zeros is reached MOSEK will automatically allocate more space for A.

A.2.78 Task.getmaxnumbarvar()

```
maxnumbarvar = Task.getmaxnumbarvar()
```

Obtains the number of semidefinite variables.

Arguments

`maxnumbarvar : int`
Obtains maximum number of semidefinite variable currently allowed.

Description:

Obtains the number of semidefinite variables.

A.2.79 Task.getmaxnumcon()

```
maxnumcon = Task.getmaxnumcon()
```

Obtains the number of preallocated constraints in the optimization task.

Arguments

`maxnumcon : int`
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.

A.2.80 Task.getmaxnumcone()

```
maxnumcone = Task.getmaxnumcone()
```

Obtains the number of preallocated cones in the optimization task.

Arguments

`maxnumcone : int`
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.

A.2.81 Task.getmaxnumqnz()

```
maxnumqnz = Task.getmaxnumqnz()
```

Obtains the number of preallocated non-zeros for all quadratic terms in objective and constraints.

Arguments

`maxnumqnz : long`
Number of non-zero elements preallocated in quadratic coefficient matrixes.

Description:

Obtains the number of preallocated non-zeros for Q (both objective and constraints). When this number of non-zeros is reached MOSEK will automatically allocate more space for Q.

A.2.82 Task.getmaxnumvar()

```
maxnumvar = Task.getmaxnumvar()
```

Obtains the maximum number variables allowed.

Arguments

`maxnumvar : int`
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.

A.2.83 Task.getmemusage()

```
meminuse,maxmemuse = Task.getmemusage()
```

Obtains information about the amount of memory used by a task.

Arguments

`maxmemuse : long`
Maximum amount of memory used by the `task` until now.

`meminuse : long`
Amount of memory currently used by the `task`.

Description:

Obtains information about the amount of memory used by a task.

A.2.84 Task.getnumanz()

```
numanz = Task.getnumanz()
```

Obtains the number of non-zeros in the coefficient matrix.

Arguments

`numanz : int`
Number of non-zero elements in the linear constraint matrix.

Description:

Obtains the number of non-zeros in A.

A.2.85 `Task.getnumanz64()`

```
numanz = Task.getnumanz64()
```

Obtains the number of non-zeros in the coefficient matrix.

Arguments

`numanz : long`
Number of non-zero elements in the linear constraint matrix.

Description:

Obtains the number of non-zeros in A.

A.2.86 `Task.getnumbarablocktriplets()`

```
num = Task.getnumbarablocktriplets()
```

Obtains an upper bound on the number of scalar elements in the block triplet form of bara.

Arguments

`num : long`
Number elements in the block triplet form of $\bar{A}$.

Description:

Obtains an upper bound on the number of elements in the block triplet form of $\bar{A}$.

A.2.87 `Task.getnumbaranz()`

```
nz = Task.getnumbaranz()
```

Get the number of nonzero elements in barA.

Arguments

`nz : long`
The number of nonzero elements in $\bar{A}$ i.e. the number of $\bar{a}_{ij}$ elements that is nonzero.

Description:

Get the number of nonzero elements in $\bar{A}$.

A.2.88 Task.getnumbarcblocktriplets()

```
num = Task.getnumbarcblocktriplets()
```

Obtains an upper bound on the number of elements in the block triplet form of barc.

Arguments

`num : long`
An upper bound on the number elements in the block trip let form of $\bar{c}$.

Description:

Obtains an upper bound on the number of elements in the block triplet form of $\bar{C}$.

A.2.89 Task.getnumbarcnz()

```
nz = Task.getnumbarcnz()
```

Obtains the number of nonzero elements in barc.

Arguments

`nz : long`
The number of nonzeros in $\bar{c}$ i.e. the number of elements $\bar{c}_j$ that is diffrent from 0.

Description:

Obtains the number of nonzero elements in $\bar{c}$.

A.2.90 Task.getnumbarvar()

```
numbarvar = Task.getnumbarvar()
```

Obtains the number of semidefinite variables.

Arguments

`numbarvar : int`
Number of semidefinite variable in the problem.

Description:

Obtains the number of semidefinite variables.

A.2.91 Task.getnumcon()

```
numcon = Task.getnumcon()
```

Obtains the number of constraints.

Arguments

numcon : int
: Number of constraints.

Description:

Obtains the number of constraints.

A.2.92 Task.getnumcone()

```
numcone = Task.getnumcone()
```

Obtains the number of cones.

Arguments

numcone : int
: Number conic constraints.

Description:

Obtains the number of cones.

A.2.93 Task.getnumconemem()

```
nummem = Task.getnumconemem(k)
```

Obtains the number of members in a cone.

Arguments

k : int
: Index of the cone.

nummem : int
: Number of member variables in the cone.

Description:

Obtains the number of members in a cone.

A.2.94 `Task.getnumintvar()`

```
numintvar = Task.getnumintvar()
```

Obtains the number of integer-constrained variables.

Arguments

`numintvar : int`
Number of integer variables.

Description:

Obtains the number of integer-constrained variables.

A.2.95 `Task.getnumparam()`

```
numparam = Task.getnumparam(partype)
```

Obtains the number of parameters of a given type.

Arguments

`numparam : int`
Identical to the number of parameters of the type `partype`.

`partype : parametertype`
Parameter type.

Description:

Obtains the number of parameters of a given type.

A.2.96 `Task.getnumqconknz()`

```
numqcnz = Task.getnumqconknz(k)
```

Obtains the number of non-zero quadratic terms in a constraint.

Arguments

`k : int`
Index of the constraint for which the number of non-zero quadratic terms should be obtained.

numqcnz : int
: Number of quadratic terms.

Description:

Obtains the number of non-zero quadratic terms in a constraint.

A.2.97 Task.getnumqconknz64()

```
numqcnz = Task.getnumqconknz64(k)
```

Obtains the number of non-zero quadratic terms in a constraint.

Arguments

k : int
: Index of the constraint for which the number quadratic terms should be obtained.

numqcnz : long
: Number of quadratic terms.

Description:

Obtains the number of non-zero quadratic terms in a constraint.

A.2.98 Task.getnumqobjnz()

```
numqonz = Task.getnumqobjnz()
```

Obtains the number of non-zero quadratic terms in the objective.

Arguments

numqonz : long
: Number of non-zero elements in the quadratic objective terms.

Description:

Obtains the number of non-zero quadratic terms in the objective.

A.2.99 Task.getnumsymmat()

```
num = Task.getnumsymmat()
```

Get the number of symmetric matrixes stored.

Arguments

`num : long`
Returns the number of symmetric sparse matrixes.

Description:

Get the number of symmetric matrixes stored in the vector E.

A.2.100 Task.getnumvar()

```
numvar = Task.getnumvar()
```

Obtains the number of variables.

Arguments

`numvar : int`
Number of variables.

Description:

Obtains the number of variables.

A.2.101 Task.getobjname()

```
objname = Task.getobjname()
```

Obtains the name assigned to the objective function.

Arguments

`objname : str`
Assigned the objective name.

Description:

Obtains the name assigned to the objective function.

A.2.102 `Task.getobjnamelen()`

```
len = Task.getobjnamelen()
```

Obtains the length of the name assigned to the objective function.

Arguments

`len : int`
Assigned the length of the objective name.

Description:

Obtains the length of the name assigned to the objective function.

A.2.103 `Task.getobjsense()`

```
sense = Task.getobjsense()
```

Gets the objective sense.

Arguments

`sense : objsense`
The returned objective sense.

Description:

Gets the objective sense of the task.

See also

- `Task.putobjsense` Sets the objective sense.

A.2.104 `Task.getpbi()`

```
Task.getpbi(
    whichsol,
    accmode,
    sub,
    pbi,
    normalize)
```

Deprecated.

Arguments

`accmode` : `accmode`
: If set to `accmode.var` return bound infeasibility for x otherwise for x^c.

`normalize` : `int`
: If non-zero, normalize with largest absolute value of the input data used to compute the individual infeasibility.

`pbi` : `double[]`
: Bound infeasibility for x or x^c.

`sub` : `int[]`
: An array of constraint or variable indexes.

`whichsol` : `soltype`
: Selects a solution.

Description:

Deprecated.

Obtains the primal bound infeasibility. If `acmode` is `accmode.con` then

$$\texttt{pbi}[i] = \max(x^c_{\texttt{sub[i]}} - u^c_{\texttt{sub[i]}}, l^c_{\texttt{sub[i]}} - x^c_{\texttt{sub[i]}}, 0) \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1$$

If `acmode` is `accmode.var` then

$$\texttt{pbi}[i] = \max(x_{\texttt{sub[i]}} - u^x_{\texttt{sub[i]}}, l^x_{\texttt{sub[i]}} - x_{\texttt{sub[i]}}, 0) \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1$$

A.2.105 Task.getpcni()

```
Task.getpcni(
    whichsol,
    sub,
    pcni)
```

Deprecated.

Arguments

`pcni` : `double[]`
: `pcni[i]` contains primal cone infeasibility for the cone with index `sub[i]`.

`sub` : `int[]`
: Constraint indexes for which to calculate the equation infeasibility.

`whichsol` : `soltype`
: Selects a solution.

Description:

Deprectaed.

A.2.106 `Task.getpeqi()`

```
Task.getpeqi(
    whichsol,
    sub,
    peqi,
    normalize)
```

Deprecated.

Arguments

`normalize : int`
: If non-zero, normalize with largest absolute value of the input data used to compute the individual infeasibility.

`peqi : double[]`
: `peqi[i]` contains equation infeasibility of constraint `sub[i]`.

`sub : int[]`
: Constraint indexes for which to calculate the equation infeasibility.

`whichsol : soltype`
: Selects a solution.

Description:

Deprecated.

Obtains the primal equation infeasibility.

$$\mathtt{peqi}[i] = \left|(Ax - x^c)_{\mathtt{sub}[i]}\right| \quad \mathtt{for} \quad i = 0, \ldots, \mathtt{len} - 1.$$

A.2.107 `Task.getprimalobj()`

```
primalobj = Task.getprimalobj(whichsol)
```

Computes the primal objective value for the desired solution.

Arguments

`primalobj : double`
: Objective value corresponding to the primal solution.

`whichsol : soltype`
: Selects a solution.

Description:

Computes the primal objective value for the desired solution. Note if the solution is an infeasibility certificate, then the fixed term in the objective is not included.

A.2.108 Task.getprobtype()

```
probtype = Task.getprobtype()
```

Obtains the problem type.

Arguments

`probtype` : `problemtype`
The problem type.

Description:

Obtains the problem type.

A.2.109 Task.getprosta()

```
prosta = Task.getprosta(whichsol)
```

Obtains the problem status.

Arguments

`prosta` : `prosta`
Problem status.

`whichsol` : `soltype`
Selects a solution.

Description:

Obtains the problem status.

A.2.110 Task.getpviolbarvar()

```
Task.getpviolbarvar(
    whichsol,
    sub,
    viol)
```

Computes the violation of a primal solution for a list of barx variables.

Arguments

`sub : int[]`
An array of indexes of $\bar{X}$ variables.

`viol : double[]`
`viol[k]` is how much the solution violate the constraint $\bar{X}_{\texttt{sub}[k]} \in \mathcal{S}^{+}$.

`whichsol : soltype`
Selects a solution.

Description:

Let $(\bar{X}_j)^*$ be the value of variable $\bar{X}_j$ for the specified solution. Then the primal violation of the solution associated with variable $\bar{X}_j$ is given by

$$\max(-\lambda_{\min}(\bar{X}_j), 0.0).$$

A.2.111 `Task.getpviolcon()`

```
Task.getpviolcon(
    whichsol,
    sub,
    viol)
```

Computes the violation of a primal solution for a list of xc variables.

Arguments

`sub : int[]`
An array of indexes of constraints.

`viol : double[]`
`viol[k]` associated with the solution for the `sub[k]`'th constraint.

`whichsol : soltype`
Selects a solution.

Description:

The primal violation of the solution associated of constraint is computed by

$$\max(l_i^c \tau - (x_i^c)^*), (x_i^c)^* \tau - u_i^c \tau, |\sum_{j=0}^{numvar-1} a_{ij} x_j^* - x_i^c|)$$

where τ is defined as follows. If the solution is a certificate of dual infeasibility, then $\tau = 0$ and otherwise $\tau = 1$. Both when the solution is a valid certificate of dual infeasibility or when it is primal feasibible solution the violation should be small. The above is only shown for linear case but is appropriately generalized for the other cases.

A.2.112 `Task.getpviolcones()`

```
Task.getpviolcones(
    whichsol,
    sub,
    viol)
```

Computes the violation of a solution for set of conic constraints.

Arguments

`sub : int[]`
An array of indexes of $\bar{X}$ variables.

`viol : double[]`
`viol[k]` violation of the solution associated with `sub[k]`'th conic constraint.

`whichsol : soltype`
Selects a solution.

Description:

Let x^* be the value of variable x for the specified solution. For simplicity let us assume that x is a member of quadratic cone, then the violation is computed as follows

$$\begin{cases} \max(0, \|x_{2;n}\| - x_1)/\sqrt{2}, & x_1 \geq -\|x_{2:n}\|, \\ \|x\|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of dual infeasibility or when it is a primal feasibible solution the violation should be small.

A.2.113 `Task.getpviolvar()`

```
Task.getpviolvar(
    whichsol,
    sub,
    viol)
```

Computes the violation of a primal solution for a list of x variables.

Arguments

`sub : int[]`
An array of indexes of x variables.

`viol : double[]`
`viol[k]` is the violation associated the solution for variable x_j.

`whichsol` : soltype

Selects a solution.

Description:

Let x_j^* be the value of variable x_j for the specified solution. Then the primal violation of the solution associated with variable x_j is given by

$$\max(l_j^x \tau - x_j^*, x_j^* - u_j^x \tau).$$

where τ is defined as follows. If the solution is a certificate of dual infeasibility, then $\tau = 0$ and otherwise $\tau = 1$. Both when the solution is a valid certificate of dual infeasibility or when it is primal feasibible solution the violation should be small.

A.2.114 `Task.getqconk()`

```
numqcnz = Task.getqconk(
    k,
    qcsubi,
    qcsubj,
    qcval)
```

Obtains all the quadratic terms in a constraint.

Arguments

`k` : `int`

Which constraint.

`numqcnz` : `long`

Number of quadratic terms.

`qcsubi` : `int[]`

Row subscripts for quadratic constraint matrix.

`qcsubj` : `int[]`

Column subscripts for quadratic constraint matrix.

`qcval` : `double[]`

Quadratic constraint coefficient values.

Description:

Obtains all the quadratic terms in a constraint. The quadratic terms are stored sequentially `qcsubi`, `qcsubj`, and `qcval`.

A.2.115 `Task.getqobj()`

```
numqonz = Task.getqobj(
    qosubi,
    qosubj,
    qoval)
```

Obtains all the quadratic terms in the objective.

Arguments

`numqonz : int`
: Number of non-zero elements in the quadratic objective terms.

`qosubi : int[]`
: Row subscripts for quadratic objective coefficients.

`qosubj : int[]`
: Column subscripts for quadratic objective coefficients.

`qoval : double[]`
: Quadratic objective coefficient values.

Description:

Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in `qosubi`, `qosubj`, and `qoval`.

A.2.116 `Task.getqobj64()`

```
numqonz = Task.getqobj64(
    qosubi,
    qosubj,
    qoval)
```

Obtains all the quadratic terms in the objective.

Arguments

`numqonz : long`
: Number of non-zero elements in the quadratic objective terms.

`qosubi : int[]`
: Row subscripts for quadratic objective coefficients.

`qosubj : int[]`
: Column subscripts for quadratic objective coefficients.

`qoval : double[]`

Quadratic objective coefficient values.

Description:

Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in `qosubi`, `qosubj`, and `qoval`.

A.2.117 `Task.getqobjij()`

```
qoij = Task.getqobjij(
    i,
    j)
```

Obtains one coefficient from the quadratic term of the objective

Arguments

`i : int`

Row index of the coefficient.

`j : int`

Column index of coefficient.

`qoij : double`

The required coefficient.

Description:

Obtains one coefficient q_{ij}^o in the quadratic term of the objective.

A.2.118 `Task.getreducedcosts()`

```
Task.getreducedcosts(
    whichsol,
    first,
    last,
    redcosts)
```

Obtains the difference of (slx-sux) for a sequence of variables.

Arguments

`first : int`

See formula (A.1) for the definition.

`last : int`

See formula (A.1) for the definition.

`redcosts : double[]`

The reduced costs in the required sequence of variables are stored sequentially in `redcosts` starting at `redcosts[0]`.

`whichsol : soltype`

Selects a solution.

Description:

Computes the reduced costs for a sequence of variables and return them in the variable `redcosts` i.e.

$$\texttt{redcosts}[j - \texttt{first}] = (s_l^x)_j - (s_u^x)_j, \; j = \texttt{first}, \ldots, last - 1. \tag{A.1}$$

A.2.119 `Task.getskc()`

```
Task.getskc(
    whichsol,
    skc)
```

Obtains the status keys for the constraints.

Arguments

`skc : mosek.stakey[]`

Status keys for the constraints.

`whichsol : soltype`

Selects a solution.

Description:

Obtains the status keys for the constraints.

See also

- `Task.getskcslice` Obtains the status keys for the constraints.

A.2.120 `Task.getskcslice()`

```
Task.getskcslice(
    whichsol,
    first,
    last,
    skc)
```

Obtains the status keys for the constraints.

Arguments

`first : int`
First index in the sequence.

`last : int`
Last index plus 1 in the sequence.

`skc : mosek.stakey[]`
Status keys for the constraints.

`whichsol : soltype`
Selects a solution.

Description:

Obtains the status keys for the constraints.

See also

- `Task.getskc` Obtains the status keys for the constraints.

A.2.121 Task.getskx()

```
Task.getskx(
    whichsol,
    skx)
```

Obtains the status keys for the scalar variables.

Arguments

`skx : mosek.stakey[]`
Status keys for the variables.

`whichsol : soltype`
Selects a solution.

Description:

Obtains the status keys for the scalar variables.

See also

- `Task.getskxslice` Obtains the status keys for the variables.

A.2.122 Task.getskxslice()

```
Task.getskxslice(
    whichsol,
    first,
    last,
    skx)
```

Obtains the status keys for the variables.

Arguments

`first : int`
First index in the sequence.

`last : int`
Last index plus 1 in the sequence.

`skx : mosek.stakey[]`
Status keys for the variables.

`whichsol : soltype`
Selects a solution.

Description:

Obtains the status keys for the variables.

A.2.123 Task.getslc()

```
Task.getslc(
    whichsol,
    slc)
```

Obtains the slc vector for a solution.

Arguments

`slc : double[]`
The s_l^c vector.

`whichsol : soltype`
Selects a solution.

Description:

Obtains the s_l^c vector for a solution.

See also

- `Task.getslcslice` Obtains a slice of the slc vector for a solution.

A.2.124 Task.getslcslice()

```
Task.getslcslice(
    whichsol,
    first,
    last,
    slc)
```

Obtains a slice of the slc vector for a solution.

Arguments

`first : int`
First index in the sequence.

`last : int`
Last index plus 1 in the sequence.

`slc : double[]`
Dual variables corresponding to the lower bounds on the constraints.

`whichsol : soltype`
Selects a solution.

Description:

Obtains a slice of the s_l^c vector for a solution.

See also

- `Task.getslc` Obtains the slc vector for a solution.

A.2.125 Task.getslx()

```
Task.getslx(
    whichsol,
    slx)
```

Obtains the slx vector for a solution.

Arguments

`slx : double[]`
The s_l^x vector.

`whichsol : soltype`
Selects a solution.

Description:

Obtains the s_l^x vector for a solution.

See also

- Task.getslx Obtains the slx vector for a solution.

A.2.126 Task.getslxslice()

```
Task.getslxslice(
    whichsol,
    first,
    last,
    slx)
```

Obtains a slice of the slx vector for a solution.

Arguments

`first : int`
: First index in the sequence.

`last : int`
: Last index plus 1 in the sequence.

`slx : double[]`
: Dual variables corresponding to the lower bounds on the variables.

`whichsol : soltype`
: Selects a solution.

Description:

Obtains a slice of the s_l^x vector for a solution.

See also

- Task.getslx Obtains the slx vector for a solution.

A.2.127 Task.getsnx()

```
Task.getsnx(
    whichsol,
    snx)
```

Obtains the snx vector for a solution.

Arguments

`snx : double[]`
The s_n^x vector.

`whichsol : soltype`
Selects a solution.

Description:

Obtains the s_n^x vector for a solution.

See also

- `Task.getsnxslice` Obtains a slice of the snx vector for a solution.

A.2.128 `Task.getsnxslice()`

```
Task.getsnxslice(
    whichsol,
    first,
    last,
    snx)
```

Obtains a slice of the snx vector for a solution.

Arguments

`first : int`
First index in the sequence.

`last : int`
Last index plus 1 in the sequence.

`snx : double[]`
Dual variables corresponding to the conic constraints on the variables.

`whichsol : soltype`
Selects a solution.

Description:

Obtains a slice of the s_n^x vector for a solution.

See also

- `Task.getsnx` Obtains the snx vector for a solution.

A.2.129 Task.getsolsta()

```
solsta = Task.getsolsta(whichsol)
```

Obtains the solution status.

Arguments

`solsta` : solsta
: Solution status.

`whichsol` : soltype
: Selects a solution.

Description:

Obtains the solution status.

A.2.130 Task.getsolution()

```
prosta,solsta = Task.getsolution(
    whichsol,
    skc,
    skx,
    skn,
    xc,
    xx,
    y,
    slc,
    suc,
    slx,
    sux,
    snx)
```

Obtains the complete solution.

Arguments

`prosta` : prosta
: Problem status.

`skc` : `mosek.stakey[]`
: Status keys for the constraints.

`skn` : `mosek.stakey[]`
: Status keys for the conic constraints.

`skx` : `mosek.stakey[]`
: Status keys for the variables.

`slc : double[]`
Dual variables corresponding to the lower bounds on the constraints.

`slx : double[]`
Dual variables corresponding to the lower bounds on the variables.

`snx : double[]`
Dual variables corresponding to the conic constraints on the variables.

`solsta : solsta`
Solution status.

`suc : double[]`
Dual variables corresponding to the upper bounds on the constraints.

`sux : double[]`
Dual variables corresponding to the upper bounds on the variables.

`whichsol : soltype`
Selects a solution.

`xc : double[]`
Primal constraint solution.

`xx : double[]`
Primal variable solution.

`y : double[]`
Vector of dual variables corresponding to the constraints.

Description:

Obtains the complete solution.

Consider the case of linear programming. The primal problem is given by

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

and the corresponding dual problem is

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

In this case the mapping between variables and arguments to the function is as follows:

`xx`:
Corresponds to variable x.

`y`:
Corresponds to variable y.

`slc`:
Corresponds to variable s_l^c.

`suc`:
Corresponds to variable s_u^c.

`slx`:
Corresponds to variable s_l^x.

`sux`:
Corresponds to variable s_u^x.

`xc`:
Corresponds to Ax.

The meaning of the values returned by this function depend on the *solution status* returned in the argument `solsta`. The most important possible values of `solsta` are:

`solsta.optimal`
An optimal solution satisfying the optimality criteria for continuous problems is returned.

`solsta.integer_optimal`
An optimal solution satisfying the optimality criteria for integer problems is returned.

`solsta.prim_feas`
A solution satisfying the feasibility criteria.

`solsta.prim_infeas_cer`
A primal certificate of infeasibility is returned.

`solsta.dual_infeas_cer`
A dual certificate of infeasibility is returned.

See also

- `Task.getsolutioni` Obtains the solution for a single constraint or variable.
- `Task.getsolutionslice` Obtains a slice of the solution.

A.2.131 `Task.getsolutioni()`

```
sk,x,sl,su,sn = Task.getsolutioni(
    accmode,
    i,
    whichsol)
```

Obtains the solution for a single constraint or variable.

Arguments

`accmode` : `accmode`
  If set to `accmode.con` the solution information for a constraint is retrieved. Otherwise for a variable.

`i` : `int`
  Index of the constraint or variable.

`sk` : `stakey`
  Status key of the constraint of variable.

`sl` : `double`
  Solution value of the dual variable associated with the lower bound.

`sn` : `double`
  Solution value of the dual variable associated with the cone constraint.

`su` : `double`
  Solution value of the dual variable associated with the upper bound.

`whichsol` : `soltype`
  Selects a solution.

`x` : `double`
  Solution value of the primal variable.

Description:

Obtains the primal and dual solution information for a single constraint or variable.

See also

- `Task.getsolution` Obtains the complete solution.
- `Task.getsolutionslice` Obtains a slice of the solution.

A.2.132 `Task.getsolutioninf()`

```
prosta,solsta,primalobj,maxpbi,maxpcni,maxpeqi,maxinti,dualobj,maxdbi,maxdcni,maxdeqi = Task.getsolutioninf(whichsol)
```

Deprecated

Arguments

`dualobj` : `double`
  Value of the dual objective.

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

`maxdbi : double`

Maximum infeasibility in bounds on dual variables.

$$\max\{0, \max_{i\in\{0,\ldots,n-1\}} -(s_l^x)_i, \max_{i\in\{0,\ldots,n-1\}} -(s_u^x)_i, \max_{i\in\{0,\ldots,m-1\}} -(s_l^c)_i, \max_{i\in\{0,\ldots,m-1\}} -(s_u^c)_i\}$$

`maxdcni : double`

Maximum infeasibility in the dual conic constraints.

`maxdeqi : double`

Maximum infeasibility in the dual equality constraints.

$$\max\left\{ \left\|A^T y + s_l^x - s_u^x - c\right\|_\infty, \left\| -y + s_l^c - s_u^c\right\|_\infty \right\}$$

`maxinti : double`

Maximum infeasibility in integer constraints.

$$\max_{i\in\{0,\ldots,n-1\}}(\min(x_i - \lfloor x_i \rfloor, \lceil x_i \rceil - x_i)).$$

`maxpbi : double`

Maximum infeasibility in primal bounds on variables.

$$\max\left\{0, \max_{i\in 1,\ldots,n-1}(x_i - u_i^x), \max_{i\in 1,\ldots,n-1}(l_i^x - x_i), \max_{i\in 1,\ldots,n-1}(x_i^c - u_i^c), \max_{i\in 1,\ldots,n-1}(l_i^c - x_i^c)\right\}$$

`maxpcni : double`

Maximum infeasibility in the primal conic constraints.

`maxpeqi : double`

Maximum infeasibility in primal equality constraints.

$$\left\|Ax - x^c\right\|_\infty$$

`primalobj : double`

Value of the primal objective.

$$c^T x + c^f$$

`prosta : prosta`

Problem status.

`solsta : solsta`

Solution status.

`whichsol : soltype`

Selects a solution.

Description:

Deprecated. Use `Task.getsolutioninfo` instead.

A.2.133 `Task.getsolutioninfo()`

```
pobj,pviolcon,pviolvar,pviolbarvar,pviolcone,pviolitg,dobj,dviolcon,dviolvar,dviolbarvar,dviolcone = Task.getsolutioninfo(wh
```

Obtains information about of a solution.

Arguments

`dobj : double`
: Dual objective value as computed as computed by `Task.getdualobj`.

`dviolbarvar : double`
: Maximal violation of the dual solution associated with the $\bar{s}$ variable as computed by as computed by `Task.getdviolbarvar`.

`dviolcon : double`
: Maximal violation of the dual solution associated with the x^c variable as computed by as computed by `Task.getdviolcon`.

`dviolcone : double`
: Maximal violation of the dual solution associated with the dual conic constraints as computed by `Task.getdviolcones`.

`dviolvar : double`
: Maximal violation of the dual solution associated with the x variable as computed by as computed by `Task.getdviolvar`.

`pobj : double`
: The primal objective value as computed by `Task.getprimalobj`.

`pviolbarvar : double`
: Maximal primal violation of solution for the $\bar{X}$ variables where the violations are computed by `Task.getpviolbarvar`.

`pviolcon : double`
: Maximal primal violation of the solution associated with the x^c variables where the violations are computed by `Task.getpviolcon`.

`pviolcone : double`
: Maximal primal violation of solution for the conic constraints where the violations are computed by `Task.getpviolcones`.

`pviolitg : double`
: Maximal violation in the integer constraints. The violation for an integer constrained variable x_j is given by

$$\min(x_j - \lfloor x_j \rfloor, \lceil x_j \rceil - x_j).$$

This number is always zero for the interior-point and the basic solutions.

`pviolvar : double`

Maximal primal violation of the solution for the x^x variables where the violations are computed by Task.getpviolvar.

`whichsol : soltype`

Selects a solution.

Description:

Obtains information about a solution.

See also

- Task.getsolsta Obtains the solution status.
- Task.getprimalobj Computes the primal objective value for the desired solution.
- Task.getpviolcon Computes the violation of a primal solution for a list of xc variables.
- Task.getpviolvar Computes the violation of a primal solution for a list of x variables.
- Task.getpviolbarvar Computes the violation of a primal solution for a list of barx variables.
- Task.getpviolcones Computes the violation of a solution for set of conic constraints.
- Task.getdualobj Computes the dual objective value associated with the solution.
- Task.getdviolcon Computes the violation of a dual solution associated with a set of constraints.
- Task.getdviolvar Computes the violation of a dual solution associated with a set of x variables.
- Task.getdviolbarvar Computes the violation of dual solution for a set of barx variables.
- Task.getdviolcones Computes the violation of a solution for set of dual conic constraints.

A.2.134 Task.getsolutionslice()

```
Task.getsolutionslice(
    whichsol,
    solitem,
    first,
    last,
    values)
```

Obtains a slice of the solution.

Arguments

`first : int`

Index of the first value in the slice.

`last : int`

Value of the last index+1 in the slice, e.g. if $xx[5,\ldots,9]$ is required `last` should be 10.

`solitem : solitem`

Which part of the solution is required.

`values : double[]`

The values in the required sequence are stored sequentially in `values` starting at `values[0]`.

`whichsol : soltype`

Selects a solution.

Description:

Obtains a slice of the solution.

Consider the case of linear programming. The primal problem is given by

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

and the corresponding dual problem is

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

The `solitem` argument determines which part of the solution is returned:

`solitem.xx`:

The variable `values` return x.

`solitem.y`:

The variable `values` return y.

`solitem.slc`:

The variable `values` return s_l^c.

`solitem.suc`:

The variable `values` return s_u^c.

`solitem.slx`:

The variable `values` return s_l^x.

`solitem.sux`:

The variable `values` return s_u^x.

A conic optimization problem has the same primal variables as in the linear case. Recall that the dual of a conic optimization problem is given by:

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

This introduces one additional dual variable s_n^x. This variable can be acceded by selecting `solitem` as `solitem.snx`.

The meaning of the values returned by this function also depends on the *solution status* which can be obtained with `Task.getsolsta`. Depending on the solution status `value` will be:

`solsta.optimal`
: A part of the optimal solution satisfying the optimality criteria for continuous problems.

`solsta.integer_optimal`
: A part of the optimal solution satisfying the optimality criteria for integer problems.

`solsta.prim_feas`
: A part of the solution satisfying the feasibility criteria.

`solsta.prim_infeas_cer`
: A part of the primal certificate of infeasibility.

`solsta.dual_infeas_cer`
: A part of the dual certificate of infeasibility.

See also

- `Task.getsolution` Obtains the complete solution.
- `Task.getsolutioni` Obtains the solution for a single constraint or variable.

A.2.135 `Task.getsparsesymmat()`

```
Task.getsparsesymmat(
    idx,
    subi,
    subj,
    valij)
```

Gets a single symmetric matrix from the matrix store.

Arguments

`idx : long`
: Index of the matrix to get.

`subi : int[]`
  Row subscripts of the matrix non-zero elements.

`subj : int[]`
  Column subscripts of the matrix non-zero elements.

`valij : double[]`
  Coefficients of the matrix non-zero elements.

Description:

Get a single symmetric matrix from the matrix store.

A.2.136 `Task.getstrparam()`

```
len,parvalue = Task.getstrparam(param)
```

Obtains the value of a string parameter.

Arguments

`len : int`
  The length of the parameter value.

`param : sparam`
  Which parameter.

`parvalue : str`
  If this is not NULL, the parameter value is stored here.

Description:

Obtains the value of a string parameter.

A.2.137 `Task.getstrparamlen()`

```
len = Task.getstrparamlen(param)
```

Obtains the length of a string parameter.

Arguments

`len : int`
  The length of the parameter value.

`param : sparam`
  Which parameter.

Description:

Obtains the length of a string parameter.

A.2.138 Task.getsuc()

```
Task.getsuc(
    whichsol,
    suc)
```

Obtains the suc vector for a solution.

Arguments

`suc : double[]`
The s_u^c vector.

`whichsol : soltype`
Selects a solution.

Description:

Obtains the s_u^c vector for a solution.

See also

- `Task.getsucslice` Obtains a slice of the suc vector for a solution.

A.2.139 Task.getsucslice()

```
Task.getsucslice(
    whichsol,
    first,
    last,
    suc)
```

Obtains a slice of the suc vector for a solution.

Arguments

`first : int`
First index in the sequence.

`last : int`
Last index plus 1 in the sequence.

`suc : double[]`
Dual variables corresponding to the upper bounds on the constraints.

`whichsol : soltype`
Selects a solution.

Description:

Obtains a slice of the s_u^c vector for a solution.

See also

- Task.getsuc Obtains the suc vector for a solution.

A.2.140 Task.getsux()

```
Task.getsux(
    whichsol,
    sux)
```

Obtains the sux vector for a solution.

Arguments

`sux : double[]`
The s_u^x vector.

`whichsol : soltype`
Selects a solution.

Description:

Obtains the s_u^x vector for a solution.

See also

- Task.getsuxslice Obtains a slice of the sux vector for a solution.

A.2.141 Task.getsuxslice()

```
Task.getsuxslice(
    whichsol,
    first,
    last,
    sux)
```

Obtains a slice of the sux vector for a solution.

Arguments

`first : int`
First index in the sequence.

`last : int`
Last index plus 1 in the sequence.

`sux : double[]`
Dual variables corresponding to the upper bounds on the variables.

`whichsol : soltype`
Selects a solution.

Description:

Obtains a slice of the s_u^x vector for a solution.

See also

- `Task.getsux` Obtains the sux vector for a solution.

A.2.142 `Task.getsymmatinfo()`

```
dim,nz,type = Task.getsymmatinfo(idx)
```

Obtains information of a matrix from the symmetric matrix storage E.

Arguments

`dim : int`
Returns the dimension of the requested matrix.

`idx : long`
Index of the matrix that is requested information about.

`nz : long`
Returns the number of non-zeros in the requested matrix.

`type : symmattype`
Returns the type of the requested matrix.

Description:

MOSEK maintains a vector denoted E of symmetric data matrixes. This function makes it possible to obtain important information about an data matrix in E.

A.2.143 `Task.gettaskname()`

```
taskname = Task.gettaskname()
```

Obtains the task name.

Arguments

`taskname : str`
Is assigned the task name.

Description:

Obtains the name assigned to the task.

A.2.144 `Task.gettasknamelen()`

```
len = Task.gettasknamelen()
```

Obtains the length the task name.

Arguments

`len : int`
Returns the length of the task name.

Description:

Obtains the length the task name.

See also

- `Task.getbarvarname` Obtains a name of a semidefinite variable.

A.2.145 `Task.getvarbound()`

```
bk,bl,bu = Task.getvarbound(i)
```

Obtains bound information for one variable.

Arguments

`bk : boundkey`
Bound keys.

`bl : double`
Values for lower bounds.

`bu : double`
Values for upper bounds.

`i : int`
Index of the variable for which the bound information should be obtained.

Description:

Obtains bound information for one variable.

A.2.146 `Task.getvarboundslice()`

```
Task.getvarboundslice(
    first,
    last,
    bk,
    bl,
    bu)
```

Obtains bounds information for a slice of the variables.

Arguments

`bk` : `mosek.boundkey[]`
: Bound keys.

`bl` : `double[]`
: Values for lower bounds.

`bu` : `double[]`
: Values for upper bounds.

`first` : `int`
: First index in the sequence.

`last` : `int`
: Last index plus 1 in the sequence.

Description:

Obtains bounds information for a slice of the variables.

A.2.147 `Task.getvarbranchdir()`

```
direction = Task.getvarbranchdir(j)
```

Obtains the branching direction for a variable.

Arguments

`direction` : `branchdir`
: The branching direction assigned to variable j.

`j` : `int`
: Index of the variable.

Description:

Obtains the branching direction for a given variable j.

A.2.148 `Task.getvarbranchpri()`

```
priority = Task.getvarbranchpri(j)
```

Obtains the branching priority for a variable.

Arguments

`j : int`
Index of the variable.

`priority : int`
The branching priority assigned to variable j.

Description:
Obtains the branching priority for a given variable j.

A.2.149 `Task.getvarname()`

```
name = Task.getvarname(j)
```

Obtains a name of a variable.

Arguments

`j : int`
Index.

`name : str`
Is assigned the required name.

Description:
Obtains a name of a variable.

A.2.150 `Task.getvarnameindex()`

```
asgn,index = Task.getvarnameindex(somename)
```

Checks whether the name somename has been assigned to any variable.

Arguments

`asgn : int`
Is non-zero if the name `somename` is assigned to a variable.

`index : int`
If the name `somename` is assigned to a variable, then `index` is the name of the variable.

`somename : str`
The name which should be checked.

Description:

Checks whether the name `somename` has been assigned to any variable. If it has been assigned to variable, then index of the variable is reported.

A.2.151 `Task.getvarnamelen()`

```
len = Task.getvarnamelen(i)
```

Obtains the length of a name of a variable variable.

Arguments

`i : int`
Index.

`len : int`
Returns the length of the indicated name.

Description:

Obtains the length of a name of a variable variable.

See also

- `Task.getbarvarname` Obtains a name of a semidefinite variable.

A.2.152 `Task.getvartype()`

```
vartype = Task.getvartype(j)
```

Gets the variable type of one variable.

Arguments

`j : int`
Index of the variable.

`vartype` : `variabletype`
Variable type of variable `j`.

Description:

Gets the variable type of one variable.

A.2.153 Task.getvartypelist()

```
Task.getvartypelist(
    subj,
    vartype)
```

Obtains the variable type for one or more variables.

Arguments

`subj` : `int[]`
A list of variable indexes.

`vartype` : `mosek.variabletype[]`
The variables types corresponding to the variables specified by `subj`.

Description:

Obtains the variable type of one or more variables.

Upon return `vartype[k]` is the variable type of variable `subj[k]`.

A.2.154 Task.getxc()

```
Task.getxc(
    whichsol,
    xc)
```

Obtains the xc vector for a solution.

Arguments

`whichsol` : `soltype`
Selects a solution.

`xc` : `double[]`
The x^c vector.

Description:

Obtains the x^c vector for a solution.

See also

- Task.getxcslice Obtains a slice of the xc vector for a solution.

A.2.155 Task.getxcslice()

```
Task.getxcslice(
    whichsol,
    first,
    last,
    xc)
```

Obtains a slice of the xc vector for a solution.

Arguments

`first : int`
: First index in the sequence.

`last : int`
: Last index plus 1 in the sequence.

`whichsol : soltype`
: Selects a solution.

`xc : double[]`
: Primal constraint solution.

Description:

Obtains a slice of the x^c vector for a solution.

See also

- Task.getxc Obtains the xc vector for a solution.

A.2.156 Task.getxx()

```
Task.getxx(
    whichsol,
    xx)
```

Obtains the xx vector for a solution.

Arguments

`whichsol` : `soltype`
  Selects a solution.

`xx` : `double[]`
  The x^x vector.

Description:

Obtains the x^x vector for a solution.

See also

- `Task.getxxslice` Obtains a slice of the xx vector for a solution.

A.2.157 `Task.getxxslice()`

```
Task.getxxslice(
    whichsol,
    first,
    last,
    xx)
```

Obtains a slice of the xx vector for a solution.

Arguments

`first` : `int`
  First index in the sequence.

`last` : `int`
  Last index plus 1 in the sequence.

`whichsol` : `soltype`
  Selects a solution.

`xx` : `double[]`
  Primal variable solution.

Description:

Obtains a slice of the x^x vector for a solution.

See also

- `Task.getxx` Obtains the xx vector for a solution.

A.2.158 Task.gety()

```
Task.gety(
    whichsol,
    y)
```

Obtains the y vector for a solution.

Arguments

whichsol : soltype
: Selects a solution.

y : double[]
: The y vector.

Description:

Obtains the y vector for a solution.

See also

- Task.getyslice Obtains a slice of the y vector for a solution.

A.2.159 Task.getyslice()

```
Task.getyslice(
    whichsol,
    first,
    last,
    y)
```

Obtains a slice of the y vector for a solution.

Arguments

first : int
: First index in the sequence.

last : int
: Last index plus 1 in the sequence.

whichsol : soltype
: Selects a solution.

y : double[]
: Vector of dual variables corresponding to the constraints.

Description:

Obtains a slice of the y vector for a solution.

See also

- Task.gety Obtains the y vector for a solution.

A.2.160 Task.initbasissolve()

```
Task.initbasissolve(basis)
```

Prepare a task for basis solver.

Arguments

basis : int[]

The array of basis indexes to use.

The array is interpreted as follows: If $\texttt{basis}[i] \leq numcon - 1$, then $x^c_{\texttt{basis}[i]}$ is in the basis at position i, otherwise $x_{\texttt{basis}[i]-\texttt{numcon}}$ is in the basis at position i.

Description:

Prepare a task for use with the Task.solvewithbasis function.

This function should be called

- immediately before the first call to Task.solvewithbasis, and
- immediately before any subsequent call to Task.solvewithbasis if the task has been modified.

If the basis is singular i.e. not invertible, then

the exception rescode.err_basis_singular is generated.

A.2.161 Task.inputdata()

```
Task.inputdata(
    maxnumcon,
    maxnumvar,
    c,
    cfix,
    aptrb,
    aptre,
    asub,
    aval,
    bkc,
    blc,
    buc,
    bkx,
    blx,
    bux)
```

Input the linear part of an optimization task in one function call.

Arguments

`aptrb : long[]`
: Row or column end pointers.

`aptre : long[]`
: Row or column start pointers.

`asub : int[]`
: Coefficient subscripts.

`aval : double[]`
: Coefficient coefficient values.

`bkc : boundkey`
: Bound keys for the constraints.

`bkx : boundkey`
: Bound keys for the variables.

`blc : double[]`
: Lower bounds for the constraints.

`blx : double[]`
: Lower bounds for the variables.

`buc : double[]`
: Upper bounds for the constraints.

`bux : double[]`
: Upper bounds for the variables.

`c : double[]`
: Linear terms of the objective as a dense vector. The lengths is the number of variables.

`cfix : double`
: Fixed term in the objective.

`maxnumcon : int`
: Number of preallocated constraints in the optimization task.

`maxnumvar : int`
: Number of preallocated variables in the optimization task.

Description:

Input the linear part of an optimization problem.

The non-zeros of A are inputted column-wise in the format described in Section 5.13.3.2.

For an explained code example see Section 5.2 and Section 5.13.3.

A.2.162 Task.isdouparname()

```
param = Task.isdouparname(parname)
```

Checks a double parameter name.

Arguments

`param` : `dparam`
Which parameter.

`parname` : `str`
Parameter name.

Description:

Checks whether `parname` is a valid double parameter name.

A.2.163 Task.isintparname()

```
param = Task.isintparname(parname)
```

Checks an integer parameter name.

Arguments

`param` : `iparam`
Which parameter.

`parname` : `str`
Parameter name.

Description:

Checks whether `parname` is a valid integer parameter name.

A.2.164 Task.isstrparname()

```
param = Task.isstrparname(parname)
```

Checks a string parameter name.

Arguments

param : sparam
Which parameter.

parname : str
Parameter name.

Description:

Checks whether `parname` is a valid string parameter name.

A.2.165 `Task.linkfiletostream()`

```
Task.linkfiletostream(
    whichstream,
    filename,
    append)
```

Directs all output from a task stream to a file.

Arguments

append : int
If this argument is 0 the output file will be overwritten, otherwise text is append to the output file.

filename : str
The name of the file where text from the stream defined by `whichstream` is written.

whichstream : streamtype
Index of the stream.

Description:

Directs all output from a task stream to a file.

A.2.166 `Task.onesolutionsummary()`

```
Task.onesolutionsummary(
    whichstream,
    whichsol)
```

Prints a short summary for the specified solution.

Arguments

whichsol : soltype
Selects a solution.

`whichstream : streamtype`
Index of the stream.

Description:

Prints a short summary for a specified solution.

A.2.167 `Task.optimize()`

```
trmcode = Task.optimize()
```

Optimizes the problem.

Arguments

`trmcode : rescode`
Is either `rescode.ok` or a termination response code.

Description:

Calls the optimizer. Depending on the problem type and the selected optimizer this will call one of the optimizers in MOSEK. By default the interior point optimizer will be selected for continuous problems. The optimizer may be selected manually by setting the parameter `iparam.optimizer`.

See also

- `Task.optimizeconcurrent` Optimize a given task with several optimizers concurrently.
- `Task.getsolution` Obtains the complete solution.
- `Task.getsolutioni` Obtains the solution for a single constraint or variable.
- `Task.getsolutioninfo` Obtains information about of a solution.
- `iparam.optimizer` Controls which optimizer is used to optimize the task.

A.2.168 `Task.optimizeconcurrent()`

```
Task.optimizeconcurrent(taskarray)
```

Optimize a given task with several optimizers concurrently.

Arguments

`taskarray : mosek.Task[]`
An array of `num` tasks.

Description:

Solves several instances of the same problem in parallel, with unique parameter settings for each task. The argument `task` contains the problem to be solved. `taskarray` is a pointer to an array of `num` empty tasks. The task `task` and the `num` tasks pointed to by `taskarray` are solved in parallel. That is $\texttt{num} + 1$ 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 `Task.optimizeconcurrent` returns `task` holds the optimal solution of the task which finished first. If all the concurrent optimizations finished without providing an optimal solution the error code from the solution of the task `task` is returned.

In summary a call to `Task.optimizeconcurrent` does the following:

- All data except task parameters (`iparam`, `dparam` and `sparam`) in `task` is copied to each of the tasks in `taskarray`. In particular this means that any solution in `task` is copied to the other tasks. Call-back functions are not copied.
- The tasks `task` and the `num` tasks in `taskarray` are started in parallel.
- When a task finishes providing an optimal solution (or a certificate of infeasibility) its solution is copied to `task` and all other tasks are stopped.

Observe the concurrent optimizer is not deterministic.

For an explained code example see Section 11.6.4.

A.2.169 `Task.optimizersummary()`

```
Task.optimizersummary(whichstream)
```

Prints a short summary with optimizer statistics for last optimization.

Arguments

`whichstream` : `streamtype`

Index of the stream.

Description:

Prints a short summary with optimizer statistics for last optimization.

A.2.170 `Task.primalrepair()`

```
Task.primalrepair(
    wlc,
    wuc,
    wlx,
    wux)
```

The function repairs a primal infeasible optimization problem by adjusting the bounds on the constraints and variables.

Arguments

`wlc : double[]`

$(w_l^c)_i$ is the weight associated with relaxing the lower bound on constraint i. If the weigth is negative, then the lower bound is not relaxed. Moreover, if the argument is `None`, then all the weights are assumed to be 1.

`wlx : double[]`

$(w_l^x)_j$ is the weight associated with relaxing the upper bound on constraint j. If the weigth is negative, then the lower bound is not relaxed. Moreover, if the argument is `None`, then all the weights are assumed to be 1.

`wuc : double[]`

$(w_u^c)_i$ is the weight associated with relaxing the upper bound on constraint i. If the weigth is negative, then the upper bound is not relaxed. Moreover, if the argument is `None`, then all the weights are assumed to be 1.

`wux : double[]`

$(w_l^x)_i$ is the weight associated with relaxing the upper bound on variable j. If the weigth is negative, then the upper bound is not relaxed. Moreover, if the argument is `None`, then all the weights are assumed to be 1.

Description:

The function repairs a primal infeasible optimization problem by adjusting the bounds on the constraints and variables where the adjustment is computed as the minimal weigthed sum relaxation to the bounds on the constraints and variables.

The function is applicable to linear and conic problems possibly having integer constrained variables.

Observe that when computing the minimal weighted relaxation then the termination tolerance specified by the parameters of the task is employed. For instance the parameter `iparam.mio_mode` can be used make MOSEK ignore the integer constraints during the repair leading to a possibly a much faster repair. However, the drawback is of course that the repaired problem may not have integer feasible solution.

Note the function modifies the bounds on the constraints and variables. If this is not a desired feature, then apply the fucntion to a cloned task.

See also

- iparam.primal_repair_optimizer Controls which optimizer that is used to find the optimal repair.
- iparam.log_feas_repair Controls the amount of output printed when performing feasibility repair. A value higher than one means extensive logging.
- dinfitem.primal_repair_penalty_obj The optimal objective value of the penalty function.

A.2.171 Task.primalsensitivity()

```
Task.primalsensitivity(
    subi,
    marki,
    subj,
    markj,
    leftpricei,
    rightpricei,
    leftrangei,
    rightrangei,
    leftpricej,
    rightpricej,
    leftrangej,
    rightrangej)
```

Perform sensitivity analysis on bounds.

Arguments

`leftpricei : double[]`

`leftpricei[i]` is the left shadow price for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

`leftpricej : double[]`

`leftpricej[j]` is the left shadow price for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]`.

`leftrangei : double[]`

`leftrangei[i]` is the left range for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

`leftrangej : double[]`

`leftrangej[j]` is the left range for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]`.

`marki : mark`

The value of `marki[i]` specifies for which bound (upper or lower) on constraint `subi[i]` sensitivity analysis should be performed.

markj : mark
The value of markj[j] specifies for which bound (upper or lower) on variable subj[j] sensitivity analysis should be performed.

rightpricei : double[]
rightpricei[i] is the right shadow price for the upper/lower bound (indicated by marki[i]) of the constraint with index subi[i].

rightpricej : double[]
rightpricej[j] is the right shadow price for the upper/lower bound (indicated by marki[j]) on variable subj[j] .

rightrangei : double[]
rightrangei[i] is the right range for the upper/lower bound (indicated by marki[i]) of the constraint with index subi[i].

rightrangej : double[]
rightrangej[j] is the right range for the upper/lower bound (indicated by marki[j]) on variable subj[j].

subi : int[]
Indexes of bounds on constraints to analyze.

subj : int[]
Indexes of bounds on variables to analyze.

Description:

Calculates sensitivity information for bounds on variables and constraints.

For details on sensitivity analysis and the definitions of *shadow price* and *linearity interval* see chapter 15.

The constraints for which sensitivity analysis is performed are given by the data structures:

- subi Index of constraint to analyze.
- marki Indicate for which bound of constraint subi[i] sensitivity analysis is performed. If marki[i] = mark.up the upper bound of constraint subi[i] is analyzed, and if marki[i] = mark.lo the lower bound is analyzed. If subi[i] is an equality constraint, either mark.lo or mark.up can be used to select the constraint for sensitivity analysis.

Consider the problem:

$$
\begin{array}{lccc}
\text{minimize} & x_1 + x_2 & & \\
\text{subject to} -1 \leq & x_1 - x_2 & \leq & 1, \\
 & x_1 & = & 0, \\
 & x_1 \geq 0, x_2 \geq 0 & &
\end{array}
$$

Suppose that

- numi = 1;
- subi = [0];

- marki = [mark.up]

then

leftpricei[0], rightpricei[0], leftrangei[0] and rightrangei[0] will contain the sensitivity information for the upper bound on constraint 0 given by the expression:

$$x_1 - x_2 \leq 1$$

Similarly, the variables for which to perform sensitivity analysis are given by the structures:

- subj Index of variables to analyze.
- markj Indicate for which bound of variable subi[j] sensitivity analysis is performed. If markj[j] = mark.up the upper bound of constraint subi[j] is analyzed, and if markj[j] = mark.lo the lower bound is analyzed. If subi[j] is an equality constraint, either mark.lo or mark.up can be used to select the constraint for sensitivity analysis.

For an example, please see Section 15.5.

The type of sensitivity analysis to be performed (basis or optimal partition) is controlled by the parameter iparam.sensitivity_type.

See also

- Task.dualsensitivity Performs sensitivity analysis on objective coefficients.
- Task.sensitivityreport Creates a sensitivity report.
- iparam.sensitivity_type Controls which type of sensitivity analysis is to be performed.
- iparam.log_sensitivity Control logging in sensitivity analyzer.
- iparam.log_sensitivity_opt Control logging in sensitivity analyzer.

A.2.172 Task.printdata()

```
Task.printdata(
    whichstream,
    firsti,
    lasti,
    firstj,
    lastj,
    firstk,
    lastk,
    c,
    qo,
    a,
    qc,
    bc,
    bx,
    vartype,
    cones)
```

Prints a part of the problem data to a stream.

Arguments

`a : int`

If non-zero A is printed.

`bc : int`

If non-zero the constraints bounds are printed.

`bx : int`

If non-zero the variable bounds are printed.

`c : int`

If non-zero c is printed.

`cones : int`

If non-zero the conic data is printed.

`firsti : int`

Index of first constraint for which data should be printed.

`firstj : int`

Index of first variable for which data should be printed.

`firstk : int`

Index of first cone for which data should be printed.

`lasti : int`

Index of last constraint plus 1 for which data should be printed.

`lastj : int`

Index of last variable plus 1 for which data should be printed.

`lastk : int`

Index of last cone plus 1 for which data should be printed.

`qc : int`

If non-zero Q^k is printed for the relevant constraints.

`qo : int`

If non-zero Q^o is printed.

`vartype : int`

If non-zero the variable types are printed.

`whichstream : streamtype`

Index of the stream.

Description:

Prints a part of the problem data to a stream. This function is normally used for debugging purposes only, e.g. to verify that the correct data has been inputted.

A.2.173 `Task.printparam()`

```
Task.printparam()
```

Prints the current parameter settings.

Description:

Prints the current parameter settings to the message stream.

A.2.174 `Task.putacol()`

```
Task.putacol(
    j,
    subj,
    valj)
```

Replaces all elements in one column of A.

Arguments

`j : int`
Index of column in A.

`subj : int[]`
Row indexes of non-zero values in column j of A.

`valj : double[]`
New non-zero values of column j in A.

Description:

Replaces all entries in column j of A. Assuming that there are no duplicate subscripts in `subj`, assignment is performed as follows:

$$A_{\mathtt{subj}[k],\mathtt{j}} = \mathtt{valj}[k], \quad k = 0, \ldots, \mathtt{nzj} - 1$$

All other entries in column j are set to zero.

See also

- `Task.putarow` Replaces all elements in one row of A.
- `Task.putaij` Changes a single value in the linear coefficient matrix.
- `Task.putmaxnumanz` The function changes the size of the preallocated storage for linear coefficients.

A.2.175 Task.putacollist()

```
Task.putacollist(
    sub,
    ptrb,
    ptre,
    asub,
    aval)
```

Replaces all elements in several columns the linear constraint matrix by new values.

Arguments

`asub : int[]`
: `asub` contains the new variable indexes.

`aval : double[]`
: Coefficient coefficient values.

`ptrb : long[]`
: Array of pointers to the first element in the columns stored in `asub` and `aval`. For an explanation of the meaning of `ptrb` see Section 5.13.3.2.

`ptre : long[]`
: Array of pointers to the last element plus one in the columns stored in `asub` and `aval`. For an explanation of the meaning of `ptre` see Section 5.13.3.2.

`sub : int[]`
: Indexes of columns that should be replaced. `sub` should not contain duplicate values.

Description:

Replaces all elements in a set of columns of A. The elements are replaced as follows

$$
\begin{array}{ll}
\texttt{for} & i = 0, \ldots, num - 1 \\
 & a_{\texttt{asub}[k],\texttt{sub}[i]} = \texttt{aval}[k], \quad k = \texttt{aptrb}[i], \ldots, \texttt{aptre}[i] - 1.
\end{array}
$$

See also

- Task.putmaxnumanz The function changes the size of the preallocated storage for linear coefficients.

A.2.176 Task.putacolslice()

```
Task.putacolslice(
    first,
    last,
    ptrb,
    ptre,
    asub,
    aval)
```

Replaces all elements in several columns the linear constraint matrix by new values.

Arguments

`asub : int[]`
: `asub` contains the new variable indexes.

`aval : double[]`
: Coefficient coefficient values.

`first : int`
: First column in the slice.

`last : int`
: Last column plus one in the slice.

`ptrb : long[]`
: Array of pointers to the first element in the columns stored in `asub` and `aval`.
  For an explanation of the meaning of `ptrb` see Section 5.13.3.2.

`ptre : long[]`
: Array of pointers to the last element plus one in the columns stored in `asub` and `aval`.
  For an explanation of the meaning of `ptre` see Section 5.13.3.2.

Description:

Replaces all elements in a set of columns of A.

See also

- `Task.putmaxnumanz` The function changes the size of the preallocated storage for linear coefficients.

A.2.177 `Task.putaij()`

```
Task.putaij(
    i,
    j,
    aij)
```

Changes a single value in the linear coefficient matrix.

Arguments

`aij : double`
: New coefficient for $a_{i,j}$.

`i : int`
: Index of the constraint in which the change should occur.

j : int
Index of the variable in which the change should occur.

Description:

Changes a coefficient in A using the method

$$a_{\mathtt{ij}} = \mathtt{aij}.$$

See also

- Task.putarow Replaces all elements in one row of A.
- Task.putacol Replaces all elements in one column of A.
- Task.putaij Changes a single value in the linear coefficient matrix.
- Task.putmaxnumanz The function changes the size of the preallocated storage for linear coefficients.

A.2.178 Task.putaijlist()

```
Task.putaijlist(
    subi,
    subj,
    valij)
```

Changes one or more coefficients in the linear constraint matrix.

Arguments

subi : int[]
Constraint indexes in which the change should occur.

subj : int[]
Variable indexes in which the change should occur.

valij : double[]
New coefficient values for $a_{i,j}$.

Description:

Changes one or more coefficients in A using the method

$$a_{\mathtt{subi[k]},\mathtt{subj[k]}} = \mathtt{valij[k]}, \quad k = 0, \ldots, \mathtt{num} - 1.$$

See also

- Task.putarow Replaces all elements in one row of A.
- Task.putacol Replaces all elements in one column of A.
- Task.putaij Changes a single value in the linear coefficient matrix.
- Task.putmaxnumanz The function changes the size of the preallocated storage for linear coefficients.

A.2.179 Task.putarow()

```
Task.putarow(
    i,
    subi,
    vali)
```

Replaces all elements in one row of A.

Arguments

i : int
Index of row in A.

subi : int[]
Row indexes of non-zero values in row i of A.

vali : double[]
New non-zero values of row i in A.

Description:

Replaces all entries in row i of A. Assuming that there are no duplicate subscripts in `subi`, assignment is performed as follows:

$$A_{\mathtt{i},\mathtt{subi}[k]} = \mathtt{vali}[k], \quad k = 0, \ldots, \mathtt{nzi} - 1$$

All other entries in row i are set to zero.

See also

- Task.putacol Replaces all elements in one column of A.
- Task.putaij Changes a single value in the linear coefficient matrix.
- Task.putmaxnumanz The function changes the size of the preallocated storage for linear coefficients.

A.2.180 Task.putarowlist()

```
Task.putarowlist(
    sub,
    ptrb,
    ptre,
    asub,
    aval)
```

Replaces all elements in several rows the linear constraint matrix by new values.

Arguments

`asub` : `int[]`
asub contains the new variable indexes.

`aval` : `double[]`
Coefficient coefficient values.

`ptrb` : `long[]`
Array of pointers to the first element in the rows stored in `asub` and `aval`. For an explanation of the meaning of `ptrb` see Section 5.13.3.2.

`ptre` : `long[]`
Array of pointers to the last element plus one in the rows stored in `asub` and `aval`. For an explanation of the meaning of `ptre` see Section 5.13.3.2.

`sub` : `int[]`
Indexes of rows or columns that should be replaced. `sub` should not contain duplicate values.

Description:

Replaces all elements in a set of rows of A. The elements are replaced as follows

$$\begin{array}{ll} \texttt{for} & i = \texttt{first}, \ldots, \texttt{last} - 1 \\ & a_{\texttt{sub[i]},\texttt{asub}[k]} = \texttt{aval}[k], \quad k = \texttt{aptrb}[i], \ldots, \texttt{aptre}[i] - 1. \end{array}$$

See also

- `Task.putmaxnumanz` The function changes the size of the preallocated storage for linear coefficients.

A.2.181 `Task.putbarablocktriplet()`

```
Task.putbarablocktriplet(
    num,
    subi,
    subj,
    subk,
    subl,
    valijkl)
```

Inputs barA in block triplet form.

Arguments

`num` : `long`
Number of elements in the block triplet form.

`subi` : `int[]`
Constraint index.

subj : int[]
Symmetric matrix variable index.

subk : int[]
Block row index.

subl : int[]
Block column index.

valijkl : double[]
The numerical value associated with the block triplet.

Description:

Inputs the $\bar{A}$ in block triplet form.

A.2.182 Task.putbaraij()

```
Task.putbaraij(
    i,
    j,
    sub,
    weights)
```

Inputs an element of barA.

Arguments

i : int
Row index of $\bar{A}$.

j : int
Column index of $\bar{A}$.

sub : long[]
See argument `weights` for an explenation.

weights : double[]
`weights[k]` times `sub[k]`'th term of E is added to $\bar{A}_{ij}$.

Description:

This function puts one element associated with $\bar{X}_j$ in the $\bar{A}$ matrix.

Each element in the $\bar{A}$ matrix is a weighted sum of symmetric matrixes, i.e. $\bar{A}_{ij}$ is a symmetric matrix with dimensions as $\bar{X}_j$. By default all elements in $\bar{A}$ are 0, so only non-zero elements need be added.

Setting the same elements again will overwrite the earlier entry.

The symmetric matrixes themselves are defined separately using the funtion `Task.appendsparsesymmat`.

A.2.183 `Task.putbarcblocktriplet()`

```
Task.putbarcblocktriplet(
    num,
    subj,
    subk,
    subl,
    valjkl)
```

Inputs barC in block triplet form.

Arguments

`num : long`
: Number of elements in the block triplet form.

`subj : int[]`
: Symmetric matrix variable index.

`subk : int[]`
: Block row index.

`subl : int[]`
: Block column index.

`valjkl : double[]`
: The numerical value associated with the block triplet.

Description:

Inputs the $\bar{C}$ in block triplet form.

A.2.184 `Task.putbarcj()`

```
Task.putbarcj(
    j,
    sub,
    weights)
```

Changes one element in barc.

Arguments

`j : int`
: Index of the element in $\bar{c}$ that should be changed.

`sub : long[]`
: `sub` is list of indexes of those symmetric matrixes appearing in sum.

weights : double[]
The weights of the terms in the weighted sum that forms $\mathtt{c}_j$.

Description:

This function puts one element associated with $\bar{X}_j$ in the $\bar{c}$ vector.

Each element in the $\bar{c}$ vector is a weighted sum of symmetric matrixes, i.e. $\bar{c}_j$ is a symmetric matrix with dimensions as $\bar{X}_j$. By default all elements in $\bar{c}$ are 0, so only non-zero elements need be added.

Setting the same elements again will overwrite the earlier entry.

The symmetric matrixes themselves are defined separately using the funtion Task.appendsparsesymmat.

A.2.185 Task.putbarsj()

```
Task.putbarsj(
    whichsol,
    j,
    barsj)
```

Sets the dual solution for a semidefinite variable.

Arguments

barsj : double[]
Value of $\bar{s}_j$.

j : int
Index of the semidefinite variable.

whichsol : soltype
Selects a solution.

Description:

Sets the dual solution for a semidefinite variable.

A.2.186 Task.putbarvarname()

```
Task.putbarvarname(
    j,
    name)
```

Puts the name of a semidefinite variable.

Arguments

`j` : `int`
Index of the variable.

`name` : `str`
The variable name.

Description:

Puts the name of a semidefinite variable.

See also

- `Task.getbarvarnamelen` Obtains the length of a name of a semidefinite variable.

A.2.187 `Task.putbarxj()`

```
Task.putbarxj(
    whichsol,
    j,
    barxj)
```

Sets the primal solution for a semidefinite variable.

Arguments

`barxj` : `double[]`
Value of $\bar{X}_j$.

`j` : `int`
Index of the semidefinite variable.

`whichsol` : `soltype`
Selects a solution.

Description:

Sets the primal solution for a semidefinite variable.

A.2.188 `Task.putbound()`

```
Task.putbound(
    accmode,
    i,
    bk,
    bl,
    bu)
```

Changes the bound for either one constraint or one variable.

Arguments

`accmode` : `accmode`
: Defines whether the bound for a constraint or a variable is changed.

`bk` : `boundkey`
: New bound key.

`bl` : `double`
: New lower bound.

`bu` : `double`
: New upper bound.

`i` : `int`
: Index of the constraint or variable.

Description:

Changes the bounds for either one constraint or one variable.

If the a bound value specified is numerically larger than `dparam.data_tol_bound_inf` it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than `dparam.data_tol_bound_wrn`, a warning will be displayed, but the bound is inputted as specified.

See also

- `Task.putboundlist` Changes the bounds of constraints or variables.

A.2.189 `Task.putboundlist()`

```
Task.putboundlist(
    accmode,
    sub,
    bk,
    bl,
    bu)
```

Changes the bounds of constraints or variables.

Arguments

`accmode` : `accmode`
: Defines whether bounds for constraints (`accmode.con`) or variables (`accmode.var`) are changed.

`bk` : `boundkey`
: Constraint or variable index `sub[t]` is assigned the bound key `bk[t]`.

`bl : double[]`
Constraint or variable index `sub[t]` is assigned the lower bound `bl[t]`.

`bu : double[]`
Constraint or variable index `sub[t]` is assigned the upper bound `bu[t]`.

`sub : int[]`
Subscripts of the bounds that should be changed.

Description:

Changes the bounds for either some constraints or variables. If multiple bound changes are specified for a constraint or a variable, only the last change takes effect.

See also

- `Task.putbound` Changes the bound for either one constraint or one variable.
- `dparam.data_tol_bound_inf` Data tolerance threshold.
- `dparam.data_tol_bound_wrn` Data tolerance threshold.

A.2.190 `Task.putboundslice()`

```
Task.putboundslice(
    con,
    first,
    last,
    bk,
    bl,
    bu)
```

Modifies bounds.

Arguments

`bk : boundkey`
Bound keys.

`bl : double[]`
Values for lower bounds.

`bu : double[]`
Values for upper bounds.

`con : accmode`
Defines whether bounds for constraints (`accmode.con`) or variables (`accmode.var`) are changed.

`first : int`
First index in the sequence.

`last : int`

Last index plus 1 in the sequence.

Description:

Changes the bounds for a sequence of variables or constraints.

See also

- `Task.putbound` Changes the bound for either one constraint or one variable.
- `dparam.data_tol_bound_inf` Data tolerance threshold.
- `dparam.data_tol_bound_wrn` Data tolerance threshold.

A.2.191 `Task.putcfix()`

```
Task.putcfix(cfix)
```

Replaces the fixed term in the objective.

Arguments

`cfix : double`

Fixed term in the objective.

Description:

Replaces the fixed term in the objective by a new one.

A.2.192 `Task.putcj()`

```
Task.putcj(
    j,
    cj)
```

Modifies one linear coefficient in the objective.

Arguments

`cj : double`

New value of c_j.

`j : int`

Index of the variable for which c should be changed.

Description:

Modifies one coefficient in the linear objective vector c, i.e.

$$c_{\mathtt{j}} = \mathtt{cj}.$$

See also

- Task.putclist Modifies a part of the linear objective coefficients.
- Task.putcslice Modifies a slice of the linear objective coefficients.

A.2.193 Task.putclist()

```
Task.putclist(
    subj,
    val)
```

Modifies a part of the linear objective coefficients.

Arguments

subj : int[]
: Index of variables for which c should be changed.

val : double[]
: 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` only the last entry is used.

A.2.194 Task.putconbound()

```
Task.putconbound(
    i,
    bk,
    bl,
    bu)
```

Changes the bound for one constraint.

Arguments

bk : boundkey
New bound key.

bl : double
New lower bound.

bu : double
New upper bound.

i : int
Index of the constraint.

Description:

Changes the bounds for one constraint.

If the a bound value specified is numerically larger than `dparam.data_tol_bound_inf` it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than `dparam.data_tol_bound_wrn`, a warning will be displayed, but the bound is inputted as specified.

See also

- `Task.putconboundslice` Changes the bounds for a slice of the constraints.

A.2.195 Task.putconboundlist()

```
Task.putconboundlist(
    sub,
    bkc,
    blc,
    buc)
```

Changes the bounds of a list of constraints.

Arguments

bkc : boundkey
New bound keys.

blc : double[]
New lower bound values.

buc : double[]
New upper bound values.

sub : int[]
List constraints indexes.

Description:

Changes the bounds for a list of constraints. If multiple bound changes are specified for a constraint, then only the last change takes effect.

See also

- `Task.putconbound` Changes the bound for one constraint.
- `Task.putconboundslice` Changes the bounds for a slice of the constraints.
- `dparam.data_tol_bound_inf` Data tolerance threshold.
- `dparam.data_tol_bound_wrn` Data tolerance threshold.

A.2.196 `Task.putconboundslice()`

```
Task.putconboundslice(
    first,
    last,
    bk,
    bl,
    bu)
```

Changes the bounds for a slice of the constraints.

Arguments

`bk` : `boundkey`
New bound keys.

`bl` : `double[]`
New lower bounds.

`bu` : `double[]`
New upper bounds.

`first` : `int`
Index of the first constraint in the slice.

`last` : `int`
Index of the last constraint in the slice plus 1.

Description:

Changes the bounds for a slice of the constraints.

See also

- `Task.putconbound` Changes the bound for one constraint.
- `Task.putconboundlist` Changes the bounds of a list of constraints.

A.2.197 `Task.putcone()`

```
Task.putcone(
    k,
    conetype,
    conepar,
    submem)
```

Replaces a conic constraint.

Arguments

`conepar : double`
This argument is currently not used. Can be set to 0.0.

`conetype : conetype`
Specifies the type of the cone.

`k : int`
Index of the cone.

`submem : int[]`
Variable subscripts of the members in the cone.

Description:

Replaces a conic constraint.

A.2.198 `Task.putconename()`

```
Task.putconename(
    j,
    name)
```

Puts the name of a cone.

Arguments

`j : int`
Index of the variable.

`name : str`
The variable name.

Description:

Puts the name of a cone.

A.2.199 `Task.putconname()`

```
Task.putconname(
    i,
    name)
```

Puts the name of a constraint.

Arguments

`i : int`
Index of the variable.

`name : str`
The variable name.

Description:

Puts the name of a constraint.

A.2.200 `Task.putcslice()`

```
Task.putcslice(
    first,
    last,
    slice)
```

Modifies a slice of the linear objective coefficients.

Arguments

`first : int`
First element in the slice of c.

`last : int`
Last element plus 1 of the slice in c to be changed.

`slice : double[]`
New numerical values for coefficients in c that should be modified.

Description:

Modifies a slice in the linear term c in the objective using the principle

$$c_{\mathtt{j}} = \mathtt{slice}[\mathtt{j} - \mathtt{first}], \quad j = first, .., last - 1$$

A.2.201 `Task.putdouparam()`

```
Task.putdouparam(
    param,
    parvalue)
```

Sets a double parameter.

Arguments

`param` : `dparam`
Which parameter.

`parvalue` : `double`
Parameter value.

Description:

Sets the value of a double parameter.

A.2.202 `Task.putintparam()`

```
Task.putintparam(
    param,
    parvalue)
```

Sets an integer parameter.

Arguments

`param` : `iparam`
Which parameter.

`parvalue` : `int`
Parameter value.

Description:

Sets the value of an integer parameter.

A.2.203 `Task.putmaxnumanz()`

```
Task.putmaxnumanz(maxnumanz)
```

The function changes the size of the preallocated storage for linear coefficients.

Arguments

`maxnumanz : long`
New size of the storage reserved for storing A.

Description:

MOSEK stores only the non-zero elements in A. Therefore, MOSEK cannot predict how much storage is required to store A. Using this function it is possible to specify the number of non-zeros to preallocate for storing A.

If the number of non-zeros in the problem is known, it is a good idea to set `maxnumanz` slightly larger than this number, otherwise a rough estimate can be used. In general, if A is inputted in many small chunks, setting this value may speed up the the data input phase.

It is not mandatory to call this function, since MOSEK will reallocate internal structures whenever it is necessary.

See also

- `iinfitem.sto_num_a_realloc` Number of times the storage for storing the linear coefficient matrix has been changed.

A.2.204 `Task.putmaxnumbarvar()`

```
Task.putmaxnumbarvar(maxnumbarvar)
```

Sets the number of preallocated symmetric matrix variables in the optimization task.

Arguments

`maxnumbarvar : int`
The maximum number of semidefinite variables.

Description:

Sets the number of preallocated symmetric matrix variables in the optimization task. When this number of variables is reached MOSEK will automatically allocate more space for variables.

It is not mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

Please note that `maxnumbarvar` must be larger than the current number of variables in the task.

A.2.205 Task.putmaxnumcon()

```
Task.putmaxnumcon(maxnumcon)
```

Sets the number of preallocated constraints in the optimization task.

Arguments

`maxnumcon : int`
Number of preallocated constraints in the optimization task.

Description:

Sets the number of preallocated constraints in the optimization task. When this number of constraints is reached MOSEK will automatically allocate more space for constraints.

It is never mandatory to call this function, since MOSEK will reallocate any internal structures whenever it is required.

Please note that `maxnumcon` must be larger than the current number of constraints in the task.

A.2.206 Task.putmaxnumcone()

```
Task.putmaxnumcone(maxnumcone)
```

Sets the number of preallocated conic constraints in the optimization task.

Arguments

`maxnumcone : int`
Number of preallocated conic constraints in the optimization task.

Description:

Sets the number of preallocated conic constraints in the optimization task. When this number of conic constraints is reached MOSEK will automatically allocate more space for conic constraints.

It is never mandatory to call this function, since MOSEK will reallocate any internal structures whenever it is required.

Please note that `maxnumcon` must be larger than the current number of constraints in the task.

A.2.207 Task.putmaxnumqnz()

```
Task.putmaxnumqnz(maxnumqnz)
```

Changes the size of the preallocated storage for quadratic terms.

Arguments

`maxnumqnz : long`
Number of non-zero elements preallocated in quadratic coefficient matrixes.

Description:

MOSEK stores only the non-zero elements in Q. Therefore, MOSEK cannot predict how much storage is required to store Q. Using this function it is possible to specify the number non-zeros to preallocate for storing Q (both objective and constraints).

It may be advantageous to reserve more non-zeros for Q than actually needed since it may improve the internal efficiency of MOSEK, however, it is never worthwhile to specify more than the double of the anticipated number of non-zeros in Q.

It is never mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

A.2.208 Task.putmaxnumvar()

```
Task.putmaxnumvar(maxnumvar)
```

Sets the number of preallocated variables in the optimization task.

Arguments

`maxnumvar : int`
Number of preallocated variables in the optimization task.

Description:

Sets the number of preallocated variables in the optimization task. When this number of variables is reached MOSEK will automatically allocate more space for variables.

It is never mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

Please note that `maxnumvar` must be larger than the current number of variables in the task.

A.2.209 Task.putnadouparam()

```
Task.putnadouparam(
    paramname,
    parvalue)
```

Sets a double parameter.

Arguments

`paramname : str`
: Name of a parameter.

`parvalue : double`
: Parameter value.

Description:

Sets the value of a named double parameter.

A.2.210 Task.putnaintparam()

```
Task.putnaintparam(
    paramname,
    parvalue)
```

Sets an integer parameter.

Arguments

`paramname : str`
: Name of a parameter.

`parvalue : int`
: Parameter value.

Description:

Sets the value of a named integer parameter.

A.2.211 Task.putnastrparam()

```
Task.putnastrparam(
    paramname,
    parvalue)
```

Sets a string parameter.

Arguments

`paramname : str`
Name of a parameter.

`parvalue : str`
Parameter value.

Description:

Sets the value of a named string parameter.

A.2.212 `Task.putobjname()`

```
Task.putobjname(objname)
```

Assigns a new name to the objective.

Arguments

`objname : str`
Name of the objective.

Description:

Assigns the name given by `objname` to the objective function.

A.2.213 `Task.putobjsense()`

```
Task.putobjsense(sense)
```

Sets the objective sense.

Arguments

`sense : objsense`
The objective sense of the task. The values `objsense.maximize` and `objsense.minimize` means that the the problem is maximized or minimized respectively.

Description:

Sets the objective sense of the task.

See also

- `Task.getobjsense` Gets the objective sense.

A.2.214 Task.putparam()

```
Task.putparam(
    parname,
    parvalue)
```

Modifies the value of parameter.

Arguments

parname : str
Parameter name.

parvalue : str
Parameter value.

Description:

Checks if a `parname` is valid parameter name. If it is, the parameter is assigned the value specified by `parvalue`.

A.2.215 Task.putqcon()

```
Task.putqcon(
    qcsubk,
    qcsubi,
    qcsubj,
    qcval)
```

Replaces all quadratic terms in constraints.

Arguments

qcsubi : int[]
Row subscripts for quadratic constraint matrix.

qcsubj : int[]
Column subscripts for quadratic constraint matrix.

qcsubk : int[]
Constraint subscripts for quadratic coefficients.

qcval : double[]
Quadratic constraint coefficient values.

Description:

Replaces all quadratic entries in the constraints. Consider constraints on the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{numvar-1} \sum_{j=0}^{numvar-1} q_{ij}^k x_i x_j + \sum_{j=0}^{numvar-1} a_{kj} x_j \leq u_k^c, \ k = 0, \ldots, m-1.$$

The function assigns values to q such that:

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

and

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

Values not assigned are set to zero.

Please note that duplicate entries are added together.

See also

- `Task.putqconk` Replaces all quadratic terms in a single constraint.
- `Task.putmaxnumqnz` Changes the size of the preallocated storage for quadratic terms.

A.2.216 `Task.putqconk()`

```
Task.putqconk(
    k,
    qcsubi,
    qcsubj,
    qcval)
```

Replaces all quadratic terms in a single constraint.

Arguments

`k : int`
: The constraint in which the new Q elements are inserted.

`qcsubi : int[]`
: Row subscripts for quadratic constraint matrix.

`qcsubj : int[]`
: Column subscripts for quadratic constraint matrix.

`qcval : double[]`
: Quadratic constraint coefficient values.

Description:

Replaces all the quadratic entries in one constraint k of the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{numvar-1} \sum_{j=0}^{numvar-1} q_{ij}^k x_i x_j + \sum_{j=0}^{numvar-1} a_{kj} x_j \leq u_k^c.$$

It is assumed that Q^k is symmetric, i.e. $q_{ij}^k = q_{ji}^k$, and therefore, only the values of q_{ij}^k for which $i \geq j$ should be inputted to MOSEK. To be precise, MOSEK uses the following procedure

1. $Q^k = 0$
2. for $t = 0$ to $numqonz - 1$
3. $\quad q^k_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} = q^k_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} + \texttt{qcval[t]}$
3. $\quad q^k_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} = q^k_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} + \texttt{qcval[t]}$

Please note that:

- For large problems it is essential for the efficiency that the function `Task.putmaxnumqnz` is employed to specify an appropriate `maxnumqnz`.
- Only the lower triangular part should be specified because Q^k is symmetric. Specifying values for q_{ij}^k where $i < j$ will result in an error.
- Only non-zero elements should be specified.
- The order in which the non-zero elements are specified is insignificant.
- Duplicate elements are added together. Hence, it is recommended not to specify the same element multiple times in `qosubi`, `qosubj`, and `qoval`.

For a code example see Section 5.5.2.

See also

- `Task.putqcon` Replaces all quadratic terms in constraints.
- `Task.putmaxnumqnz` Changes the size of the preallocated storage for quadratic terms.

A.2.217 `Task.putqobj()`

```
Task.putqobj(
    qosubi,
    qosubj,
    qoval)
```

Replaces all quadratic terms in the objective.

Arguments

`qosubi : int[]`

Row subscripts for quadratic objective coefficients.

qosubj : int[]
: Column subscripts for quadratic objective coefficients.

qoval : double[]
: Quadratic objective coefficient values.

Description:

Replaces all the quadratic terms in the objective

$$\frac{1}{2}\sum_{i=0}^{numvar-1}\sum_{j=0}^{numvar-1} q_{ij}^o x_i x_j + \sum_{j=0}^{numvar-1} c_j x_j + c^f.$$

It is assumed that Q^o is symmetric, i.e. $q_{ij}^o = q_{ji}^o$, and therefore, only the values of q_{ij}^o for which $i \geq j$ should be specified. To be precise, MOSEK uses the following procedure

1. $Q^o = 0$
2. for $t = 0$ to $numqonz - 1$
3. $\quad q^o_{\mathtt{qosubi[t]},\mathtt{qosubj[t]}} = q^o_{\mathtt{qosubi[t]},\mathtt{qosubj[t]}} + \mathtt{qoval}[\mathtt{t}]$
3. $\quad q^o_{\mathtt{qosubj[t]},\mathtt{qosubi[t]}} = q^o_{\mathtt{qosubj[t]},\mathtt{qosubi[t]}} + \mathtt{qoval}[\mathtt{t}]$

Please note that:

- Only the lower triangular part should be specified because Q^o is symmetric. Specifying values for q_{ij}^o where $i < j$ will result in an error.
- Only non-zero elements should be specified.
- The order in which the non-zero elements are specified is insignificant.
- Duplicate entries are added to together.

For a code example see Section 5.5.1.

A.2.218 Task.putqobjij()

```
Task.putqobjij(
    i,
    j,
    qoij)
```

Replaces one coefficient in the quadratic term in the objective.

Arguments

i : int
: Row index for the coefficient to be replaced.

j : int
: Column index for the coefficient to be replaced.

qoij : double
The new value for q_{ij}^o.

Description:

Replaces one coefficient in the quadratic term in the objective. The function performs the assignment

$$q_{\mathtt{ij}}^o = \mathtt{qoij}.$$

Only the elements in the lower triangular part are accepted. Setting q_{ij} with $j > i$ will cause an error.

Please note that replacing all quadratic element, one at a time, is more computationally expensive than replacing all elements at once. Use `Task.putqobj` instead whenever possible.

A.2.219 Task.putskc()

```
Task.putskc(
    whichsol,
    skc)
```

Sets the status keys for the constraints.

Arguments

skc : stakey
Status keys for the constraints.

whichsol : soltype
Selects a solution.

Description:

Sets the status keys for the constraints.

See also

- `Task.putskcslice` Sets the status keys for the constraints.

A.2.220 Task.putskcslice()

```
Task.putskcslice(
    whichsol,
    first,
    last,
    skc)
```

Sets the status keys for the constraints.

Arguments

`first` : `int`
: First index in the sequence.

`last` : `int`
: Last index plus 1 in the sequence.

`skc` : `stakey`
: Status keys for the constraints.

`whichsol` : `soltype`
: Selects a solution.

Description:

Sets the status keys for the constraints.

See also

- `Task.putskc` Sets the status keys for the constraints.

A.2.221 `Task.putskx()`

```
Task.putskx(
    whichsol,
    skx)
```

Sets the status keys for the scalar variables.

Arguments

`skx` : `stakey`
: Status keys for the variables.

`whichsol` : `soltype`
: Selects a solution.

Description:

Sets the status keys for the scalar variables.

See also

- `Task.putskxslice` Sets the status keys for the variables.

A.2.222 `Task.putskxslice()`

```
Task.putskxslice(
    whichsol,
    first,
    last,
    skx)
```

Sets the status keys for the variables.

Arguments

`first : int`
First index in the sequence.

`last : int`
Last index plus 1 in the sequence.

`skx : stakey`
Status keys for the variables.

`whichsol : soltype`
Selects a solution.

Description:

Sets the status keys for the variables.

A.2.223 `Task.putslc()`

```
Task.putslc(
    whichsol,
    slc)
```

Sets the slc vector for a solution.

Arguments

`slc : double[]`
The s_l^c vector.

`whichsol : soltype`
Selects a solution.

Description:

Sets the s_l^c vector for a solution.

See also

- `Task.putslcslice` Sets a slice of the slc vector for a solution.

A.2.224 Task.putslcslice()

```
Task.putslcslice(
    whichsol,
    first,
    last,
    slc)
```

Sets a slice of the slc vector for a solution.

Arguments

`first : int`
First index in the sequence.

`last : int`
Last index plus 1 in the sequence.

`slc : double[]`
Dual variables corresponding to the lower bounds on the constraints.

`whichsol : soltype`
Selects a solution.

Description:

Sets a slice of the s_l^c vector for a solution.

See also

- `Task.putslc` Sets the slc vector for a solution.

A.2.225 Task.putslx()

```
Task.putslx(
    whichsol,
    slx)
```

Sets the slx vector for a solution.

Arguments

`slx : double[]`
The s_l^x vector.

`whichsol : soltype`
Selects a solution.

Description:

Sets the s_l^x vector for a solution.

See also

- Task.putslx Sets the slx vector for a solution.

A.2.226 Task.putslxslice()

```
Task.putslxslice(
    whichsol,
    first,
    last,
    slx)
```

Sets a slice of the slx vector for a solution.

Arguments

`first : int`
: First index in the sequence.

`last : int`
: Last index plus 1 in the sequence.

`slx : double[]`
: Dual variables corresponding to the lower bounds on the variables.

`whichsol : soltype`
: Selects a solution.

Description:

Sets a slice of the s_l^x vector for a solution.

See also

- Task.putslx Sets the slx vector for a solution.

A.2.227 Task.putsnx()

```
Task.putsnx(
    whichsol,
    sux)
```

Sets the snx vector for a solution.

Arguments

`sux` : `double[]`
The s_n^x vector.

`whichsol` : `soltype`
Selects a solution.

Description:

Sets the s_n^x vector for a solution.

See also

- `Task.putsnxslice` Sets a slice of the snx vector for a solution.

A.2.228 `Task.putsnxslice()`

```
Task.putsnxslice(
    whichsol,
    first,
    last,
    snx)
```

Sets a slice of the snx vector for a solution.

Arguments

`first` : `int`
First index in the sequence.

`last` : `int`
Last index plus 1 in the sequence.

`snx` : `double[]`
Dual variables corresponding to the conic constraints on the variables.

`whichsol` : `soltype`
Selects a solution.

Description:

Sets a slice of the s_n^x vector for a solution.

See also

- `Task.putsnx` Sets the snx vector for a solution.

A.2.229 `Task.putsolution()`

```
Task.putsolution(
    whichsol,
    skc,
    skx,
    skn,
    xc,
    xx,
    y,
    slc,
    suc,
    slx,
    sux,
    snx)
```

Inserts a solution.

Arguments

`skc` : `stakey`
: Status keys for the constraints.

`skn` : `stakey`
: Status keys for the conic constraints.

`skx` : `stakey`
: Status keys for the variables.

`slc` : `double[]`
: Dual variables corresponding to the lower bounds on the constraints.

`slx` : `double[]`
: Dual variables corresponding to the lower bounds on the variables.

`snx` : `double[]`
: Dual variables corresponding to the conic constraints on the variables.

`suc` : `double[]`
: Dual variables corresponding to the upper bounds on the constraints.

`sux` : `double[]`
: Dual variables corresponding to the upper bounds on the variables.

`whichsol` : `soltype`
: Selects a solution.

`xc` : `double[]`
: Primal constraint solution.

`xx` : `double[]`
: Primal variable solution.

y : double[]
Vector of dual variables corresponding to the constraints.

Description:
Inserts a solution into the task.

A.2.230 Task.putsolutioni()

```
Task.putsolutioni(
    accmode,
    i,
    whichsol,
    sk,
    x,
    sl,
    su,
    sn)
```

Sets the primal and dual solution information for a single constraint or variable.

Arguments

accmode : accmode
If set to accmode.con the solution information for a constraint is modified. Otherwise for a variable.

i : int
Index of the constraint or variable.

sk : stakey
Status key of the constraint or variable.

sl : double
Solution value of the dual variable associated with the lower bound.

sn : double
Solution value of the dual variable associated with the cone constraint.

su : double
Solution value of the dual variable associated with the upper bound.

whichsol : soltype
Selects a solution.

x : double
Solution value of the primal constraint or variable.

Description:
Sets the primal and dual solution information for a single constraint or variable.

A.2.231 Task.putsolutionyi()

```
Task.putsolutionyi(
    i,
    whichsol,
    y)
```

Inputs the dual variable of a solution.

Arguments

`i : int`
: Index of the dual variable.

`whichsol : soltype`
: Selects a solution.

`y : double`
: Solution value of the dual variable.

Description:

Inputs the dual variable of a solution.

See also

- `Task.putsolutioni` Sets the primal and dual solution information for a single constraint or variable.

A.2.232 Task.putstrparam()

```
Task.putstrparam(
    param,
    parvalue)
```

Sets a string parameter.

Arguments

`param : sparam`
: Which parameter.

`parvalue : str`
: Parameter value.

Description:

Sets the value of a string parameter.

A.2.233 Task.putsuc()

```
Task.putsuc(
    whichsol,
    suc)
```

Sets the suc vector for a solution.

Arguments

`suc` : `double[]`
    The s_u^c vector.

`whichsol` : `soltype`
    Selects a solution.

Description:

Sets the s_u^c vector for a solution.

See also

- `Task.putsucslice` Sets a slice of the suc vector for a solution.

A.2.234 Task.putsucslice()

```
Task.putsucslice(
    whichsol,
    first,
    last,
    suc)
```

Sets a slice of the suc vector for a solution.

Arguments

`first` : `int`
    First index in the sequence.

`last` : `int`
    Last index plus 1 in the sequence.

`suc` : `double[]`
    Dual variables corresponding to the upper bounds on the constraints.

`whichsol` : `soltype`
    Selects a solution.

Description:

Sets a slice of the s_u^c vector for a solution.

See also

- Task.putsuc Sets the suc vector for a solution.

A.2.235 Task.putsux()

```
Task.putsux(
    whichsol,
    sux)
```

Sets the sux vector for a solution.

Arguments

`sux` : `double[]`
The s_u^x vector.

`whichsol` : `soltype`
Selects a solution.

Description:

Sets the s_u^x vector for a solution.

See also

- Task.putsuxslice Sets a slice of the sux vector for a solution.

A.2.236 Task.putsuxslice()

```
Task.putsuxslice(
    whichsol,
    first,
    last,
    sux)
```

Sets a slice of the sux vector for a solution.

Arguments

`first` : `int`
First index in the sequence.

`last` : `int`
Last index plus 1 in the sequence.

`sux` : `double[]`
Dual variables corresponding to the upper bounds on the variables.

`whichsol` : `soltype`
Selects a solution.

Description:

Sets a slice of the s_u^x vector for a solution.

See also

- `Task.putsux` Sets the sux vector for a solution.

A.2.237 `Task.puttaskname()`

```
Task.puttaskname(taskname)
```

Assigns a new name to the task.

Arguments

`taskname` : `str`
Name assigned to the task.

Description:

Assigns the name `taskname` to the task.

A.2.238 `Task.putvarbound()`

```
Task.putvarbound(
    j,
    bk,
    bl,
    bu)
```

Changes the bound for one variable.

Arguments

`bk` : `boundkey`
New bound key.

`bl` : `double`
New lower bound.

`bu` : `double`
New upper bound.

`j` : `int`
Index of the variable.

Description:

Changes the bounds for one variable.

If the a bound value specified is numerically larger than `dparam.data_tol_bound_inf` it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than `dparam.data_tol_bound_wrn`, a warning will be displayed, but the bound is inputted as specified.

See also

- `Task.putvarboundslice` Changes the bounds for a slice of the variables.

A.2.239 `Task.putvarboundlist()`

```
Task.putvarboundlist(
    sub,
    bkx,
    blx,
    bux)
```

Changes the bounds of a list of variables.

Arguments

`bkx` : `boundkey`
New bound keys.

`blx` : `double[]`
New lower bound values.

`bux` : `double[]`
New upper bound values.

`sub` : `int[]`
List of variable indexes.

Description:

Changes the bounds for one or more variables. If multiple bound changes are specified for a variable, then only the last change takes effect.

See also

- Task.putvarbound Changes the bound for one variable.
- Task.putvarboundslice Changes the bounds for a slice of the variables.
- dparam.data_tol_bound_inf Data tolerance threshold.
- dparam.data_tol_bound_wrn Data tolerance threshold.

A.2.240 Task.putvarboundslice()

```
Task.putvarboundslice(
    first,
    last,
    bk,
    bl,
    bu)
```

Changes the bounds for a slice of the variables.

Arguments

bk : boundkey
: New bound keys.

bl : double[]
: New lower bounds.

bu : double[]
: New upper bounds.

first : int
: Index of the first variable in the slice.

last : int
: Index of the last variable in the slice plus 1.

Description:

Changes the bounds for a slice of the variables.

See also

- Task.putconbound Changes the bound for one constraint.

A.2.241 `Task.putvarbranchorder()`

```
Task.putvarbranchorder(
    j,
    priority,
    direction)
```

Assigns a branching priority and direction to a variable.

Arguments

`direction : branchdir`
: Specifies the preferred branching direction for variable j.

`j : int`
: Index of the variable.

`priority : int`
: The branching priority that should be assigned to 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.

A.2.242 `Task.putvarname()`

```
Task.putvarname(
    j,
    name)
```

Puts the name of a variable.

Arguments

`j : int`
: Index of the variable.

`name : str`
: The variable name.

Description:

Puts the name of a variable.

A.2.243 Task.putvartype()

```
Task.putvartype(
    j,
    vartype)
```

Sets the variable type of one variable.

Arguments

`j` : `int`
    Index of the variable.

`vartype` : `variabletype`
    The new variable type.

Description:

Sets the variable type of one variable.

See also

- `Task.putvartypelist` Sets the variable type for one or more variables.

A.2.244 Task.putvartypelist()

```
Task.putvartypelist(
    subj,
    vartype)
```

Sets the variable type for one or more variables.

Arguments

`subj` : `int[]`
    A list of variable indexes for which the variable type should be changed.

`vartype` : `variabletype`
    A list of variable types that should be assigned to the variables specified by `subj`. See section `variabletype` for the possible values of `vartype`.

Description:

Sets the variable type for one or more variables, i.e. variable number `subj`[k] is assigned the variable type `vartype`[k].

If the same index is specified multiple times in `subj` only the last entry takes effect.

See also

- `Task.putvartype` Sets the variable type of one variable.

A.2.245 Task.putxc()

```
Task.putxc(
    whichsol,
    xc)
```

Sets the xc vector for a solution.

Arguments

whichsol : soltype
Selects a solution.

xc : double[]
The x^c vector.

Description:

Sets the x^c vector for a solution.

See also

- Task.putxcslice Sets a slice of the xc vector for a solution.

A.2.246 Task.putxcslice()

```
Task.putxcslice(
    whichsol,
    first,
    last,
    xc)
```

Sets a slice of the xc vector for a solution.

Arguments

first : int
First index in the sequence.

last : int
Last index plus 1 in the sequence.

whichsol : soltype
Selects a solution.

xc : double[]
Primal constraint solution.

Description:

Sets a slice of the x^c vector for a solution.

See also

- Task.putxc Sets the xc vector for a solution.

A.2.247 Task.putxx()

```
Task.putxx(
    whichsol,
    xx)
```

Sets the xx vector for a solution.

Arguments

whichsol : soltype
: Selects a solution.

xx : double[]
: The x^x vector.

Description:

Sets the x^x vector for a solution.

See also

- Task.putxxslice Obtains a slice of the xx vector for a solution.

A.2.248 Task.putxxslice()

```
Task.putxxslice(
    whichsol,
    first,
    last,
    xx)
```

Obtains a slice of the xx vector for a solution.

Arguments

first : int
: First index in the sequence.

last : int
: Last index plus 1 in the sequence.

whichsol : soltype
: Selects a solution.

xx : double[]
: Primal variable solution.

Description:

Obtains a slice of the x^x vector for a solution.

See also

- Task.putxx Sets the xx vector for a solution.

A.2.249 Task.puty()

```
Task.puty(
    whichsol,
    y)
```

Sets the y vector for a solution.

Arguments

whichsol : soltype
: Selects a solution.

y : double[]
: The y vector.

Description:

Sets the y vector for a solution.

See also

- Task.putyslice Sets a slice of the y vector for a solution.

A.2.250 Task.putyslice()

```
Task.putyslice(
    whichsol,
    first,
    last,
    y)
```

Sets a slice of the y vector for a solution.

Arguments

`first : int`
First index in the sequence.

`last : int`
Last index plus 1 in the sequence.

`whichsol : soltype`
Selects a solution.

`y : double[]`
Vector of dual variables corresponding to the constraints.

Description:

Sets a slice of the y vector for a solution.

See also

- `Task.puty` Sets the y vector for a solution.

A.2.251 `Task.readbranchpriorities()`

```
Task.readbranchpriorities(filename)
```

Reads branching priority data from a file.

Arguments

`filename : str`
Data is read from the file `filename`.

Description:

Reads branching priority data from a file.

See also

- `Task.writebranchpriorities` Writes branching priority data to a file.

A.2.252 Task.readdata()

```
Task.readdata(filename)
```

Reads problem data from a file.

Arguments

`filename : str`
Data is read from the file `filename`.

Description:

Reads an optimization problem and associated data from a file.

See also

- `iparam.read_data_format` Format of the data file to be read.

A.2.253 Task.readdataformat()

```
Task.readdataformat(
    filename,
    format,
    compress)
```

Reads problem data from a file.

Arguments

`compress : compresstype`
File compression type.

`filename : str`
Data is read from the file `filename`.

`format : dataformat`
File data format.

Description:

Reads an optimization problem and associated data from a file.

See also

- `iparam.read_data_format` Format of the data file to be read.

A.2.254 `Task.readparamfile()`

```
Task.readparamfile()
```

Reads a parameter file.

Description:

Reads a parameter file.

A.2.255 `Task.readsolution()`

```
Task.readsolution(
    whichsol,
    filename)
```

Reads a solution from a file.

Arguments

`filename : str`
A valid file name.

`whichsol : soltype`
Selects a solution.

Description:

Reads a solution file and inserts the solution into the solution `whichsol`.

A.2.256 `Task.readsummary()`

```
Task.readsummary(whichstream)
```

Prints information about last file read.

Arguments

`whichstream : streamtype`
Index of the stream.

Description:

Prints a short summary of last file that was read.

A.2.257 `Task.readtask()`

```
Task.readtask(filename)
```

Load task data from a file.

Arguments

`filename : str`
Input file name.

Description:

Load task data from a file, replacing any data that already is in the task object. All problem data are resotred, but if the file contains solutions, the solution status after loading a file is still unknown, even if it was optimal or otherwise well-defined when the file was dumped.

See section F.4 for a description of the Task format.

A.2.258 `Task.relaxprimal()`

```
relaxedtask = Task.relaxprimal(
    wlc,
    wuc,
    wlx,
    wux)
```

Deprecated.

Arguments

`relaxedtask : Task`
The returned task.

`wlc : double[]`
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.

`wlx : double[]`
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.

`wuc : double[]`
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.

`wux : double[]`

Weights associated with upper bounds on the activity of variables. If negative, the bound is strictly enforced, i.e. if $(w_u^x)_j < 0$ then $(v_u^x)_j$ is fixed to zero. On return `wux[i]` contains the relaxed bound.

Description:

Deprecated. Please use `Task.primalrepair` instead.

See also

- `dparam.feasrepair_tol` Tolerance for constraint enforcing upper bound on sum of weighted violations in feasibility repair.
- `iparam.feasrepair_optimize` Controls which type of feasibility analysis is to be performed.
- `sparam.feasrepair_name_separator` Feasibility repair name separator.
- `sparam.feasrepair_name_prefix` Feasibility repair name prefix.

A.2.259 `Task.removebarvars()`

```
Task.removebarvars(subset)
```

The function removes a number of symmetric matrix.

Arguments

`subset : int[]`

Indexes of symmetric matrix which should be removed.

Description:

The function removes a subset of the symmetric matrix from the optimization task. This implies that the existing symmetric matrix are renumbered, for instance if constraint 5 is removed then constraint 6 becomes constraint 5 and so forth.

See also

- `Task.appendbarvars` Appends a semidefinite variable of dimension dim to the problem.

A.2.260 `Task.removecones()`

```
Task.removecones(subset)
```

Removes a conic constraint from the problem.

Arguments

`subset : int[]`
Indexes of cones which should be removed.

Description:

Removes a number conic constraint from the problem. In general, it is much more efficient to remove a cone with a high index than a low index.

A.2.261 Task.removecons()

```
Task.removecons(subset)
```

The function removes a number of constraints.

Arguments

`subset : int[]`
Indexes of constraints which should be removed.

Description:

The function removes a subset of the constraints from the optimization task. This implies that the existing constraints are renumbered, for instance if constraint 5 is removed then constraint 6 becomes constraint 5 and so forth.

See also

- `Task.appendcons` Appends a number of constraints to the optimization task.

A.2.262 Task.removevars()

```
Task.removevars(subset)
```

The function removes a number of variables.

Arguments

`subset : int[]`
Indexes of variables which should be removed.

Description:

The function removes a subset of the variables from the optimization task. This implies that the existing variables are renumbered, for instance if constraint 5 is removed then constraint 6 becomes constraint 5 and so forth.

See also

- Task.appendvars Appends a number of variables to the optimization task.

A.2.263 Task.resizetask()

```
Task.resizetask(
    maxnumcon,
    maxnumvar,
    maxnumcone,
    maxnumanz,
    maxnumqnz)
```

Resizes an optimization task.

Arguments

maxnumanz : long
New maximum number of non-zeros in A.

maxnumcon : int
New maximum number of constraints.

maxnumcone : int
New maximum number of cones.

maxnumqnz : long
New maximum number of non-zeros in all Q matrixes.

maxnumvar : int
New maximum number of variables.

Description:

Sets the amount of preallocated space assigned for each type of data in an optimization task.

It is never mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

Please note that the procedure is **destructive** in the sense that all existing data stored in the task is destroyed.

See also

- Task.putmaxnumvar Sets the number of preallocated variables in the optimization task.
- Task.putmaxnumcon Sets the number of preallocated constraints in the optimization task.
- Task.putmaxnumcone Sets the number of preallocated conic constraints in the optimization task.
- Task.putmaxnumanz The function changes the size of the preallocated storage for linear coefficients.
- Task.putmaxnumqnz Changes the size of the preallocated storage for quadratic terms.

A.2.264 Task.sensitivityreport()

```
Task.sensitivityreport(whichstream)
```

Creates a sensitivity report.

Arguments

whichstream : streamtype
Index of the stream.

Description:

Reads a sensitivity format file from a location given by sparam.sensitivity_file_name and writes the result to the stream whichstream. If sparam.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

- Task.dualsensitivity Performs sensitivity analysis on objective coefficients.
- Task.primalsensitivity Perform sensitivity analysis on bounds.
- iparam.log_sensitivity Control logging in sensitivity analyzer.
- iparam.log_sensitivity_opt Control logging in sensitivity analyzer.
- iparam.sensitivity_type Controls which type of sensitivity analysis is to be performed.

A.2.265 Task.setdefaults()

```
Task.setdefaults()
```

Resets all parameters values.

Description:

Resets all the parameters to their default values.

A.2.266 Task.solutiondef()

```
isdef = Task.solutiondef(whichsol)
```

Checks whether a solution is defined.

Arguments

`isdef` : `int`
Is non-zero if the requested solution is defined.

`whichsol` : `soltype`
Selects a solution.

Description:

Checks whether a solution is defined.

A.2.267 `Task.solutionsummary()`

```
Task.solutionsummary(whichstream)
```

Prints a short summary of the current solutions.

Arguments

`whichstream` : `streamtype`
Index of the stream.

Description:

Prints a short summary of the current solutions.

A.2.268 `Task.solvewithbasis()`

```
numnz = Task.solvewithbasis(
    transp,
    numnz,
    sub,
    val)
```

Solve a linear equation system involving a basis matrix.

Arguments

`numnz` : `int`
As input it is the number of non-zeros in b. As output it is the number of non-zeros in $\bar{X}$.

`sub` : `int[]`
As input it contains the positions of the non-zeros in b, i.e.

$$b[\mathtt{sub}[k]] \neq 0, \ k = 0, \ldots, numnz[0] - 1.$$

As output it contains the positions of the non-zeros in $\bar{X}$. It is important that `sub` has room for `numcon` elements.

`transp : int`

If this argument is non-zero, then (A.3) is solved. Otherwise the system (A.2) is solved.

`val : double[]`

As input it is the vector b. Although the positions of the non-zero elements are specified in `sub` it is required that `val`$[i] = 0$ if $b[i] = 0$. As output `val` is the vector $\bar{X}$.

Please note that `val` is a dense vector — not a packed sparse vector. This implies that `val` has room for `numcon` elements.

Description:

If a basic solution is available, then exactly `numcon` basis variables are defined. These `numcon` basis variables are denoted the basis. Associated with the basis is a basis matrix denoted B. This function solves either the linear equation system

$$B\bar{X} = b \tag{A.2}$$

or the system

$$B^T\bar{X} = b \tag{A.3}$$

for the unknowns $\bar{X}$, with b being a user-defined vector.

In order to make sense of the solution $\bar{X}$ it is important to know the ordering of the variables in the basis because the ordering specifies how B is constructed. When calling `Task.initbasissolve` an ordering of the basis variables is obtained, whicd can be used to deduce how MOSEK has constructed B. Indeed if the kth basis variable is variable x_j it implies that

$$B_{i,k} = A_{i,j}, \ i = 0, \ldots, numcon - 1.$$

Otherwise if the kth basis variable is variable x_j^c it implies that'

$$B_{i,k} = \left\{ \begin{array}{ll} -1, & i = j, \\ 0, & i \neq j. \end{array} \right.$$

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

- `Task.initbasissolve` Prepare a task for basis solver.
- `iparam.basis_solve_use_plus_one` Controls the sign of the columns in the basis matrix corresponding to slack variables.

A.2.269 `Task.startstat()`

```
Task.startstat()
```

Starts the statistics file.

Description:

Starts the statistics file.

A.2.270 `Task.stopstat()`

```
Task.stopstat()
```

Stops the statistics file.

Description:

Stops the statistics file.

A.2.271 `Task.strtoconetype()`

```
conetype = Task.strtoconetype(str)
```

Obtains a cone type code.

Arguments

`conetype` : `conetype`
: The cone type corresponding to the string `str`.

`str` : `str`
: String corresponding to the cone type code `codetype`.

Description:

Obtains cone type code corresponding to a cone type string.

A.2.272 Task.strtosk()

```
sk = Task.strtosk(str)
```

Obtains a status key.

Arguments

`sk : int`
Status key corresponding to the string.

`str : str`
Status key string.

Description:

Obtains the status key corresponding to an explanatory string.

A.2.273 Task.toconic()

```
Task.toconic()
```

Inplace reformulation of a QCQP to a COP

Description:

This function tries to reformulate a given Quadratically Constrained Quadratic Optimization problem (QCQP) as a Conic Quadratic Optimization problem (CQO). The first step of the reformulation is to convert the quadratic term of the objective function as a constraint, if any. Then the following steps are repeated for each quadratic constraint:

- a conic constraint is added along with a suitable number of auxiliary variables and constraints;
- the original quadratic constraint is not removed, but all its coefficients are zeroed out.

Note that the reformulation preserves all the original variables.

The conversion is performed in-place, i.e. the task passed as argument is modified on exit. That also means that if the reformulation fails, i.e. the given QCQP is not representable as a CQO, then the task has an undefined state. In some cases, users may want to clone the task to ensure a clean copy is preserved.

A.2.274 Task.updatesolutioninfo()

```
Task.updatesolutioninfo(whichsol)
```

Update the information items related to the solution.

Arguments

whichsol : soltype
Selects a solution.

Description:

Update the information items related to the solution.

A.2.275 Task.writebranchpriorities()

```
Task.writebranchpriorities(filename)
```

Writes branching priority data to a file.

Arguments

filename : str
Data is written to the file `filename`.

Description:

Writes branching priority data to a file.

See also

- `Task.readbranchpriorities` Reads branching priority data from a file.

A.2.276 Task.writedata()

```
Task.writedata(filename)
```

Writes problem data to a file.

Arguments

`filename : str`
Data is written to the file `filename` if it is a nonempty string. Otherwise data is written to the file specified by `sparam.data_file_name`.

Description:

Writes problem data associated with the optimization task to a file in one of four formats:

LP:
A text based row oriented format. File extension `.lp`. See Appendix F.2.

MPS:
A text based column oriented format. File extension `.mps`. See Appendix F.1.

OPF:
A text based row oriented format. File extension `.opf`. Supports more problem types than MPS and LP. See Appendix F.3.

TASK:
A MOSEK specific binary format for fast reading and writing. File extension `.task`.

By default the data file format is determined by the file name extension. This behaviour can be overridden by setting the `iparam.write_data_format` parameter.

MOSEK is able to read and write files in a compressed format (gzip). To write in the compressed format append the extension ".gz". E.g to write a gzip compressed MPS file use the extension `mps.gz`.

Please note that MPS, LP and OPF files require all variables to have unique names. If a task contains no names, it is possible to write the file with automaticly generated anonymous names by setting the `iparam.write_generic_names` parameter to `onoffkey.on`.

Please note that if a general nonlinear function appears in the problem then such function *cannot* be written to file and MOSEK will issue a warning.

See also

- `iparam.write_data_format` Controls the output file format.

A.2.277 `Task.writeparamfile()`

`Task.writeparamfile(filename)`

Writes all the parameters to a parameter file.

Arguments

`filename : str`
The name of parameter file.

Description:

Writes all the parameters to a parameter file.

A.2.278 `Task.writesolution()`

```
Task.writesolution(
    whichsol,
    filename)
```

Write a solution to a file.

Arguments

`filename : str`
A valid file name.

`whichsol : soltype`
Selects a solution.

Description:

Saves the current basic, interior-point, or integer solution to a file.

See also

- `iparam.write_sol_ignore_invalid_names` Controls whther the user specified names are employed even if they are invalid names.
- `iparam.write_sol_head` Controls solution file format.
- `iparam.write_sol_constraints` Controls the solution file format.
- `iparam.write_sol_variables` Controls the solution file format.
- `iparam.write_sol_barvariables` Controls the solution file format.
- `iparam.write_bas_head` Controls the basic solution file format.
- `iparam.write_bas_constraints` Controls the basic solution file format.
- `iparam.write_bas_variables` Controls the basic solution file format.

A.2.279 `Task.writetask()`

```
Task.writetask(filename)
```

Write a complete binary dump of the task data.

Arguments

`filename : str`
Output file name.

Description:

Write a binary dump of the task data. This format saves all problem data, but not callback-funktions and general non-linear terms.

See section F.4 for a description of the Task format.

A.3 Class Env

A.3.1 `Env.axpy()`

```
Env.axpy(
    n,
    alpha,
    x,
    y)
```

Adds alpha times x to y.

Arguments

`alpha : double`
: The scalar that multiplies x.

`n : int`
: Length of the vectors.

`x : double[]`
: The vector.

`y : double[]`
: The vector.

Description:

Adds αx to y.

A.3.2 `Env.checkinlicense()`

```
Env.checkinlicense(feature)
```

Check in a license feature from the license server ahead of time.

Arguments

`feature : feature`
: Feature to check in to the license system.

Description:

Check in a license feature to the license server. By default all licenses consumed by functions using a single environment is kept checked out for the lifetime of the MOSEK environment. This function checks in a given license feature to the license server immidiatly.

If the given license feature is not checked out or is in use by a call to `Task.optimize` calling this function has no effect.

Please note that returning a license to the license server incurs a small overhead, so frequent calls to this function should be avoided.

A.3.3 `Env.checkoutlicense()`

```
Env.checkoutlicense(feature)
```

Check out a license feature from the license server ahead of time.

Arguments

`feature` : `feature`
Feature to check out from the license system.

Description:

Check out a license feature from the license server. Normally the required license features will be automatically checked out the first time it is needed by the function `Task.optimize`. This function can be used to check out one or more features ahead of time.

The license will remain checked out for the lifetime of the MOSEK environment or until the function `Env.checkinlicense` is called.

If a given feature is already checked out when this function is called, only one feature will be checked out from the license server.

A.3.4 `Env.dot()`

```
xty = Env.dot(
    n,
    x,
    y)
```

Computes the inner product of two vectors.

Arguments

`n` : `int`
Length of the vectors.

`x : double[]`
The x vector.

`xty : double`
The result of the inner product between x and y.

`y : double[]`
The y vector.

Description:

Computes the inner product of two vectors x, y of lenght $n \geq 0$, i.e

$$x \cdot y = \sum_{i=1} x_i y_i.$$

Note that if $n = 0$, then the results of the operation is 0.

A.3.5 `Env.echointro()`

```
Env.echointro(longver)
```

Prints an intro to message stream.

Arguments

`longver : int`
If non-zero, then the intro is slightly longer.

Description:

Prints an intro to message stream.

A.3.6 `Env.gemm()`

```
Env.gemm(
    transa,
    transb,
    m,
    n,
    k,
    alpha,
    a,
    b,
    beta,
    c)
```

Performs a dense matrix multiplication.

Arguments

`a : double[]`
: The pointer to the array storing matrix A in a column-major format.

`alpha : double`
: A scalar value multipling the result of the matrix multiplication.

`b : double[]`
: Indicates the number of rows of matrix B and columns of matrix A.

`beta : double`
: A scalar value that multiplies C.

`c : double[]`
: The pointer to the array storing matrix C in a column-major format.

`k : int`
: Specifies the number of columns of the matrix A and the number of rows of the matrix B.

`m : int`
: Indicates the number of rows of matrices A and C.

`n : int`
: Indicates the number of columns of matrices B and C.

`transa : transpose`
: Indicates whether the matrix A must be transposed.

`transb : transpose`
: Indicates whether the matrix B must be transposed.

Description:

Performs a matrix multiplication plus addition of dense matrices. Given A, B and C of compatible dimensions, this function computes

$$C := \alpha op(A) op(B) + \beta C$$

where α, β are two scalar values. The function $op(X)$ return X if transX is YES, or X^T if set to NO. Dimensions of A, b must therefore match those of C.

The result of this operation is stored in C.

A.3.7 `Env.gemv()`

```
Env.gemv(
    transa,
    m,
    n,
    alpha,
    a,
    x,
    beta,
    y)
```

Computes dense matrix times a dense vector product.

Arguments

`a : double[]`
A pointer to the array storing matrix A in a column-major format.

`alpha : double`
A scalar value multipling the matrix A.

`beta : double`
A scalar value multipling the vector y.

`m : int`
Specifies the number of rows of the matrix A.

`n : int`
Specifies the number of columns of the matrix A.

`transa : transpose`
Indicates whether the matrix A must be transposed.

`x : double[]`
A pointer to the array storing the vector x.

`y : double[]`
A pointer to the array storing the vector y.

Description:

Computes the multiplication of a scaled dense matrix times a dense vector product, plus a scaled dense vector. In formula

$$y = \alpha A x + \beta y,$$

or if trans is set to transpose.yes

$$y = \alpha A^T x + \beta y,$$

where α, β are scalar values. A is an $n \times m$ matrix, $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$.

Note that the result is stored overwriting y.

A.3.8 `Env.getcodedesc()`

```
symname,str = Env.getcodedesc(code)
```

Obtains a short description of a response code.

Arguments

`code` : `rescode`
: A valid MOSEK response code.

`str` : `str`
: Obtains a short description of a response code.

`symname` : `str`
: Symbolic name corresponding to `code`.

Description:

Obtains a short description of the meaning of the response code given by `code`.

A.3.9 `Env.getversion()`

```
major,minor,build,revision = Env.getversion()
```

Obtains MOSEK version information.

Arguments

`build` : `int`
: Build number.

`major` : `int`
: Major version number.

`minor` : `int`
: Minor version number.

`revision` : `int`
: Revision number.

Description:

Obtains MOSEK version information.

A.3.10 `Env.licensecleanup()`

```
Env.licensecleanup()
```

Stops all threads and delete all handles used by the license system.

Description:

Stops all threads and delete all handles used by the license system. If this function is called, it must be called as the last MOSEK API call. No other MOSEK API calls are valid after this.

A.3.11 Env.linkfiletostream()

```
Env.linkfiletostream(
    whichstream,
    filename,
    append)
```

Directs all output from a stream to a file.

Arguments

`append : int`
: If this argument is non-zero, the output is appended to the file.

`filename : str`
: Sends all output from the stream defined by `whichstream` to the file given by `filename`.

`whichstream : streamtype`
: Index of the stream.

Description:

Directs all output from a stream to a file.

A.3.12 Env.potrf()

```
Env.potrf(
    uplo,
    n,
    a)
```

Computes a Cholesky factorization a dense matrix.

Arguments

`a : double[]`
: A symmetric matrix stored in column-major order. Only the lower or the upper triangular part is used, accordingly with the uplo parameter. It will contain the result on exit.

`n : int`
: Dimension of the symmetric matrix.

`uplo : uplo`
: Indicates whether the upper or lower triangular part of the matrix is stored.

Description:

Computes a Cholesky factorization of a real symmetric positive definite dense matrix.

A.3.13 `Env.putdllpath()`

```
Env.putdllpath(dllpath)
```

Sets the path to the DLL/shared libraries that MOSEK is loading.

Arguments

`dllpath : str`
A path to where the MOSEK dynamic link/shared libraries are located. If `dllpath` is `NULL`, then MOSEK assumes that the operating system can locate the libraries.

Description:

Sets the path to the DLL/shared libraries that MOSEK are loading.

A.3.14 `Env.putkeepdlls()`

```
Env.putkeepdlls(keepdlls)
```

Controls whether explicitly loaded DLLs should be kept.

Arguments

`keepdlls : int`
Controls whether explicitly loaded DLLs should be kept.

Description:

Controls whether explicitly loaded DLLs should be kept when they no longer are in use.

A.3.15 `Env.putlicensecode()`

```
Env.putlicensecode(code)
```

The purpose of this function is to input a runtime license code.

Arguments

`code : int[]`
A runtime license code.

Description:

The purpose of this function is to input a runtime license code.

A.3.16 Env.putlicensedebug()

```
Env.putlicensedebug(licdebug)
```

Enables debug information for the license system.

Arguments

`licdebug : int`
: If this argument is non-zero, then MOSEK will print debug info regarding the license checkout.

Description:

If `licdebug` is non-zero, then MOSEK will print debug info regarding the license checkout.

A.3.17 Env.putlicensepath()

```
Env.putlicensepath(licensepath)
```

Set the path to the license file.

Arguments

`licensepath : str`
: A path specifycing where to search for the license.

Description:

Set the path to the license file.

A.3.18 Env.putlicensewait()

```
Env.putlicensewait(licwait)
```

Control whether mosek should wait for an available license if no license is available.

Arguments

`licwait : int`
: If this argument is non-zero, then MOSEK will wait for a license if no license is available. Moreover, `licwait-1` is the number of milliseconds to wait between each check for an available license.

Description:

If `licwait` is non-zero, then MOSEK will wait for a license if no license is available. Moreover, `licwait-1` is the number of milliseconds to wait between each check for an available license.

A.3.19 Env.syeig()

```
Env.syeig(
    uplo,
    n,
    a,
    w)
```

Computes all eigenvalues of a symmetric dense matrix.

Arguments

`a : double[]`
A symmetric matrix stored in column-major order. Only the lower-triangular part is used.

`n : int`
Dimension of the symmetric input matrix.

`uplo : uplo`
Indicates whether the upper or lower triangular part is used.

`w : double[]`
Array of minimum dimension n where eigenvalues will be stored.

Description:

Computes all eigenvalues of a real symmetric matrix A. Eigenvalues are stored in the w array.

A.3.20 Env.syevd()

```
Env.syevd(
    uplo,
    n,
    a,
    w)
```

Computes all the eigenvalue and eigenvectors of a symmetric dense matrix, and thus its eigenvalue decomposition.

Arguments

`a : double[]`

A symmetric matrix stored in column-major order. Only the lower-triangular part is used. It will be overwritten on exit.

`n : int`

Dimension of symmetric input matrix.

`uplo : uplo`

Indicates whether the upper or lower triangular part is used.

`w : double[]`

An array where eigenvalues will be stored. Its lenght must be at least the dimension of the input matrix.

Description:

Computes all the eigenvalues and eigenvectors a real symmetric matrix.

Given the input matrix $A \in \mathbb{R}^{n\times n}$, this function returns a vector $w \in \mathbb{R}^n$ containing the eigenvalues of A and the corresponding eigenvectors, stored in A as well.

Therefore, this function compute the eigenvalue decomposition of A as

$$A = UVU^T,$$

where $V = diag(w)$ and U contains the eigen-vectors of A.

A.3.21 `Env.syrk()`

```
Env.syrk(
    uplo,
    trans,
    n,
    k,
    alpha,
    a,
    beta,
    c)
```

Performs a rank-k update of a symmetric matrix.

Arguments

`a : double[]`

The pointer to the array storing matrix A in a column-major format.

`alpha : double`

A scalar value multipling the result of the matrix multiplication.

`beta : double`

A scalar value that multiplies C.

c : double[]
: The pointer to the array storing matrix C in a column-major format.

k : int
: Indicates the number of rows or columns of A, and its rank.

n : int
: Specifies the order of C.

trans : transpose
: Indicates whether the matrix A must be transposed.

uplo : uplo
: Indicates whether the upper or lower triangular part of C is stored.

Description:

Performs a symmetric rank-k update for a symmetric matrix.

Given a symmetric matrix $C \in \mathbb{R}^{n\times n}$, two scalars α, β and a matrix A of rank $k \leq n$, it computes either

$$C = \alpha AA^T + \beta C,$$

or

$$C = \alpha A^T A + \beta C.$$

In the first case $A \in \mathbb{R}^{k\times n}$, in the second $A \in \mathbb{R}^{n\times k}$.

Note that the results overwrite the matrix C.

A.4 Callback functions and related methods

Callbacks from task methods are performed by attaching a function to the task object.

A.4.1 Progress callback

When MOSEK is optimizing or doing other tasks that may potentially take a long time, it is possible to recieve periodical calls from MOSEK indicating the current status. This achieved by attaching a callback handler to the task. A callback handler is simply a function that accepts one integer argument and returns an integer.

The argument is an integer indicating the current location in MOSEK. If the function returns a non-zero value or raises an exception, MOSEK will attempt to stop the current task and return as fast as possible.

```
def myCallback(caller, dinf, iinf, liinf):
    print "Caller : %d, intpnt iteration = %d" % (caller, iinf[iinfitem.intpnt_iter])
```

```
    return 0
task.set_Progress(myCallback)
```

The arguments dinf, iinf and liinf are arrays of values containing information items for the current state of the solver.

The Progress callback can be detached by calling

```
task.set_Progress(None)
```

NOTE: Due to the way the Python garbage collector works, it is necessary to hold a reference to the callback function for the duration of the lifetime of the Task object. This means that a construction as

```
#NOTE: WRONG way to attach callback!
task.set_Progress(lambda caller, dinf, iinf, liinf: print "Caller : %d" % caller)
```

will *not work*.

A.4.2 Stream callback

Text written to specific MOSEK streams can be intercepted by attaching a stream handler to the specific stream. A stream handler is simply a function that accepts one string argument.

To attach a stream callback to a Task, use:

```
def myStream(msg):
    sys.stdout.write(msg)
    sys.stdout.flush()
task.set_Stream(streamtype.log,myStream)
```

The Stream object can be detached by calling

```
task.set_Stream(None)
```

In this example we attached to the log stream; see streamtype for other options.

NOTE: As for Progress callbacks, it is necessary to hold a reference to the callback function for the duration of the lifetime of the Task or ttEnv object.

A.5 Dispose and garbage collection

The garbage collecter in Python will automatically dispose and reuse both Task and Env objects, but since these objects may allocate substantial amounts of memory or reserve MOSEK licenses outside

Python, in some cases it is desireable to explicitly free up this memory when the task or environment will not be used anymore.

The native memory associated with `Task` and `Env` can be free'd by calling the `__del__()` method:

```
task.__del__()
env.__del__()
```

Notice that even if an `Env` is disposed before all `Tasks` that belongs to it were disposed, its memory will not be free'd before that last task is disposed.

A.6 All functions by name

`Task.analyzenames`

Analyze the names and issue an error for the first invalid name.

`Task.analyzeproblem`

Analyze the data of a task.

`Task.analyzesolution`

Print information related to the quality of the solution.

`Task.appendbarvars`

Appends a semidefinite variable of dimension dim to the problem.

`Task.appendcone`

Appends a new cone constraint to the problem.

`Task.appendconeseq`

Appends a new conic constraint to the problem.

`Task.appendconesseq`

Appends multiple conic constraints to the problem.

`Task.appendcons`

Appends a number of constraints to the optimization task.

`Task.appendsparsesymmat`

Appends a general sparse symmetric matrix to the vector E of symmetric matrixes.

`Task.appendstat`

Appends a record the statistics file.

`Task.appendvars`

Appends a number of variables to the optimization task.

`Env.axpy`

Adds alpha times x to y.

`Task.basiscond`

Computes conditioning information for the basis matrix.

`Task.checkconvexity`

Checks if a quadratic optimization problem is convex.

`Env.checkinlicense`

Check in a license feature from the license server ahead of time.

`Task.checkmem`

Checks the memory allocated by the task.

`Env.checkoutlicense`

Check out a license feature from the license server ahead of time.

`Task.chgbound`

Changes the bounds for one constraint or variable.

`Task.commitchanges`

Commits all cached problem changes.

`Task.deletesolution`

Undefines a solution and frees the memory it uses.

`Env.dot`

Computes the inner product of two vectors.

`Task.dualsensitivity`

Performs sensitivity analysis on objective coefficients.

`Env.echointro`

Prints an intro to message stream.

`Env.gemm`

Performs a dense matrix multiplication.

`Env.gemv`

Computes dense matrix times a dense vector product.

`Task.getacol`

Obtains one column of the linear constraint matrix.

`Task.getacolnumnz`

Obtains the number of non-zero elements in one column of the linear constraint matrix

`Task.getacolslicetrip`

Obtains a sequence of columns from the coefficient matrix in triplet format.

`Task.getaij`

Obtains a single coefficient in linear constraint matrix.

`Task.getarow`

Obtains one row of the linear constraint matrix.

`Task.getarownumnz`

Obtains the number of non-zero elements in one row of the linear constraint matrix

`Task.getarowslicetrip`

Obtains a sequence of rows from the coefficient matrix in triplet format.

`Task.getaslice`

Obtains a sequence of rows or columns from the coefficient matrix.

`Task.getbarablocktriplet`

Obtains barA in block triplet form.

`Task.getbaraidx`

Obtains information about an element barA.

`Task.getbaraidxij`

Obtains information about an element barA.

`Task.getbaraidxinfo`

Obtains the number terms in the weighted sum that forms a particular element in barA.

`Task.getbarasparsity`

Obtains the sparsity pattern of the barA matrix.

`Task.getbarcblocktriplet`

Obtains barc in block triplet form.

`Task.getbarcidx`

Obtains information about an element in barc.

`Task.getbarcidxinfo`

Obtains information about an element in barc.

`Task.getbarcidxj`

Obtains the row index of an element in barc.

`Task.getbarcsparsity`

Get the positions of the nonzero elements in barc.

Task.getbarsj

Obtains the dual solution for a semidefinite variable.

Task.getbarvarname

Obtains a name of a semidefinite variable.

Task.getbarvarnameindex

Obtains the index of name of semidefinite variable.

Task.getbarvarnamelen

Obtains the length of a name of a semidefinite variable.

Task.getbarxj

Obtains the primal solution for a semidefinite variable.

Task.getbound

Obtains bound information for one constraint or variable.

Task.getboundslice

Obtains bounds information for a sequence of variables or constraints.

Task.getc

Obtains all objective coefficients.

Task.getcfix

Obtains the fixed term in the objective.

Task.getcj

Obtains one coefficient of c.

Env.getcodedesc

Obtains a short description of a response code.

Task.getconbound

Obtains bound information for one constraint.

Task.getconboundslice

Obtains bounds information for a slice of the constraints.

Task.getcone

Obtains a conic constraint.

Task.getconeinfo

Obtains information about a conic constraint.

Task.getconename

Obtains a name of a cone.

`Task.getconenameindex`

Checks whether the name somename has been assigned to any cone.

`Task.getconenamelen`

Obtains the length of a name of a cone.

`Task.getconname`

Obtains a name of a constraint.

`Task.getconnameindex`

Checks whether the name somename has been assigned to any constraint.

`Task.getconnamelen`

Obtains the length of a name of a constraint variable.

`Task.getcslice`

Obtains a sequence of coefficients from the objective.

`Task.getdbi`

Deprecated.

`Task.getdcni`

Deprecated.

`Task.getdeqi`

Deprecated.

`Task.getdimbarvarj`

Obtains the dimension of a symmetric matrix variable.

`Task.getdouinf`

Obtains a double information item.

`Task.getdouparam`

Obtains a double parameter.

`Task.getdualobj`

Computes the dual objective value associated with the solution.

`Task.getdviolbarvar`

Computes the violation of dual solution for a set of barx variables.

`Task.getdviolcon`

Computes the violation of a dual solution associated with a set of constraints.

`Task.getdviolcones`

Computes the violation of a solution for set of dual conic constraints.

`Task.getdviolvar`

Computes the violation of a dual solution associated with a set of x variables.

`Task.getinfeasiblesubproblem`

Obtains an infeasible sub problem.

`Task.getinti`

Deprecated.

`Task.getintinf`

Obtains an integer information item.

`Task.getintparam`

Obtains an integer parameter.

`Task.getlenbarvarj`

Obtains the length if the j'th semidefinite variables.

`Task.getlintinf`

Obtains an integer information item.

`Task.getmaxnumanz`

Obtains number of preallocated non-zeros in the linear constraint matrix.

`Task.getmaxnumbarvar`

Obtains the number of semidefinite variables.

`Task.getmaxnumcon`

Obtains the number of preallocated constraints in the optimization task.

`Task.getmaxnumcone`

Obtains the number of preallocated cones in the optimization task.

`Task.getmaxnumqnz`

Obtains the number of preallocated non-zeros for all quadratic terms in objective and constraints.

`Task.getmaxnumvar`

Obtains the maximum number variables allowed.

`Task.getmemusage`

Obtains information about the amount of memory used by a task.

`Task.getnumanz`

Obtains the number of non-zeros in the coefficient matrix.

`Task.getnumanz64`

Obtains the number of non-zeros in the coefficient matrix.

`Task.getnumbarablocktriplets`

Obtains an upper bound on the number of scalar elements in the block triplet form of bara.

`Task.getnumbaranz`

Get the number of nonzero elements in barA.

`Task.getnumbarcblocktriplets`

Obtains an upper bound on the number of elements in the block triplet form of barc.

`Task.getnumbarcnz`

Obtains the number of nonzero elements in barc.

`Task.getnumbarvar`

Obtains the number of semidefinite variables.

`Task.getnumcon`

Obtains the number of constraints.

`Task.getnumcone`

Obtains the number of cones.

`Task.getnumconemem`

Obtains the number of members in a cone.

`Task.getnumintvar`

Obtains the number of integer-constrained variables.

`Task.getnumparam`

Obtains the number of parameters of a given type.

`Task.getnumqconknz`

Obtains the number of non-zero quadratic terms in a constraint.

`Task.getnumqconknz64`

Obtains the number of non-zero quadratic terms in a constraint.

`Task.getnumqobjnz`

Obtains the number of non-zero quadratic terms in the objective.

`Task.getnumsymmat`

Get the number of symmetric matrixes stored.

`Task.getnumvar`

Obtains the number of variables.

`Task.getobjname`

Obtains the name assigned to the objective function.

`Task.getobjnamelen`

Obtains the length of the name assigned to the objective function.

`Task.getobjsense`

Gets the objective sense.

`Task.getpbi`

Deprecated.

`Task.getpcni`

Deprecated.

`Task.getpeqi`

Deprecated.

`Task.getprimalobj`

Computes the primal objective value for the desired solution.

`Task.getprobtype`

Obtains the problem type.

`Task.getprosta`

Obtains the problem status.

`Task.getpviolbarvar`

Computes the violation of a primal solution for a list of barx variables.

`Task.getpviolcon`

Computes the violation of a primal solution for a list of xc variables.

`Task.getpviolcones`

Computes the violation of a solution for set of conic constraints.

`Task.getpviolvar`

Computes the violation of a primal solution for a list of x variables.

`Task.getqconk`

Obtains all the quadratic terms in a constraint.

`Task.getqobj`

Obtains all the quadratic terms in the objective.

`Task.getqobj64`

Obtains all the quadratic terms in the objective.

`Task.getqobjij`

Obtains one coefficient from the quadratic term of the objective

`Task.getreducedcosts`

Obtains the difference of (slx-sux) for a sequence of variables.

`Task.getskc`

Obtains the status keys for the constraints.

`Task.getskcslice`

Obtains the status keys for the constraints.

`Task.getskx`

Obtains the status keys for the scalar variables.

`Task.getskxslice`

Obtains the status keys for the variables.

`Task.getslc`

Obtains the slc vector for a solution.

`Task.getslcslice`

Obtains a slice of the slc vector for a solution.

`Task.getslx`

Obtains the slx vector for a solution.

`Task.getslxslice`

Obtains a slice of the slx vector for a solution.

`Task.getsnx`

Obtains the snx vector for a solution.

`Task.getsnxslice`

Obtains a slice of the snx vector for a solution.

`Task.getsolsta`

Obtains the solution status.

`Task.getsolution`

Obtains the complete solution.

`Task.getsolutioni`

Obtains the solution for a single constraint or variable.

`Task.getsolutioninf`

Deprecated

`Task.getsolutioninfo`

Obtains information about of a solution.

`Task.getsolutionslice`

Obtains a slice of the solution.

`Task.getsparsesymmat`

Gets a single symmetric matrix from the matrix store.

`Task.getstrparam`

Obtains the value of a string parameter.

`Task.getstrparamlen`

Obtains the length of a string parameter.

`Task.getsuc`

Obtains the suc vector for a solution.

`Task.getsucslice`

Obtains a slice of the suc vector for a solution.

`Task.getsux`

Obtains the sux vector for a solution.

`Task.getsuxslice`

Obtains a slice of the sux vector for a solution.

`Task.getsymmatinfo`

Obtains information of a matrix from the symmetric matrix storage E.

`Task.gettaskname`

Obtains the task name.

`Task.gettasknamelen`

Obtains the length the task name.

`Task.getvarbound`

Obtains bound information for one variable.

`Task.getvarboundslice`

Obtains bounds information for a slice of the variables.

`Task.getvarbranchdir`

Obtains the branching direction for a variable.

`Task.getvarbranchpri`

Obtains the branching priority for a variable.

`Task.getvarname`

Obtains a name of a variable.

`Task.getvarnameindex`

Checks whether the name somename has been assigned to any variable.

`Task.getvarnamelen`

Obtains the length of a name of a variable variable.

`Task.getvartype`

Gets the variable type of one variable.

`Task.getvartypelist`

Obtains the variable type for one or more variables.

`Task.getxc`

Obtains the xc vector for a solution.

`Task.getxcslice`

Obtains a slice of the xc vector for a solution.

`Task.getxx`

Obtains the xx vector for a solution.

`Task.getxxslice`

Obtains a slice of the xx vector for a solution.

`Task.gety`

Obtains the y vector for a solution.

`Task.getyslice`

Obtains a slice of the y vector for a solution.

`Task.initbasissolve`

Prepare a task for basis solver.

`Task.inputdata`

Input the linear part of an optimization task in one function call.

`Task.isdouparname`

Checks a double parameter name.

`Task.isintparname`

Checks an integer parameter name.

`Task.isstrparname`

Checks a string parameter name.

`Env.licensecleanup`

Stops all threads and delete all handles used by the license system.

Env.linkfiletostream

Directs all output from a stream to a file.

Task.linkfiletostream

Directs all output from a task stream to a file.

Task.onesolutionsummary

Prints a short summary for the specified solution.

Task.optimizeconcurrent

Optimize a given task with several optimizers concurrently.

Task.optimizersummary

Prints a short summary with optimizer statistics for last optimization.

Task.optimize

Optimizes the problem.

Env.potrf

Computes a Cholesky factorization a dense matrix.

Task.primalrepair

The function repairs a primal infeasible optimization problem by adjusting the bounds on the constraints and variables.

Task.primalsensitivity

Perform sensitivity analysis on bounds.

Task.printdata

Prints a part of the problem data to a stream.

Task.printparam

Prints the current parameter settings.

Task.putacol

Replaces all elements in one column of A.

Task.putacollist

Replaces all elements in several columns the linear constraint matrix by new values.

Task.putacolslice

Replaces all elements in several columns the linear constraint matrix by new values.

Task.putaij

Changes a single value in the linear coefficient matrix.

`Task.putaijlist`

Changes one or more coefficients in the linear constraint matrix.

`Task.putarow`

Replaces all elements in one row of A.

`Task.putarowlist`

Replaces all elements in several rows the linear constraint matrix by new values.

`Task.putbarablocktriplet`

Inputs barA in block triplet form.

`Task.putbaraij`

Inputs an element of barA.

`Task.putbarcblocktriplet`

Inputs barC in block triplet form.

`Task.putbarcj`

Changes one element in barc.

`Task.putbarsj`

Sets the dual solution for a semidefinite variable.

`Task.putbarvarname`

Puts the name of a semidefinite variable.

`Task.putbarxj`

Sets the primal solution for a semidefinite variable.

`Task.putbound`

Changes the bound for either one constraint or one variable.

`Task.putboundlist`

Changes the bounds of constraints or variables.

`Task.putboundslice`

Modifies bounds.

`Task.putcfix`

Replaces the fixed term in the objective.

`Task.putcj`

Modifies one linear coefficient in the objective.

`Task.putclist`

Modifies a part of the linear objective coefficients.

`Task.putconbound`

Changes the bound for one constraint.

`Task.putconboundlist`

Changes the bounds of a list of constraints.

`Task.putconboundslice`

Changes the bounds for a slice of the constraints.

`Task.putcone`

Replaces a conic constraint.

`Task.putconename`

Puts the name of a cone.

`Task.putconname`

Puts the name of a constraint.

`Task.putcslice`

Modifies a slice of the linear objective coefficients.

`Task.putdouparam`

Sets a double parameter.

`Task.putintparam`

Sets an integer parameter.

`Env.putlicensecode`

The purpose of this function is to input a runtime license code.

`Env.putlicensedebug`

Enables debug information for the license system.

`Env.putlicensepath`

Set the path to the license file.

`Env.putlicensewait`

Control whether mosek should wait for an available license if no license is available.

`Task.putmaxnumanz`

The function changes the size of the preallocated storage for linear coefficients.

`Task.putmaxnumbarvar`

Sets the number of preallocated symmetric matrix variables in the optimization task.

`Task.putmaxnumcon`

Sets the number of preallocated constraints in the optimization task.

Task.putmaxnumcone

Sets the number of preallocated conic constraints in the optimization task.

Task.putmaxnumqnz

Changes the size of the preallocated storage for quadratic terms.

Task.putmaxnumvar

Sets the number of preallocated variables in the optimization task.

Task.putnadouparam

Sets a double parameter.

Task.putnaintparam

Sets an integer parameter.

Task.putnastrparam

Sets a string parameter.

Task.putobjname

Assigns a new name to the objective.

Task.putobjsense

Sets the objective sense.

Task.putparam

Modifies the value of parameter.

Task.putqcon

Replaces all quadratic terms in constraints.

Task.putqconk

Replaces all quadratic terms in a single constraint.

Task.putqobj

Replaces all quadratic terms in the objective.

Task.putqobjij

Replaces one coefficient in the quadratic term in the objective.

Task.putskc

Sets the status keys for the constraints.

Task.putskcslice

Sets the status keys for the constraints.

Task.putskx

Sets the status keys for the scalar variables.

`Task.putskxslice`

Sets the status keys for the variables.

`Task.putslc`

Sets the slc vector for a solution.

`Task.putslcslice`

Sets a slice of the slc vector for a solution.

`Task.putslx`

Sets the slx vector for a solution.

`Task.putslxslice`

Sets a slice of the slx vector for a solution.

`Task.putsnx`

Sets the snx vector for a solution.

`Task.putsnxslice`

Sets a slice of the snx vector for a solution.

`Task.putsolution`

Inserts a solution.

`Task.putsolutioni`

Sets the primal and dual solution information for a single constraint or variable.

`Task.putstrparam`

Sets a string parameter.

`Task.putsuc`

Sets the suc vector for a solution.

`Task.putsucslice`

Sets a slice of the suc vector for a solution.

`Task.putsux`

Sets the sux vector for a solution.

`Task.putsuxslice`

Sets a slice of the sux vector for a solution.

`Task.puttaskname`

Assigns a new name to the task.

`Task.putvarbound`

Changes the bound for one variable.

`Task.putvarboundlist`

Changes the bounds of a list of variables.

`Task.putvarboundslice`

Changes the bounds for a slice of the variables.

`Task.putvarbranchorder`

Assigns a branching priority and direction to a variable.

`Task.putvarname`

Puts the name of a variable.

`Task.putvartype`

Sets the variable type of one variable.

`Task.putvartypelist`

Sets the variable type for one or more variables.

`Task.putxc`

Sets the xc vector for a solution.

`Task.putxcslice`

Sets a slice of the xc vector for a solution.

`Task.putxx`

Sets the xx vector for a solution.

`Task.putxxslice`

Obtains a slice of the xx vector for a solution.

`Task.puty`

Sets the y vector for a solution.

`Task.putyslice`

Sets a slice of the y vector for a solution.

`Task.readbranchpriorities`

Reads branching priority data from a file.

`Task.readdata`

Reads problem data from a file.

`Task.readdataformat`

Reads problem data from a file.

`Task.readparamfile`

Reads a parameter file.

`Task.readsolution`

Reads a solution from a file.

`Task.readsummary`

Prints information about last file read.

`Task.relaxprimal`

Deprecated.

`Task.removebarvars`

The function removes a number of symmetric matrix.

`Task.removecones`

Removes a conic constraint from the problem.

`Task.removecons`

The function removes a number of constraints.

`Task.removevars`

The function removes a number of variables.

`Task.sensitivityreport`

Creates a sensitivity report.

`Task.setdefaults`

Resets all parameters values.

`Task.solutiondef`

Checks whether a solution is defined.

`Task.solutionsummary`

Prints a short summary of the current solutions.

`Task.solvewithbasis`

Solve a linear equation system involving a basis matrix.

`Task.startstat`

Starts the statistics file.

`Task.stopstat`

Stops the statistics file.

`Env.syeig`

Computes all eigenvalues of a symmetric dense matrix.

`Env.syevd`

Computes all the eigenvalue and eigenvectors of a symmetric dense matrix, and thus its eigenvalue decomposition.

`Env.syrk`

Performs a rank-k update of a symmetric matrix.

`Task.updatesolutioninfo`

Update the information items related to the solution.

`Task.writebranchpriorities`

Writes branching priority data to a file.

`Task.writedata`

Writes problem data to a file.

`Task.writeparamfile`

Writes all the parameters to a parameter file.

`Task.writesolution`

Write a solution to a file.

A.6.1 Env()

```
Env()
```

The environment construction takes no arguments. Please note that each process should only construct one environment, even when multiple task object are constructed.

A.6.2 Task()

The task object is created from an environment object and, optionally, the problem maximum dimensions. The the dimensions are not given they default to 0, put they can be changed afterwards.

If a `Task` object is given instead of an `Env` object, the new task is created using the data from the old task. Callback objects are not copied.

Appendix B

Parameters

Parameters grouped by functionality.

Analysis parameters.

Parameters controling the behaviour of the problem and solution analyzers.

- dparam.ana_sol_infeas_tol. If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer.

Basis identification parameters.

- iparam.bi_clean_optimizer. Controls which simplex optimizer is used in the clean-up phase.
- iparam.bi_ignore_max_iter. Turns on basis identification in case the interior-point optimizer is terminated due to maximum number of iterations.
- iparam.bi_ignore_num_error. Turns on basis identification in case the interior-point optimizer is terminated due to a numerical problem.
- iparam.bi_max_iterations. Maximum number of iterations after basis identification.
- iparam.intpnt_basis. Controls whether basis identification is performed.
- iparam.log_bi. Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.
- iparam.log_bi_freq. Controls the logging frequency.
- dparam.sim_lu_tol_rel_piv. Relative pivot tolerance employed when computing the LU factorization of the basis matrix.

Behavior of the optimization task.

Parameters defining the behavior of an optimization task when loading data.

- sparam.feasrepair_name_prefix. Feasibility repair name prefix.
- sparam.feasrepair_name_separator. Feasibility repair name separator.

- sparam.feasrepair_name_wsumviol. Feasibility repair name violation name.

Conic interior-point method parameters.

Parameters defining the behavior of the interior-point method for conic problems.

- dparam.intpnt_co_tol_dfeas. Dual feasibility tolerance used by the conic interior-point optimizer.
- dparam.intpnt_co_tol_infeas. Infeasibility tolerance for the conic solver.
- dparam.intpnt_co_tol_mu_red. Optimality tolerance for the conic solver.
- dparam.intpnt_co_tol_near_rel. Optimality tolerance for the conic solver.
- dparam.intpnt_co_tol_pfeas. Primal feasibility tolerance used by the conic interior-point optimizer.
- dparam.intpnt_co_tol_rel_gap. Relative gap termination tolerance used by the conic interior-point optimizer.

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.

- dparam.data_tol_aij. Data tolerance threshold.
- dparam.data_tol_aij_huge. Data tolerance threshold.
- dparam.data_tol_aij_large. Data tolerance threshold.
- dparam.data_tol_bound_inf. Data tolerance threshold.
- dparam.data_tol_bound_wrn. Data tolerance threshold.
- dparam.data_tol_c_huge. Data tolerance threshold.
- dparam.data_tol_cj_large. Data tolerance threshold.
- dparam.data_tol_qij. Data tolerance threshold.
- dparam.data_tol_x. Data tolerance threshold.
- iparam.log_check_convexity. Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on.

  If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

Data input/output parameters.

Parameters defining the behavior of data readers and writers.

- sparam.bas_sol_file_name. Name of the bas solution file.
- sparam.data_file_name. Data are read and written to this file.
- sparam.debug_file_name. MOSEK debug file.
- sparam.int_sol_file_name. Name of the int solution file.
- sparam.itr_sol_file_name. Name of the itr solution file.

- `iparam.log_file`. If turned on, then some log info is printed when a file is written or read.
- `sparam.mio_debug_string`. For internal use only.
- `sparam.param_comment_sign`. Solution file comment character.
- `sparam.param_read_file_name`. Modifications to the parameter database is read from this file.
- `sparam.param_write_file_name`. The parameter database is written to this file.
- `sparam.read_mps_bou_name`. Name of the BOUNDS vector used. An empty name means that the first BOUNDS vector is used.
- `sparam.read_mps_obj_name`. Objective name in the MPS file.
- `sparam.read_mps_ran_name`. Name of the RANGE vector used. An empty name means that the first RANGE vector is used.
- `sparam.read_mps_rhs_name`. Name of the RHS used. An empty name means that the first RHS vector is used.
- `sparam.sol_filter_xc_low`. Solution file filter.
- `sparam.sol_filter_xc_upr`. Solution file filter.
- `sparam.sol_filter_xx_low`. Solution file filter.
- `sparam.sol_filter_xx_upr`. Solution file filter.
- `sparam.stat_file_name`. Statistics file name.
- `sparam.stat_key`. Key used when writing the summary file.
- `sparam.stat_name`. Name used when writing the statistics file.
- `sparam.write_lp_gen_var_name`. Added variable names in the LP files.

Debugging parameters.

These parameters defines that can be used when debugging a problem.

- `iparam.auto_sort_a_before_opt`. Controls whether the elements in each column of A are sorted before an optimization is performed.

Dual simplex optimizer parameters.

Parameters defining the behavior of the dual simplex optimizer for linear problems.

- `iparam.sim_dual_crash`. Controls whether crashing is performed in the dual simplex optimizer.
- `iparam.sim_dual_restrict_selection`. Controls how aggressively restricted selection is used.
- `iparam.sim_dual_selection`. Controls the dual simplex strategy.

Feasibility repair parameters.

- `dparam.feasrepair_tol`. Tolerance for constraint enforcing upper bound on sum of weighted violations in feasibility repair.

Infeasibility report parameters.

- iparam.log_infeas_ana. Controls log level for the infeasibility analyzer.

Interior-point method parameters.

Parameters defining the behavior of the interior-point method for linear, conic and convex problems.

- iparam.bi_ignore_max_iter. Turns on basis identification in case the interior-point optimizer is terminated due to maximum number of iterations.
- iparam.bi_ignore_num_error. Turns on basis identification in case the interior-point optimizer is terminated due to a numerical problem.
- dparam.check_convexity_rel_tol. Convexity check tolerance.
- iparam.intpnt_basis. Controls whether basis identification is performed.
- dparam.intpnt_co_tol_dfeas. Dual feasibility tolerance used by the conic interior-point optimizer.
- dparam.intpnt_co_tol_infeas. Infeasibility tolerance for the conic solver.
- dparam.intpnt_co_tol_mu_red. Optimality tolerance for the conic solver.
- dparam.intpnt_co_tol_near_rel. Optimality tolerance for the conic solver.
- dparam.intpnt_co_tol_pfeas. Primal feasibility tolerance used by the conic interior-point optimizer.
- dparam.intpnt_co_tol_rel_gap. Relative gap termination tolerance used by the conic interior-point optimizer.
- iparam.intpnt_diff_step. Controls whether different step sizes are allowed in the primal and dual space.
- iparam.intpnt_max_iterations. Controls the maximum number of iterations allowed in the interior-point optimizer.
- iparam.intpnt_max_num_cor. Maximum number of correction steps.
- iparam.intpnt_max_num_refinement_steps. Maximum number of steps to be used by the iterative search direction refinement.
- dparam.intpnt_nl_merit_bal. Controls if the complementarity and infeasibility is converging to zero at about equal rates.
- dparam.intpnt_nl_tol_dfeas. Dual feasibility tolerance used when a nonlinear model is solved.
- dparam.intpnt_nl_tol_mu_red. Relative complementarity gap tolerance.
- dparam.intpnt_nl_tol_near_rel. Nonlinear solver optimality tolerance parameter.
- dparam.intpnt_nl_tol_pfeas. Primal feasibility tolerance used when a nonlinear model is solved.
- dparam.intpnt_nl_tol_rel_gap. Relative gap termination tolerance for nonlinear problems.

- dparam.intpnt_nl_tol_rel_step. Relative step size to the boundary for general nonlinear optimization problems.
- iparam.intpnt_off_col_trh. Controls the aggressiveness of the offending column detection.
- iparam.intpnt_order_method. Controls the ordering strategy.
- iparam.intpnt_regularization_use. Controls whether regularization is allowed.
- iparam.intpnt_scaling. Controls how the problem is scaled before the interior-point optimizer is used.
- iparam.intpnt_solve_form. Controls whether the primal or the dual problem is solved.
- iparam.intpnt_starting_point. Starting point used by the interior-point optimizer.
- dparam.intpnt_tol_dfeas. Dual feasibility tolerance used for linear and quadratic optimization problems.
- dparam.intpnt_tol_dsafe. Controls the interior-point dual starting point.
- dparam.intpnt_tol_infeas. Nonlinear solver infeasibility tolerance parameter.
- dparam.intpnt_tol_mu_red. Relative complementarity gap tolerance.
- dparam.intpnt_tol_path. interior-point centering aggressiveness.
- dparam.intpnt_tol_pfeas. Primal feasibility tolerance used for linear and quadratic optimization problems.
- dparam.intpnt_tol_psafe. Controls the interior-point primal starting point.
- dparam.intpnt_tol_rel_gap. Relative gap termination tolerance.
- dparam.intpnt_tol_rel_step. Relative step size to the boundary for linear and quadratic optimization problems.
- dparam.intpnt_tol_step_size. If the step size falls below the value of this parameter, then the interior-point optimizer assumes that it is stalled. In other words the interior-point optimizer does not make any progress and therefore it is better stop.
- iparam.log_intpnt. Controls the amount of log information from the interior-point optimizers.
- iparam.log_presolve. Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.
- dparam.qcqo_reformulate_rel_drop_tol. This parameter determines when columns are dropped in incomplete cholesky factorization doing reformulation of quadratic problems.

License manager parameters.

- iparam.cache_license. Control license caching.
- iparam.license_debug. Controls the license manager client debugging behavior.
- iparam.license_pause_time. Controls license manager client behavior.
- iparam.license_suppress_expire_wrns. Controls license manager client behavior.
- iparam.license_wait. Controls if MOSEK should queue for a license if none is available.

Logging parameters.

- iparam.log. Controls the amount of log information.
- iparam.log_bi. Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.
- iparam.log_bi_freq. Controls the logging frequency.
- iparam.log_concurrent. Controls amount of output printed by the concurrent optimizer.
- iparam.log_expand. Controls the amount of logging when a data item such as the maximum number constrains is expanded.
- iparam.log_factor. If turned on, then the factor log lines are added to the log.
- iparam.log_feas_repair. Controls the amount of output printed when performing feasibility repair. A value higher than one means extensive logging.
- iparam.log_file. If turned on, then some log info is printed when a file is written or read.
- iparam.log_head. If turned on, then a header line is added to the log.
- iparam.log_infeas_ana. Controls log level for the infeasibility analyzer.
- iparam.log_intpnt. Controls the amount of log information from the interior-point optimizers.
- iparam.log_mio. Controls the amount of log information from the mixed-integer optimizers.
- iparam.log_mio_freq. The mixed-integer solver logging frequency.
- iparam.log_nonconvex. Controls amount of output printed by the nonconvex optimizer.
- iparam.log_optimizer. Controls the amount of general optimizer information that is logged.
- iparam.log_order. If turned on, then factor lines are added to the log.
- iparam.log_param. Controls the amount of information printed out about parameter changes.
- iparam.log_presolve. Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.
- iparam.log_response. Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.
- iparam.log_sim. Controls the amount of log information from the simplex optimizers.
- iparam.log_sim_freq. Controls simplex logging frequency.
- iparam.log_sim_network_freq. Controls the network simplex logging frequency.
- iparam.log_storage. Controls the memory related log information.

Mixed-integer optimization parameters.

- iparam.log_mio. Controls the amount of log information from the mixed-integer optimizers.
- iparam.log_mio_freq. The mixed-integer solver logging frequency.
- iparam.mio_branch_dir. Controls whether the mixed-integer optimizer is branching up or down by default.

- iparam.mio_construct_sol. Controls if an initial mixed integer solution should be constructed from the values of the integer variables.
- iparam.mio_cont_sol. Controls the meaning of interior-point and basic solutions in mixed integer problems.
- iparam.mio_cut_cg. Controls whether CG cuts should be generated.
- iparam.mio_cut_cmir. Controls whether mixed integer rounding cuts should be generated.
- iparam.mio_cut_level_root. Controls the cut level employed by the mixed-integer optimizer at the root node.
- iparam.mio_cut_level_tree. Controls the cut level employed by the mixed-integer optimizer in the tree.
- dparam.mio_disable_term_time. Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.
- iparam.mio_feaspump_level. Controls the feasibility pump heuristic which is used to construct a good initial feasible solution.
- iparam.mio_heuristic_level. Controls the heuristic employed by the mixed-integer optimizer to locate an initial integer feasible solution.
- dparam.mio_heuristic_time. Time limit for the mixed-integer heuristics.
- iparam.mio_hotstart. Controls whether the integer optimizer is hot-started.
- iparam.mio_keep_basis. Controls whether the integer presolve keeps bases in memory.
- iparam.mio_max_num_branches. Maximum number of branches allowed during the branch and bound search.
- iparam.mio_max_num_relaxs. Maximum number of relaxations in branch and bound search.
- iparam.mio_max_num_solutions. Controls how many feasible solutions the mixed-integer optimizer investigates.
- dparam.mio_max_time. Time limit for the mixed-integer optimizer.
- dparam.mio_max_time_aprx_opt. Time limit for the mixed-integer optimizer.
- dparam.mio_near_tol_abs_gap. Relaxed absolute optimality tolerance employed by the mixed-integer optimizer.
- dparam.mio_near_tol_rel_gap. The mixed-integer optimizer is terminated when this tolerance is satisfied.
- iparam.mio_node_optimizer. Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.
- iparam.mio_node_selection. Controls the node selection strategy employed by the mixed-integer optimizer.
- iparam.mio_optimizer_mode. An exprimental feature.
- iparam.mio_presolve_aggregate. Controls whether problem aggregation is performed in the mixed-integer presolve.
- iparam.mio_presolve_probing. Controls whether probing is employed by the mixed-integer presolve.

- iparam.mio_presolve_use. Controls whether presolve is performed by the mixed-integer optimizer.
- iparam.mio_probing_level. Controls the amount of probing employed by the mixed-integer optimizer in presolve.
- dparam.mio_rel_add_cut_limited. Controls cut generation for mixed-integer optimizer.
- dparam.mio_rel_gap_const. This value is used to compute the relative gap for the solution to an integer optimization problem.
- iparam.mio_rins_max_nodes. Maximum number of nodes in each call to the RINS heuristic.
- iparam.mio_root_optimizer. Controls which optimizer is employed at the root node in the mixed-integer optimizer.
- iparam.mio_strong_branch. The depth from the root in which strong branching is employed.
- dparam.mio_tol_abs_gap. Absolute optimality tolerance employed by the mixed-integer optimizer.
- dparam.mio_tol_abs_relax_int. Integer constraint tolerance.
- dparam.mio_tol_feas. Feasibility tolerance for mixed integer solver. Any solution with maximum infeasibility below this value will be considered feasible.
- dparam.mio_tol_max_cut_frac_rhs. Controls cut generation for mixed-integer optimizer.
- dparam.mio_tol_min_cut_frac_rhs. Controls cut generation for mixed-integer optimizer.
- dparam.mio_tol_rel_dual_bound_improvement. Controls cut generation for mixed-integer optimizer.
- dparam.mio_tol_rel_gap. Relative optimality tolerance employed by the mixed-integer optimizer.
- dparam.mio_tol_rel_relax_int. Integer constraint tolerance.
- dparam.mio_tol_x. Absolute solution tolerance used in mixed-integer optimizer.
- iparam.mio_use_multithreaded_optimizer. Controls wheter the new multithreaded optimizer should be used for Mixed integer problems.

Network simplex optimizer parameters.

Parameters defining the behavior of the network simplex optimizer for linear problems.

- iparam.log_sim_network_freq. Controls the network simplex logging frequency.
- iparam.sim_refactor_freq. Controls the basis refactoring frequency.

Non-convex solver parameters.

- iparam.log_nonconvex. Controls amount of output printed by the nonconvex optimizer.
- iparam.nonconvex_max_iterations. Maximum number of iterations that can be used by the nonconvex optimizer.
- dparam.nonconvex_tol_feas. Feasibility tolerance used by the nonconvex optimizer.
- dparam.nonconvex_tol_opt. Optimality tolerance used by the nonconvex optimizer.

Nonlinear convex method parameters.

Parameters defining the behavior of the interior-point method for nonlinear convex problems.

- dparam.intpnt_nl_merit_bal. Controls if the complementarity and infeasibility is converging to zero at about equal rates.
- dparam.intpnt_nl_tol_dfeas. Dual feasibility tolerance used when a nonlinear model is solved.
- dparam.intpnt_nl_tol_mu_red. Relative complementarity gap tolerance.
- dparam.intpnt_nl_tol_near_rel. Nonlinear solver optimality tolerance parameter.
- dparam.intpnt_nl_tol_pfeas. Primal feasibility tolerance used when a nonlinear model is solved.
- dparam.intpnt_nl_tol_rel_gap. Relative gap termination tolerance for nonlinear problems.
- dparam.intpnt_nl_tol_rel_step. Relative step size to the boundary for general nonlinear optimization problems.
- dparam.intpnt_tol_infeas. Nonlinear solver infeasibility tolerance parameter.
- iparam.log_check_convexity. Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on.

  If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

Optimization system parameters.

Parameters defining the overall solver system environment. This includes system and platform related information and behavior.

- iparam.license_wait. Controls if MOSEK should queue for a license if none is available.
- iparam.log_storage. Controls the memory related log information.
- iparam.num_threads. Controls the number of threads employed by the optimizer. If set to 0 the number of threads used will be equal to the number of cores detected on the machine.

Output information parameters.

- iparam.license_suppress_expire_wrns. Controls license manager client behavior.
- iparam.log. Controls the amount of log information.
- iparam.log_bi. Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.
- iparam.log_bi_freq. Controls the logging frequency.
- iparam.log_expand. Controls the amount of logging when a data item such as the maximum number constrains is expanded.
- iparam.log_factor. If turned on, then the factor log lines are added to the log.

- `iparam.log_feas_repair`. Controls the amount of output printed when performing feasibility repair. A value higher than one means extensive logging.
- `iparam.log_file`. If turned on, then some log info is printed when a file is written or read.
- `iparam.log_head`. If turned on, then a header line is added to the log.
- `iparam.log_infeas_ana`. Controls log level for the infeasibility analyzer.
- `iparam.log_intpnt`. Controls the amount of log information from the interior-point optimizers.
- `iparam.log_mio`. Controls the amount of log information from the mixed-integer optimizers.
- `iparam.log_mio_freq`. The mixed-integer solver logging frequency.
- `iparam.log_nonconvex`. Controls amount of output printed by the nonconvex optimizer.
- `iparam.log_optimizer`. Controls the amount of general optimizer information that is logged.
- `iparam.log_order`. If turned on, then factor lines are added to the log.
- `iparam.log_param`. Controls the amount of information printed out about parameter changes.
- `iparam.log_response`. Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.
- `iparam.log_sim`. Controls the amount of log information from the simplex optimizers.
- `iparam.log_sim_freq`. Controls simplex logging frequency.
- `iparam.log_sim_minor`. Currently not in use.
- `iparam.log_sim_network_freq`. Controls the network simplex logging frequency.
- `iparam.log_storage`. Controls the memory related log information.
- `iparam.max_num_warnings`. A negtive number means all warnings are logged. Otherwise the parameter specifies the maximum number times each warning is logged.
- `iparam.warning_level`. Deprecated and not in use

Overall solver parameters.

- `iparam.bi_clean_optimizer`. Controls which simplex optimizer is used in the clean-up phase.
- `iparam.concurrent_num_optimizers`. The maximum number of simultaneous optimizations that will be started by the concurrent optimizer.
- `iparam.concurrent_priority_dual_simplex`. Priority of the dual simplex algorithm when selecting solvers for concurrent optimization.
- `iparam.concurrent_priority_free_simplex`. Priority of the free simplex optimizer when selecting solvers for concurrent optimization.
- `iparam.concurrent_priority_intpnt`. Priority of the interior-point algorithm when selecting solvers for concurrent optimization.
- `iparam.concurrent_priority_primal_simplex`. Priority of the primal simplex algorithm when selecting solvers for concurrent optimization.

- iparam.infeas_prefer_primal. Controls which certificate is used if both primal- and dual- certificate of infeasibility is available.
- iparam.license_wait. Controls if MOSEK should queue for a license if none is available.
- iparam.mio_cont_sol. Controls the meaning of interior-point and basic solutions in mixed integer problems.
- iparam.mio_local_branch_number. Controls the size of the local search space when doing local branching.
- iparam.mio_mode. Turns on/off the mixed-integer mode.
- iparam.optimizer. Controls which optimizer is used to optimize the task.
- iparam.presolve_level. Currently not used.
- iparam.presolve_use. Controls whether the presolve is applied to a problem before it is optimized.
- iparam.solution_callback. Indicates whether solution call-backs will be performed during the optimization.

Presolve parameters.

- iparam.presolve_elim_fill. Maximum amount of fill-in in the elimination phase.
- iparam.presolve_eliminator_max_num_tries. Control the maximum number of times the eliminator is tried.
- iparam.presolve_eliminator_use. Controls whether free or implied free variables are eliminated from the problem.
- iparam.presolve_level. Currently not used.
- iparam.presolve_lindep_abs_work_trh. Controls linear dependency check in presolve.
- iparam.presolve_lindep_rel_work_trh. Controls linear dependency check in presolve.
- iparam.presolve_lindep_use. Controls whether the linear constraints are checked for linear dependencies.
- dparam.presolve_tol_abs_lindep. Absolute tolerance employed by the linear dependency checker.
- dparam.presolve_tol_aij. Absolute zero tolerance employed for constraint coefficients in the presolve.
- dparam.presolve_tol_rel_lindep. Relative tolerance employed by the linear dependency checker.
- dparam.presolve_tol_s. Absolute zero tolerance employed for slack variables in the presolve.
- dparam.presolve_tol_x. Absolute zero tolerance employed for variables in the presolve.
- iparam.presolve_use. Controls whether the presolve is applied to a problem before it is optimized.

Primal simplex optimizer parameters.

Parameters defining the behavior of the primal simplex optimizer for linear problems.

- iparam.sim_primal_crash. Controls the simplex crash.
- iparam.sim_primal_restrict_selection. Controls how aggressively restricted selection is used.
- iparam.sim_primal_selection. Controls the primal simplex strategy.

Progress call-back parameters.

- iparam.solution_callback. Indicates whether solution call-backs will be performed during the optimization.

Simplex optimizer parameters.

Parameters defining the behavior of the simplex optimizer for linear problems.

- dparam.basis_rel_tol_s. Maximum relative dual bound violation allowed in an optimal basic solution.
- dparam.basis_tol_s. Maximum absolute dual bound violation in an optimal basic solution.
- dparam.basis_tol_x. Maximum absolute primal bound violation allowed in an optimal basic solution.
- iparam.log_sim. Controls the amount of log information from the simplex optimizers.
- iparam.log_sim_freq. Controls simplex logging frequency.
- iparam.log_sim_minor. Currently not in use.
- iparam.sim_basis_factor_use. Controls whether a (LU) factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penanlty.
- iparam.sim_degen. Controls how aggressively degeneration is handled.
- iparam.sim_dual_phaseone_method. An exprimental feature.
- iparam.sim_exploit_dupvec. Controls if the simplex optimizers are allowed to exploit duplicated columns.
- iparam.sim_hotstart. Controls the type of hot-start that the simplex optimizer perform.
- iparam.sim_integer. An exprimental feature.
- dparam.sim_lu_tol_rel_piv. Relative pivot tolerance employed when computing the LU factorization of the basis matrix.
- iparam.sim_max_iterations. Maximum number of iterations that can be used by a simplex optimizer.
- iparam.sim_max_num_setbacks. Controls how many set-backs that are allowed within a simplex optimizer.
- iparam.sim_non_singular. Controls if the simplex optimizer ensures a non-singular basis, if possible.
- iparam.sim_primal_phaseone_method. An exprimental feature.
- iparam.sim_reformulation. Controls if the simplex optimizers are allowed to reformulate the problem.

- iparam.sim_save_lu. Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.
- iparam.sim_scaling. Controls how much effort is used in scaling the problem before a simplex optimizer is used.
- iparam.sim_scaling_method. Controls how the problem is scaled before a simplex optimizer is used.
- iparam.sim_solve_form. Controls whether the primal or the dual problem is solved by the primal-/dual- simplex optimizer.
- iparam.sim_stability_priority. Controls how high priority the numerical stability should be given.
- iparam.sim_switch_optimizer. Controls the simplex behavior.
- dparam.simplex_abs_tol_piv. Absolute pivot tolerance employed by the simplex optimizers.

Solution input/output parameters.

Parameters defining the behavior of solution reader and writer.

- sparam.bas_sol_file_name. Name of the bas solution file.
- sparam.int_sol_file_name. Name of the int solution file.
- sparam.itr_sol_file_name. Name of the itr solution file.
- iparam.sol_filter_keep_basic. Controls the license manager client behavior.
- sparam.sol_filter_xc_low. Solution file filter.
- sparam.sol_filter_xc_upr. Solution file filter.
- sparam.sol_filter_xx_low. Solution file filter.
- sparam.sol_filter_xx_upr. Solution file filter.

Termination criterion parameters.

Parameters which define termination and optimality criteria and related information.

- dparam.basis_rel_tol_s. Maximum relative dual bound violation allowed in an optimal basic solution.
- dparam.basis_tol_s. Maximum absolute dual bound violation in an optimal basic solution.
- dparam.basis_tol_x. Maximum absolute primal bound violation allowed in an optimal basic solution.
- iparam.bi_max_iterations. Maximum number of iterations after basis identification.
- dparam.intpnt_co_tol_dfeas. Dual feasibility tolerance used by the conic interior-point optimizer.
- dparam.intpnt_co_tol_infeas. Infeasibility tolerance for the conic solver.
- dparam.intpnt_co_tol_mu_red. Optimality tolerance for the conic solver.
- dparam.intpnt_co_tol_near_rel. Optimality tolerance for the conic solver.

- dparam.intpnt_co_tol_pfeas. Primal feasibility tolerance used by the conic interior-point optimizer.
- dparam.intpnt_co_tol_rel_gap. Relative gap termination tolerance used by the conic interior-point optimizer.
- iparam.intpnt_max_iterations. Controls the maximum number of iterations allowed in the interior-point optimizer.
- dparam.intpnt_nl_tol_dfeas. Dual feasibility tolerance used when a nonlinear model is solved.
- dparam.intpnt_nl_tol_mu_red. Relative complementarity gap tolerance.
- dparam.intpnt_nl_tol_near_rel. Nonlinear solver optimality tolerance parameter.
- dparam.intpnt_nl_tol_pfeas. Primal feasibility tolerance used when a nonlinear model is solved.
- dparam.intpnt_nl_tol_rel_gap. Relative gap termination tolerance for nonlinear problems.
- dparam.intpnt_tol_dfeas. Dual feasibility tolerance used for linear and quadratic optimization problems.
- dparam.intpnt_tol_infeas. Nonlinear solver infeasibility tolerance parameter.
- dparam.intpnt_tol_mu_red. Relative complementarity gap tolerance.
- dparam.intpnt_tol_pfeas. Primal feasibility tolerance used for linear and quadratic optimization problems.
- dparam.intpnt_tol_rel_gap. Relative gap termination tolerance.
- dparam.lower_obj_cut. Objective bound.
- dparam.lower_obj_cut_finite_trh. Objective bound.
- dparam.mio_disable_term_time. Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.
- iparam.mio_max_num_branches. Maximum number of branches allowed during the branch and bound search.
- iparam.mio_max_num_solutions. Controls how many feasible solutions the mixed-integer optimizer investigates.
- dparam.mio_max_time. Time limit for the mixed-integer optimizer.
- dparam.mio_near_tol_rel_gap. The mixed-integer optimizer is terminated when this tolerance is satisfied.
- dparam.mio_rel_gap_const. This value is used to compute the relative gap for the solution to an integer optimization problem.
- dparam.mio_tol_rel_gap. Relative optimality tolerance employed by the mixed-integer optimizer.
- dparam.optimizer_max_time. Solver time limit.
- iparam.sim_max_iterations. Maximum number of iterations that can be used by a simplex optimizer.

- `dparam.upper_obj_cut`. Objective bound.
- `dparam.upper_obj_cut_finite_trh`. Objective bound.

- Integer parameters
- Double parameters
- String parameters

B.1 dparam: Double parameters

B.1.1 dparam.ana_sol_infeas_tol

Corresponding constant:

dparam.ana_sol_infeas_tol

Description:

If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer.

Possible Values:

Any number between 0.0 and +inf.

Default value:

`1e-6`

B.1.2 dparam.basis_rel_tol_s

Corresponding constant:

dparam.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`

B.1.3 dparam.basis_tol_s

Corresponding constant:

dparam.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`

B.1.4 dparam.basis_tol_x

Corresponding constant:

dparam.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`

B.1.5 dparam.check_convexity_rel_tol

Corresponding constant:

dparam.check_convexity_rel_tol

Description:

This parameter controls when the full convexity check declares a problem to be non-convex. Increasing this tolerance relaxes the criteria for declaring the problem non-convex.

A problem is declared non-convex if negative (positive) pivot elements are detected in the cholesky factor of a matrix which is required to be PSD (NSD). This parameter controles how much this non-negativity requirement may be violated.

If d_i is the pivot element for column i, then the matrix Q is considered to not be PSD if:

$$d_i \leq -|Q_{ii}| * \texttt{check\_convexity\_rel\_tol}$$

Possible Values:

Any number between 0 and +inf.

Default value:

`1e-10`

B.1.6 dparam.data_tol_aij

Corresponding constant:

dparam.data_tol_aij

Description:

Absolute zero tolerance for elements in A. If any value A_{ij} is smaller than this parameter in absolute terms MOSEK will treat the values as zero and generate a warning.

Possible Values:

Any number between 1.0e-16 and 1.0e-6.

Default value:

`1.0e-12`

B.1.7 dparam.data_tol_aij_huge

Corresponding constant:

dparam.data_tol_aij_huge

Description:

An element in A which is larger than this value in absolute size causes an error.

Possible Values:

Any number between 0.0 and +inf.

Default value:

`1.0e20`

B.1.8 dparam.data_tol_aij_large

Corresponding constant:

dparam.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`

B.1.9 dparam.data_tol_bound_inf

Corresponding constant:

dparam.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`

B.1.10 dparam.data_tol_bound_wrn

Corresponding constant:

dparam.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`

B.1.11 dparam.data_tol_c_huge

Corresponding constant:

dparam.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`

B.1.12 dparam.data_tol_cj_large

Corresponding constant:

dparam.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`

B.1.13 dparam.data_tol_qij

Corresponding constant:

dparam.data_tol_qij

Description:

Absolute zero tolerance for elements in Q matrixes.

Possible Values:

Any number between 0.0 and +inf.

Default value:

`1.0e-16`

B.1.14 dparam.data_tol_x

Corresponding constant:

dparam.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`

B.1.15 dparam.feasrepair_tol

Corresponding constant:

dparam.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`

B.1.16 dparam.intpnt_co_tol_dfeas

Corresponding constant:

dparam.intpnt_co_tol_dfeas

Description:

Dual feasibility tolerance used by the conic interior-point optimizer.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

`1.0e-8`

See also:

- `dparam.intpnt_co_tol_near_rel` Optimality tolerance for the conic solver.

B.1.17 dparam.intpnt_co_tol_infeas

Corresponding constant:

dparam.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-10`

B.1.18 dparam.intpnt_co_tol_mu_red

Corresponding constant:

dparam.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`

B.1.19 dparam.intpnt_co_tol_near_rel

Corresponding constant:

dparam.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:

`1000`

B.1.20 dparam.intpnt_co_tol_pfeas

Corresponding constant:

dparam.intpnt_co_tol_pfeas

Description:

Primal feasibility tolerance used by the conic interior-point optimizer.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

`1.0e-8`

See also:

- `dparam.intpnt_co_tol_near_rel` Optimality tolerance for the conic solver.

B.1.21 dparam.intpnt_co_tol_rel_gap

Corresponding constant:

dparam.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-7`

See also:

- `dparam.intpnt_co_tol_near_rel` Optimality tolerance for the conic solver.

B.1.22 dparam.intpnt_nl_merit_bal

Corresponding constant:

dparam.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`

B.1.23 dparam.intpnt_nl_tol_dfeas

Corresponding constant:

dparam.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`

B.1.24 dparam.intpnt_nl_tol_mu_red

Corresponding constant:

dparam.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`

B.1.25 dparam.intpnt_nl_tol_near_rel

Corresponding constant:

dparam.intpnt_nl_tol_near_rel

Description:

If the MOSEK nonlinear interior-point optimizer cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Possible Values:

Any number between 1.0 and +inf.

Default value:

`1000.0`

B.1.26 dparam.intpnt_nl_tol_pfeas

Corresponding constant:

dparam.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`

B.1.27 dparam.intpnt_nl_tol_rel_gap

Corresponding constant:

dparam.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`

B.1.28 dparam.intpnt_nl_tol_rel_step

Corresponding constant:

dparam.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`

B.1.29 dparam.intpnt_tol_dfeas

Corresponding constant:

dparam.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`

B.1.30 dparam.intpnt_tol_dsafe

Corresponding constant:

dparam.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`

B.1.31 dparam.intpnt_tol_infeas

Corresponding constant:

dparam.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-10`

B.1.32 dparam.intpnt_tol_mu_red

Corresponding constant:

dparam.intpnt_tol_mu_red

Description:

Relative complementarity gap tolerance.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

`1.0e-16`

B.1.33 dparam.intpnt_tol_path

Corresponding constant:

dparam.intpnt_tol_path

Description:

Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central is followed very closely. On numerical unstable problems it may be worthwhile to increase this parameter.

Possible Values:

Any number between 0.0 and 0.9999.

Default value:

`1.0e-8`

B.1.34 dparam.intpnt_tol_pfeas

Corresponding constant:

dparam.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`

B.1.35 dparam.intpnt_tol_psafe

Corresponding constant:

dparam.intpnt_tol_psafe

Description:

Controls the initial primal starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it may be worthwhile to increase this value.

Possible Values:

Any number between 1.0e-4 and +inf.

Default value:

`1.0`

B.1.36 dparam.intpnt_tol_rel_gap

Corresponding constant:

dparam.intpnt_tol_rel_gap

Description:

Relative gap termination tolerance.

Possible Values:

Any number between 1.0e-14 and +inf.

Default value:

`1.0e-8`

B.1.37 dparam.intpnt_tol_rel_step

Corresponding constant:

dparam.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`

B.1.38 dparam.intpnt_tol_step_size

Corresponding constant:

dparam.intpnt_tol_step_size

Description:

If the step size falls below the value of this parameter, then the interior-point optimizer assumes that it is stalled. In other words the interior-point optimizer does not make any progress and therefore it is better stop.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

`1.0e-6`

B.1.39 dparam.lower_obj_cut

Corresponding constant:

dparam.lower_obj_cut

Description:

If either a primal or dual feasible solution is found proving that the optimal objective value is outside, the interval [`dparam.lower_obj_cut`, `dparam.upper_obj_cut`], then MOSEK is terminated.

Possible Values:

Any number between -inf and +inf.

Default value:

`-1.0e30`

See also:

- `dparam.lower_obj_cut_finite_trh` Objective bound.

B.1.40 dparam.lower_obj_cut_finite_trh

Corresponding constant:

dparam.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. `dparam.lower_obj_cut` is treated as $-\infty$.

Possible Values:

Any number between -inf and +inf.

Default value:

-0.5e30

B.1.41 dparam.mio_disable_term_time

Corresponding constant:

dparam.mio_disable_term_time

Description:

The termination criteria governed by

- iparam.mio_max_num_relaxs
- iparam.mio_max_num_branches
- dparam.mio_near_tol_abs_gap
- dparam.mio_near_tol_rel_gap

is disabled the first n seconds. This parameter specifies the number n. A negative value is identical to infinity i.e. the termination criteria are never checked.

Possible Values:

Any number between -inf and +inf.

Default value:

-1.0

See also:

- iparam.mio_max_num_relaxs Maximum number of relaxations in branch and bound search.
- iparam.mio_max_num_branches Maximum number of branches allowed during the branch and bound search.
- dparam.mio_near_tol_abs_gap Relaxed absolute optimality tolerance employed by the mixed-integer optimizer.
- dparam.mio_near_tol_rel_gap The mixed-integer optimizer is terminated when this tolerance is satisfied.

B.1.42 dparam.mio_heuristic_time

Corresponding constant:

dparam.mio_heuristic_time

Description:

Minimum amount of time to be used in the heuristic search for a good feasible integer solution. A negative values implies that the optimizer decides the amount of time to be spent in the heuristic.

Possible Values:

Any number between -inf and +inf.

Default value:

`-1.0`

B.1.43 dparam.mio_max_time

Corresponding constant:

dparam.mio_max_time

Description:

This parameter limits the maximum time spent by the mixed-integer optimizer. A negative number means infinity.

Possible Values:

Any number between -inf and +inf.

Default value:

`-1.0`

B.1.44 dparam.mio_max_time_aprx_opt

Corresponding constant:

dparam.mio_max_time_aprx_opt

Description:

Number of seconds spent by the mixed-integer optimizer before the `dparam.mio_tol_rel_relax_int` is applied.

Possible Values:

Any number between 0.0 and +inf.

Default value:

`60`

B.1.45 dparam.mio_near_tol_abs_gap

Corresponding constant:

dparam.mio_near_tol_abs_gap

Description:

Relaxed absolute optimality tolerance employed by the mixed-integer optimizer. This termination criteria is delayed. See `dparam.mio_disable_term_time` for details.

Possible Values:

Any number between 0.0 and +inf.

Default value:

`0.0`

See also:

- `dparam.mio_disable_term_time` Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

B.1.46 dparam.mio_near_tol_rel_gap

Corresponding constant:

dparam.mio_near_tol_rel_gap

Description:

The mixed-integer optimizer is terminated when this tolerance is satisfied. This termination criteria is delayed. See `dparam.mio_disable_term_time` for details.

Possible Values:

Any number between 0.0 and +inf.

Default value:

`1.0e-3`

See also:

- `dparam.mio_disable_term_time` Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

B.1.47 dparam.mio_rel_add_cut_limited

Corresponding constant:

dparam.mio_rel_add_cut_limited

Description:

Controls how many cuts the mixed-integer optimizer is allowed to add to the problem. Let α be the value of this parameter and m the number constraints, then mixed-integer optimizer is allowed to αm cuts.

Possible Values:

Any number between 0.0 and 2.0.

Default value:

`0.75`

B.1.48 dparam.mio_rel_gap_const

Corresponding constant:

dparam.mio_rel_gap_const

Description:

This value is used to compute the relative gap for the solution to an integer optimization problem.

Possible Values:

Any number between 1.0e-15 and +inf.

Default value:

`1.0e-10`

B.1.49 dparam.mio_tol_abs_gap

Corresponding constant:

dparam.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`

B.1.50 dparam.mio_tol_abs_relax_int

Corresponding constant:

dparam.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 1e-9 and +inf.

Default value:

`1.0e-5`

B.1.51 dparam.mio_tol_feas

Corresponding constant:

dparam.mio_tol_feas

Description:

Feasibility tolerance for mixed integer solver. Any solution with maximum infeasibility below this value will be considered feasible.

Possible Values:

Any number between 0.0 and +inf.

Default value:

`1.0e-7`

B.1.52 dparam.mio_tol_max_cut_frac_rhs

Corresponding constant:

dparam.mio_tol_max_cut_frac_rhs

Description:

Maximum value of fractional part of right hand side to generate CMIR and CG cuts for. A value of 0.0 means that the value is selected automatically.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

`0.0`

B.1.53 dparam.mio_tol_min_cut_frac_rhs

Corresponding constant:

dparam.mio_tol_min_cut_frac_rhs

Description:

Minimum value of fractional part of right hand side to generate CMIR and CG cuts for. A value of 0.0 means that the value is selected automatically.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

`0.0`

B.1.54 dparam.mio_tol_rel_dual_bound_improvement

Corresponding constant:

dparam.mio_tol_rel_dual_bound_improvement

Description:

If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation. A value of 0.0 means that the value is selected automatically.

Possible Values:

Any number between 0.0 and 1.0.

Default value:

`0.0`

B.1.55 dparam.mio_tol_rel_gap

Corresponding constant:

dparam.mio_tol_rel_gap

Description:

Relative optimality tolerance employed by the mixed-integer optimizer.

Possible Values:

Any number between 0.0 and +inf.

Default value:

`1.0e-4`

B.1.56 dparam.mio_tol_rel_relax_int

Corresponding constant:

dparam.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`

B.1.57 dparam.mio_tol_x

Corresponding constant:

dparam.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`

B.1.58 dparam.nonconvex_tol_feas

Corresponding constant:

dparam.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`

B.1.59 dparam.nonconvex_tol_opt

Corresponding constant:

dparam.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`

B.1.60 dparam.optimizer_max_time

Corresponding constant:

dparam.optimizer_max_time

Description:

Maximum amount of time the optimizer is allowed to spent on the optimization. A negative number means infinity.

Possible Values:

Any number between -inf and +inf.

Default value:

`-1.0`

B.1.61 dparam.presolve_tol_abs_lindep

Corresponding constant:

dparam.presolve_tol_abs_lindep

Description:

Absolute tolerance employed by the linear dependency checker.

Possible Values:

Any number between 0.0 and +inf.

Default value:

`1.0e-6`

B.1.62 dparam.presolve_tol_aij

Corresponding constant:

dparam.presolve_tol_aij

Description:

Absolute zero tolerance employed for a_{ij} in the presolve.

Possible Values:

Any number between 1.0e-15 and +inf.

Default value:

`1.0e-12`

B.1.63 dparam.presolve_tol_rel_lindep

Corresponding constant:

dparam.presolve_tol_rel_lindep

Description:

Relative tolerance employed by the linear dependency checker.

Possible Values:

Any number between 0.0 and +inf.

Default value:

`1.0e-10`

B.1.64 dparam.presolve_tol_s

Corresponding constant:

dparam.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`

B.1.65 dparam.presolve_tol_x

Corresponding constant:

dparam.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`

B.1.66 dparam.qcqo_reformulate_rel_drop_tol

Corresponding constant:

dparam.qcqo_reformulate_rel_drop_tol

Description:

This parameter determines when columns are dropped in incomplete cholesky factorization doing reformulation of quadratic problems.

Possible Values:

Any number between 0 and +inf.

Default value:

`1e-15`

B.1.67 dparam.sim_lu_tol_rel_piv

Corresponding constant:

dparam.sim_lu_tol_rel_piv

Description:

Relative pivot tolerance employed when computing the LU factorization of the basis in the simplex optimizers and in the basis identification procedure.

A value closer to 1.0 generally improves numerical stability but typically also implies an increase in the computational work.

Possible Values:

Any number between 1.0e-6 and 0.999999.

Default value:

`0.01`

B.1.68 dparam.simplex_abs_tol_piv

Corresponding constant:

dparam.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`

B.1.69 dparam.upper_obj_cut

Corresponding constant:

dparam.upper_obj_cut

Description:

If either a primal or dual feasible solution is found proving that the optimal objective value is outside, [`dparam.lower_obj_cut`, `dparam.upper_obj_cut`], then MOSEK is terminated.

Possible Values:

Any number between -inf and +inf.

Default value:

`1.0e30`

See also:

- `dparam.upper_obj_cut_finite_trh` Objective bound.

B.1.70 dparam.upper_obj_cut_finite_trh

Corresponding constant:

dparam.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 `dparam.upper_obj_cut` is treated as ∞.

Possible Values:

Any number between -inf and +inf.

Default value:

`0.5e30`

B.2 iparam: Integer parameters

B.2.1 iparam.alloc_add_qnz

Corresponding constant:

iparam.alloc_add_qnz

Description:

Additional number of Q non-zeros that are allocated space for when `numanz` exceeds `maxnumqnz` during addition of new Q entries.

Possible Values:

Any number between 0 and +inf.

Default value:

`5000`

B.2.2 iparam.ana_sol_basis

Corresponding constant:

iparam.ana_sol_basis

Description:

Controls whether the basis matrix is analyzed in solaution analyzer.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.3 iparam.ana_sol_print_violated

Corresponding constant:

iparam.ana_sol_print_violated

Description:

Controls whether a list of violated constraints is printed when calling `Task.analyzesolution`. All constraints violated by more than the value set by the parameter `dparam.ana_sol_infeas_tol` will be printed.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.off

B.2.4 iparam.auto_sort_a_before_opt

Corresponding constant:

iparam.auto_sort_a_before_opt

Description:

Controls whether the elements in each column of A are sorted before an optimization is performed. This is not required but makes the optimization more deterministic.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.off

B.2.5 iparam.auto_update_sol_info

Corresponding constant:

iparam.auto_update_sol_info

Description:

Controls whether the solution information items are automatically updated after an optimization is performed.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.off

B.2.6 iparam.basis_solve_use_plus_one

Corresponding constant:

iparam.basis_solve_use_plus_one

Description:

If a slack variable is in the basis, then the corresponding column in the basis is a unit vector with -1 in the right position. However, if this parameter is set to `onoffkey.on`, -1 is replaced by 1.

This has siginificance for the results returned by the `Task.solvewithbasis` function.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.7 iparam.bi_clean_optimizer

Corresponding constant:

iparam.bi_clean_optimizer

Description:

Controls which simplex optimizer is used in the clean-up phase.

Possible values:

- `optimizertype.concurrent` The optimizer for nonconvex nonlinear problems.
- `optimizertype.conic` The optimizer for problems having conic constraints.
- `optimizertype.dual_simplex` The dual simplex optimizer is used.
- `optimizertype.free` The optimizer is chosen automatically.
- `optimizertype.free_simplex` One of the simplex optimizers is used.
- `optimizertype.intpnt` The interior-point optimizer is used.
- `optimizertype.mixed_int` The mixed-integer optimizer.
- `optimizertype.mixed_int_conic` The mixed-integer optimizer for conic and linear problems.
- `optimizertype.network_primal_simplex` The network primal simplex optimizer is used. It is only applicable to pure network problems.
- `optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.
- `optimizertype.primal_dual_simplex` The primal dual simplex optimizer is used.
- `optimizertype.primal_simplex` The primal simplex optimizer is used.

Default value:

`optimizertype.free`

B.2.8 iparam.bi_ignore_max_iter

Corresponding constant:

iparam.bi_ignore_max_iter

Description:

If the parameter `iparam.intpnt_basis` has the value `basindtype.no_error` and the interior-point optimizer has terminated due to maximum number of iterations, then basis identification is performed if this parameter has the value `onoffkey.on`.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.9 iparam.bi_ignore_num_error

Corresponding constant:

iparam.bi_ignore_num_error

Description:

If the parameter `iparam.intpnt_basis` has the value `basindtype.no_error` and the interior-point optimizer has terminated due to a numerical problem, then basis identification is performed if this parameter has the value `onoffkey.on`.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.10 iparam.bi_max_iterations

Corresponding constant:

iparam.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`

B.2.11 iparam.cache_license

Corresponding constant:

iparam.cache_license

Description:

Specifies if the license is kept checked out for the lifetime of the mosek environment (on) or returned to the server immediately after the optimization (off).

Check-in and check-out of licenses have an overhead. Frequent communication with the license server should be avoided.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.12 iparam.check_convexity

Corresponding constant:

iparam.check_convexity

Description:

Specify the level of convexity check on quadratic problems

Possible values:

- `checkconvexitytype.full` Perform a full convexity check.
- `checkconvexitytype.none` No convexity check.
- `checkconvexitytype.simple` Perform simple and fast convexity check.

Default value:

`checkconvexitytype.full`

B.2.13 iparam.compress_statfile

Corresponding constant:

iparam.compress_statfile

Description:

Control compression of stat files.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.14 iparam.concurrent_num_optimizers

Corresponding constant:

iparam.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

B.2.15 iparam.concurrent_priority_dual_simplex

Corresponding constant:

iparam.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

B.2.16 iparam.concurrent_priority_free_simplex

Corresponding constant:

iparam.concurrent_priority_free_simplex

Description:

Priority of the free simplex optimizer when selecting solvers for concurrent optimization.

Possible Values:

Any number between 0 and +inf.

Default value:

3

B.2.17 iparam.concurrent_priority_intpnt

Corresponding constant:

iparam.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

B.2.18 iparam.concurrent_priority_primal_simplex

Corresponding constant:

iparam.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

B.2.19 iparam.feasrepair_optimize

Corresponding constant:

iparam.feasrepair_optimize

Description:

Controls which type of feasibility analysis is to be performed.

Possible values:

- `feasrepairtype.optimize_combined` Minimize with original objective subject to minimal weighted violation of bounds.
- `feasrepairtype.optimize_none` Do not optimize the feasibility repair problem.
- `feasrepairtype.optimize_penalty` Minimize weighted sum of violations.

Default value:

`feasrepairtype.optimize_none`

B.2.20 iparam.infeas_generic_names

Corresponding constant:

iparam.infeas_generic_names

Description:

Controls whether generic names are used when an infeasible subproblem is created.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.21 iparam.infeas_prefer_primal

Corresponding constant:

iparam.infeas_prefer_primal

Description:

If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.22 iparam.infeas_report_auto

Corresponding constant:

iparam.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:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.23 iparam.infeas_report_level

Corresponding constant:

iparam.infeas_report_level

Description:

Controls the amount of information presented in an infeasibility report. Higher values imply more information.

Possible Values:

Any number between 0 and +inf.

Default value:

1

B.2.24 iparam.intpnt_basis

Corresponding constant:

iparam.intpnt_basis

Description:

Controls whether the interior-point optimizer also computes an optimal basis.

Possible values:

- `basindtype.always` Basis identification is always performed even if the interior-point optimizer terminates abnormally.
- `basindtype.if_feasible` Basis identification is not performed if the interior-point optimizer terminates with a problem status saying that the problem is primal or dual infeasible.
- `basindtype.never` Never do basis identification.
- `basindtype.no_error` Basis identification is performed if the interior-point optimizer terminates without an error.
- `basindtype.reservered` Not currently in use.

Default value:

`basindtype.always`

See also:

- `iparam.bi_ignore_max_iter` Turns on basis identification in case the interior-point optimizer is terminated due to maximum number of iterations.
- `iparam.bi_ignore_num_error` Turns on basis identification in case the interior-point optimizer is terminated due to a numerical problem.

B.2.25 iparam.intpnt_diff_step

Corresponding constant:

iparam.intpnt_diff_step

Description:

Controls whether different step sizes are allowed in the primal and dual space.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.26 iparam.intpnt_factor_debug_lvl

Corresponding constant:

iparam.intpnt_factor_debug_lvl

Description:

Controls factorization debug level.

Possible Values:

Any number between 0 and +inf.

Default value:

`0`

B.2.27 iparam.intpnt_factor_method

Corresponding constant:

iparam.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`

B.2.28 iparam.intpnt_hotstart

Corresponding constant:

iparam.intpnt_hotstart

Description:

Currently not in use.

Possible values:

- `intpnthotstart.dual` The interior-point optimizer exploits the dual solution only.
- `intpnthotstart.none` The interior-point optimizer performs a coldstart.
- `intpnthotstart.primal` The interior-point optimizer exploits the primal solution only.
- `intpnthotstart.primal_dual` The interior-point optimizer exploits both the primal and dual solution.

Default value:

`intpnthotstart.none`

B.2.29 iparam.intpnt_max_iterations

Corresponding constant:

iparam.intpnt_max_iterations

Description:

Controls the maximum number of iterations allowed in the interior-point optimizer.

Possible Values:

Any number between 0 and +inf.

Default value:

`400`

B.2.30 iparam.intpnt_max_num_cor

Corresponding constant:

iparam.intpnt_max_num_cor

Description:

Controls the maximum number of correctors allowed by the multiple corrector procedure. A negative value means that MOSEK is making the choice.

Possible Values:

Any number between -1 and +inf.

Default value:

`-1`

B.2.31 iparam.intpnt_max_num_refinement_steps

Corresponding constant:

iparam.intpnt_max_num_refinement_steps

Description:

Maximum number of steps to be used by the iterative refinement of the search direction. A negative value implies that the optimizer Chooses the maximum number of iterative refinement steps.

Possible Values:

Any number between -inf and +inf.

Default value:

`-1`

B.2.32 iparam.intpnt_off_col_trh

Corresponding constant:

iparam.intpnt_off_col_trh

Description:

Controls how many offending columns are detected in the Jacobian of the constraint matrix.

1 means aggressive detection, higher values mean less aggressive detection.

0 means no detection.

Possible Values:

Any number between 0 and +inf.

Default value:

`40`

B.2.33 iparam.intpnt_order_method

Corresponding constant:

iparam.intpnt_order_method

Description:

Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.

Possible values:

- `orderingtype.appminloc` Approximate minimum local fill-in ordering is employed.
- `orderingtype.experimental` This option should not be used.
- `orderingtype.force_graphpar` Always use the graph partitioning based ordering even if it is worse that the approximate minimum local fill ordering.
- `orderingtype.free` The ordering method is chosen automatically.
- `orderingtype.none` No ordering is used.
- `orderingtype.try_graphpar` Always try the the graph partitioning based ordering.

Default value:

`orderingtype.free`

B.2.34 iparam.intpnt_regularization_use

Corresponding constant:

iparam.intpnt_regularization_use

Description:

Controls whether regularization is allowed.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.35 iparam.intpnt_scaling

Corresponding constant:

iparam.intpnt_scaling

Description:

Controls how the problem is scaled before the interior-point optimizer is used.

Possible values:

- `scalingtype.aggressive` A very aggressive scaling is performed.
- `scalingtype.free` The optimizer chooses the scaling heuristic.
- `scalingtype.moderate` A conservative scaling is performed.
- `scalingtype.none` No scaling is performed.

Default value:

`scalingtype.free`

B.2.36 iparam.intpnt_solve_form

Corresponding constant:

iparam.intpnt_solve_form

Description:

Controls whether the primal or the dual problem is solved.

Possible values:

- `solveform.dual` The optimizer should solve the dual problem.
- `solveform.free` The optimizer is free to solve either the primal or the dual problem.
- `solveform.primal` The optimizer should solve the primal problem.

Default value:

`solveform.free`

B.2.37 iparam.intpnt_starting_point

Corresponding constant:

iparam.intpnt_starting_point

Description:

Starting point used by the interior-point optimizer.

Possible values:

- `startpointtype.constant` The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.
- `startpointtype.free` The starting point is chosen automatically.
- `startpointtype.guess` The optimizer guesses a starting point.
- `startpointtype.satisfy_bounds` The starting point is choosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

Default value:

`startpointtype.free`

B.2.38 iparam.lic_trh_expiry_wrn

Corresponding constant:

iparam.lic_trh_expiry_wrn

Description:

If a license feature expires in a numbers days less than the value of this parameter then a warning will be issued.

Possible Values:

Any number between 0 and +inf.

Default value:

7

B.2.39 iparam.license_debug

Corresponding constant:

iparam.license_debug

Description:

This option is used to turn on debugging of the incense manager.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.off

B.2.40 iparam.license_pause_time

Corresponding constant:

iparam.license_pause_time

Description:

If iparam.license_wait=onoffkey.on and no license is available, then MOSEK sleeps a number of milliseconds between each check of whether a license has become free.

Possible Values:

Any number between 0 and 1000000.

Default value:

100

B.2.41 iparam.license_suppress_expire_wrns

Corresponding constant:

iparam.license_suppress_expire_wrns

Description:

Controls whether license features expire warnings are suppressed.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.off

B.2.42 iparam.license_wait

Corresponding constant:

iparam.license_wait

Description:

If all licenses are in use MOSEK returns with an error code. However, by turning on this parameter MOSEK will wait for an available license.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.43 iparam.log

Corresponding constant:

iparam.log

Description:

Controls the amount of log information. The value 0 implies that all log information is suppressed. A higher level implies that more information is logged.

Please note that if a task is employed to solve a sequence of optimization problems the value of this parameter is reduced by the value of `iparam.log_cut_second_opt` for the second and any subsequent optimizations.

Possible Values:

Any number between 0 and +inf.

Default value:

`10`

See also:

- `iparam.log_cut_second_opt` Controls the reduction in the log levels for the second and any subsequent optimizations.

B.2.44 iparam.log_bi

Corresponding constant:

iparam.log_bi

Description:

Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and +inf.

Default value:

4

B.2.45 iparam.log_bi_freq

Corresponding constant:

iparam.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

B.2.46 iparam.log_check_convexity

Corresponding constant:

iparam.log_check_convexity

Description:

Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on.

If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

Possible Values:

Any number between 0 and +inf.

Default value:

`0`

B.2.47 iparam.log_concurrent

Corresponding constant:

iparam.log_concurrent

Description:

Controls amount of output printed by the concurrent optimizer.

Possible Values:

Any number between 0 and +inf.

Default value:

`1`

B.2.48 iparam.log_cut_second_opt

Corresponding constant:

iparam.log_cut_second_opt

Description:

If a task is employed to solve a sequence of optimization problems, then the value of the log levels is reduced by the value of this parameter. E.g `iparam.log` and `iparam.log_sim` are reduced by the value of this parameter for the second and any subsequent optimizations.

Possible Values:

Any number between 0 and +inf.

Default value:

`1`

See also:

- `iparam.log` Controls the amount of log information.
- `iparam.log_intpnt` Controls the amount of log information from the interior-point optimizers.
- `iparam.log_mio` Controls the amount of log information from the mixed-integer optimizers.
- `iparam.log_sim` Controls the amount of log information from the simplex optimizers.

B.2.49 iparam.log_expand

Corresponding constant:

iparam.log_expand

Description:

Controls the amount of logging when a data item such as the maximum number constrains is expanded.

Possible Values:

Any number between 0 and +inf.

Default value:

`0`

B.2.50 iparam.log_factor

Corresponding constant:

iparam.log_factor

Description:

If turned on, then the factor log lines are added to the log.

Possible Values:

Any number between 0 and +inf.

Default value:

`1`

B.2.51 iparam.log_feas_repair

Corresponding constant:

iparam.log_feas_repair

Description:

Controls the amount of output printed when performing feasibility repair. A value higher than one means extensive logging.

Possible Values:

Any number between 0 and +inf.

Default value:

`1`

B.2.52 iparam.log_file

Corresponding constant:

iparam.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

B.2.53 iparam.log_head

Corresponding constant:

iparam.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

B.2.54 iparam.log_infeas_ana

Corresponding constant:

iparam.log_infeas_ana

Description:

Controls amount of output printed by the infeasibility analyzer procedures. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and +inf.

Default value:

1

B.2.55 iparam.log_intpnt

Corresponding constant:

iparam.log_intpnt

Description:

Controls amount of output printed printed by the interior-point optimizer. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and +inf.

Default value:

4

B.2.56 iparam.log_mio

Corresponding constant:

iparam.log_mio

Description:

Controls the log level for the mixed-integer optimizer. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and +inf.

Default value:

4

B.2.57 iparam.log_mio_freq

Corresponding constant:

iparam.log_mio_freq

Description:

Controls how frequent the mixed-integer optimizer prints the log line. It will print line every time `iparam.log_mio_freq` relaxations have been solved.

Possible Values:

A integer value.

Default value:

1000

B.2.58 iparam.log_nonconvex

Corresponding constant:

iparam.log_nonconvex

Description:

Controls amount of output printed by the nonconvex optimizer.

Possible Values:

Any number between 0 and +inf.

Default value:

`1`

B.2.59 iparam.log_optimizer

Corresponding constant:

iparam.log_optimizer

Description:

Controls the amount of general optimizer information that is logged.

Possible Values:

Any number between 0 and +inf.

Default value:

`1`

B.2.60 iparam.log_order

Corresponding constant:

iparam.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`

B.2.61 iparam.log_param

Corresponding constant:

iparam.log_param

Description:

Controls the amount of information printed out about parameter changes.

Possible Values:

Any number between 0 and +inf.

Default value:

0

B.2.62 iparam.log_presolve

Corresponding constant:

iparam.log_presolve

Description:

Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and +inf.

Default value:

1

B.2.63 iparam.log_response

Corresponding constant:

iparam.log_response

Description:

Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and +inf.

Default value:

0

B.2.64 iparam.log_sensitivity

Corresponding constant:

iparam.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`

B.2.65 iparam.log_sensitivity_opt

Corresponding constant:

iparam.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`

B.2.66 iparam.log_sim

Corresponding constant:

iparam.log_sim

Description:

Controls amount of output printed by the simplex optimizer. A higher level implies that more information is logged.

Possible Values:

Any number between 0 and +inf.

Default value:

`4`

B.2.67 iparam.log_sim_freq

Corresponding constant:

iparam.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:

`1000`

B.2.68 iparam.log_sim_minor

Corresponding constant:

iparam.log_sim_minor

Description:

Currently not in use.

Possible Values:

Any number between 0 and +inf.

Default value:

`1`

B.2.69 iparam.log_sim_network_freq

Corresponding constant:

iparam.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 `iparam.log_sim_freq` times `iparam.log_sim_network_freq`.

Possible Values:

Any number between 0 and +inf.

Default value:

`1000`

B.2.70 iparam.log_storage

Corresponding constant:

iparam.log_storage

Description:

When turned on, MOSEK prints messages regarding the storage usage and allocation.

Possible Values:

Any number between 0 and +inf.

Default value:

`0`

B.2.71 iparam.max_num_warnings

Corresponding constant:

iparam.max_num_warnings

Description:

A negtive number means all warnings are logged. Otherwise the parameter specifies the maximum number times each warning is logged.

Possible Values:

Any number between -inf and +inf.

Default value:

`6`

B.2.72 iparam.mio_branch_dir

Corresponding constant:

iparam.mio_branch_dir

Description:

Controls whether the mixed-integer optimizer is branching up or down by default.

Possible values:

- `branchdir.down` The mixed-integer optimizer always chooses the down branch first.
- `branchdir.free` The mixed-integer optimizer decides which branch to choose.
- `branchdir.up` The mixed-integer optimizer always chooses the up branch first.

Default value:

`branchdir.free`

B.2.73 iparam.mio_branch_priorities_use

Corresponding constant:

iparam.mio_branch_priorities_use

Description:

Controls whether branching priorities are used by the mixed-integer optimizer.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.74 iparam.mio_construct_sol

Corresponding constant:

iparam.mio_construct_sol

Description:

If set to `onoffkey.on` and all integer variables have been given a value for which a feasible mixed integer solution exists, then MOSEK generates an initial solution to the mixed integer problem by fixing all integer values and solving the remaining problem.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.75 iparam.mio_cont_sol

Corresponding constant:

iparam.mio_cont_sol

Description:

Controls the meaning of the interior-point and basic solutions in mixed integer problems.

Possible values:

- miocontsoltype.itg The reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. A solution is only reported in case the problem has a primal feasible solution.
- miocontsoltype.itg_rel In case the problem is primal feasible then the reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. If the problem is primal infeasible, then the solution to the root node problem is reported.
- miocontsoltype.none No interior-point or basic solution are reported when the mixed-integer optimizer is used.
- miocontsoltype.root The reported interior-point and basic solutions are a solution to the root node problem when mixed-integer optimizer is used.

Default value:

miocontsoltype.none

B.2.76 iparam.mio_cut_cg

Corresponding constant:

iparam.mio_cut_cg

Description:

Controls whether CG cuts should be generated.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.on

B.2.77 iparam.mio_cut_cmir

Corresponding constant:

iparam.mio_cut_cmir

Description:

Controls whether mixed integer rounding cuts should be generated.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.on

B.2.78 iparam.mio_cut_level_root

Corresponding constant:

iparam.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`

B.2.79 iparam.mio_cut_level_tree

Corresponding constant:

iparam.mio_cut_level_tree

Description:

Controls the cut level employed by the mixed-integer optimizer at the tree. See `iparam.mio_cut_level_root` for an explanation of the parameter values.

Possible Values:

Any value.

Default value:

`-1`

B.2.80 iparam.mio_feaspump_level

Corresponding constant:

iparam.mio_feaspump_level

Description:

Feasibility pump is a heuristic designed to compute an initial feasible solution. A value of 0 implies that the feasibility pump heuristic is not used. A value of -1 implies that the mixed-integer optimizer decides how the feasibility pump heuristic is used. A larger value than 1 implies that the feasibility pump is employed more aggressively. Normally a value beyond 3 is not worthwhile.

Possible Values:

Any number between -inf and 3.

Default value:

`-1`

B.2.81 iparam.mio_heuristic_level

Corresponding constant:

iparam.mio_heuristic_level

Description:

Controls the heuristic employed by the mixed-integer optimizer to locate an initial good integer feasible solution. A value of zero means the heuristic is not used at all. A larger value than 0 means that a gradually more sophisticated heuristic is used which is computationally more expensive. A negative value implies that the optimizer chooses the heuristic. Normally a value around 3 to 5 should be optimal.

Possible Values:

Any value.

Default value:

`-1`

B.2.82 iparam.mio_hotstart

Corresponding constant:

iparam.mio_hotstart

Description:

Controls whether the integer optimizer is hot-started.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.83 iparam.mio_keep_basis

Corresponding constant:

iparam.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:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.84 iparam.mio_local_branch_number

Corresponding constant:

iparam.mio_local_branch_number

Description:

Controls the size of the local search space when doing local branching.

Possible Values:

Any number between -inf and +inf.

Default value:

`-1`

B.2.85 iparam.mio_max_num_branches

Corresponding constant:

iparam.mio_max_num_branches

Description:

Maximum number of branches allowed during the branch and bound search. A negative value means infinite.

Possible Values:

Any number between -inf and +inf.

Default value:

`-1`

See also:

- `dparam.mio_disable_term_time` Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

B.2.86 iparam.mio_max_num_relaxs

Corresponding constant:

iparam.mio_max_num_relaxs

Description:

Maximum number of relaxations allowed during the branch and bound search. A negative value means infinite.

Possible Values:

Any number between -inf and +inf.

Default value:

`-1`

See also:

- `dparam.mio_disable_term_time` Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

B.2.87 iparam.mio_max_num_solutions

Corresponding constant:

iparam.mio_max_num_solutions

Description:

The mixed-integer optimizer can be terminated after a certain number of different feasible solutions has been located. If this parameter has the value n and n is strictly positive, then the mixed-integer optimizer will be terminated when n feasible solutions have been located.

Possible Values:

Any number between -inf and +inf.

Default value:

-1

See also:

- dparam.mio_disable_term_time Certain termination criteria is disabled within the mixed-integer optimizer for period time specified by the parameter.

B.2.88 iparam.mio_mode

Corresponding constant:

iparam.mio_mode

Description:

Controls whether the optimizer includes the integer restrictions when solving a (mixed) integer optimization problem.

Possible values:

- miomode.ignored The integer constraints are ignored and the problem is solved as a continuous problem.
- miomode.lazy Integer restrictions should be satisfied if an optimizer is available for the problem.
- miomode.satisfied Integer restrictions should be satisfied.

Default value:

miomode.satisfied

B.2.89 iparam.mio_mt_user_cb

Corresponding constant:

iparam.mio_mt_user_cb

Description:

It true user callbacks are called from each thread used by this optimizer. If false the user callback is only called from a single thread.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.on

B.2.90 iparam.mio_node_optimizer

Corresponding constant:

iparam.mio_node_optimizer

Description:

Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.

Possible values:

- `optimizertype.concurrent` The optimizer for nonconvex nonlinear problems.
- `optimizertype.conic` The optimizer for problems having conic constraints.
- `optimizertype.dual_simplex` The dual simplex optimizer is used.
- `optimizertype.free` The optimizer is chosen automatically.
- `optimizertype.free_simplex` One of the simplex optimizers is used.
- `optimizertype.intpnt` The interior-point optimizer is used.
- `optimizertype.mixed_int` The mixed-integer optimizer.
- `optimizertype.mixed_int_conic` The mixed-integer optimizer for conic and linear problems.
- `optimizertype.network_primal_simplex` The network primal simplex optimizer is used. It is only applicable to pure network problems.
- `optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.
- `optimizertype.primal_dual_simplex` The primal dual simplex optimizer is used.
- `optimizertype.primal_simplex` The primal simplex optimizer is used.

Default value:

`optimizertype.free`

B.2.91 iparam.mio_node_selection

Corresponding constant:

iparam.mio_node_selection

Description:

Controls the node selection strategy employed by the mixed-integer optimizer.

Possible values:

- `mionodeseltype.best` The optimizer employs a best bound node selection strategy.
- `mionodeseltype.first` The optimizer employs a depth first node selection strategy.
- `mionodeseltype.free` The optimizer decides the node selection strategy.
- `mionodeseltype.hybrid` The optimizer employs a hybrid strategy.

- `mionodeseltype.pseudo` The optimizer employs selects the node based on a pseudo cost estimate.
- `mionodeseltype.worst` The optimizer employs a worst bound node selection strategy.

Default value:

`mionodeseltype.free`

B.2.92 iparam.mio_optimizer_mode

Corresponding constant:

iparam.mio_optimizer_mode

Description:

An exprimental feature.

Possible Values:

Any number between 0 and 1.

Default value:

`0`

B.2.93 iparam.mio_presolve_aggregate

Corresponding constant:

iparam.mio_presolve_aggregate

Description:

Controls whether the presolve used by the mixed-integer optimizer tries to aggregate the constraints.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.94 iparam.mio_presolve_probing

Corresponding constant:

iparam.mio_presolve_probing

Description:

Controls whether the mixed-integer presolve performs probing. Probing can be very time consuming.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.95 iparam.mio_presolve_use

Corresponding constant:

iparam.mio_presolve_use

Description:

Controls whether presolve is performed by the mixed-integer optimizer.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.96 iparam.mio_probing_level

Corresponding constant:

iparam.mio_probing_level

Description:

Controls the amount of probing employed by the mixed-integer optimizer in presolve.

- -1 The optimizer chooses the level of probing employed.
- 0 Probing is disabled.
- 1 A low amount of probing is employed.

- 2 A medium amount of probing is employed.
- 3 A high amount of probing is employed.

Possible Values:

An integer value in the range of -1 to 3.

Default value:

`-1`

B.2.97 iparam.mio_rins_max_nodes

Corresponding constant:

iparam.mio_rins_max_nodes

Description:

Controls the maximum number of nodes allowed in each call to the RINS heuristic. The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Possible Values:

Any number between -1 and +inf.

Default value:

`-1`

B.2.98 iparam.mio_root_optimizer

Corresponding constant:

iparam.mio_root_optimizer

Description:

Controls which optimizer is employed at the root node in the mixed-integer optimizer.

Possible values:

- `optimizertype.concurrent` The optimizer for nonconvex nonlinear problems.
- `optimizertype.conic` The optimizer for problems having conic constraints.
- `optimizertype.dual_simplex` The dual simplex optimizer is used.
- `optimizertype.free` The optimizer is chosen automatically.
- `optimizertype.free_simplex` One of the simplex optimizers is used.
- `optimizertype.intpnt` The interior-point optimizer is used.
- `optimizertype.mixed_int` The mixed-integer optimizer.

- optimizertype.mixed_int_conic The mixed-integer optimizer for conic and linear problems.
- optimizertype.network_primal_simplex The network primal simplex optimizer is used. It is only applicable to pure network problems.
- optimizertype.nonconvex The optimizer for nonconvex nonlinear problems.
- optimizertype.primal_dual_simplex The primal dual simplex optimizer is used.
- optimizertype.primal_simplex The primal simplex optimizer is used.

Default value:

optimizertype.free

B.2.99 iparam.mio_strong_branch

Corresponding constant:

iparam.mio_strong_branch

Description:

The value specifies the depth from the root in which strong branching is used. A negative value means that the optimizer chooses a default value automatically.

Possible Values:

Any number between -inf and +inf.

Default value:

-1

B.2.100 iparam.mio_use_multithreaded_optimizer

Corresponding constant:

iparam.mio_use_multithreaded_optimizer

Description:

Controls wheter the new multithreaded optimizer should be used for Mixed integer problems.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.off

B.2.101 iparam.mt_spincount

Corresponding constant:

iparam.mt_spincount

Description:

Set the number of iterations to spin before sleeping.

Possible Values:

Any integer greater or equal to 0.

Default value:

`0`

B.2.102 iparam.nonconvex_max_iterations

Corresponding constant:

iparam.nonconvex_max_iterations

Description:

Maximum number of iterations that can be used by the nonconvex optimizer.

Possible Values:

Any number between 0 and +inf.

Default value:

`100000`

B.2.103 iparam.num_threads

Corresponding constant:

iparam.num_threads

Description:

Controls the number of threads employed by the optimizer. If set to 0 the number of threads used will be equal to the number of cores detected on the machine.

Possible Values:

Any integer greater or equal to 0.

Default value:

`0`

B.2.104 iparam.opf_max_terms_per_line

Corresponding constant:

iparam.opf_max_terms_per_line

Description:

The maximum number of terms (linear and quadratic) per line when an OPF file is written.

Possible Values:

Any number between 0 and +inf.

Default value:

5

B.2.105 iparam.opf_write_header

Corresponding constant:

iparam.opf_write_header

Description:

Write a text header with date and MOSEK version in an OPF file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.106 iparam.opf_write_hints

Corresponding constant:

iparam.opf_write_hints

Description:

Write a hint section with problem dimensions in the beginning of an OPF file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.107 iparam.opf_write_parameters

Corresponding constant:

iparam.opf_write_parameters

Description:

Write a parameter section in an OPF file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.108 iparam.opf_write_problem

Corresponding constant:

iparam.opf_write_problem

Description:

Write objective, constraints, bounds etc. to an OPF file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.109 iparam.opf_write_sol_bas

Corresponding constant:

iparam.opf_write_sol_bas

Description:

If `iparam.opf_write_solutions` is `onoffkey.on` and a basic solution is defined, include the basic solution in OPF files.

Possible values:

- `onoffkey.off` Switch the option off.

- onoffkey.on Switch the option on.

Default value:

onoffkey.on

B.2.110 iparam.opf_write_sol_itg

Corresponding constant:

iparam.opf_write_sol_itg

Description:

If iparam.opf_write_solutions is onoffkey.on and an integer solution is defined, write the integer solution in OPF files.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.on

B.2.111 iparam.opf_write_sol_itr

Corresponding constant:

iparam.opf_write_sol_itr

Description:

If iparam.opf_write_solutions is onoffkey.on and an interior solution is defined, write the interior solution in OPF files.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.on

B.2.112 iparam.opf_write_solutions

Corresponding constant:

iparam.opf_write_solutions

Description:

Enable inclusion of solutions in the OPF files.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.113 iparam.optimizer

Corresponding constant:

iparam.optimizer

Description:

The paramter controls which optimizer is used to optimize the task.

Possible values:

- `optimizertype.concurrent` The optimizer for nonconvex nonlinear problems.
- `optimizertype.conic` The optimizer for problems having conic constraints.
- `optimizertype.dual_simplex` The dual simplex optimizer is used.
- `optimizertype.free` The optimizer is chosen automatically.
- `optimizertype.free_simplex` One of the simplex optimizers is used.
- `optimizertype.intpnt` The interior-point optimizer is used.
- `optimizertype.mixed_int` The mixed-integer optimizer.
- `optimizertype.mixed_int_conic` The mixed-integer optimizer for conic and linear problems.
- `optimizertype.network_primal_simplex` The network primal simplex optimizer is used. It is only applicable to pure network problems.
- `optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.
- `optimizertype.primal_dual_simplex` The primal dual simplex optimizer is used.
- `optimizertype.primal_simplex` The primal simplex optimizer is used.

Default value:

`optimizertype.free`

B.2.114 iparam.param_read_case_name

Corresponding constant:

iparam.param_read_case_name

Description:

If turned on, then names in the parameter file are case sensitive.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.115 iparam.param_read_ign_error

Corresponding constant:

iparam.param_read_ign_error

Description:

If turned on, then errors in paramter settings is ignored.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.116 iparam.presolve_elim_fill

Corresponding constant:

iparam.presolve_elim_fill

Description:

Controls the maximum amount of fill-in that can be created during the elimination phase of the presolve. This parameter times (`numcon`+`numvar`) denotes the amount of fill-in.

Possible Values:

Any number between 0 and +inf.

Default value:

`1`

B.2.117 iparam.presolve_eliminator_max_num_tries

Corresponding constant:

iparam.presolve_eliminator_max_num_tries

Description:

Control the maximum number of times the eliminator is tried.

Possible Values:

A negative value implies MOSEK decides maximum number of times.

Default value:

`-1`

B.2.118 iparam.presolve_eliminator_use

Corresponding constant:

iparam.presolve_eliminator_use

Description:

Controls whether free or implied free variables are eliminated from the problem.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.119 iparam.presolve_level

Corresponding constant:

iparam.presolve_level

Description:

Currently not used.

Possible Values:

Any number between -inf and +inf.

Default value:

`-1`

B.2.120 iparam.presolve_lindep_abs_work_trh

Corresponding constant:

iparam.presolve_lindep_abs_work_trh

Description:

The linear dependency check is potentially computationally expensive.

Possible Values:

Any number between 0 and +inf.

Default value:

`100`

B.2.121 iparam.presolve_lindep_rel_work_trh

Corresponding constant:

iparam.presolve_lindep_rel_work_trh

Description:

The linear dependency check is potentially computationally expensive.

Possible Values:

Any number between 0 and +inf.

Default value:

`100`

B.2.122 iparam.presolve_lindep_use

Corresponding constant:

iparam.presolve_lindep_use

Description:

Controls whether the linear constraints are checked for linear dependencies.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.123 iparam.presolve_max_num_reductions

Corresponding constant:

iparam.presolve_max_num_reductions

Description:

Controls the maximum number reductions performed by the presolve. The value of the parameter is normally only changed in connection with debugging. A negative value implies that an infinite number of reductions are allowed.

Possible Values:

Any number between -inf and +inf.

Default value:

`-1`

B.2.124 iparam.presolve_use

Corresponding constant:

iparam.presolve_use

Description:

Controls whether the presolve is applied to a problem before it is optimized.

Possible values:

- `presolvemode.free` It is decided automatically whether to presolve before the problem is optimized.
- `presolvemode.off` The problem is not presolved before it is optimized.
- `presolvemode.on` The problem is presolved before it is optimized.

Default value:

`presolvemode.free`

B.2.125 iparam.primal_repair_optimizer

Corresponding constant:

iparam.primal_repair_optimizer

Description:

Controls which optimizer that is used to find the optimal repair.

Possible values:

- `optimizertype.concurrent` The optimizer for nonconvex nonlinear problems.

- optimizertype.conic The optimizer for problems having conic constraints.
- optimizertype.dual_simplex The dual simplex optimizer is used.
- optimizertype.free The optimizer is chosen automatically.
- optimizertype.free_simplex One of the simplex optimizers is used.
- optimizertype.intpnt The interior-point optimizer is used.
- optimizertype.mixed_int The mixed-integer optimizer.
- optimizertype.mixed_int_conic The mixed-integer optimizer for conic and linear problems.
- optimizertype.network_primal_simplex The network primal simplex optimizer is used. It is only applicable to pure network problems.
- optimizertype.nonconvex The optimizer for nonconvex nonlinear problems.
- optimizertype.primal_dual_simplex The primal dual simplex optimizer is used.
- optimizertype.primal_simplex The primal simplex optimizer is used.

Default value:

optimizertype.free

B.2.126 iparam.qo_separable_reformulation

Corresponding constant:

iparam.qo_separable_reformulation

Description:

Determine if Quadratic programing problems should be reformulated to separable form.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.off

B.2.127 iparam.read_anz

Corresponding constant:

iparam.read_anz

Description:

Expected maximum number of A non-zeros to be read. The option is used only by fast MPS and LP file readers.

Possible Values:

Any number between 0 and +inf.

Default value:

100000

B.2.128 iparam.read_con

Corresponding constant:

iparam.read_con

Description:

Expected maximum number of constraints to be read. The option is only used by fast MPS and LP file readers.

Possible Values:

Any number between 0 and +inf.

Default value:

10000

B.2.129 iparam.read_cone

Corresponding constant:

iparam.read_cone

Description:

Expected maximum number of conic constraints to be read. The option is used only by fast MPS and LP file readers.

Possible Values:

Any number between 0 and +inf.

Default value:

2500

B.2.130 iparam.read_data_compressed

Corresponding constant:

iparam.read_data_compressed

Description:

If this option is turned on,it is assumed that the data file is compressed.

Possible values:

- compresstype.free The type of compression used is chosen automatically.
- compresstype.gzip The type of compression used is gzip compatible.
- compresstype.none No compression is used.

Default value:

compresstype.free

B.2.131 iparam.read_data_format

Corresponding constant:

iparam.read_data_format

Description:

Format of the data file to be read.

Possible values:

- dataformat.cb Conic benchmark format.
- dataformat.extension The file extension is used to determine the data file format.
- dataformat.free_mps The data data a free MPS formatted file.
- dataformat.lp The data file is LP formatted.
- dataformat.mps The data file is MPS formatted.
- dataformat.op The data file is an optimization problem formatted file.
- dataformat.task Generic task dump file.
- dataformat.xml The data file is an XML formatted file.

Default value:

dataformat.extension

B.2.132 iparam.read_debug

Corresponding constant:

iparam.read_debug

Description:

Turns on additional debugging information when reading files.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.off

B.2.133 iparam.read_keep_free_con

Corresponding constant:

iparam.read_keep_free_con

Description:

Controls whether the free constraints are included in the problem.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.134 iparam.read_lp_drop_new_vars_in_bou

Corresponding constant:

iparam.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:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.135 iparam.read_lp_quoted_names

Corresponding constant:

iparam.read_lp_quoted_names

Description:

If a name is in quotes when reading an LP file, the quotes will be removed.

Possible values:

- `onoffkey.off` Switch the option off.

- onoffkey.on Switch the option on.

Default value:

onoffkey.on

B.2.136 iparam.read_mps_format

Corresponding constant:

iparam.read_mps_format

Description:

Controls how strictly the MPS file reader interprets the MPS format.

Possible values:

- mpsformat.free It is assumed that the input file satisfies the free MPS format. This implies that spaces are not allowed in names. Otherwise the format is free.
- mpsformat.relaxed It is assumed that the input file satisfies a slightly relaxed version of the MPS format.
- mpsformat.strict It is assumed that the input file satisfies the MPS format strictly.

Default value:

mpsformat.relaxed

B.2.137 iparam.read_mps_keep_int

Corresponding constant:

iparam.read_mps_keep_int

Description:

Controls whether MOSEK should keep the integer restrictions on the variables while reading the MPS file.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.on

B.2.138 iparam.read_mps_obj_sense

Corresponding constant:

iparam.read_mps_obj_sense

Description:

If turned on, the MPS reader uses the objective sense section. Otherwise the MPS reader ignores it.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.139 iparam.read_mps_relax

Corresponding constant:

iparam.read_mps_relax

Description:

If this option is turned on, then mixed integer constraints are ignored when a problem is read.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.140 iparam.read_mps_width

Corresponding constant:

iparam.read_mps_width

Description:

Controls the maximal number of characters allowed in one line of the MPS file.

Possible Values:

Any positive number greater than 80.

Default value:

`1024`

B.2.141 iparam.read_qnz

Corresponding constant:

iparam.read_qnz

Description:

Expected maximum number of Q non-zeros to be read. The option is used only by MPS and LP file readers.

Possible Values:

Any number between 0 and +inf.

Default value:

`20000`

B.2.142 iparam.read_task_ignore_param

Corresponding constant:

iparam.read_task_ignore_param

Description:

Controls whether MOSEK should ignore the parameter setting defined in the task file and use the default parameter setting instead.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.143 iparam.read_var

Corresponding constant:

iparam.read_var

Description:

Expected maximum number of variable to be read. The option is used only by MPS and LP file readers.

Possible Values:

Any number between 0 and +inf.

Default value:

`10000`

B.2.144 iparam.sensitivity_all

Corresponding constant:

iparam.sensitivity_all

Description:

If set to `onoffkey.on`, then `Task.sensitivityreport` analyzes all bounds and variables instead of reading a specification from the file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.145 iparam.sensitivity_optimizer

Corresponding constant:

iparam.sensitivity_optimizer

Description:

Controls which optimizer is used for optimal partition sensitivity analysis.

Possible values:

- `optimizertype.concurrent` The optimizer for nonconvex nonlinear problems.
- `optimizertype.conic` The optimizer for problems having conic constraints.
- `optimizertype.dual_simplex` The dual simplex optimizer is used.
- `optimizertype.free` The optimizer is chosen automatically.
- `optimizertype.free_simplex` One of the simplex optimizers is used.
- `optimizertype.intpnt` The interior-point optimizer is used.
- `optimizertype.mixed_int` The mixed-integer optimizer.
- `optimizertype.mixed_int_conic` The mixed-integer optimizer for conic and linear problems.
- `optimizertype.network_primal_simplex` The network primal simplex optimizer is used. It is only applicable to pure network problems.
- `optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.
- `optimizertype.primal_dual_simplex` The primal dual simplex optimizer is used.
- `optimizertype.primal_simplex` The primal simplex optimizer is used.

Default value:

`optimizertype.free_simplex`

B.2.146 iparam.sensitivity_type

Corresponding constant:

iparam.sensitivity_type

Description:

Controls which type of sensitivity analysis is to be performed.

Possible values:

- `sensitivitytype.basis` Basis sensitivity analysis is performed.
- `sensitivitytype.optimal_partition` Optimal partition sensitivity analysis is performed.

Default value:

`sensitivitytype.basis`

B.2.147 iparam.sim_basis_factor_use

Corresponding constant:

iparam.sim_basis_factor_use

Description:

Controls whether a (LU) factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penanlty.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.148 iparam.sim_degen

Corresponding constant:

iparam.sim_degen

Description:

Controls how aggressively degeneration is handled.

Possible values:

- simdegen.aggressive The simplex optimizer should use an aggressive degeneration strategy.
- simdegen.free The simplex optimizer chooses the degeneration strategy.
- simdegen.minimum The simplex optimizer should use a minimum degeneration strategy.
- simdegen.moderate The simplex optimizer should use a moderate degeneration strategy.
- simdegen.none The simplex optimizer should use no degeneration strategy.

Default value:

simdegen.free

B.2.149 iparam.sim_dual_crash

Corresponding constant:

iparam.sim_dual_crash

Description:

Controls whether crashing is performed in the dual simplex optimizer.

In general if a basis consists of more than (100-this parameter value)% fixed variables, then a crash will be performed.

Possible Values:

Any number between 0 and +inf.

Default value:

90

B.2.150 iparam.sim_dual_phaseone_method

Corresponding constant:

iparam.sim_dual_phaseone_method

Description:

An exprimental feature.

Possible Values:

Any number between 0 and 10.

Default value:

0

B.2.151 iparam.sim_dual_restrict_selection

Corresponding constant:

iparam.sim_dual_restrict_selection

Description:

The dual simplex optimizer can use a so-called restricted selection/pricing strategy to chooses the outgoing variable. Hence, if restricted selection is applied, then the dual simplex optimizer first choose a subset of all the potential outgoing variables. Next, for some time it will choose the outgoing variable only among the subset. From time to time the subset is redefined.

A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Possible Values:

Any number between 0 and 100.

Default value:

`50`

B.2.152 iparam.sim_dual_selection

Corresponding constant:

iparam.sim_dual_selection

Description:

Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer.

Possible values:

- `simseltype.ase` The optimizer uses approximate steepest-edge pricing.
- `simseltype.devex` The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).
- `simseltype.free` The optimizer chooses the pricing strategy.
- `simseltype.full` The optimizer uses full pricing.
- `simseltype.partial` The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.
- `simseltype.se` The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

Default value:

`simseltype.free`

B.2.153 iparam.sim_exploit_dupvec

Corresponding constant:

iparam.sim_exploit_dupvec

Description:

Controls if the simplex optimizers are allowed to exploit duplicated columns.

Possible values:

- `simdupvec.free` The simplex optimizer can choose freely.
- `simdupvec.off` Disallow the simplex optimizer to exploit duplicated columns.
- `simdupvec.on` Allow the simplex optimizer to exploit duplicated columns.

Default value:

`simdupvec.off`

B.2.154 iparam.sim_hotstart

Corresponding constant:

iparam.sim_hotstart

Description:

Controls the type of hot-start that the simplex optimizer perform.

Possible values:

- `simhotstart.free` The simplex optimize chooses the hot-start type.
- `simhotstart.none` The simplex optimizer performs a coldstart.
- `simhotstart.status_keys` Only the status keys of the constraints and variables are used to choose the type of hot-start.

Default value:

`simhotstart.free`

B.2.155 iparam.sim_hotstart_lu

Corresponding constant:

iparam.sim_hotstart_lu

Description:

Determines if the simplex optimizer should exploit the initial factorization.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.156 iparam.sim_integer

Corresponding constant:

iparam.sim_integer

Description:

An exprimental feature.

Possible Values:

Any number between 0 and 10.

Default value:

`0`

B.2.157 iparam.sim_max_iterations

Corresponding constant:

iparam.sim_max_iterations

Description:

Maximum number of iterations that can be used by a simplex optimizer.

Possible Values:

Any number between 0 and +inf.

Default value:

`10000000`

B.2.158 iparam.sim_max_num_setbacks

Corresponding constant:

iparam.sim_max_num_setbacks

Description:

Controls how many set-backs are allowed within a simplex optimizer. A set-back is an event where the optimizer moves in the wrong direction. This is impossible in theory but may happen due to numerical problems.

Possible Values:

Any number between 0 and +inf.

Default value:

250

B.2.159 iparam.sim_non_singular

Corresponding constant:

iparam.sim_non_singular

Description:

Controls if the simplex optimizer ensures a non-singular basis, if possible.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.160 iparam.sim_primal_crash

Corresponding constant:

iparam.sim_primal_crash

Description:

Controls whether crashing is performed in the primal simplex optimizer.

In general, if a basis consists of more than (100-this parameter value)% fixed variables, then a crash will be performed.

Possible Values:

Any nonnegative integer value.

Default value:

90

B.2.161 iparam.sim_primal_phaseone_method

Corresponding constant:

iparam.sim_primal_phaseone_method

Description:

An exprimental feature.

Possible Values:

Any number between 0 and 10.

Default value:

0

B.2.162 iparam.sim_primal_restrict_selection

Corresponding constant:

iparam.sim_primal_restrict_selection

Description:

The primal simplex optimizer can use a so-called restricted selection/pricing strategy to chooses the outgoing variable. Hence, if restricted selection is applied, then the primal simplex optimizer first choose a subset of all the potential incoming variables. Next, for some time it will choose the incoming variable only among the subset. From time to time the subset is redefined.

A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Possible Values:

Any number between 0 and 100.

Default value:

50

B.2.163 iparam.sim_primal_selection

Corresponding constant:

iparam.sim_primal_selection

Description:

Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.

Possible values:

- simseltype.ase The optimizer uses approximate steepest-edge pricing.

- simseltype.devex The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).
- simseltype.free The optimizer chooses the pricing strategy.
- simseltype.full The optimizer uses full pricing.
- simseltype.partial The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.
- simseltype.se The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

Default value:

simseltype.free

B.2.164 iparam.sim_refactor_freq

Corresponding constant:

iparam.sim_refactor_freq

Description:

Controls how frequent the basis is refactorized. The value 0 means that the optimizer determines the best point of refactorization.

It is strongly recommended NOT to change this parameter.

Possible Values:

Any number between 0 and +inf.

Default value:

0

B.2.165 iparam.sim_reformulation

Corresponding constant:

iparam.sim_reformulation

Description:

Controls if the simplex optimizers are allowed to reformulate the problem.

Possible values:

- simreform.aggressive The simplex optimizer should use an aggressive reformulation strategy.
- simreform.free The simplex optimizer can choose freely.
- simreform.off Disallow the simplex optimizer to reformulate the problem.

- `simreform.on` Allow the simplex optimizer to reformulate the problem.

Default value:

`simreform.off`

B.2.166 iparam.sim_save_lu

Corresponding constant:

iparam.sim_save_lu

Description:

Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.167 iparam.sim_scaling

Corresponding constant:

iparam.sim_scaling

Description:

Controls how much effort is used in scaling the problem before a simplex optimizer is used.

Possible values:

- `scalingtype.aggressive` A very aggressive scaling is performed.
- `scalingtype.free` The optimizer chooses the scaling heuristic.
- `scalingtype.moderate` A conservative scaling is performed.
- `scalingtype.none` No scaling is performed.

Default value:

`scalingtype.free`

B.2.168 iparam.sim_scaling_method

Corresponding constant:

iparam.sim_scaling_method

Description:

Controls how the problem is scaled before a simplex optimizer is used.

Possible values:

- `scalingmethod.free` The optimizer chooses the scaling heuristic.
- `scalingmethod.pow2` Scales only with power of 2 leaving the mantissa untouched.

Default value:

`scalingmethod.pow2`

B.2.169 iparam.sim_solve_form

Corresponding constant:

iparam.sim_solve_form

Description:

Controls whether the primal or the dual problem is solved by the primal-/dual- simplex optimizer.

Possible values:

- `solveform.dual` The optimizer should solve the dual problem.
- `solveform.free` The optimizer is free to solve either the primal or the dual problem.
- `solveform.primal` The optimizer should solve the primal problem.

Default value:

`solveform.free`

B.2.170 iparam.sim_stability_priority

Corresponding constant:

iparam.sim_stability_priority

Description:

Controls how high priority the numerical stability should be given.

Possible Values:

Any number between 0 and 100.

Default value:

50

B.2.171 iparam.sim_switch_optimizer

Corresponding constant:

iparam.sim_switch_optimizer

Description:

The simplex optimizer sometimes chooses to solve the dual problem instead of the primal problem. This implies that if you have chosen to use the dual simplex optimizer and the problem is dualized, then it actually makes sense to use the primal simplex optimizer instead. If this parameter is on and the problem is dualized and furthermore the simplex optimizer is chosen to be the primal (dual) one, then it is switched to the dual (primal).

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.172 iparam.sol_filter_keep_basic

Corresponding constant:

iparam.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:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.173 iparam.sol_filter_keep_ranged

Corresponding constant:

iparam.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:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.174 iparam.sol_read_name_width

Corresponding constant:

iparam.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`

B.2.175 iparam.sol_read_width

Corresponding constant:

iparam.sol_read_width

Description:

Controls the maximal acceptable width of line in the solutions when read by MOSEK.

Possible Values:

Any positive number greater than 80.

Default value:

`1024`

B.2.176 iparam.solution_callback

Corresponding constant:

iparam.solution_callback

Description:

Indicates whether solution call-backs will be performed during the optimization.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.177 iparam.timing_level

Corresponding constant:

iparam.timing_level

Description:

Controls the a amount of timing performed inside MOSEK.

Possible Values:

Any integer greater or equal to 0.

Default value:

`1`

B.2.178 iparam.warning_level

Corresponding constant:

iparam.warning_level

Description:

Deprecated and not in use

Possible Values:

Any number between 0 and +inf.

Default value:

`1`

B.2.179 iparam.write_bas_constraints

Corresponding constant:

iparam.write_bas_constraints

Description:

Controls whether the constraint section is written to the basic solution file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.180 iparam.write_bas_head

Corresponding constant:

iparam.write_bas_head

Description:

Controls whether the header section is written to the basic solution file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.181 iparam.write_bas_variables

Corresponding constant:

iparam.write_bas_variables

Description:

Controls whether the variables section is written to the basic solution file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.182 iparam.write_data_compressed

Corresponding constant:

iparam.write_data_compressed

Description:

Controls whether the data file is compressed while it is written. 0 means no compression while higher values mean more compression.

Possible Values:

Any number between 0 and +inf.

Default value:

`0`

B.2.183 iparam.write_data_format

Corresponding constant:

iparam.write_data_format

Description:

Controls the data format when a task is written using `Task.writedata`.

Possible values:

- `dataformat.cb` Conic benchmark format.
- `dataformat.extension` The file extension is used to determine the data file format.
- `dataformat.free_mps` The data data a free MPS formatted file.
- `dataformat.lp` The data file is LP formatted.
- `dataformat.mps` The data file is MPS formatted.
- `dataformat.op` The data file is an optimization problem formatted file.
- `dataformat.task` Generic task dump file.
- `dataformat.xml` The data file is an XML formatted file.

Default value:

`dataformat.extension`

B.2.184 iparam.write_data_param

Corresponding constant:

iparam.write_data_param

Description:

If this option is turned on the parameter settings are written to the data file as parameters.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.185 iparam.write_free_con

Corresponding constant:

iparam.write_free_con

Description:

Controls whether the free constraints are written to the data file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.186 iparam.write_generic_names

Corresponding constant:

iparam.write_generic_names

Description:

Controls whether the generic names or user-defined names are used in the data file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.187 iparam.write_generic_names_io

Corresponding constant:

iparam.write_generic_names_io

Description:

Index origin used in generic names.

Possible Values:

Any number between 0 and +inf.

Default value:

`1`

B.2.188 iparam.write_ignore_incompatible_conic_items

Corresponding constant:

iparam.write_ignore_incompatible_conic_items

Description:

If the output format is not compatible with conic quadratic problems this parameter controls if the writer ignores the conic parts or produces an error.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.189 iparam.write_ignore_incompatible_items

Corresponding constant:

iparam.write_ignore_incompatible_items

Description:

Controls if the writer ignores incompatible problem items when writing files.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.190 iparam.write_ignore_incompatible_nl_items

Corresponding constant:

iparam.write_ignore_incompatible_nl_items

Description:

Controls if the writer ignores general non-linear terms or produces an error.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.191 iparam.write_ignore_incompatible_psd_items

Corresponding constant:

iparam.write_ignore_incompatible_psd_items

Description:

If the output format is not compatible with semidefinite problems this parameter controls if the writer ignores the conic parts or produces an error.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.192 iparam.write_int_constraints

Corresponding constant:

iparam.write_int_constraints

Description:

Controls whether the constraint section is written to the integer solution file.

Possible values:

- `onoffkey.off` Switch the option off.

- onoffkey.on Switch the option on.

Default value:

onoffkey.on

B.2.193 iparam.write_int_head

Corresponding constant:

iparam.write_int_head

Description:

Controls whether the header section is written to the integer solution file.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.on

B.2.194 iparam.write_int_variables

Corresponding constant:

iparam.write_int_variables

Description:

Controls whether the variables section is written to the integer solution file.

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

Default value:

onoffkey.on

B.2.195 iparam.write_lp_line_width

Corresponding constant:

iparam.write_lp_line_width

Description:

Maximum width of line in an LP file written by MOSEK.

Possible Values:

Any positive number.

Default value:

`80`

B.2.196 iparam.write_lp_quoted_names

Corresponding constant:

iparam.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:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.197 iparam.write_lp_strict_format

Corresponding constant:

iparam.write_lp_strict_format

Description:

Controls whether LP output files satisfy the LP format strictly.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.198 iparam.write_lp_terms_per_line

Corresponding constant:

iparam.write_lp_terms_per_line

Description:

Maximum number of terms on a single line in an LP file written by MOSEK. 0 means unlimited.

Possible Values:

Any number between 0 and +inf.

Default value:

`10`

B.2.199 iparam.write_mps_int

Corresponding constant:

iparam.write_mps_int

Description:

Controls if marker records are written to the MPS file to indicate whether variables are integer restricted.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.200 iparam.write_precision

Corresponding constant:

iparam.write_precision

Description:

Controls the precision with which `double` numbers are printed in the MPS data file. In general it is not worthwhile to use a value higher than 15.

Possible Values:

Any number between 0 and +inf.

Default value:

`8`

B.2.201 iparam.write_sol_barvariables

Corresponding constant:

iparam.write_sol_barvariables

Description:

Controls whether the symmetric matrix variables section is written to the solution file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.202 iparam.write_sol_constraints

Corresponding constant:

iparam.write_sol_constraints

Description:

Controls whether the constraint section is written to the solution file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.203 iparam.write_sol_head

Corresponding constant:

iparam.write_sol_head

Description:

Controls whether the header section is written to the solution file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.204 iparam.write_sol_ignore_invalid_names

Corresponding constant:

iparam.write_sol_ignore_invalid_names

Description:

Even if the names are invalid MPS names, then they are employed when writing the solution file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.off`

B.2.205 iparam.write_sol_variables

Corresponding constant:

iparam.write_sol_variables

Description:

Controls whether the variables section is written to the solution file.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.206 iparam.write_task_inc_sol

Corresponding constant:

iparam.write_task_inc_sol

Description:

Controls whether the solutions are stored in the task file too.

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

Default value:

`onoffkey.on`

B.2.207 iparam.write_xml_mode

Corresponding constant:

iparam.write_xml_mode

Description:

Controls if linear coefficients should be written by row or column when writing in the XML file format.

Possible values:

- `xmlwriteroutputtype.col` Write in column order.
- `xmlwriteroutputtype.row` Write in row order.

Default value:

`xmlwriteroutputtype.row`

B.3 sparam: String parameter types

B.3.1 sparam.bas_sol_file_name

Corresponding constant:

sparam.bas_sol_file_name

Description:

Name of the `bas` solution file.

Possible Values:

Any valid file name.

Default value:

`""`

B.3.2 sparam.data_file_name

Corresponding constant:

sparam.data_file_name

Description:

Data are read and written to this file.

Possible Values:

Any valid file name.

Default value:

`""`

B.3.3 sparam.debug_file_name

Corresponding constant:

sparam.debug_file_name

Description:

MOSEK debug file.

Possible Values:

Any valid file name.

Default value:

`""`

B.3.4 sparam.feasrepair_name_prefix

Corresponding constant:

sparam.feasrepair_name_prefix

Description:

If the function `Task.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-"`

B.3.5 sparam.feasrepair_name_separator

Corresponding constant:

sparam.feasrepair_name_separator

Description:

Separator string for names of constraints and variables generated by `Task.relaxprimal`.

Possible Values:

Any valid string.

Default value:

`"-"`

B.3.6 sparam.feasrepair_name_wsumviol

Corresponding constant:

sparam.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"`

B.3.7 sparam.int_sol_file_name

Corresponding constant:

sparam.int_sol_file_name

Description:

Name of the `int` solution file.

Possible Values:

Any valid file name.

Default value:

`""`

B.3.8 sparam.itr_sol_file_name

Corresponding constant:

sparam.itr_sol_file_name

Description:

Name of the `itr` solution file.

Possible Values:

Any valid file name.

Default value:

`""`

B.3.9 sparam.mio_debug_string

Corresponding constant:

sparam.mio_debug_string

Description:

For internal use only.

Possible Values:

Any valid string.

Default value:

""

B.3.10 sparam.param_comment_sign

Corresponding constant:

sparam.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:

"%%"

B.3.11 sparam.param_read_file_name

Corresponding constant:

sparam.param_read_file_name

Description:

Modifications to the parameter database is read from this file.

Possible Values:

Any valid file name.

Default value:

""

B.3.12 sparam.param_write_file_name

Corresponding constant:

sparam.param_write_file_name

Description:

The parameter database is written to this file.

Possible Values:

Any valid file name.

Default value:

""

B.3.13 sparam.read_mps_bou_name

Corresponding constant:

sparam.read_mps_bou_name

Description:

Name of the BOUNDS vector used. An empty name means that the first BOUNDS vector is used.

Possible Values:

Any valid MPS name.

Default value:

""

B.3.14 sparam.read_mps_obj_name

Corresponding constant:

sparam.read_mps_obj_name

Description:

Name of the free constraint used as objective function. An empty name means that the first constraint is used as objective function.

Possible Values:

Any valid MPS name.

Default value:

""

B.3.15 sparam.read_mps_ran_name

Corresponding constant:

sparam.read_mps_ran_name

Description:

Name of the RANGE vector used. An empty name means that the first RANGE vector is used.

Possible Values:

Any valid MPS name.

Default value:

""

B.3.16 sparam.read_mps_rhs_name

Corresponding constant:

sparam.read_mps_rhs_name

Description:

Name of the RHS used. An empty name means that the first RHS vector is used.

Possible Values:

Any valid MPS name.

Default value:

""

B.3.17 sparam.sensitivity_file_name

Corresponding constant:

sparam.sensitivity_file_name

Description:

If defined `Task.sensitivityreport` reads this file as a sensitivity analysis data file specifying the type of analysis to be done.

Possible Values:

Any valid string.

Default value:

""

B.3.18 sparam.sensitivity_res_file_name

Corresponding constant:

sparam.sensitivity_res_file_name

Description:

If this is a nonempty string, then `Task.sensitivityreport` writes results to this file.

Possible Values:

Any valid string.

Default value:

""

B.3.19 sparam.sol_filter_xc_low

Corresponding constant:

sparam.sol_filter_xc_low

Description:

A filter used to determine which constraints should be listed in the solution file. A value of "0.5" means that all constraints having `xc[i]>0.5` should be listed, whereas "+0.5" means that all constraints having `xc[i]>=blc[i]+0.5` should be listed. An empty filter means that no filter is applied.

Possible Values:

Any valid filter.

Default value:

""

B.3.20 sparam.sol_filter_xc_upr

Corresponding constant:

sparam.sol_filter_xc_upr

Description:

A filter used to determine which constraints should be listed in the solution file. A value of "0.5" means that all constraints having `xc[i]<0.5` should be listed, whereas "-0.5" means all constraints having `xc[i]<=buc[i]-0.5` should be listed. An empty filter means that no filter is applied.

Possible Values:

Any valid filter.

Default value:

""

B.3.21 sparam.sol_filter_xx_low

Corresponding constant:

sparam.sol_filter_xx_low

Description:

A filter used to determine which variables should be listed in the solution file. A value of "0.5" means that all constraints having `xx[j]>=0.5` should be listed, whereas "+0.5" means that all constraints having `xx[j]>=blx[j]+0.5` should be listed. An empty filter means no filter is applied.

Possible Values:

Any valid filter.

Default value:

""

B.3.22 sparam.sol_filter_xx_upr

Corresponding constant:

sparam.sol_filter_xx_upr

Description:

A filter used to determine which variables should be listed in the solution file. A value of "0.5" means that all constraints having `xx[j]<0.5` should be printed, whereas "-0.5" means all constraints having `xx[j]<=bux[j]-0.5` should be listed. An empty filter means no filter is applied.

Possible Values:

Any valid file name.

Default value:

""

B.3.23 sparam.stat_file_name

Corresponding constant:

sparam.stat_file_name

Description:

Statistics file name.

Possible Values:

Any valid file name.

Default value:

""

B.3.24 sparam.stat_key

Corresponding constant:

sparam.stat_key

Description:

Key used when writing the summary file.

Possible Values:

Any valid XML string.

Default value:

""

B.3.25 sparam.stat_name

Corresponding constant:

sparam.stat_name

Description:

Name used when writing the statistics file.

Possible Values:

Any valid XML string.

Default value:

""

B.3.26 sparam.write_lp_gen_var_name

Corresponding constant:

sparam.write_lp_gen_var_name

Description:

Sometimes when an LP file is written additional variables must be inserted. They will have the prefix denoted by this parameter.

Possible Values:

Any valid string.

Default value:

`"xmskgen"`

Appendix C

Response codes

Response codes ordered by name.

`rescode.err_ad_invalid_codelist`
: The code list data was invalid.

`rescode.err_ad_invalid_operand`
: The code list data was invalid. An unknown operand was used.

`rescode.err_ad_invalid_operator`
: The code list data was invalid. An unknown operator was used.

`rescode.err_ad_missing_operand`
: The code list data was invalid. Missing operand for operator.

`rescode.err_ad_missing_return`
: The code list data was invalid. Missing return operation in function.

`rescode.err_api_array_too_small`
: An input array was too short.

`rescode.err_api_cb_connect`
: Failed to connect a callback object.

`rescode.err_api_fatal_error`
: An internal error occurred in the API. Please report this problem.

`rescode.err_api_internal`
: An internal fatal error occurred in an interface function.

`rescode.err_arg_is_too_large`
: The value of a argument is too small.

`rescode.err_arg_is_too_small`

The value of a argument is too small.

`rescode.err_argument_dimension`

A function argument is of incorrect dimension.

`rescode.err_argument_is_too_large`

The value of a function argument is too large.

`rescode.err_argument_lenneq`

Incorrect length of arguments.

`rescode.err_argument_perm_array`

An invalid permutation array is specified.

`rescode.err_argument_type`

Incorrect argument type.

`rescode.err_bar_var_dim`

The dimension of a symmetric matrix variable has to greater than 0.

`rescode.err_basis`

An invalid basis is specified. Either too many or too few basis variables are specified.

`rescode.err_basis_factor`

The factorization of the basis is invalid.

`rescode.err_basis_singular`

The basis is singular and hence cannot be factored.

`rescode.err_blank_name`

An all blank name has been specified.

`rescode.err_cannot_clone_nl`

A task with a nonlinear function call-back cannot be cloned.

`rescode.err_cannot_handle_nl`

A function cannot handle a task with nonlinear function call-backs.

`rescode.err_cbf_duplicate_acoord`

Duplicate index in ACOORD.

`rescode.err_cbf_duplicate_bcoord`

Duplicate index in BCOORD.

`rescode.err_cbf_duplicate_con`

Duplicate CON keyword.

`rescode.err_cbf_duplicate_int`
  Duplicate INT keyword.

`rescode.err_cbf_duplicate_obj`
  Duplicate OBJ keyword.

`rescode.err_cbf_duplicate_objacoord`
  Duplicate index in OBJCOORD.

`rescode.err_cbf_duplicate_var`
  Duplicate VAR keyword.

`rescode.err_cbf_invalid_con_type`
  Invalid constraint type.

`rescode.err_cbf_invalid_domain_dimension`
  Invalid domain dimension.

`rescode.err_cbf_invalid_int_index`
  Invalid INT index.

`rescode.err_cbf_invalid_var_type`
  Invalid variable type.

`rescode.err_cbf_no_variables`
  No variables are specified.

`rescode.err_cbf_no_version_specified`
  No version specified.

`rescode.err_cbf_obj_sense`
  An invalid objective sense is specified.

`rescode.err_cbf_parse`
  An error occurred while parsing an CBF file.

`rescode.err_cbf_syntax`
  Invalid syntax.

`rescode.err_cbf_too_few_constraints`
  Too few constraints defined.

`rescode.err_cbf_too_few_ints`
  Too few ints are specified.

`rescode.err_cbf_too_few_variables`
  Too few variables defined.

`rescode.err_cbf_too_many_constraints`

Too many constraints specified.

`rescode.err_cbf_too_many_ints`

Too many ints are specified.

`rescode.err_cbf_too_many_variables`

Too many variables specified.

`rescode.err_cbf_unsupported`

Unsupported feature is present.

`rescode.err_con_q_not_nsd`

The quadratic constraint matrix is not negative semidefinite as expected for a constraint with finite lower bound. This results in a nonconvex problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.

`rescode.err_con_q_not_psd`

The quadratic constraint matrix is not positive semidefinite as expected for a constraint with finite upper bound. This results in a nonconvex problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.

`rescode.err_concurrent_optimizer`

An unsupported optimizer was chosen for use with the concurrent optimizer.

`rescode.err_cone_index`

An index of a non-existing cone has been specified.

`rescode.err_cone_overlap`

A new cone which variables overlap with an existing cone has been specified.

`rescode.err_cone_overlap_append`

The cone to be appended has one variable which is already memeber of another cone.

`rescode.err_cone_rep_var`

A variable is included multiple times in the cone.

`rescode.err_cone_size`

A cone with too few members is specified.

`rescode.err_cone_type`

Invalid cone type specified.

`rescode.err_cone_type_str`

Invalid cone type specified.

`rescode.err_data_file_ext`

The data file format cannot be determined from the file name.

`rescode.err_dup_name`

The same name was used multiple times for the same problem item type.

`rescode.err_duplicate_barvariable_names`

Two barvariable names are identical.

`rescode.err_duplicate_cone_names`

Two cone names are identical.

`rescode.err_duplicate_constraint_names`

Two constraint names are identical.

`rescode.err_duplicate_variable_names`

Two variable names are identical.

`rescode.err_end_of_file`

End of file reached.

`rescode.err_factor`

An error occurred while factorizing a matrix.

`rescode.err_feasrepair_cannot_relax`

An optimization problem cannot be relaxed. This is the case e.g. for general nonlinear optimization problems.

`rescode.err_feasrepair_inconsistent_bound`

The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.

`rescode.err_feasrepair_solving_relaxed`

The relaxed problem could not be solved to optimality. Please consult the log file for further details.

`rescode.err_file_license`

Invalid license file.

`rescode.err_file_open`

Error while opening a file.

`rescode.err_file_read`

File read error.

`rescode.err_file_write`

File write error.

rescode.err_first

Invalid first.

rescode.err_firsti

Invalid firsti.

rescode.err_firstj

Invalid firstj.

rescode.err_fixed_bound_values

A fixed constraint/variable has been specified using the bound keys but the numerical value of the lower and upper bound is different.

rescode.err_flexlm

The FLEXlm license manager reported an error.

rescode.err_global_inv_conic_problem

The global optimizer can only be applied to problems without semidefinite variables.

rescode.err_huge_aij

A numerically huge value is specified for an $a_{i,j}$ element in A. The parameter dparam.data_tol_aij_huge controls when an $a_{i,j}$ is considered huge.

rescode.err_huge_c

A huge value in absolute size is specified for one c_j.

rescode.err_identical_tasks

Some tasks related to this function call were identical. Unique tasks were expected.

rescode.err_in_argument

A function argument is incorrect.

rescode.err_index

An index is out of range.

rescode.err_index_arr_is_too_large

An index in an array argument is too large.

rescode.err_index_arr_is_too_small

An index in an array argument is too small.

rescode.err_index_is_too_large

An index in an argument is too large.

rescode.err_index_is_too_small

An index in an argument is too small.

`rescode.err_inf_dou_index`

A double information index is out of range for the specified type.

`rescode.err_inf_dou_name`

A double information name is invalid.

`rescode.err_inf_int_index`

An integer information index is out of range for the specified type.

`rescode.err_inf_int_name`

An integer information name is invalid.

`rescode.err_inf_lint_index`

A long integer information index is out of range for the specified type.

`rescode.err_inf_lint_name`

A long integer information name is invalid.

`rescode.err_inf_type`

The information type is invalid.

`rescode.err_infeas_undefined`

The requested value is not defined for this solution type.

`rescode.err_infinite_bound`

A numerically huge bound value is specified.

`rescode.err_int64_to_int32_cast`

An 32 bit integer could not cast to a 64 bit integer.

`rescode.err_internal`

An internal error occurred. Please report this problem.

`rescode.err_internal_test_failed`

An internal unit test function failed.

`rescode.err_inv_aptre`

`aptre[j]` is strictly smaller than `aptrb[j]` for some `j`.

`rescode.err_inv_bk`

Invalid bound key.

`rescode.err_inv_bkc`

Invalid bound key is specified for a constraint.

`rescode.err_inv_bkx`

An invalid bound key is specified for a variable.

`rescode.err_inv_cone_type`

Invalid cone type code is encountered.

`rescode.err_inv_cone_type_str`

Invalid cone type string encountered.

`rescode.err_inv_conic_problem`

The conic optimizer can only be applied to problems with linear objective and constraints. Many problems such convex quadratically constrained problems can easily be reformulated to conic problems. See the appropriate MOSEK manual for details.

`rescode.err_inv_marki`

Invalid value in marki.

`rescode.err_inv_markj`

Invalid value in markj.

`rescode.err_inv_name_item`

An invalid name item code is used.

`rescode.err_inv_numi`

Invalid numi.

`rescode.err_inv_numj`

Invalid numj.

`rescode.err_inv_optimizer`

An invalid optimizer has been chosen for the problem. This means that the simplex or the conic optimizer is chosen to optimize a nonlinear problem.

`rescode.err_inv_problem`

Invalid problem type. Probably a nonconvex problem has been specified.

`rescode.err_inv_qcon_subi`

Invalid value in `qcsubi`.

`rescode.err_inv_qcon_subj`

Invalid value in `qcsubj`.

`rescode.err_inv_qcon_subk`

Invalid value in `qcsubk`.

`rescode.err_inv_qcon_val`

Invalid value in `qcval`.

`rescode.err_inv_qobj_subi`

Invalid value in `qosubi`.

rescode.err_inv_qobj_subj

Invalid value in qosubj.

rescode.err_inv_qobj_val

Invalid value in qoval.

rescode.err_inv_sk

Invalid status key code.

rescode.err_inv_sk_str

Invalid status key string encountered.

rescode.err_inv_skc

Invalid value in skc.

rescode.err_inv_skn

Invalid value in skn.

rescode.err_inv_skx

Invalid value in skx.

rescode.err_inv_var_type

An invalid variable type is specified for a variable.

rescode.err_invalid_accmode

An invalid access mode is specified.

rescode.err_invalid_aij

$a_{i,j}$ contains an invalid floating point value, i.e. a NaN or an infinite value.

rescode.err_invalid_ampl_stub

Invalid AMPL stub.

rescode.err_invalid_barvar_name

An invalid symmetric matrix variable name is used.

rescode.err_invalid_branch_direction

An invalid branching direction is specified.

rescode.err_invalid_branch_priority

An invalid branching priority is specified. It should be nonnegative.

rescode.err_invalid_compression

Invalid compression type.

rescode.err_invalid_con_name

An invalid constraint name is used.

`rescode.err_invalid_cone_name`

An invalid cone name is used.

`rescode.err_invalid_file_format_for_cones`

The file format does not support a problem with conic constraints.

`rescode.err_invalid_file_format_for_general_nl`

The file format does not support a problem with general nonlinear terms.

`rescode.err_invalid_file_format_for_sym_mat`

The file format does not support a problem with symmetric matrix variables.

`rescode.err_invalid_file_name`

An invalid file name has been specified.

`rescode.err_invalid_format_type`

Invalid format type.

`rescode.err_invalid_idx`

A specified index is invalid.

`rescode.err_invalid_iomode`

Invalid io mode.

`rescode.err_invalid_max_num`

A specified index is invalid.

`rescode.err_invalid_name_in_sol_file`

An invalid name occurred in a solution file.

`rescode.err_invalid_network_problem`

The problem is not a network problem as expected. The error occurs if a network optimizer is applied to a problem that cannot (easily) be converted to a network problem.

`rescode.err_invalid_obj_name`

An invalid objective name is specified.

`rescode.err_invalid_objective_sense`

An invalid objective sense is specified.

`rescode.err_invalid_problem_type`

An invalid problem type.

`rescode.err_invalid_sol_file_name`

An invalid file name has been specified.

`rescode.err_invalid_stream`

An invalid stream is referenced.

`rescode.err_invalid_surplus`

Invalid surplus.

`rescode.err_invalid_sym_mat_dim`

A sparse symmetric matrix of invalid dimension is specified.

`rescode.err_invalid_task`

The `task` is invalid.

`rescode.err_invalid_utf8`

An invalid UTF8 string is encountered.

`rescode.err_invalid_var_name`

An invalid variable name is used.

`rescode.err_invalid_wchar`

An invalid `wchar` string is encountered.

`rescode.err_invalid_whichsol`

`whichsol` is invalid.

`rescode.err_last`

Invalid index `last`. A given index was out of expected range.

`rescode.err_lasti`

Invalid `lasti`.

`rescode.err_lastj`

Invalid `lastj`.

`rescode.err_lau_arg_k`

Invalid argument k.

`rescode.err_lau_arg_m`

Invalid argument m.

`rescode.err_lau_arg_n`

Invalid argument n.

`rescode.err_lau_arg_trans`

Invalid argument trans.

`rescode.err_lau_arg_transa`

Invalid argument transa.

`rescode.err_lau_arg_transb`

Invalid argument transb.

`rescode.err_lau_arg_uplo`

Invalid argument uplo.

`rescode.err_lau_singular_matrix`

A matrix is singular.

`rescode.err_lau_unknown`

An unknown error.

`rescode.err_license`

Invalid license.

`rescode.err_license_cannot_allocate`

The license system cannot allocate the memory required.

`rescode.err_license_cannot_connect`

MOSEK cannot connect to the license server. Most likely the license server is not up and running.

`rescode.err_license_expired`

The license has expired.

`rescode.err_license_feature`

A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.

`rescode.err_license_invalid_hostid`

The host ID specified in the license file does not match the host ID of the computer.

`rescode.err_license_max`

Maximum number of licenses is reached.

`rescode.err_license_moseklm_daemon`

The MOSEKLM license manager daemon is not up and running.

`rescode.err_license_no_server_line`

There is no `SERVER` line in the license file. All non-zero license count features need at least one `SERVER` line.

`rescode.err_license_no_server_support`

The license server does not support the requested feature. Possible reasons for this error include:

- The feature has expired.
- The feature's start date is later than today's date.

- The version requested is higher than feature's the highest supported version.
- A corrupted license file.

Try restarting the license and inspect the license server debug file, usually called `lmgrd.log`.

`rescode.err_license_server`

The license server is not responding.

`rescode.err_license_server_version`

The version specified in the checkout request is greater than the highest version number the daemon supports.

`rescode.err_license_version`

The license is valid for another version of MOSEK.

`rescode.err_link_file_dll`

A file cannot be linked to a stream in the DLL version.

`rescode.err_living_tasks`

All tasks associated with an enviroment must be deleted before the environment is deleted. There are still some undeleted tasks.

`rescode.err_lower_bound_is_a_nan`

The lower bound specificied is not a number (nan).

`rescode.err_lp_dup_slack_name`

The name of the slack variable added to a ranged constraint already exists.

`rescode.err_lp_empty`

The problem cannot be written to an LP formatted file.

`rescode.err_lp_file_format`

Syntax error in an LP file.

`rescode.err_lp_format`

Syntax error in an LP file.

`rescode.err_lp_free_constraint`

Free constraints cannot be written in LP file format.

`rescode.err_lp_incompatible`

The problem cannot be written to an LP formatted file.

`rescode.err_lp_invalid_con_name`

A constraint name is invalid when used in an LP formatted file.

`rescode.err_lp_invalid_var_name`

A variable name is invalid when used in an LP formatted file.

`rescode.err_lp_write_conic_problem`

The problem contains cones that cannot be written to an LP formatted file.

`rescode.err_lp_write_geco_problem`

The problem contains general convex terms that cannot be written to an LP formatted file.

`rescode.err_lu_max_num_tries`

Could not compute the LU factors of the matrix within the maximum number of allowed tries.

`rescode.err_max_len_is_too_small`

An maximum length that is too small has been specfified.

`rescode.err_maxnumbarvar`

The maximum number of semidefinite variables specified is smaller than the number of semidefinite variables in the task.

`rescode.err_maxnumcon`

The maximum number of constraints specified is smaller than the number of constraints in the task.

`rescode.err_maxnumcone`

The value specified for `maxnumcone` is too small.

`rescode.err_maxnumqnz`

The maximum number of non-zeros specified for the Q matrixes is smaller than the number of non-zeros in the current Q matrixes.

`rescode.err_maxnumvar`

The maximum number of variables specified is smaller than the number of variables in the task.

`rescode.err_mbt_incompatible`

The MBT file is incompatible with this platform. This results from reading a file on a 32 bit platform generated on a 64 bit platform.

`rescode.err_mbt_invalid`

The MBT file is invalid.

`rescode.err_mio_internal`

A fatal error occurred in the mixed integer optimizer. Please contact MOSEK support.

`rescode.err_mio_invalid_node_optimizer`

An invalid node optimizer was selected for the problem type.

`rescode.err_mio_invalid_root_optimizer`

An invalid root optimizer was selected for the problem type.

`rescode.err_mio_no_optimizer`

No optimizer is available for the current class of integer optimization problems.

`rescode.err_mio_not_loaded`

The mixed-integer optimizer is not loaded.

`rescode.err_missing_license_file`

MOSEK cannot license file or a token server. See the MOSEK installation manual for details.

`rescode.err_mixed_problem`

The problem contains both conic and nonlinear constraints.

`rescode.err_mps_cone_overlap`

A variable is specified to be a member of several cones.

`rescode.err_mps_cone_repeat`

A variable is repeated within the `CSECTION`.

`rescode.err_mps_cone_type`

Invalid cone type specified in a `CSECTION`.

`rescode.err_mps_duplicate_q_element`

Duplicate elements is specfied in a Q matrix.

`rescode.err_mps_file`

An error occurred while reading an MPS file.

`rescode.err_mps_inv_bound_key`

An invalid bound key occurred in an MPS file.

`rescode.err_mps_inv_con_key`

An invalid constraint key occurred in an MPS file.

`rescode.err_mps_inv_field`

A field in the MPS file is invalid. Probably it is too wide.

`rescode.err_mps_inv_marker`

An invalid marker has been specified in the MPS file.

`rescode.err_mps_inv_sec_name`

An invalid section name occurred in an MPS file.

`rescode.err_mps_inv_sec_order`

The sections in the MPS data file are not in the correct order.

`rescode.err_mps_invalid_obj_name`

An invalid objective name is specified.

`rescode.err_mps_invalid_objsense`

An invalid objective sense is specified.

`rescode.err_mps_mul_con_name`

A constraint name was specified multiple times in the `ROWS` section.

`rescode.err_mps_mul_csec`

Multiple `CSECTION`s are given the same name.

`rescode.err_mps_mul_qobj`

The Q term in the objective is specified multiple times in the MPS data file.

`rescode.err_mps_mul_qsec`

Multiple `QSECTION`s are specified for a constraint in the MPS data file.

`rescode.err_mps_no_objective`

No objective is defined in an MPS file.

`rescode.err_mps_non_symmetric_q`

A non symmetric matrice has been speciefied.

`rescode.err_mps_null_con_name`

An empty constraint name is used in an MPS file.

`rescode.err_mps_null_var_name`

An empty variable name is used in an MPS file.

`rescode.err_mps_splitted_var`

All elements in a column of the A matrix must be specified consecutively. Hence, it is illegal to specify non-zero elements in A for variable 1, then for variable 2 and then variable 1 again.

`rescode.err_mps_tab_in_field2`

A tab char occurred in field 2.

`rescode.err_mps_tab_in_field3`

A tab char occurred in field 3.

`rescode.err_mps_tab_in_field5`

A tab char occurred in field 5.

`rescode.err_mps_undef_con_name`

An undefined constraint name occurred in an MPS file.

`rescode.err_mps_undef_var_name`

An undefined variable name occurred in an MPS file.

`rescode.err_mul_a_element`

An element in A is defined multiple times.

`rescode.err_name_is_null`

The name buffer is a NULL pointer.

`rescode.err_name_max_len`

A name is longer than the buffer that is supposed to hold it.

`rescode.err_nan_in_blc`

l^c contains an invalid floating point value, i.e. a `NaN`.

`rescode.err_nan_in_blx`

l^x contains an invalid floating point value, i.e. a `NaN`.

`rescode.err_nan_in_buc`

u^c contains an invalid floating point value, i.e. a `NaN`.

`rescode.err_nan_in_bux`

u^x contains an invalid floating point value, i.e. a `NaN`.

`rescode.err_nan_in_c`

c contains an invalid floating point value, i.e. a `NaN`.

`rescode.err_nan_in_double_data`

An invalid floating point value was used in some double data.

`rescode.err_negative_append`

Cannot append a negative number.

`rescode.err_negative_surplus`

Negative surplus.

`rescode.err_newer_dll`

The dynamic link library is newer than the specified version.

`rescode.err_no_bars_for_solution`

There is no $\bar{s}$ available for the solution specified. In particular note there are no $\bar{s}$ defined for the basic and integer solutions.

`rescode.err_no_barx_for_solution`

There is no $\bar{X}$ available for the solution specified. In particular note there are no $\bar{X}$ defined for the basic and integer solutions.

rescode.err_no_basis_sol

No basic solution is defined.

rescode.err_no_dual_for_itg_sol

No dual information is available for the integer solution.

rescode.err_no_dual_infeas_cer

A certificate of infeasibility is not available.

rescode.err_no_dual_info_for_itg_sol

Dual information is not available for the integer solution.

rescode.err_no_init_env

env is not initialized.

rescode.err_no_optimizer_var_type

No optimizer is available for this class of optimization problems.

rescode.err_no_primal_infeas_cer

A certificate of primal infeasibility is not available.

rescode.err_no_snx_for_bas_sol

s_n^x is not available for the basis solution.

rescode.err_no_solution_in_callback

The required solution is not available.

rescode.err_non_unique_array

An array does not contain unique elements.

rescode.err_nonconvex

The optimization problem is nonconvex.

rescode.err_nonlinear_equality

The model contains a nonlinear equality which defines a nonconvex set.

rescode.err_nonlinear_functions_not_allowed

An operation that is invalid for problems with nonlinear functions defined has been attempted.

rescode.err_nonlinear_ranged

The model contains a nonlinear ranged constraint which by definition defines a nonconvex set.

rescode.err_nr_arguments

Incorrect number of function arguments.

rescode.err_null_env

env is a NULL pointer.

rescode.err_null_pointer

An argument to a function is unexpectedly a NULL pointer.

rescode.err_null_task

task is a NULL pointer.

rescode.err_numconlim

Maximum number of constraints limit is exceeded.

rescode.err_numvarlim

Maximum number of variables limit is exceeded.

rescode.err_obj_q_not_nsd

The quadratic coefficient matrix in the objective is not negative semidefinite as expected for a maximization problem. The parameter dparam.check_convexity_rel_tol can be used to relax the convexity check.

rescode.err_obj_q_not_psd

The quadratic coefficient matrix in the objective is not positive semidefinite as expected for a minimization problem. The parameter dparam.check_convexity_rel_tol can be used to relax the convexity check.

rescode.err_objective_range

Empty objective range.

rescode.err_older_dll

The dynamic link library is older than the specified version.

rescode.err_open_dl

A dynamic link library could not be opened.

rescode.err_opf_format

Syntax error in an OPF file

rescode.err_opf_new_variable

Introducing new variables is now allowed. When a [variables] section is present, it is not allowed to introduce new variables later in the problem.

rescode.err_opf_premature_eof

Premature end of file in an OPF file.

rescode.err_optimizer_license

The optimizer required is not licensed.

rescode.err_ord_invalid

Invalid content in branch ordering file.

`rescode.err_ord_invalid_branch_dir`

An invalid branch direction key is specified.

`rescode.err_overflow`

A computation produced an overflow i.e. a very large number.

`rescode.err_param_index`

Parameter index is out of range.

`rescode.err_param_is_too_large`

The parameter value is too large.

`rescode.err_param_is_too_small`

The parameter value is too small.

`rescode.err_param_name`

The parameter name is not correct.

`rescode.err_param_name_dou`

The parameter name is not correct for a double parameter.

`rescode.err_param_name_int`

The parameter name is not correct for an integer parameter.

`rescode.err_param_name_str`

The parameter name is not correct for a string parameter.

`rescode.err_param_type`

The parameter type is invalid.

`rescode.err_param_value_str`

The parameter value string is incorrect.

`rescode.err_platform_not_licensed`

A requested license feature is not available for the required platform.

`rescode.err_postsolve`

An error occurred during the postsolve. Please contact MOSEK support.

`rescode.err_pro_item`

An invalid problem is used.

`rescode.err_prob_license`

The software is not licensed to solve the problem.

`rescode.err_qcon_subi_too_large`

Invalid value in `qcsubi`.

`rescode.err_qcon_subi_too_small`

Invalid value in `qcsubi`.

`rescode.err_qcon_upper_triangle`

An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.

`rescode.err_qobj_upper_triangle`

An element in the upper triangle of Q^o is specified. Only elements in the lower triangle should be specified.

`rescode.err_read_format`

The specified format cannot be read.

`rescode.err_read_lp_missing_end_tag`

Syntax error in LP file. Possibly missing End tag.

`rescode.err_read_lp_nonexisting_name`

A variable never occurred in objective or constraints.

`rescode.err_remove_cone_variable`

A variable cannot be removed because it will make a cone invalid.

`rescode.err_repair_invalid_problem`

The feasibility repair does not support the specified problem type.

`rescode.err_repair_optimization_failed`

Computation the optimal relaxation failed. The cause may have been numerical problems.

`rescode.err_sen_bound_invalid_lo`

Analysis of lower bound requested for an index, where no lower bound exists.

`rescode.err_sen_bound_invalid_up`

Analysis of upper bound requested for an index, where no upper bound exists.

`rescode.err_sen_format`

Syntax error in sensitivity analysis file.

`rescode.err_sen_index_invalid`

Invalid range given in the sensitivity file.

`rescode.err_sen_index_range`

Index out of range in the sensitivity analysis file.

`rescode.err_sen_invalid_regexp`

Syntax error in regexp or regexp longer than 1024.

`rescode.err_sen_numerical`

Numerical difficulties encountered performing the sensitivity analysis.

`rescode.err_sen_solution_status`

No optimal solution found to the original problem given for sensitivity analysis.

`rescode.err_sen_undef_name`

An undefined name was encountered in the sensitivity analysis file.

`rescode.err_sen_unhandled_problem_type`

Sensitivity analysis cannot be performed for the spcified problem. Sensitivity analysis is only possible for linear problems.

`rescode.err_size_license`

The problem is bigger than the license.

`rescode.err_size_license_con`

The problem has too many constraints to be solved with the available license.

`rescode.err_size_license_intvar`

The problem contains too many integer variables to be solved with the available license.

`rescode.err_size_license_numcores`

The computer contains more cpu cores than the license allows for.

`rescode.err_size_license_var`

The problem has too many variables to be solved with the available license.

`rescode.err_sol_file_invalid_number`

An invalid number is specified in a solution file.

`rescode.err_solitem`

The solution item number `solitem` is invalid. Please note that `solitem.snx` is invalid for the basic solution.

`rescode.err_solver_probtype`

Problem type does not match the chosen optimizer.

`rescode.err_space`

Out of space.

`rescode.err_space_leaking`

MOSEK is leaking memory. This can be due to either an incorrect use of MOSEK or a bug.

`rescode.err_space_no_info`

No available information about the space usage.

`rescode.err_sym_mat_duplicate`

A value in a symmetric matric as been specified more than once.

`rescode.err_sym_mat_invalid_col_index`

A column index specified for sparse symmetric maxtrix is invalid.

`rescode.err_sym_mat_invalid_row_index`

A row index specified for sparse symmetric maxtrix is invalid.

`rescode.err_sym_mat_invalid_value`

The numerical value specified in a sparse symmetric matrix is not a value floating value.

`rescode.err_sym_mat_not_lower_tringular`

Only the lower triangular part of sparse symmetric matrix should be specified.

`rescode.err_task_incompatible`

The Task file is incompatible with this platform. This results from reading a file on a 32 bit platform generated on a 64 bit platform.

`rescode.err_task_invalid`

The Task file is invalid.

`rescode.err_thread_cond_init`

Could not initialize a condition.

`rescode.err_thread_create`

Could not create a thread. This error may occur if a large number of environments are created and not deleted again. In any case it is a good practice to minimize the number of environments created.

`rescode.err_thread_mutex_init`

Could not initialize a mutex.

`rescode.err_thread_mutex_lock`

Could not lock a mutex.

`rescode.err_thread_mutex_unlock`

Could not unlock a mutex.

`rescode.err_toconic_conversion_fail`

A constraint could not be converted in conic form.

`rescode.err_too_many_concurrent_tasks`

Too many concurrent tasks specified.

`rescode.err_too_small_max_num_nz`

The maximum number of non-zeros specified is too small.

`rescode.err_too_small_maxnumanz`

The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A.

`rescode.err_unb_step_size`

A step size in an optimizer was unexpectedly unbounded. For instance, if the step-size becomes unbounded in phase 1 of the simplex algorithm then an error occurs. Normally this will happen only if the problem is badly formulated. Please contact MOSEK support if this error occurs.

`rescode.err_undef_solution`

MOSEK has the following solution types:

- an interior-point solution,
- an basic solution,
- and an integer solution.

Each optimizer may set one or more of these solutions; e.g by default a successful optimization with the interior-point optimizer defines the interior-point solution, and, for linear problems, also the basic solution. This error occurs when asking for a solution or for information about a solution that is not defined.

`rescode.err_undefined_objective_sense`

The objective sense has not been specified before the optimization.

`rescode.err_unhandled_solution_status`

Unhandled solution status.

`rescode.err_unknown`

Unknown error.

`rescode.err_upper_bound_is_a_nan`

The upper bound specificied is not a number (nan).

`rescode.err_upper_triangle`

An element in the upper triangle of a lower triangular matrix is specified.

`rescode.err_user_func_ret`

An user function reported an error.

`rescode.err_user_func_ret_data`

An user function returned invalid data.

`rescode.err_user_nlo_eval`

The user-defined nonlinear function reported an error.

`rescode.err_user_nlo_eval_hessubi`

The user-defined nonlinear function reported an invalid subscript in the Hessian.

`rescode.err_user_nlo_eval_hessubj`

The user-defined nonlinear function reported an invalid subscript in the Hessian.

`rescode.err_user_nlo_func`

The user-defined nonlinear function reported an error.

`rescode.err_whichitem_not_allowed`

`whichitem` is unacceptable.

`rescode.err_whichsol`

The solution defined by compwhichsol does not exists.

`rescode.err_write_lp_format`

Problem cannot be written as an LP file.

`rescode.err_write_lp_non_unique_name`

An auto-generated name is not unique.

`rescode.err_write_mps_invalid_name`

An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.

`rescode.err_write_opf_invalid_var_name`

Empty variable names cannot be written to OPF files.

`rescode.err_writing_file`

An error occurred while writing file

`rescode.err_xml_invalid_problem_type`

The problem type is not supported by the XML format.

`rescode.err_y_is_undefined`

The solution item y is undefined.

`rescode.ok`

No error occurred.

`rescode.trm_internal`

The optimizer terminated due to some internal reason. Please contact MOSEK support.

`rescode.trm_internal_stop`

The optimizer terminated for internal reasons. Please contact MOSEK support.

`rescode.trm_max_iterations`

The optimizer terminated at the maximum number of iterations.

`rescode.trm_max_num_setbacks`

The optimizer terminated as the maximum number of set-backs was reached. This indicates numerical problems and a possibly badly formulated problem.

`rescode.trm_max_time`

The optimizer terminated at the maximum amount of time.

`rescode.trm_mio_near_abs_gap`

The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied.

`rescode.trm_mio_near_rel_gap`

The mixed-integer optimizer terminated because the near optimal relative gap tolerance was satisfied.

`rescode.trm_mio_num_branches`

The mixed-integer optimizer terminated as to the maximum number of branches was reached.

`rescode.trm_mio_num_relaxs`

The mixed-integer optimizer terminated as the maximum number of relaxations was reached.

`rescode.trm_num_max_num_int_solutions`

The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.

`rescode.trm_numerical_problem`

The optimizer terminated due to numerical problems.

`rescode.trm_objective_range`

The optimizer terminated on the bound of the objective range.

`rescode.trm_stall`

The optimizer is terminated due to slow progress.

Stalling means that numerical problems prevent the optimizer from making reasonable progress and that it make no sense to continue. In many cases this happens if the problem is badly scaled or otherwise ill-conditioned. There is no guarantee that the solution will be (near) feasible or near optimal. However, often stalling happens near the optimum, and the returned solution may be of good quality. Therefore, it is recommended to check the status of then solution. If the solution near optimal the solution is most likely good enough for most practical purposes.

Please note that if a linear optimization problem is solved using the interior-point optimizer with basis identification turned on, the returned basic solution likely to have high accuracy, even though the optimizer stalled.

Some common causes of stalling are a) badly scaled models, b) near feasible or near infeasible problems and c) a non-convex problems. Case c) is only relevant for general non-linear problems. It is not possible in general for MOSEK to check if a specific problems is convex since such a check would be NP hard in itself. This implies that care should be taken when solving problems involving general user defined functions.

`rescode.trm_user_callback`

The optimizer terminated due to the return of the user-defined call-back function.

`rescode.wrn_ana_almost_int_bounds`

This warning is issued by the problem analyzer if a constraint is bound nearly integral.

`rescode.wrn_ana_c_zero`

This warning is issued by the problem analyzer, if the coefficients in the linear part of the objective are all zero.

`rescode.wrn_ana_close_bounds`

This warning is issued by problem analyzer, if ranged constraints or variables with very close upper and lower bounds are detected. One should consider treating such constraints as equalities and such variables as constants.

`rescode.wrn_ana_empty_cols`

This warning is issued by the problem analyzer, if columns, in which all coefficients are zero, are found.

`rescode.wrn_ana_large_bounds`

This warning is issued by the problem analyzer, if one or more constraint or variable bounds are very large. One should consider omitting these bounds entirely by setting them to +inf or -inf.

`rescode.wrn_construct_invalid_sol_itg`

The intial value for one or more of the integer variables is not feasible.

`rescode.wrn_construct_no_sol_itg`

The construct solution requires an integer solution.

`rescode.wrn_construct_solution_infeas`

After fixing the integer variables at the suggested values then the problem is infeasible.

`rescode.wrn_dropped_nz_qobj`

One or more non-zero elements were dropped in the Q matrix in the objective.

`rescode.wrn_duplicate_barvariable_names`

Two barvariable names are identical.

`rescode.wrn_duplicate_cone_names`

Two cone names are identical.

`rescode.wrn_duplicate_constraint_names`

Two constraint names are identical.

`rescode.wrn_duplicate_variable_names`

Two variable names are identical.

`rescode.wrn_eliminator_space`

The eliminator is skipped at least once due to lack of space.

`rescode.wrn_empty_name`

A variable or constraint name is empty. The output file may be invalid.

`rescode.wrn_ignore_integer`

Ignored integer constraints.

`rescode.wrn_incomplete_linear_dependency_check`

The linear dependency check(s) is not completed. Normally this is not an important warning unless the optimization problem has been formulated with linear dependencies which is bad practice.

`rescode.wrn_large_aij`

A numerically large value is specified for an $a_{i,j}$ element in A. The parameter `dparam.data_tol_aij_large` controls when an $a_{i,j}$ is considered large.

`rescode.wrn_large_bound`

A numerically large bound value is specified.

`rescode.wrn_large_cj`

A numerically large value is specified for one c_j.

`rescode.wrn_large_con_fx`

An equality constraint is fixed to a numerically large value. This can cause numerical problems.

`rescode.wrn_large_lo_bound`

A numerically large lower bound value is specified.

`rescode.wrn_large_up_bound`

A numerically large upper bound value is specified.

`rescode.wrn_license_expire`

The license expires.

`rescode.wrn_license_feature_expire`

The license expires.

`rescode.wrn_license_server`

The license server is not responding.

`rescode.wrn_lp_drop_variable`

Ignored a variable because the variable was not previously defined. Usually this implies that a variable appears in the bound section but not in the objective or the constraints.

rescode.wrn_lp_old_quad_format

Missing '/2' after quadratic expressions in bound or objective.

rescode.wrn_mio_infeasible_final

The final mixed-integer problem with all the integer variables fixed at their optimal values is infeasible.

rescode.wrn_mps_split_bou_vector

A BOUNDS vector is split into several nonadjacent parts in an MPS file.

rescode.wrn_mps_split_ran_vector

A RANGE vector is split into several nonadjacent parts in an MPS file.

rescode.wrn_mps_split_rhs_vector

An RHS vector is split into several nonadjacent parts in an MPS file.

rescode.wrn_name_max_len

A name is longer than the buffer that is supposed to hold it.

rescode.wrn_no_dualizer

No automatic dualizer is available for the specified problem. The primal problem is solved.

rescode.wrn_no_global_optimizer

No global optimizer is available.

rescode.wrn_no_nonlinear_function_write

The problem contains a general nonlinear function in either the objective or the constraints. Such a nonlinear function cannot be written to a disk file. Note that quadratic terms when inputted explicitly can be written to disk.

rescode.wrn_nz_in_upr_tri

Non-zero elements specified in the upper triangle of a matrix were ignored.

rescode.wrn_open_param_file

The parameter file could not be opened.

rescode.wrn_param_ignored_cmio

A parameter was ignored by the conic mixed integer optimizer.

rescode.wrn_param_name_dou

The parameter name is not recognized as a double parameter.

rescode.wrn_param_name_int

The parameter name is not recognized as a integer parameter.

rescode.wrn_param_name_str

The parameter name is not recognized as a string parameter.

`rescode.wrn_param_str_value`

The string is not recognized as a symbolic value for the parameter.

`rescode.wrn_presolve_outofspace`

The presolve is incomplete due to lack of space.

`rescode.wrn_quad_cones_with_root_fixed_at_zero`

For at least one quadratic cone the root is fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problems, or to fix all the variables in the cone to 0.

`rescode.wrn_rquad_cones_with_root_fixed_at_zero`

For at least one rotated quadratic cone at least one of the root variables are fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problems, or to fix all the variables in the cone to 0.

`rescode.wrn_sol_file_ignored_con`

One or more lines in the constraint section were ignored when reading a solution file.

`rescode.wrn_sol_file_ignored_var`

One or more lines in the variable section were ignored when reading a solution file.

`rescode.wrn_sol_filter`

Invalid solution filter is specified.

`rescode.wrn_spar_max_len`

A value for a string parameter is longer than the buffer that is supposed to hold it.

`rescode.wrn_too_few_basis_vars`

An incomplete basis has been specified. Too few basis variables are specified.

`rescode.wrn_too_many_basis_vars`

A basis with too many variables has been specified.

`rescode.wrn_too_many_threads_concurrent`

The concurrent optimizer employs more threads than available. This will lead to poor performance.

`rescode.wrn_undef_sol_file_name`

Undefined name occurred in a solution.

`rescode.wrn_using_generic_names`

Generic names are used because a name is not valid. For instance when writing an LP file the names must not contain blanks or start with a digit.

`rescode.wrn_write_changed_names`

Some names were changed because they were invalid for the output file format.

`rescode.wrn_write_discarded_cfix`

The fixed objective term could not be converted to a variable and was discarded in the output file.

`rescode.wrn_zero_aij`

One or more zero elements are specified in A.

`rescode.wrn_zeros_in_sparse_col`

One or more (near) zero elements are specified in a sparse column of a matrix. It is redundant to specify zero elements. Hence, it may indicate an error.

`rescode.wrn_zeros_in_sparse_row`

One or more (near) zero elements are specified in a sparse row of a matrix. It is redundant to specify zero elements. Hence it may indicate an error.

Appendix D

API constants

D.1 Constraint or variable access modes

`accmode.var`

Access data by columns (variable oriented)

`accmode.con`

Access data by rows (constraint oriented)

D.2 Basis identification

`basindtype.never`

Never do basis identification.

`basindtype.always`

Basis identification is always performed even if the interior-point optimizer terminates abnormally.

`basindtype.no_error`

Basis identification is performed if the interior-point optimizer terminates without an error.

`basindtype.if_feasible`

Basis identification is not performed if the interior-point optimizer terminates with a problem status saying that the problem is primal or dual infeasible.

`basindtype.reservered`

Not currently in use.

D.3 Bound keys

`boundkey.lo`

The constraint or variable has a finite lower bound and an infinite upper bound.

`boundkey.up`

The constraint or variable has an infinite lower bound and an finite upper bound.

`boundkey.fx`

The constraint or variable is fixed.

`boundkey.fr`

The constraint or variable is free.

`boundkey.ra`

The constraint or variable is ranged.

D.4 Specifies the branching direction.

`branchdir.free`

The mixed-integer optimizer decides which branch to choose.

`branchdir.up`

The mixed-integer optimizer always chooses the up branch first.

`branchdir.down`

The mixed-integer optimizer always chooses the down branch first.

D.5 Progress call-back codes

`callbackcode.begin_bi`

The basis identification procedure has been started.

`callbackcode.begin_concurrent`

Concurrent optimizer is started.

`callbackcode.begin_conic`

The call-back function is called when the conic optimizer is started.

`callbackcode.begin_dual_bi`

The call-back function is called from within the basis identification procedure when the dual phase is started.

`callbackcode.begin_dual_sensitivity`

Dual sensitivity analysis is started.

`callbackcode.begin_dual_setup_bi`

The call-back function is called when the dual BI phase is started.

`callbackcode.begin_dual_simplex`

The call-back function is called when the dual simplex optimizer started.

`callbackcode.begin_dual_simplex_bi`

The call-back function is called from within the basis identification procedure when the dual simplex clean-up phase is started.

`callbackcode.begin_full_convexity_check`

Begin full convexity check.

`callbackcode.begin_infeas_ana`

The call-back function is called when the infeasibility analyzer is started.

`callbackcode.begin_intpnt`

The call-back function is called when the interior-point optimizer is started.

`callbackcode.begin_license_wait`

Begin waiting for license.

`callbackcode.begin_mio`

The call-back function is called when the mixed-integer optimizer is started.

`callbackcode.begin_network_dual_simplex`

The call-back function is called when the dual network simplex optimizer is started.

`callbackcode.begin_network_primal_simplex`

The call-back function is called when the primal network simplex optimizer is started.

`callbackcode.begin_network_simplex`

The call-back function is called when the simplex network optimizer is started.

`callbackcode.begin_nonconvex`

The call-back function is called when the nonconvex optimizer is started.

`callbackcode.begin_optimizer`

The call-back function is called when the optimizer is started.

`callbackcode.begin_presolve`

The call-back function is called when the presolve is started.

`callbackcode.begin_primal_bi`

The call-back function is called from within the basis identification procedure when the primal phase is started.

`callbackcode.begin_primal_dual_simplex`

The call-back function is called when the primal-dual simplex optimizer is started.

`callbackcode.begin_primal_dual_simplex_bi`

The call-back function is called from within the basis identification procedure when the primal-dual simplex clean-up phase is started.

`callbackcode.begin_primal_repair`

Begin primal feasibility repair.

`callbackcode.begin_primal_sensitivity`

Primal sensitivity analysis is started.

`callbackcode.begin_primal_setup_bi`

The call-back function is called when the primal BI setup is started.

`callbackcode.begin_primal_simplex`

The call-back function is called when the primal simplex optimizer is started.

`callbackcode.begin_primal_simplex_bi`

The call-back function is called from within the basis identification procedure when the primal simplex clean-up phase is started.

`callbackcode.begin_qcqo_reformulate`

Begin QCQO reformulation.

`callbackcode.begin_read`

MOSEK has started reading a problem file.

`callbackcode.begin_simplex`

The call-back function is called when the simplex optimizer is started.

`callbackcode.begin_simplex_bi`

The call-back function is called from within the basis identification procedure when the simplex clean-up phase is started.

`callbackcode.begin_simplex_network_detect`

The call-back function is called when the network detection procedure is started.

`callbackcode.begin_write`

MOSEK has started writing a problem file.

`callbackcode.conic`

The call-back function is called from within the conic optimizer after the information database has been updated.

`callbackcode.dual_simplex`

The call-back function is called from within the dual simplex optimizer.

`callbackcode.end_bi`

The call-back function is called when the basis identification procedure is terminated.

`callbackcode.end_concurrent`

Concurrent optimizer is terminated.

`callbackcode.end_conic`

The call-back function is called when the conic optimizer is terminated.

`callbackcode.end_dual_bi`

The call-back function is called from within the basis identification procedure when the dual phase is terminated.

`callbackcode.end_dual_sensitivity`

Dual sensitivity analysis is terminated.

`callbackcode.end_dual_setup_bi`

The call-back function is called when the dual BI phase is terminated.

`callbackcode.end_dual_simplex`

The call-back function is called when the dual simplex optimizer is terminated.

`callbackcode.end_dual_simplex_bi`

The call-back function is called from within the basis identification procedure when the dual clean-up phase is terminated.

`callbackcode.end_full_convexity_check`

End full convexity check.

`callbackcode.end_infeas_ana`

The call-back function is called when the infeasibility analyzer is terminated.

`callbackcode.end_intpnt`

The call-back function is called when the interior-point optimizer is terminated.

`callbackcode.end_license_wait`

End waiting for license.

`callbackcode.end_mio`

The call-back function is called when the mixed-integer optimizer is terminated.

`callbackcode.end_network_dual_simplex`

The call-back function is called when the dual network simplex optimizer is terminated.

`callbackcode.end_network_primal_simplex`

The call-back function is called when the primal network simplex optimizer is terminated.

`callbackcode.end_network_simplex`

The call-back function is called when the simplex network optimizer is terminated.

`callbackcode.end_nonconvex`

The call-back function is called when the nonconvex optimizer is terminated.

`callbackcode.end_optimizer`

The call-back function is called when the optimizer is terminated.

`callbackcode.end_presolve`

The call-back function is called when the presolve is completed.

`callbackcode.end_primal_bi`

The call-back function is called from within the basis identification procedure when the primal phase is terminated.

`callbackcode.end_primal_dual_simplex`

The call-back function is called when the primal-dual simplex optimizer is terminated.

`callbackcode.end_primal_dual_simplex_bi`

The call-back function is called from within the basis identification procedure when the primal-dual clean-up phase is terminated.

`callbackcode.end_primal_repair`

End primal feasibility repair.

`callbackcode.end_primal_sensitivity`

Primal sensitivity analysis is terminated.

`callbackcode.end_primal_setup_bi`

The call-back function is called when the primal BI setup is terminated.

`callbackcode.end_primal_simplex`

The call-back function is called when the primal simplex optimizer is terminated.

`callbackcode.end_primal_simplex_bi`

The call-back function is called from within the basis identification procedure when the primal clean-up phase is terminated.

`callbackcode.end_qcqo_reformulate`

End QCQO reformulation.

`callbackcode.end_read`

MOSEK has finished reading a problem file.

`callbackcode.end_simplex`

The call-back function is called when the simplex optimizer is terminated.

`callbackcode.end_simplex_bi`

The call-back function is called from within the basis identification procedure when the simplex clean-up phase is terminated.

`callbackcode.end_simplex_network_detect`

The call-back function is called when the network detection procedure is terminated.

`callbackcode.end_write`

MOSEK has finished writing a problem file.

`callbackcode.im_bi`

The call-back function is called from within the basis identification procedure at an intermediate point.

`callbackcode.im_conic`

The call-back function is called at an intermediate stage within the conic optimizer where the information database has not been updated.

`callbackcode.im_dual_bi`

The call-back function is called from within the basis identification procedure at an intermediate point in the dual phase.

`callbackcode.im_dual_sensivity`

The call-back function is called at an intermediate stage of the dual sensitivity analysis.

`callbackcode.im_dual_simplex`

The call-back function is called at an intermediate point in the dual simplex optimizer.

`callbackcode.im_full_convexity_check`

The call-back function is called at an intermediate stage of the full convexity check.

`callbackcode.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.

`callbackcode.im_license_wait`

MOSEK is waiting for a license.

`callbackcode.im_lu`

The call-back function is called from within the LU factorization procedure at an intermediate point.

callbackcode.im_mio

The call-back function is called at an intermediate point in the mixed-integer optimizer.

callbackcode.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.

callbackcode.im_mio_intpnt

The call-back function is called at an intermediate point in the mixed-integer optimizer while running the interior-point optimizer.

callbackcode.im_mio_presolve

The call-back function is called at an intermediate point in the mixed-integer optimizer while running the presolve.

callbackcode.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.

callbackcode.im_network_dual_simplex

The call-back function is called at an intermediate point in the dual network simplex optimizer.

callbackcode.im_network_primal_simplex

The call-back function is called at an intermediate point in the primal network simplex optimizer.

callbackcode.im_nonconvex

The call-back function is called at an intermediate stage within the nonconvex optimizer where the information database has not been updated.

callbackcode.im_order

The call-back function is called from within the matrix ordering procedure at an intermediate point.

callbackcode.im_presolve

The call-back function is called from within the presolve procedure at an intermediate stage.

callbackcode.im_primal_bi

The call-back function is called from within the basis identification procedure at an intermediate point in the primal phase.

callbackcode.im_primal_dual_simplex

The call-back function is called at an intermediate point in the primal-dual simplex optimizer.

callbackcode.im_primal_sensivity

The call-back function is called at an intermediate stage of the primal sensitivity analysis.

`callbackcode.im_primal_simplex`

The call-back function is called at an intermediate point in the primal simplex optimizer.

`callbackcode.im_qo_reformulate`

The call-back function is called at an intermediate stage of the conic quadratic reformulation.

`callbackcode.im_read`

Intermediate stage in reading.

`callbackcode.im_simplex`

The call-back function is called from within the simplex optimizer at an intermediate point.

`callbackcode.im_simplex_bi`

The call-back function is called from within the basis identification procedure at an intermediate point in the simplex clean-up phase. The frequency of the call-backs is controlled by the `iparam.log_sim_freq` parameter.

`callbackcode.intpnt`

The call-back function is called from within the interior-point optimizer after the information database has been updated.

`callbackcode.new_int_mio`

The call-back function is called after a new integer solution has been located by the mixed-integer optimizer.

`callbackcode.noncovex`

The call-back function is called from within the nonconvex optimizer after the information database has been updated.

`callbackcode.primal_simplex`

The call-back function is called from within the primal simplex optimizer.

`callbackcode.read_opf`

The call-back function is called from the OPF reader.

`callbackcode.read_opf_section`

A chunk of Q non-zeos has been read from a problem file.

`callbackcode.update_dual_bi`

The call-back function is called from within the basis identification procedure at an intermediate point in the dual phase.

`callbackcode.update_dual_simplex`

The call-back function is called in the dual simplex optimizer.

callbackcode.update_dual_simplex_bi

The call-back function is called from within the basis identification procedure at an intermediate point in the dual simplex clean-up phase. The frequency of the call-backs is controlled by the iparam.log_sim_freq parameter.

callbackcode.update_network_dual_simplex

The call-back function is called in the dual network simplex optimizer.

callbackcode.update_network_primal_simplex

The call-back function is called in the primal network simplex optimizer.

callbackcode.update_nonconvex

The call-back function is called at an intermediate stage within the nonconvex optimizer where the information database has been updated.

callbackcode.update_presolve

The call-back function is called from within the presolve procedure.

callbackcode.update_primal_bi

The call-back function is called from within the basis identification procedure at an intermediate point in the primal phase.

callbackcode.update_primal_dual_simplex

The call-back function is called in the primal-dual simplex optimizer.

callbackcode.update_primal_dual_simplex_bi

The call-back function is called from within the basis identification procedure at an intermediate point in the primal-dual simplex clean-up phase. The frequency of the call-backs is controlled by the iparam.log_sim_freq parameter.

callbackcode.update_primal_simplex

The call-back function is called in the primal simplex optimizer.

callbackcode.update_primal_simplex_bi

The call-back function is called from within the basis identification procedure at an intermediate point in the primal simplex clean-up phase. The frequency of the call-backs is controlled by the iparam.log_sim_freq parameter.

callbackcode.write_opf

The call-back function is called from the OPF writer.

D.6 Types of convexity checks.

`checkconvexitytype.none`

No convexity check.

`checkconvexitytype.simple`

Perform simple and fast convexity check.

`checkconvexitytype.full`

Perform a full convexity check.

D.7 Compression types

`compresstype.none`

No compression is used.

`compresstype.free`

The type of compression used is chosen automatically.

`compresstype.gzip`

The type of compression used is gzip compatible.

D.8 Cone types

`conetype.quad`

The cone is a quadratic cone.

`conetype.rquad`

The cone is a rotated quadratic cone.

D.9 Data format types

`dataformat.extension`

The file extension is used to determine the data file format.

`dataformat.mps`

The data file is MPS formatted.

`dataformat.lp`

The data file is LP formatted.

dataformat.op

The data file is an optimization problem formatted file.

dataformat.xml

The data file is an XML formatted file.

dataformat.free_mps

The data data a free MPS formatted file.

dataformat.task

Generic task dump file.

dataformat.cb

Conic benchmark format.

D.10 Double information items

dinfitem.bi_clean_dual_time

Time spent within the dual clean-up optimizer of the basis identification procedure since its invocation.

dinfitem.bi_clean_primal_dual_time

Time spent within the primal-dual clean-up optimizer of the basis identification procedure since its invocation.

dinfitem.bi_clean_primal_time

Time spent within the primal clean-up optimizer of the basis identification procedure since its invocation.

dinfitem.bi_clean_time

Time spent within the clean-up phase of the basis identification procedure since its invocation.

dinfitem.bi_dual_time

Time spent within the dual phase basis identification procedure since its invocation.

dinfitem.bi_primal_time

Time spent within the primal phase of the basis identification procedure since its invocation.

dinfitem.bi_time

Time spent within the basis identification procedure since its invocation.

dinfitem.concurrent_time

Time spent within the concurrent optimizer since its invocation.

`dinfitem.intpnt_dual_feas`

Dual feasibility measure reported by the interior-point optimizer. (For the interior-point optimizer this measure does not directly related to the original problem because a homogeneous model is employed.)

`dinfitem.intpnt_dual_obj`

Dual objective value reported by the interior-point optimizer.

`dinfitem.intpnt_factor_num_flops`

An estimate of the number of flops used in the factorization.

`dinfitem.intpnt_opt_status`

This measure should converge to +1 if the problem has a primal-dual optimal solution, and converge to -1 if problem is (strictly) primal or dual infeasible. Furthermore, if the measure converges to 0 the problem is usually ill-posed.

`dinfitem.intpnt_order_time`

Order time (in seconds).

`dinfitem.intpnt_primal_feas`

Primal feasibility measure reported by the interior-point optimizers. (For the interior-point optimizer this measure does not directly related to the original problem because a homogeneous model is employed).

`dinfitem.intpnt_primal_obj`

Primal objective value reported by the interior-point optimizer.

`dinfitem.intpnt_time`

Time spent within the interior-point optimizer since its invocation.

`dinfitem.mio_cg_seperation_time`

Seperation time for CG cuts.

`dinfitem.mio_cmir_seperation_time`

Seperation time for CMIR cuts.

`dinfitem.mio_construct_solution_obj`

If MOSEK has successfully constructed an integer feasible solution, then this item contains the optimal objective value corresponding to the feasible solution.

`dinfitem.mio_dual_bound_after_presolve`

Value of the dual bound after presolve but before cut generation.

`dinfitem.mio_heuristic_time`

Time spent in the optimizer while solving the relaxtions.

`dinfitem.mio_obj_abs_gap`

Given the mixed-integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the absolute gap defined by

$$|(\text{objective value of feasible solution}) - (\text{objective bound})|.$$

Otherwise it has the value -1.0.

`dinfitem.mio_obj_bound`

The best known bound on the objective function. This value is undefined until at least one relaxation has been solved: To see if this is the case check that `iinfitem.mio_num_relax` is stricly positive.

`dinfitem.mio_obj_int`

The primal objective value corresponding to the best integer feasible solution. Please note that at least one integer feasible solution must have located i.e. check `iinfitem.mio_num_int_solutions`.

`dinfitem.mio_obj_rel_gap`

Given that the mixed-integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the relative gap defined by

$$\frac{|(\text{objective value of feasible solution}) - (\text{objective bound})|}{\max(\delta, |(\text{objective value of feasible solution})|)}.$$

where δ is given by the paramater `dparam.mio_rel_gap_const`. Otherwise it has the value -1.0.

`dinfitem.mio_optimizer_time`

Time spent in the optimizer while solving the relaxtions.

`dinfitem.mio_probing_time`

Total time for probing.

`dinfitem.mio_root_cutgen_time`

Total time for cut generation.

`dinfitem.mio_root_optimizer_time`

Time spent in the optimizer while solving the root relaxation.

`dinfitem.mio_root_presolve_time`

Time spent in while presolveing the root relaxation.

`dinfitem.mio_time`

Time spent in the mixed-integer optimizer.

`dinfitem.mio_user_obj_cut`

If the objective cut is used, then this information item has the value of the cut.

`dinfitem.optimizer_time`

Total time spent in the optimizer since it was invoked.

`dinfitem.presolve_eli_time`

Total time spent in the eliminator since the presolve was invoked.

`dinfitem.presolve_lindep_time`

Total time spent in the linear dependency checker since the presolve was invoked.

`dinfitem.presolve_time`

Total time (in seconds) spent in the presolve since it was invoked.

`dinfitem.primal_repair_penalty_obj`

The optimal objective value of the penalty function.

`dinfitem.qcqo_reformulate_time`

Time spent with conic quadratic reformulation.

`dinfitem.rd_time`

Time spent reading the data file.

`dinfitem.sim_dual_time`

Time spent in the dual simplex optimizer since invoking it.

`dinfitem.sim_feas`

Feasibility measure reported by the simplex optimizer.

`dinfitem.sim_network_dual_time`

Time spent in the dual network simplex optimizer since invoking it.

`dinfitem.sim_network_primal_time`

Time spent in the primal network simplex optimizer since invoking it.

`dinfitem.sim_network_time`

Time spent in the network simplex optimizer since invoking it.

`dinfitem.sim_obj`

Objective value reported by the simplex optimizer.

`dinfitem.sim_primal_dual_time`

Time spent in the primal-dual simplex optimizer optimizer since invoking it.

`dinfitem.sim_primal_time`

Time spent in the primal simplex optimizer since invoking it.

`dinfitem.sim_time`

Time spent in the simplex optimizer since invoking it.

dinfitem.sol_bas_dual_obj

Dual objective value of the basic solution.

dinfitem.sol_bas_dviolcon

Maximal dual bound violation for x^c in the basic solution.

dinfitem.sol_bas_dviolvar

Maximal dual bound violation for x^x in the basic solution.

dinfitem.sol_bas_primal_obj

Primal objective value of the basic solution.

dinfitem.sol_bas_pviolcon

Maximal primal bound violation for x^c in the basic solution.

dinfitem.sol_bas_pviolvar

Maximal primal bound violation for x^x in the basic solution.

dinfitem.sol_itg_primal_obj

Primal objective value of the integer solution.

dinfitem.sol_itg_pviolbarvar

Maximal primal bound violation for $\bar{X}$ in the integer solution.

dinfitem.sol_itg_pviolcon

Maximal primal bound violation for x^c in the integer solution.

dinfitem.sol_itg_pviolcones

Maximal primal violation for primal conic constraints in the integer solution.

dinfitem.sol_itg_pviolitg

Maximal violation for the integer constraints in the integer solution.

dinfitem.sol_itg_pviolvar

Maximal primal bound violation for x^x in the integer solution.

dinfitem.sol_itr_dual_obj

Dual objective value of the interior-point solution.

dinfitem.sol_itr_dviolbarvar

Maximal dual bound violation for $\bar{X}$ in the interior-point solution.

dinfitem.sol_itr_dviolcon

Maximal dual bound violation for x^c in the interior-point solution.

dinfitem.sol_itr_dviolcones

Maximal dual violation for dual conic constraints in the interior-point solution.

`dinfitem.sol_itr_dviolvar`

Maximal dual bound violation for x^x in the interior-point solution.

`dinfitem.sol_itr_primal_obj`

Primal objective value of the interior-point solution.

`dinfitem.sol_itr_pviolbarvar`

Maximal primal bound violation for $\bar{X}$ in the interior-point solution.

`dinfitem.sol_itr_pviolcon`

Maximal primal bound violation for x^c in the interior-point solution.

`dinfitem.sol_itr_pviolcones`

Maximal primal violation for primal conic constraints in the interior-point solution.

`dinfitem.sol_itr_pviolvar`

Maximal primal bound violation for x^x in the interior-point solution.

D.11 Feasibility repair types

`feasrepairtype.optimize_none`

Do not optimize the feasibility repair problem.

`feasrepairtype.optimize_penalty`

Minimize weighted sum of violations.

`feasrepairtype.optimize_combined`

Minimize with original objective subject to minimal weighted violation of bounds.

D.12 License feature

`feature.pts`

Base system.

`feature.pton`

Nonlinear extension.

`feature.ptom`

Mixed-integer extension.

`feature.ptox`

Non-convex extension.

D.13 Integer information items.

`iinfitem.ana_pro_num_con`

Number of constraints in the problem.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_con_eq`

Number of equality constraints.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_con_fr`

Number of unbounded constraints.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_con_lo`

Number of constraints with a lower bound and an infinite upper bound.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_con_ra`

Number of constraints with finite lower and upper bounds.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_con_up`

Number of constraints with an upper bound and an infinite lower bound.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_var`

Number of variables in the problem.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_var_bin`

Number of binary (0-1) variables.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_var_cont`

Number of continuous variables.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_var_eq`

Number of fixed variables.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_var_fr`

Number of free variables.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_var_int`

Number of general integer variables.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_var_lo`

Number of variables with a lower bound and an infinite upper bound.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_var_ra`

Number of variables with finite lower and upper bounds.

This value is set by `Task.analyzeproblem`.

`iinfitem.ana_pro_num_var_up`

Number of variables with an upper bound and an infinite lower bound. This value is set by

This value is set by `Task.analyzeproblem`.

`iinfitem.concurrent_fastest_optimizer`

The type of the optimizer that finished first in a concurrent optimization.

`iinfitem.intpnt_factor_dim_dense`

Dimension of the dense sub system in factorization.

`iinfitem.intpnt_iter`

Number of interior-point iterations since invoking the interior-point optimizer.

`iinfitem.intpnt_num_threads`

Number of threads that the interior-point optimizer is using.

`iinfitem.intpnt_solve_dual`

Non-zero if the interior-point optimizer is solving the dual problem.

`iinfitem.mio_construct_num_roundings`

Number of values in the integer solution that is rounded to an integer value.

`iinfitem.mio_construct_solution`

If this item has the value 0, then MOSEK did not try to construct an initial integer feasible solution. If the item has a positive value, then MOSEK successfully constructed an initial integer feasible solution.

`iinfitem.mio_initial_solution`

Is non-zero if an initial integer solution is specified.

iinfitem.mio_num_active_nodes
: Number of active brabch bound nodes.

iinfitem.mio_num_basis_cuts
: Number of basis cuts.

iinfitem.mio_num_branch
: Number of branches performed during the optimization.

iinfitem.mio_num_cardgub_cuts
: Number of cardgub cuts.

iinfitem.mio_num_clique_cuts
: Number of clique cuts.

iinfitem.mio_num_coef_redc_cuts
: Number of coef. redc. cuts.

iinfitem.mio_num_contra_cuts
: Number of contra cuts.

iinfitem.mio_num_disagg_cuts
: Number of diasagg cuts.

iinfitem.mio_num_flow_cover_cuts
: Number of flow cover cuts.

iinfitem.mio_num_gcd_cuts
: Number of gcd cuts.

iinfitem.mio_num_gomory_cuts
: Number of Gomory cuts.

iinfitem.mio_num_gub_cover_cuts
: Number of GUB cover cuts.

iinfitem.mio_num_int_solutions
: Number of integer feasible solutions that has been found.

iinfitem.mio_num_knapsur_cover_cuts
: Number of knapsack cover cuts.

iinfitem.mio_num_lattice_cuts
: Number of lattice cuts.

iinfitem.mio_num_lift_cuts
: Number of lift cuts.

`iinfitem.mio_num_obj_cuts`

Number of obj cuts.

`iinfitem.mio_num_plan_loc_cuts`

Number of loc cuts.

`iinfitem.mio_num_relax`

Number of relaxations solved during the optimization.

`iinfitem.mio_numcon`

Number of constraints in the problem solved be the mixed-integer optimizer.

`iinfitem.mio_numint`

Number of integer variables in the problem solved be the mixed-integer optimizer.

`iinfitem.mio_numvar`

Number of variables in the problem solved be the mixed-integer optimizer.

`iinfitem.mio_obj_bound_defined`

Non-zero if a valid objective bound has been found, otherwise zero.

`iinfitem.mio_total_num_cuts`

Total number of cuts generated by the mixed-integer optimizer.

`iinfitem.mio_user_obj_cut`

If it is non-zero, then the objective cut is used.

`iinfitem.opt_numcon`

Number of constraints in the problem solved when the optimizer is called.

`iinfitem.opt_numvar`

Number of variables in the problem solved when the optimizer is called

`iinfitem.optimize_response`

The reponse code returned by optimize.

`iinfitem.rd_numbarvar`

Number of variables read.

`iinfitem.rd_numcon`

Number of constraints read.

`iinfitem.rd_numcone`

Number of conic constraints read.

`iinfitem.rd_numintvar`

Number of integer-constrained variables read.

`iinfitem.rd_numq`

Number of nonempty Q matrixes read.

`iinfitem.rd_numvar`

Number of variables read.

`iinfitem.rd_protype`

Problem type.

`iinfitem.sim_dual_deg_iter`

The number of dual degenerate iterations.

`iinfitem.sim_dual_hotstart`

If 1 then the dual simplex algorithm is solving from an advanced basis.

`iinfitem.sim_dual_hotstart_lu`

If 1 then a valid basis factorization of full rank was located and used by the dual simplex algorithm.

`iinfitem.sim_dual_inf_iter`

The number of iterations taken with dual infeasibility.

`iinfitem.sim_dual_iter`

Number of dual simplex iterations during the last optimization.

`iinfitem.sim_network_dual_deg_iter`

The number of dual network degenerate iterations.

`iinfitem.sim_network_dual_hotstart`

If 1 then the dual network simplex algorithm is solving from an advanced basis.

`iinfitem.sim_network_dual_hotstart_lu`

If 1 then a valid basis factorization of full rank was located and used by the dual network simplex algorithm.

`iinfitem.sim_network_dual_inf_iter`

The number of iterations taken with dual infeasibility in the network optimizer.

`iinfitem.sim_network_dual_iter`

Number of dual network simplex iterations during the last optimization.

`iinfitem.sim_network_primal_deg_iter`

The number of primal network degenerate iterations.

`iinfitem.sim_network_primal_hotstart`

If 1 then the primal network simplex algorithm is solving from an advanced basis.

`iinfitem.sim_network_primal_hotstart_lu`

If 1 then a valid basis factorization of full rank was located and used by the primal network simplex algorithm.

`iinfitem.sim_network_primal_inf_iter`

The number of iterations taken with primal infeasibility in the network optimizer.

`iinfitem.sim_network_primal_iter`

Number of primal network simplex iterations during the last optimization.

`iinfitem.sim_numcon`

Number of constraints in the problem solved by the simplex optimizer.

`iinfitem.sim_numvar`

Number of variables in the problem solved by the simplex optimizer.

`iinfitem.sim_primal_deg_iter`

The number of primal degenerate iterations.

`iinfitem.sim_primal_dual_deg_iter`

The number of degenerate major iterations taken by the primal dual simplex algorithm.

`iinfitem.sim_primal_dual_hotstart`

If 1 then the primal dual simplex algorithm is solving from an advanced basis.

`iinfitem.sim_primal_dual_hotstart_lu`

If 1 then a valid basis factorization of full rank was located and used by the primal dual simplex algorithm.

`iinfitem.sim_primal_dual_inf_iter`

The number of master iterations with dual infeasibility taken by the primal dual simplex algorithm.

`iinfitem.sim_primal_dual_iter`

Number of primal dual simplex iterations during the last optimization.

`iinfitem.sim_primal_hotstart`

If 1 then the primal simplex algorithm is solving from an advanced basis.

`iinfitem.sim_primal_hotstart_lu`

If 1 then a valid basis factorization of full rank was located and used by the primal simplex algorithm.

`iinfitem.sim_primal_inf_iter`

The number of iterations taken with primal infeasibility.

`iinfitem.sim_primal_iter`

Number of primal simplex iterations during the last optimization.

`iinfitem.sim_solve_dual`

Is non-zero if dual problem is solved.

`iinfitem.sol_bas_prosta`

Problem status of the basic solution. Updated after each optimization.

`iinfitem.sol_bas_solsta`

Solution status of the basic solution. Updated after each optimization.

`iinfitem.sol_int_prosta`

Deprecated.

`iinfitem.sol_int_solsta`

Degrecated.

`iinfitem.sol_itg_prosta`

Problem status of the integer solution. Updated after each optimization.

`iinfitem.sol_itg_solsta`

Solution status of the integer solution. Updated after each optimization.

`iinfitem.sol_itr_prosta`

Problem status of the interior-point solution. Updated after each optimization.

`iinfitem.sol_itr_solsta`

Solution status of the interior-point solution. Updated after each optimization.

`iinfitem.sto_num_a_cache_flushes`

Number of times the cache of A elements is flushed. A large number implies that `maxnumanz` is too small as well as an inefficient usage of MOSEK.

`iinfitem.sto_num_a_realloc`

Number of times the storage for storing A has been changed. A large value may indicates that memory fragmentation may occur.

`iinfitem.sto_num_a_transposes`

Number of times the A matrix is transposed. A large number implies that `maxnumanz` is too small or an inefficient usage of MOSEK. This will occur in particular if the code alternate between accessing rows and columns of A.

D.14 Information item types

`inftype.dou_type`

Is a double information type.

`inftype.int_type`

Is an integer.

`inftype.lint_type`

Is a long integer.

D.15 Hot-start type employed by the interior-point optimizers.

`intpnthotstart.none`

The interior-point optimizer performs a coldstart.

`intpnthotstart.primal`

The interior-point optimizer exploits the primal solution only.

`intpnthotstart.dual`

The interior-point optimizer exploits the dual solution only.

`intpnthotstart.primal_dual`

The interior-point optimizer exploits both the primal and dual solution.

D.16 Input/output modes

`iomode.read`

The file is read-only.

`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.

`iomode.readwrite`

The file is to read and written.

D.17 Language selection constants

`language.eng`

English language selection

`language.dan`

Danish language selection

D.18 Long integer information items.

`liinfitem.bi_clean_dual_deg_iter`

Number of dual degenerate clean iterations performed in the basis identification.

`liinfitem.bi_clean_dual_iter`

Number of dual clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_deg_iter`

Number of primal degenerate clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_dual_deg_iter`

Number of primal-dual degenerate clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_dual_iter`

Number of primal-dual clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_dual_sub_iter`

Number of primal-dual subproblem clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_iter`

Number of primal clean iterations performed in the basis identification.

`liinfitem.bi_dual_iter`

Number of dual pivots performed in the basis identification.

`liinfitem.bi_primal_iter`

Number of primal pivots performed in the basis identification.

`liinfitem.intpnt_factor_num_nz`

Number of non-zeros in factorization.

`liinfitem.mio_intpnt_iter`

Number of interior-point iterations performed by the mixed-integer optimizer.

`liinfitem.mio_simplex_iter`

Number of simplex iterations performed by the mixed-integer optimizer.

`liinfitem.rd_numanz`

Number of non-zeros in A that is read.

`liinfitem.rd_numqnz`

Number of Q non-zeros.

D.19 Mark

`mark.lo`

The lower bound is selected for sensitivity analysis.

`mark.up`

The upper bound is selected for sensitivity analysis.

D.20 Continuous mixed-integer solution type

`miocontsoltype.none`

No interior-point or basic solution are reported when the mixed-integer optimizer is used.

`miocontsoltype.root`

The reported interior-point and basic solutions are a solution to the root node problem when mixed-integer optimizer is used.

`miocontsoltype.itg`

The reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. A solution is only reported in case the problem has a primal feasible solution.

`miocontsoltype.itg_rel`

In case the problem is primal feasible then the reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. If the problem is primal infeasible, then the solution to the root node problem is reported.

D.21 Integer restrictions

`miomode.ignored`

The integer constraints are ignored and the problem is solved as a continuous problem.

`miomode.satisfied`

Integer restrictions should be satisfied.

`miomode.lazy`

Integer restrictions should be satisfied if an optimizer is available for the problem.

D.22 Mixed-integer node selection types

`mionodeseltype.free`

The optimizer decides the node selection strategy.

`mionodeseltype.first`

The optimizer employs a depth first node selection strategy.

`mionodeseltype.best`

The optimizer employs a best bound node selection strategy.

`mionodeseltype.worst`

The optimizer employs a worst bound node selection strategy.

`mionodeseltype.hybrid`

The optimizer employs a hybrid strategy.

`mionodeseltype.pseudo`

The optimizer employs selects the node based on a pseudo cost estimate.

D.23 MPS file format type

`mpsformat.strict`

It is assumed that the input file satisfies the MPS format strictly.

`mpsformat.relaxed`

It is assumed that the input file satisfies a slightly relaxed version of the MPS format.

`mpsformat.free`

It is assumed that the input file satisfies the free MPS format. This implies that spaces are not allowed in names. Otherwise the format is free.

D.24 Message keys

`msgkey.reading_file`

`msgkey.writing_file`

`msgkey.mps_selected`

D.25 Name types

`nametype.gen`

General names. However, no duplicate and blank names are allowed.

`nametype.mps`

MPS type names.

`nametype.lp`

LP type names.

D.26 Objective sense types

`objsense.minimize`

The problem should be minimized.

`objsense.maximize`

The problem should be maximized.

D.27 On/off

`onoffkey.off`

Switch the option off.

`onoffkey.on`

Switch the option on.

D.28 Optimizer types

`optimizertype.free`

The optimizer is chosen automatically.

`optimizertype.intpnt`

The interior-point optimizer is used.

`optimizertype.conic`

The optimizer for problems having conic constraints.

`optimizertype.primal_simplex`

The primal simplex optimizer is used.

`optimizertype.dual_simplex`

The dual simplex optimizer is used.

`optimizertype.primal_dual_simplex`

The primal dual simplex optimizer is used.

`optimizertype.free_simplex`

One of the simplex optimizers is used.

`optimizertype.network_primal_simplex`

The network primal simplex optimizer is used. It is only applicable to pure network problems.

`optimizertype.mixed_int_conic`

The mixed-integer optimizer for conic and linear problems.

`optimizertype.mixed_int`

The mixed-integer optimizer.

`optimizertype.concurrent`

The optimizer for nonconvex nonlinear problems.

`optimizertype.nonconvex`

The optimizer for nonconvex nonlinear problems.

D.29 Ordering strategies

`orderingtype.free`

The ordering method is chosen automatically.

`orderingtype.appminloc`

Approximate minimum local fill-in ordering is employed.

`orderingtype.experimental`

This option should not be used.

`orderingtype.try_graphpar`

Always try the the graph partitioning based ordering.

`orderingtype.force_graphpar`

Always use the graph partitioning based ordering even if it is worse that the approximate minimum local fill ordering.

`orderingtype.none`

No ordering is used.

D.30 Parameter type

`parametertype.invalid_type`

Not a valid parameter.

`parametertype.dou_type`

Is a double parameter.

`parametertype.int_type`

Is an integer parameter.

`parametertype.str_type`

Is a string parameter.

D.31 Presolve method.

`presolvemode.off`

The problem is not presolved before it is optimized.

`presolvemode.on`

The problem is presolved before it is optimized.

`presolvemode.free`

It is decided automatically whether to presolve before the problem is optimized.

D.32 Problem data items

`problemitem.var`

Item is a variable.

`problemitem.con`

Item is a constraint.

`problemitem.cone`

Item is a cone.

D.33 Problem types

`problemtype.lo`

The problem is a linear optimization problem.

`problemtype.qo`

The problem is a quadratic optimization problem.

`problemtype.qcqo`

The problem is a quadratically constrained optimization problem.

`problemtype.geco`

General convex optimization.

`problemtype.conic`

A conic optimization.

`problemtype.mixed`

General nonlinear constraints and conic constraints. This combination can not be solved by MOSEK.

D.34 Problem status keys

`prosta.unknown`

Unknown problem status.

`prosta.prim_and_dual_feas`

The problem is primal and dual feasible.

`prosta.prim_feas`

The problem is primal feasible.

`prosta.dual_feas`

The problem is dual feasible.

`prosta.prim_infeas`

The problem is primal infeasible.

`prosta.dual_infeas`

The problem is dual infeasible.

`prosta.prim_and_dual_infeas`

The problem is primal and dual infeasible.

`prosta.ill_posed`

The problem is ill-posed. For example, it may be primal and dual feasible but have a positive duality gap.

`prosta.near_prim_and_dual_feas`

The problem is at least nearly primal and dual feasible.

`prosta.near_prim_feas`

The problem is at least nearly primal feasible.

`prosta.near_dual_feas`

The problem is at least nearly dual feasible.

`prosta.prim_infeas_or_unbounded`

The problem is either primal infeasible or unbounded. This may occur for mixed-integer problems.

D.35 Response code type

`rescodetype.ok`

The response code is OK.

`rescodetype.wrn`

The response code is a warning.

`rescodetype.trm`

The response code is an optimizer termination status.

`rescodetype.err`

The response code is an error.

`rescodetype.unk`

The response code does not belong to any class.

D.36 Scaling type

`scalingmethod.pow2`

Scales only with power of 2 leaving the mantissa untouched.

`scalingmethod.free`

The optimizer chooses the scaling heuristic.

D.37 Scaling type

`scalingtype.free`

The optimizer chooses the scaling heuristic.

`scalingtype.none`

No scaling is performed.

`scalingtype.moderate`

A conservative scaling is performed.

`scalingtype.aggressive`

A very aggressive scaling is performed.

D.38 Sensitivity types

`sensitivitytype.basis`

Basis sensitivity analysis is performed.

`sensitivitytype.optimal_partition`

Optimal partition sensitivity analysis is performed.

D.39 Degeneracy strategies

`simdegen.none`

The simplex optimizer should use no degeneration strategy.

`simdegen.free`

The simplex optimizer chooses the degeneration strategy.

`simdegen.aggressive`

The simplex optimizer should use an aggressive degeneration strategy.

`simdegen.moderate`

The simplex optimizer should use a moderate degeneration strategy.

`simdegen.minimum`

The simplex optimizer should use a minimum degeneration strategy.

D.40 Exploit duplicate columns.

`simdupvec.off`

Disallow the simplex optimizer to exploit duplicated columns.

`simdupvec.on`

Allow the simplex optimizer to exploit duplicated columns.

`simdupvec.free`

The simplex optimizer can choose freely.

D.41 Hot-start type employed by the simplex optimizer

`simhotstart.none`

The simplex optimizer performs a coldstart.

`simhotstart.free`

The simplex optimize chooses the hot-start type.

`simhotstart.status_keys`

Only the status keys of the constraints and variables are used to choose the type of hot-start.

D.42 Problem reformulation.

`simreform.off`

Disallow the simplex optimizer to reformulate the problem.

`simreform.on`

Allow the simplex optimizer to reformulate the problem.

`simreform.free`

The simplex optimizer can choose freely.

`simreform.aggressive`

The simplex optimizer should use an aggressive reformulation strategy.

D.43 Simplex selection strategy

`simseltype.free`

The optimizer chooses the pricing strategy.

`simseltype.full`

The optimizer uses full pricing.

`simseltype.ase`

The optimizer uses approximate steepest-edge pricing.

`simseltype.devex`

The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).

`simseltype.se`

The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

`simseltype.partial`

The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.

D.44 Solution items

`solitem.xc`

Solution for the constraints.

`solitem.xx`

Variable solution.

`solitem.y`

Lagrange multipliers for equations.

`solitem.slc`

Lagrange multipliers for lower bounds on the constraints.

`solitem.suc`

Lagrange multipliers for upper bounds on the constraints.

`solitem.slx`

Lagrange multipliers for lower bounds on the variables.

`solitem.sux`

Lagrange multipliers for upper bounds on the variables.

`solitem.snx`

Lagrange multipliers corresponding to the conic constraints on the variables.

D.45 Solution status keys

`solsta.unknown`

Status of the solution is unknown.

`solsta.optimal`

The solution is optimal.

`solsta.prim_feas`

The solution is primal feasible.

`solsta.dual_feas`

The solution is dual feasible.

`solsta.prim_and_dual_feas`

The solution is both primal and dual feasible.

`solsta.prim_infeas_cer`

The solution is a certificate of primal infeasibility.

`solsta.dual_infeas_cer`

The solution is a certificate of dual infeasibility.

`solsta.near_optimal`

The solution is nearly optimal.

`solsta.near_prim_feas`

The solution is nearly primal feasible.

`solsta.near_dual_feas`

The solution is nearly dual feasible.

`solsta.near_prim_and_dual_feas`

The solution is nearly both primal and dual feasible.

`solsta.near_prim_infeas_cer`

The solution is almost a certificate of primal infeasibility.

`solsta.near_dual_infeas_cer`

The solution is almost a certificate of dual infeasibility.

`solsta.integer_optimal`

The primal solution is integer optimal.

`solsta.near_integer_optimal`

The primal solution is near integer optimal.

D.46 Solution types

`soltype.itr`
The interior solution.

`soltype.bas`
The basic solution.

`soltype.itg`
The integer solution.

D.47 Solve primal or dual form

`solveform.free`
The optimizer is free to solve either the primal or the dual problem.

`solveform.primal`
The optimizer should solve the primal problem.

`solveform.dual`
The optimizer should solve the dual problem.

D.48 Status keys

`stakey.unk`
The status for the constraint or variable is unknown.

`stakey.bas`
The constraint or variable is in the basis.

`stakey.supbas`
The constraint or variable is super basic.

`stakey.low`
The constraint or variable is at its lower bound.

`stakey.upr`
The constraint or variable is at its upper bound.

`stakey.fix`
The constraint or variable is fixed.

`stakey.inf`
The constraint or variable is infeasible in the bounds.

D.49 Starting point types

`startpointtype.free`

The starting point is chosen automatically.

`startpointtype.guess`

The optimizer guesses a starting point.

`startpointtype.constant`

The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.

`startpointtype.satisfy_bounds`

The starting point is choosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

D.50 Stream types

`streamtype.log`

Log stream. Contains the aggregated contents of all other streams. This means that a message written to any other stream will also be written to this stream.

`streamtype.msg`

Message stream. Log information relating to performance and progress of the optimization is written to this stream.

`streamtype.err`

Error stream. Error messages are written to this stream.

`streamtype.wrn`

Warning stream. Warning messages are written to this stream.

D.51 Symmetric matrix types

`symmattype.sparse`

Sparse symmetric matrix.

D.52 Transposed matrix.

`transpose.no`

No transpose is applied.

`transpose.yes`

A transpose is applied.

D.53 Triangular part of a symmetric matrix.

`uplo.lo`

Lower part.

`uplo.up`

Upper part

D.54 Integer values

`value.license_buffer_length`

The length of a license key buffer.

`value.max_str_len`

Maximum string length allowed in MOSEK.

D.55 Variable types

`variabletype.type_cont`

Is a continuous variable.

`variabletype.type_int`

Is an integer variable.

D.56 XML writer output mode

`xmlwriteroutputtype.row`

Write in row order.

`xmlwriteroutputtype.col`

Write in column order.

Appendix E

Troubleshooting

When creating multiple tasks and running for a long time memory usage grows, and the Task and Env objects are never garbage collected.

The Task and Environment objects cannot always be automatically garbage collected by Python, when when they are no longer in use. There are two ways to ensure that they are destroyed. Use the `with`-statement:

```
with Task(env,0,0) as t:
    t.readdata("somefile.task")
    # use the task
```

This will ensure that the create Task is automatically destroyed when the `with`-statement exits.

Alternatively, call the `__del__()` method directly when the object should not be used anymore.

Appendix F

Mosek file formats

MOSEK supports a range of problem and solution formats. The Task formats is MOSEK's native binary format and it supports all features that MOSEK supports. OPF is the corresponding ASCII format and this supports nearly all features (everything except semidefinite problems). In general, the text formats are significantly slower to read, but they can be examined and edited directly in any text editor.

MOSEK supports GZIP compression of files. Problem files with an additional ".gz" extension are assumed to be compressed when read, and is automatically compressed when written. For example, a file called

```
problem.mps.gz
```

will be read as a GZIP compressed MPS file.

F.1 The MPS file format

MOSEK supports the standard MPS format with some extensions. For a detailed description of the MPS format see the book by Nazareth [2].

F.1.1 MPS file structure

The version of the MPS format supported by MOSEK allows specification of an optimization problem on the form

$$\begin{array}{ccccc} l^c & \leq & Ax + q(x) & \leq & u^c, \\ l^x & \leq & x & \leq & u^x, \\ & & x \in \mathcal{C}, & & \\ & & x_{\mathcal{J}} \text{ integer}, & & \end{array} \tag{F.1}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \to \mathbb{R}$ is a vector of quadratic functions. Hence,

  $$q_i(x) = 1/2 x^T Q^i x$$

  where it is assumed that

  $$Q^i = (Q^i)^T.$$

  Please note the explicit 1/2 in the quadratic term and that Q^i is required to be symmetric.
- $\mathcal{C}$ is a convex cone.
- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer-constrained variables.

An MPS file with one row and one column can be illustrated like this:

```
*        1         2         3         4         5         6
*23456789012345678901234567890123456789012345678901234567890
NAME          [name]
OBJSENSE
    [objsense]
OBJNAME
    [objname]
ROWS
 ?  [cname1]
COLUMNS
    [vname1]  [cname1]    [value1]     [vname3]  [value2]
RHS
    [name]    [cname1]    [value1]     [cname2]  [value2]
RANGES
    [name]    [cname1]    [value1]     [cname2]  [value2]
QSECTION      [cname1]
    [vname1]  [vname2]    [value1]     [vname3]  [value2]
BOUNDS
 ?? [name]    [vname1]    [value1]
CSECTION      [kname1]    [value1]     [ktype]
    [vname1]
ENDATA
```

Here the names in capitals are keywords of the MPS format and names in brackets are custom defined names or values. A couple of notes on the structure:

Fields:

All items surrounded by brackets appear in *fields*. The fields named "valueN" are numerical values. Hence, they must have the format

```
[+|-]XXXXXXX.XXXXXX[[e|E][+|-]XXX]
```

where

```
X = [0|1|2|3|4|5|6|7|8|9].
```

Sections:

The MPS file consists of several sections where the names in capitals indicate the beginning of a new section. For example, COLUMNS denotes the beginning of the columns section.

Comments:

Lines starting with an "*" are comment lines and are ignored by MOSEK.

Keys:

The question marks represent keys to be specified later.

Extensions:

The sections QSECTION and CSECTION are MOSEK specific extensions of the MPS format.

The standard MPS format is a fixed format, i.e. everything in the MPS file must be within certain fixed positions. MOSEK also supports a *free format*. See Section F.1.5 for details.

F.1.1.1 Linear example lo1.mps

A concrete example of a MPS file is presented below:

```
* File: lo1.mps
NAME          lo1
OBJSENSE
    MAX
ROWS
 N  obj
 E  c1
 G  c2
 L  c3
COLUMNS
    x1        obj       3
    x1        c1        3
    x1        c2        2
    x2        obj       1
    x2        c1        1
    x2        c2        1
    x2        c3        2
    x3        obj       5
    x3        c1        2
    x3        c2        3
    x4        obj       1
    x4        c2        1
    x4        c3        3
```

```
RHS
    rhs       c1        30
    rhs       c2        15
    rhs       c3        25
RANGES
BOUNDS
 UP bound     x2        10
ENDATA
```

Subsequently each individual section in the MPS format is discussed.

F.1.1.2 NAME

In this section a name (`[name]`) is assigned to the problem.

F.1.1.3 OBJSENSE (optional)

This is an optional section that can be used to specify the sense of the objective function. The `OBJSENSE` section contains one line at most which can be one of the following

```
MIN
MINIMIZE
MAX
MAXIMIZE
```

It should be obvious what the implication is of each of these four lines.

F.1.1.4 OBJNAME (optional)

This is an optional section that can be used to specify the name of the row that is used as objective function. The `OBJNAME` section contains one line at most which has the form

```
objname
```

`objname` should be a valid row name.

F.1.1.5 ROWS

A record in the `ROWS` section has the form

```
?  [cname1]
```

where the requirements for the fields are as follows:

Field	Starting position	Maximum width	Required	Description
?	2	1	Yes	Constraint key
`[cname1]`	5	8	Yes	Constraint name

Hence, in this section each constraint is assigned an unique name denoted by `[cname1]`. Please note that `[cname1]` starts in position 5 and the field can be at most 8 characters wide. An initial key (?)

must be present to specify the type of the constraint. The key can have the values `E`, `G`, `L`, or `N` with the following interpretation:

Constraint type	l_i^c	u_i^c
`E`	finite	l_i^c
`G`	finite	∞
`L`	$-\infty$	finite
`N`	$-\infty$	∞

In the MPS format an objective vector is not specified explicitly, but one of the constraints having the key `N` will be used as the objective vector c . In general, if multiple `N` type constraints are specified, then the first will be used as the objective vector c .

F.1.1.6 `COLUMNS`

In this section the elements of A are specified using one or more records having the form

```
 [vname1]  [cname1]    [value1]     [cname2]  [value2]
```

where the requirements for each field are as follows:

Field	Starting position	Maximum width	Required	Description
`[vname1]`	5	8	Yes	Variable name
`[cname1]`	15	8	Yes	Constraint name
`[value1]`	25	12	Yes	Numerical value
`[cname2]`	40	8	No	Constraint name
`[value2]`	50	12	No	Numerical value

Hence, a record specifies one or two elements a_{ij} of A using the principle that `[vname1]` and `[cname1]` determines j and i respectively. Please note that `[cname1]` must be a constraint name specified in the `ROWS` section. Finally, `[value1]` denotes the numerical value of a_{ij} . Another optional element is specified by `[cname2]`, and `[value2]` for the variable specified by `[vname1]`. Some important comments are:

- All elements belonging to one variable must be grouped together.
- Zero elements of A should not be specified.
- At least one element for each variable should be specified.

F.1.1.7 `RHS` (optional)

A record in this section has the format

```
 [name]    [cname1]    [value1]     [cname2]  [value2]
```

where the requirements for each field are as follows:

Field	Starting position	Maximum width	Required	Description
[name]	5	8	Yes	Name of the RHS vector
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

The interpretation of a record is that [name] is the name of the RHS vector to be specified. In general, several vectors can be specified. [cname1] denotes a constraint name previously specified in the ROWS section. Now, assume that this name has been assigned to the 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] and is interpreted in the same way. Please note that it is not necessary to specify zero elements, because elements are assumed to be zero.

F.1.1.8 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 and [cname1] is a valid constraint name. Assume that [cname1] is assigned to the i th constraint and let v_1 be the value specified by [value1], then a record has the interpretation:

Constraint type	Sign of v_1	l_i^c	u_i^c
E	-	$u_i^c + v_1$	
E	+		$l_i^c + v_1$
G	- or +		$l_i^c + \|v_1\|$
L	- or +	$u_i^c - \|v_1\|$	
N			

F.1.1.9 `QSECTION` (optional)

Within the `QSECTION` the label `[cname1]` must be a constraint name previously specified in the `ROWS` section. The label `[cname1]` denotes the constraint to which the quadratic term belongs. A record in the `QSECTION` has the form

```
 [vname1]  [vname2]    [value1]     [vname3]  [value2]
```

where the requirements for each field are:

Field	Starting position	Maximum width	Required	Description
`[vname1]`	5	8	Yes	Variable name
`[vname2]`	15	8	Yes	Variable name
`[value1]`	25	12	Yes	Numerical value
`[vname3]`	40	8	No	Variable name
`[value2]`	50	12	No	Numerical value

A record specifies one or two elements in the lower triangular part of the Q^i matrix where `[cname1]` specifies the i . Hence, if the names `[vname1]` and `[vname2]` have been assigned to the k th and j th variable, then Q^i_{kj} is assigned the value given by `[value1]` An optional second element is specified in the same way by the fields `[vname1]`, `[vname3]`, and `[value2]`.

The example

$$
\begin{array}{lccc}
\text{minimize} & -x_2 + 0.5(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & & \\
\text{subject to} & x_1 + x_2 + x_3 & \geq & 1, \\
& x \geq 0 & &
\end{array}
$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
```

```
QSECTION      obj
    x1        x1        2.0
    x1        x3        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

Regarding the QSECTIONs please note that:

- Only one QSECTION is allowed for each constraint.
- The QSECTIONs can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- All entries specified in a QSECTION are assumed to belong to the lower triangular part of the quadratic term of Q .

F.1.1.10 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	Numerical value

Hence, a record in the BOUNDS section has the following interpretation: [name] is the name of the bound vector and [vname1] is the name of the variable which bounds are modified by the record. ?? and [value1] are used to modify the bound vectors according to the following table:

??	l_j^x	u_j^x	Made integer (added to $\mathcal{J}$)
FR	$-\infty$	∞	No
FX	v_1	v_1	No
LO	v_1	unchanged	No
MI	$-\infty$	unchanged	No
PL	unchanged	∞	No
UP	unchanged	v_1	No
BV	0	1	Yes
LI	$\lceil v_1 \rceil$	*unchanged*	Yes
UI	unchanged	$\lfloor v_1 \rfloor$	Yes

v_1 is the value specified by `[value1]`.

F.1.1.11 `CSECTION` (optional)

The purpose of the `CSECTION` is to specify the constraint

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

in (F.1).

It is assumed that $\mathcal{C}$ satisfies the following requirements. Let

$$x^t \in \mathbb{R}^{n^t}, \ t = 1, \ldots, k$$

be vectors comprised of parts of the decision variables x so that each decision variable is a member of exactly **one** vector x^t , for example

$$x^1 = \begin{bmatrix} x_1 \\ x_4 \\ x_7 \end{bmatrix} \text{ and } x^2 = \begin{bmatrix} x_6 \\ x_5 \\ x_3 \\ x_2 \end{bmatrix}.$$

Next define

$$\mathcal{C} := \left\{ x \in \mathbb{R}^n : \ x^t \in \mathcal{C}_t, \ t = 1, \ldots, k \right\}$$

where $\mathcal{C}_t$ must have one of the following forms

- $\mathbb{R}$ set:

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

- Quadratic cone:

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

- Rotated quadratic cone:

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

In general, only quadratic and rotated quadratic cones are specified in the MPS file whereas membership of the $\mathbb{R}$ set is not. If a variable is not a member of any other cone then it is assumed to be a member of an $\mathbb{R}$ cone.

Next, let us study an example. Assume that the quadratic cone

$$x_4 \geq \sqrt{x_5^2 + x_8^2} \tag{F.4}$$

and the rotated quadratic cone

$$2x_3x_7 \geq x_1^2 + x_0^2, \; x_3, x_7 \geq 0, \tag{F.5}$$

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 (F.4) which is given the name `konea`. This is a quadratic cone which is specified by the keyword `QUAD` in the `CSECTION` header. The 0.0 value in the `CSECTION` header is not used by the `QUAD` cone.

The second `CSECTION` specifies the rotated quadratic cone (F.5). Please note the keyword `RQUAD` in the `CSECTION` which is used to specify that the cone is a rotated quadratic cone instead of a quadratic cone. The 0.0 value in the `CSECTION` header is not used by the `RQUAD` cone.

In general, a `CSECTION` header has the format

```
CSECTION      [kname1]    [value1]     [ktype]
```

where the requirement for each field are as follows:

Field	Starting position	Maximum width	Re-quired	Description
`[kname1]`	5	8	Yes	Name of the cone
`[value1]`	15	12	No	Cone parameter
`[ktype]`	25		Yes	Type of the cone.

The possible cone type keys are:

Cone type key	Members	Interpretation.
`QUAD`	≥ 1	Quadratic cone i.e. (F.2).
`RQUAD`	≥ 2	Rotated quadratic cone i.e. (F.3).

Please note that a quadratic cone must have at least one member whereas a rotated quadratic cone must have at least two members. A record in the CSECTION has the format

```
[vname1]
```

where the requirements for each field are

Field	Starting position	Maximum width	Required	Description
[vname1]	2	8	Yes	A valid variable name

The most important restriction with respect to the CSECTION is that a variable must occur in only one CSECTION.

F.1.1.12 ENDATA

This keyword denotes the end of the MPS file.

F.1.2 Integer variables

Using special bound keys in the BOUNDS section it is possible to specify that some or all of the variables should be integer-constrained i.e. be members of $\mathcal{J}$. However, an alternative method is available.

This method is available only for backward compatibility and we recommend that it is not used. This method requires that markers are placed in the COLUMNS section as in the example:

```
COLUMNS
    x1       obj      -10.0          c1       0.7
    x1       c2       0.5            c3       1.0
    x1       c4       0.1
* Start of integer-constrained variables.
    MARK000  'MARKER'                'INTORG'
    x2       obj      -9.0           c1       1.0
    x2       c2       0.8333333333   c3       0.66666667
    x2       c4       0.25
    x3       obj      1.0            c6       2.0
    MARK001  'MARKER'                'INTEND'
* End of integer-constrained variables.
```

Please note that special marker lines are used to indicate the start and the end of the integer variables. Furthermore be aware of the following

- IMPORTANT: All variables between the markers are assigned a default lower bound of 0 and a default upper bound of 1. **This may not be what is intended.** If it is not intended, the correct bounds should be defined in the BOUNDS section of the MPS formatted file.
- MOSEK ignores field 1, i.e. MARK0001 and MARK001, however, other optimization systems require them.
- Field 2, i.e. 'MARKER', must be specified including the single quotes. This implies that no row can be assigned the name 'MARKER'.

- Field 3 is ignored and should be left blank.
- Field 4, i.e. `'INTORG'` and `'INTEND'`, must be specified.
- It is possible to specify several such integer marker sections within the `COLUMNS` section.

F.1.3 General limitations

- An MPS file should be an ASCII file.

F.1.4 Interpretation of the MPS format

Several issues related to the MPS format are not well-defined by the industry standard. However, MOSEK uses the following interpretation:

- If a matrix element in the `COLUMNS` section is specified multiple times, then the multiple entries are added together.
- If a matrix element in a `QSECTION` section is specified multiple times, then the multiple entries are added together.

F.1.5 The free MPS format

MOSEK supports a free format variation of the MPS format. The free format is similar to the MPS file format but less restrictive, e.g. it allows longer names. However, it also presents two main limitations:

- By default a line in the MPS file must not contain more than 1024 characters. However, by modifying the parameter `iparam.read_mps_width` an arbitrary large line width will be accepted.
- A name must not contain any blanks.

To use the free MPS format instead of the default MPS format the MOSEK parameter `iparam.read_mps_format` should be changed.

F.2 The LP file format

MOSEK supports the LP file format with some extensions i.e. MOSEK can read and write LP formatted files.

Please note that the LP format is not a completely well-defined standard and hence different optimization packages may interpret the same LP file in slightly different ways. MOSEK tries to emulate as closely as possible CPLEX's behavior, but tries to stay backward compatible.

The LP file format can specify problems on the form

$$
\begin{array}{lcccl}
\text{minimize/maximize} & & c^T x + \frac{1}{2} q^o(x) & & \\
\text{subject to} & l^c \leq & Ax + \frac{1}{2} q(x) & \leq & u^c, \\
 & l^x \leq & x & \leq & u^x, \\
 & & x_{\mathcal{J}} \text{integer}, & &
\end{array}
$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear term in the objective.
- $q^o :\in \mathbb{R}^n \rightarrow \mathbb{R}$ is the quadratic term in the objective where

  $$q^o(x) = x^T Q^o x$$

  and it is assumed that

  $$Q^o = (Q^o)^T.$$

- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \rightarrow \mathbb{R}$ is a vector of quadratic functions. Hence,

  $$q_i(x) = x^T Q^i x$$

  where it is assumed that

  $$Q^i = (Q^i)^T.$$

- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer constrained variables.

F.2.1 The sections

An LP formatted file contains a number of sections specifying the objective, constraints, variable bounds, and variable types. The section keywords may be any mix of upper and lower case letters.

F.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

```
4 x1 + x2 - 0.1 x3
```

and so forth. The quadratic terms are written in square brackets (`[]`) and are either squared or multiplied as in the examples

```
x1^2
```

and

```
x1 * x2
```

There may be zero or more pairs of brackets containing quadratic expressions.

An example of an objective section is:

```
minimize
myobj: 4 x1 + x2 - 0.1 x3 + [ x1^2 + 2.1 x1 * x2 ]/2
```

Please note that the quadratic expressions are multiplied with $\frac{1}{2}$, so that the above expression means

$$\text{minimize} \quad 4x_1 + x_2 - 0.1 \cdot x_3 + \frac{1}{2}(x_1^2 + 2.1 \cdot x_1 \cdot x_2)$$

If the same variable occurs more than once in the linear part, the coefficients are added, so that `4 x1 + 2 x1` is equivalent to `6 x1`. In the quadratic expressions `x1 * x2` is equivalent to `x2 * x1` and as in the linear part , if the same variables multiplied or squared occur several times their coefficients are added.

F.2.1.2 The constraints

The second section beginning with one of the keywords

```
subj to
subject to
s.t.
```

```
st
```

defines the linear constraint matrix (A) and the quadratic matrices (Q^i).

A constraint contains a name (optional), expressions adhering to the same rules as in the objective and a bound:

```
subject to
con1: x1 + x2 + [ x3^2 ]/2 <= 5.1
```

The bound type (here `<=`) may be any of `<`, `<=`, `=`, `>`, `>=` (`<` and `<=` mean the same), and the bound may be any number.

In the standard LP format it is not possible to define more than one bound, but MOSEK supports defining ranged constraints by using double-colon (`''::''`) instead of a single-colon (”`:`”) after the constraint name, i.e.

$$-5 \leq x_1 + x_2 \leq 5 \tag{F.6}$$

may be written as

```
con:: -5 < x_1 + x_2 < 5
```

By default MOSEK writes ranged constraints this way.

If the files must adhere to the LP standard, ranged constraints must either be split into upper bounded and lower bounded constraints or be written as en equality with a slack variable. For example the expression (F.6) may be written as

$$x_1 + x_2 - sl_1 = 0, -5 \leq sl_1 \leq 5.$$

F.2.1.3 Bounds

Bounds on the variables can be specified in the bound section beginning with one of the keywords

```
bound
bounds
```

The bounds section is optional but should, if present, follow the `subject to` section. All variables listed in the bounds section must occur in either the objective or a constraint.

The default lower and upper bounds are 0 and $+\infty$. A variable may be declared free with the keyword `free`, which means that the lower bound is $-\infty$ and the upper bound is $+\infty$. Furthermore it may be assigned a finite lower and upper bound. The bound definitions for a given variable may be written in one or two lines, and bounds can be any number or $\pm\infty$ (written as `+inf`/`-inf`/`+infinity`/`-infinity`) as in the example

```
bounds
x1 free
x2 <= 5
0.1 <= x2
x3 = 42
2 <= x4 < +inf
```

F.2.1.4 Variable types

The final two sections are optional and must begin with one of the keywords

```
bin
binaries
binary
```

and

```
gen
general
```

Under `general` all integer variables are listed, and under `binary` all binary (integer variables with bounds 0 and 1) are listed:

```
general
x1 x2
binary
x3 x4
```

Again, all variables listed in the binary or general sections must occur in either the objective or a constraint.

F.2.1.5 Terminating section

Finally, an LP formatted file must be terminated with the keyword

```
end
```

F.2.1.6 Linear example `lo1.lp`

A simple example of an LP file is:

```
\ File: lo1.lp
maximize
obj: 3 x1 + x2 + 5 x3 + x4
subject to
c1:  3 x1 + x2 + 2 x3 = 30
c2:  2 x1 + x2 + 3 x3 + x4 >= 15
c3:  2 x2 + 3 x4 <= 25
bounds
 0 <= x1 <= +infinity
 0 <= x2 <= 10
 0 <= x3 <= +infinity
 0 <= x4 <= +infinity
end
```

F.2.1.7 Mixed integer example `milo1.lp`

```
maximize
obj: x1 + 6.4e-01 x2
subject to
c1:  5e+01 x1 + 3.1e+01 x2 <= 2.5e+02
c2:  3e+00 x1 - 2e+00 x2 >= -4e+00
```

```
bounds
 0 <= x1 <= +infinity
 0 <= x2 <= +infinity
general
 x1 x2
end
```

F.2.2 LP format peculiarities

F.2.2.1 Comments

Anything on a line after a "\" is ignored and is treated as a comment.

F.2.2.2 Names

A name for an objective, a constraint or a variable may contain the letters a-z, A-Z, the digits 0-9 and the characters

```
!"#$%&()/,.;?@_'`|~
```

The first character in a name must not be a number, a period or the letter 'e' or 'E'. Keywords must not be used as names.

MOSEK accepts any character as valid for names, except `\0`. When writing a name that is not allowed in LP files, it is changed and a warning is issued.

The algorithm for making names LP valid works as follows: The name is interpreted as an `utf-8` string. For a unicode character `c`:

- If `c=='_'` (underscore), the output is `'__'` (two underscores).
- If `c` is a valid LP name character, the output is just `c`.
- If `c` is another character in the ASCII range, the output is `_XX`, where `XX` is the hexadecimal code for the character.
- If `c` is a character in the range 127—65535, the output is `_uXXXX`, where `XXXX` is the hexadecimal code for the character.
- If `c` is a character above 65535, the output is `_UXXXXXXXX`, where `XXXXXXXX` is the hexadecimal code for the character.

Invalid `utf-8` substrings are escaped as `'_XX'`, and if a name starts with a period, 'e' or 'E', that character is escaped as `'_XX'`.

F.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.

F.2.2.4 MOSEK specific extensions to the LP format

Some optimization software packages employ a more strict definition of the LP format that the one used by MOSEK. The limitations imposed by the strict LP format are the following:

- Quadratic terms in the constraints are not allowed.
- Names can be only 16 characters long.
- Lines must not exceed 255 characters in length.

If an LP formatted file created by MOSEK should satisfies the strict definition, then the parameter

`iparam.write_lp_strict_format`

should be set; note, however, that some problems cannot be written correctly as a strict LP formatted file. For instance, all names are truncated to 16 characters and hence they may loose their uniqueness and change the problem.

To get around some of the inconveniences converting from other problem formats, MOSEK allows lines to contain 1024 characters and names may have any length (shorter than the 1024 characters).

Internally in MOSEK names may contain any (printable) character, many of which cannot be used in LP names. Setting the parameters

`iparam.read_lp_quoted_names`

and

`iparam.write_lp_quoted_names`

allows MOSEK to use quoted names. The first parameter tells MOSEK to remove quotes from quoted names e.g, `"x1"`, when reading LP formatted files. The second parameter tells MOSEK to put quotes around any semi-illegal name (names beginning with a number or a period) and fully illegal name (containing illegal characters). As double quote is a legal character in the LP format, quoting semi-illegal names makes them legal in the pure LP format as long as they are still shorter than 16 characters. Fully illegal names are still illegal in a pure LP file.

F.2.3 The strict LP format

The LP format is not a formal standard and different vendors have slightly different interpretations of the LP format. To make MOSEK's definition of the LP format more compatible with the definitions of other vendors, use the parameter setting

`iparam.write_lp_strict_format` = `onoffkey.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 parameter setting iparam.write_generic_names = onoffkey.on

which will cause all names to be renamed systematically in the output file.

F.2.4 Formatting of an LP file

A few parameters control the visual formatting of LP files written by MOSEK in order to make it easier to read the files. These parameters are

iparam.write_lp_line_width

iparam.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.

F.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

iparam.read_con

iparam.read_var

iparam.read_anz

iparam.read_qnz

F.2.4.2 Unnamed constraints

Reading and writing an LP file with MOSEK may change it superficially. If an LP file contains unnamed constraints or objective these are given their generic names when the file is read (however unnamed constraints in MOSEK are written without names).

F.3 The OPF format

The Optimization Problem Format (OPF) is an alternative to LP and MPS files for specifying optimization problems. It is row-oriented, inspired by the CPLEX LP format.

Apart from containing objective, constraints, bounds etc. it may contain complete or partial solutions, comments and extra information relevant for solving the problem. It is designed to be easily read and modified by hand and to be forward compatible with possible future extensions.

F.3.1 Intended use

The OPF file format is meant to replace several other files:

- The LP file format. Any problem that can be written as an LP file can be written as an OPF file to; furthermore it naturally accommodates ranged constraints and variables as well as arbitrary characters in names, fixed expressions in the objective, empty constraints, and conic constraints.
- Parameter files. It is possible to specify integer, double and string parameters along with the problem (or in a separate OPF file).
- Solution files. It is possible to store a full or a partial solution in an OPF file and later reload it.

F.3.2 The file format

The format uses tags to structure data. A simple example with the basic sections may look like this:

```
[comment]
  This is a comment. You may write almost anything here...
[/comment]

# This is a single-line comment.

[objective min 'myobj']
  x + 3 y + x^2 + 3 y^2 + z + 1
[/objective]

[constraints]
  [con 'con01'] 4 <= x + y  [/con]
[/constraints]

[bounds]
  [b] -10 <= x,y <= 10  [/b]

  [cone quad] x,y,z [/cone]
[/bounds]
```

A scope is opened by a tag of the form `[tag]` and closed by a tag of the form `[/tag]`. An opening tag may accept a list of unnamed and named arguments, for examples

```
[tag value] tag with one unnamed argument [/tag]
[tag arg=value] tag with one named argument in quotes [/tag]
```

Unnamed arguments are identified by their order, while named arguments may appear in any order, but never before an unnamed argument. The `value` can be a quoted, single-quoted or double-quoted text string, i.e.

```
[tag 'value']     single-quoted value [/tag]
[tag arg='value'] single-quoted value [/tag]
```

```
[tag "value"]     double-quoted value [/tag]
[tag arg="value"] double-quoted value [/tag]
```

F.3.2.1 Sections

The recognized tags are

- `[comment]` A comment section. This can contain *almost* any text: Between single quotes (`'`) or double quotes (`"`) any text may appear. Outside quotes the markup characters (`[` and `]`) must be prefixed by backslashes. Both single and double quotes may appear alone or inside a pair of quotes if it is prefixed by a backslash.

- `[objective]` The objective function: This accepts one or two parameters, where the first one (in the above example '`min`') is either `min` or `max` (regardless of case) and defines the objective sense, and the second one (above '`myobj`'), if present, is the objective name. The section may contain linear and quadratic expressions.

  If several objectives are specified, all but the last are ignored.

- `[constraints]` This does not directly contain any data, but may contain the subsection '`con`' defining a linear constraint.

  `[con]` defines a single constraint; if an argument is present (`[con NAME]`) this is used as the name of the constraint, otherwise it is given a null-name. The section contains a constraint definition written as linear and quadratic expressions with a lower bound, an upper bound, with both or with an equality. Examples:

  ```
  [constraints]
    [con 'con1'] 0 <= x + y       [/con]
    [con 'con2'] 0 >= x + y       [/con]
    [con 'con3'] 0 <= x + y <= 10 [/con]
    [con 'con4']      x + y  = 10 [/con]
  [/constraints]
  ```

  Constraint names are unique. If a constraint is specified which has the same name as a previously defined constraint, the new constraint replaces the existing one.

- `[bounds]` This does not directly contain any data, but may contain 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

    $$-10 \leq x, y \leq 10.$$

- [cone]. Currently, the supported cones are the *quadratic cone* and the *rotated quadratic cone* (see section 5.3). A conic constraint is defined as a set of variables which belongs to a single unique cone.

  A quadratic cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1^2 > \sum_{i=2}^{n} x_i^2.$$

  A rotated quadratic cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1 x_2 > \sum_{i=3}^{n} x_i^2.$$

A [bounds]-section example:

```
[bounds]
  [b]  0 <= x,y <= 10  [/b] # ranged bound
  [b] 10 >= x,y >=  0  [/b] # ranged bound
  [b]  0 <= x,y <= inf [/b] # using inf
  [b]       x,y free   [/b] # free variables
  # Let (x,y,z,w) belong to the cone K
  [cone quad]  x,y,z,w  [/cone] # quadratic cone
  [cone rquad] x,y,z,w  [/cone] # rotated quadratic cone
[/bounds]
```

By default all variables are free.

- [variables] This defines an ordering of variables as they should appear in the problem. This is simply a space-separated list of variable names.

- [integer] This contains a space-separated list of variables and defines the constraint that the listed variables must be integer values.

- [hints] This may contain only non-essential data; for example estimates of the number of variables, constraints and non-zeros. Placed before all other sections containing data this may reduce the time spent reading the file.

  In the hints section, any subsection which is not recognized by MOSEK is simply ignored. In this section a hint in a subsection is defined as follows:

```
[hint ITEM] value [/hint]
```

  where ITEM may be replaced by numvar (number of variables), numcon (number of linear/quadratic constraints), numanz (number of linear non-zeros in constraints) and numqnz (number of quadratic non-zeros in constraints).

- [solutions] This section can contain a set of full or partial solutions to a problem. Each solution must be specified using a [solution]-section, i.e.

```
[solutions]
       [solution]...[/solution] #solution 1
       [solution]...[/solution] #solution 2
                        #other solutions....
       [solution]...[/solution] #solution n
[/solutions]
```

Note that a [solution]-section must be always specified inside a [solutions]-section. The syntax of a [solution]-section is the following:

```
[solution SOLTYPE status=STATUS]...[/solution]
```

where SOLTYPE is one of the strings

- 'interior', a non-basic solution,
- 'basic', a basic solution,
- 'integer', an integer solution,

and STATUS is one of the strings

- 'UNKNOWN',
- 'OPTIMAL',
- 'INTEGER_OPTIMAL',
- 'PRIM_FEAS',
- 'DUAL_FEAS',
- 'PRIM_AND_DUAL_FEAS',
- 'NEAR_OPTIMAL',
- 'NEAR_PRIM_FEAS',
- 'NEAR_DUAL_FEAS',
- 'NEAR_PRIM_AND_DUAL_FEAS',
- 'PRIM_INFEAS_CER',
- 'DUAL_INFEAS_CER',
- 'NEAR_PRIM_INFEAS_CER',
- 'NEAR_DUAL_INFEAS_CER',
- 'NEAR_INTEGER_OPTIMAL'.

Most of these values are irrelevant for input solutions; when constructing a solution for simplex hot-start or an initial solution for a mixed integer problem the safe setting is UNKNOWN.

A [solution]-section contains [con] and [var] sections. Each [con] and [var] section defines solution information for a single variable or constraint, specified as list of KEYWORD/value pairs, in any order, written as

```
KEYWORD=value
```

Allowed keywords are as follows:

- sk. The status of the item, where the value is one of the following strings:
  * LOW, the item is on its lower bound.
  * UPR, the item is on its upper bound.
  * FIX, it is a fixed item.
  * BAS, the item is in the basis.
  * SUPBAS, the item is super basic.

    * `UNK`, the status is unknown.
    * `INF`, the item is outside its bounds (infeasible).
  - `lvl` Defines the level of the item.
  - `sl` Defines the level of the dual variable associated with its lower bound.
  - `su` Defines the level of the dual variable associated with its upper bound.
  - `sn` Defines the level of the variable associated with its cone.
  - `y` Defines the level of the corresponding dual variable (for constraints only).

  A `[var]` section should always contain the items `sk`, `lvl`, `sl` and `su`. Items `sl` and `su` are not required for `integer` solutions.

  A `[con]` section should always contain `sk`, `lvl`, `sl`, `su` and `y`.

  An example of a solution section

```
[solution basic status=UNKNOWN]
    [var x0] sk=LOW    lvl=5.0      [/var]
    [var x1] sk=UPR    lvl=10.0     [/var]
    [var x2] sk=SUPBAS lvl=2.0  sl=1.5 su=0.0 [/var]

    [con c0] sk=LOW    lvl=3.0 y=0.0 [/con]
    [con c0] sk=UPR    lvl=0.0 y=5.0 [/con]
[/solution]
```

- `[vendor]` This contains solver/vendor specific data. It accepts one argument, which is a vendor ID – for MOSEK the ID is simply `mosek` – and the section contains the subsection `parameters` defining solver parameters. When reading a vendor section, any unknown vendor can be safely ignored. This is described later.

Comments using the '`#`' may appear anywhere in the file. Between the '`#`' and the following line-break any text may be written, including markup characters.

F.3.2.2 Numbers

Numbers, when used for parameter values or coefficients, are written in the usual way by the `printf` function. That is, they may be prefixed by a sign (`+` or `-`) and may contain an integer part, decimal part and an exponent. The decimal point is always '`.`' (a dot). Some examples are

```
1
1.0
 .0
1.
1e10
1e+10
1e-10
```

Some *invalid* examples are

```
e10   # invalid, must contain either integer or decimal part
.     # invalid
.e10  # invalid
```

More formally, the following standard regular expression describes numbers as used:

```
[+|-]?([0-9]+[.][0-9]*|[.][0-9]+)([eE][+|-]?[0-9]+)?
```

F.3.2.3 Names

Variable names, constraint names and objective name may contain arbitrary characters, which in some cases must be enclosed by quotes (single or double) that in turn must be preceded by a backslash. Unquoted names must begin with a letter (`a-z` or `A-Z`) and contain only the following characters: the letters `a-z` and `A-Z`, the digits `0-9`, braces ({ and }) and underscore (_).

Some examples of legal names:

```
an_unquoted_name
another_name{123}
'single quoted name'
"double quoted name"
"name with \\"quote\\" in it"
"name with []s in it"
```

F.3.3 Parameters section

In the `vendor` section solver parameters are defined inside the `parameters` subsection. Each parameter is written as

```
[p PARAMETER_NAME] value [/p]
```

where `PARAMETER_NAME` is replaced by a MOSEK parameter name, usually of the form `MSK_IPAR_...`, `MSK_DPAR_...` or `MSK_SPAR_...`, and the `value` is replaced by the value of that parameter; both integer values and named values may be used. Some simple examples are:

```
[vendor mosek]
  [parameters]
    [p MSK_IPAR_OPF_MAX_TERMS_PER_LINE] 10     [/p]
    [p MSK_IPAR_OPF_WRITE_PARAMETERS]   MSK_ON [/p]
    [p MSK_DPAR_DATA_TOL_BOUND_INF]     1.0e18 [/p]
  [/parameters]
[/vendor]
```

F.3.4 Writing OPF files from MOSEK

The function `Task.writedata` can be used to produce an OPF file from a task.

To write an OPF file set the parameter `iparam.write_data_format` to `dataformat.op` as this ensures that `OPF` format is used. Then modify the following parameters to define what the file should contain:

- `iparam.opf_write_header`, include a small header with comments.
- `iparam.opf_write_hints`, include hints about the size of the problem.
- `iparam.opf_write_problem`, include the problem itself — objective, constraints and bounds.
- `iparam.opf_write_solutions`, include solutions if they are defined. If this is off, no solutions are included.
- `iparam.opf_write_sol_bas`, include basic solution, if defined.

- iparam.opf_write_sol_itg, include integer solution, if defined.
- iparam.opf_write_sol_itr, include interior solution, if defined.
- iparam.opf_write_parameters, include all parameter settings.

F.3.5 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

F.3.5.1 Linear example lo1.opf

Consider the example:

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

having the bounds

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

In the OPF format the example is displayed as shown below:

```
[comment]
  The lo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 4 [/hint]
  [hint NUMCON] 3 [/hint]
  [hint NUMANZ] 9 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4
[/variables]

[objective maximize 'obj']
   3 x1 + x2 + 5 x3 + x4
[/objective]

[constraints]
  [con 'c1'] 3 x1 +   x2 + 2 x3         = 30 [/con]
  [con 'c2'] 2 x1 +   x2 + 3 x3 +   x4 >= 15 [/con]
  [con 'c3']        2 x2        + 3 x4 <= 25 [/con]
[/constraints]
```

```
[bounds]
  [b] 0 <= * [/b]
  [b] 0 <= x2 <= 10 [/b]
[/bounds]
```

F.3.5.2 Quadratic example qo1.opf

An example of a quadratic optimization problem is

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

This can be formulated in `opf` as shown below.

```
[comment]
  The qo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 3 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
  [hint NUMQNZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3
[/variables]

[objective minimize 'obj']
  # The quadratic terms are often written with a factor of 1/2 as here,
  # but this is not required.

   - x2 + 0.5 ( 2.0 x1 ^ 2 - 2.0 x3 * x1 + 0.2 x2 ^ 2 + 2.0 x3 ^ 2 )
[/objective]

[constraints]
  [con 'c1'] 1.0 <= x1 + x2 + x3 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]
```

F.3.5.3 Conic quadratic example cqo1.opf

Consider the example:

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

Please note that the type of the cones is defined by the parameter to `[cone ...]`; the content of the `cone`-section is the names of variables that belong to the cone.

```
[comment]
  The cqo1 example in OPF format.
[/comment]

[hints]
  [hint NUMVAR] 6 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4 x5 x6
[/variables]

[objective minimize 'obj']
   x4 + x5 + x6
[/objective]

[constraints]
  [con 'c1']  x1 + x2 + 2e+00 x3 = 1e+00 [/con]
[/constraints]

[bounds]
  # We let all variables default to the positive orthant
  [b] 0 <= * [/b]

  # ...and change those that differ from the default
  [b] x4,x5,x6 free [/b]

  # Define quadratic cone: x4 >= sqrt( x1^2 + x2^2 )
  [cone quad 'k1'] x4, x1, x2 [/cone]

  # Define rotated quadratic cone: 2 x5 x6 >= x3^2
  [cone rquad 'k2'] x5, x6, x3 [/cone]
[/bounds]
```

F.3.5.4 Mixed integer example `milo1.opf`

Consider the mixed integer problem:

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

This can be implemented in OPF with:

```
[comment]
  The milo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 2 [/hint]
  [hint NUMCON] 2 [/hint]
  [hint NUMANZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2
[/variables]

[objective maximize 'obj']
   x1 + 6.4e-1 x2
[/objective]

[constraints]
  [con 'c1'] 5e+1 x1 + 3.1e+1 x2 <= 2.5e+2 [/con]
  [con 'c2'] -4 <= 3 x1 - 2 x2 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]

[integer]
  x1 x2
[/integer]
```

F.4 The Task format

The Task format is MOSEK's native binary format. It contains a complete image of a MOSEK task, i.e.

- Problem data: Linear, conic quadratic, semidefinite and quadratic data
- Problem item names: Variable names, constraints names, cone names etc.
- Parameter settings
- Solutions

There are a few things to be aware of:

- The task format *does not* support General Convex problems since these are defined by arbitrary user-defined functions.
- Status of a solution read from a file will *always* be unknown.

```
*        1         2         3         4         5         6
*23456789012345678901234567890123456789012345678901234567890
NAME          [name]
 ?? [vname1]              [value1]
ENDATA
```

Figure F.1: The standard ORD format.

The format is based on the TAR (USTar) file format. This means that the individual pieces of data in a `.task` file can be examined by unpacking it as a TAR file. Please note that the inverse may not work: Creating a file using TAR will most probably not create a valid MOSEK Task file since the order of the entries is important.

F.5 The XML (OSiL) format

MOSEK can write data in the standard `OSiL` xml format. For a definition of the `OSiL` format please see http://www.optimizationservices.org/. Only linear constraints (possibly with integer variables) are supported. By default output files with the extension `.xml` are written in the `OSiL` format.

The parameter `iparam.write_xml_mode` controls if the linear coefficients in the A matrix are written in row or column order.

F.6 The ORD file format

An ORD formatted file specifies in which order the mixed integer optimizer branches on variables. The format of an ORD file is shown in Figure F.1. In the figure names in capitals are keywords of the ORD format, whereas names in brackets are custom names or values. The `??` is an optional key specifying the preferred branching direction. The possible keys are `DN` and `UP` which indicate that down or up is the preferred branching direction respectively. The branching direction key is optional and is left blank the mixed integer optimizer will decide whether to branch up or down.

F.6.1 An example

A concrete example of a ORD file is presented below:

```
NAME          EXAMPLE
 DN x1                2
 UP x2                1
    x3                10
ENDATA
```

This implies that the priorities 2, 1, and 10 are assigned to variable `x1`, `x2`, and `x3` respectively. The higher the priority value assigned to a variable the earlier the mixed integer optimizer will branch on that variable. The key `DN` implies that the mixed integer optimizer first will branch down on variable whereas the key `UP` implies that the mixed integer optimizer will first branch up on a variable.

If no branch direction is specified for a variable then the mixed integer optimizer will automatically choose the branching direction for that variable. Similarly, if no priority is assigned to a variable then it is automatically assigned the priority of 0.

F.7 The solution file format

MOSEK provides one or two solution files depending on the problem type and the optimizer used. If a problem is optimized using the interior-point optimizer and no basis identification is required, then a file named `probname.sol` is provided. `probname` is the name of the problem and `.sol` is the file extension. If the problem is optimized using the simplex optimizer or basis identification is performed, then a file named `probname.bas` is created presenting the optimal basis solution. Finally, if the problem contains integer constrained variables then a file named `probname.int` is created. It contains the integer solution.

F.7.1 The basic and interior solution files

In general both the interior-point and the basis solution files have the format:

```
NAME                : <problem name>
PROBLEM STATUS      : <status of the problem>
SOLUTION STATUS     : <status of the solution>
OBJECTIVE NAME      : <name of the objective function>
PRIMAL OBJECTIVE    : <primal objective value corresponding to the solution>
DUAL OBJECTIVE      : <dual objective value corresponding to the solution>
CONSTRAINTS
INDEX  NAME     AT ACTIVITY    LOWER LIMIT   UPPER LIMIT   DUAL LOWER    DUAL UPPER
?      <name>   ?? <a value>   <a value>     <a value>     <a value>     <a value>
VARIABLES
INDEX  NAME     AT ACTIVITY    LOWER LIMIT   UPPER LIMIT   DUAL LOWER    DUAL UPPER    CONIC DUAL
?      <name>   ?? <a value>   <a value>     <a value>     <a value>     <a value>     <a value>
```

In the example the fields `?` and `<>` will be filled with problem and solution specific information. As can be observed a solution report consists of three sections, i.e.

`HEADER`

In this section, first the name of the problem is listed and afterwards the problem and solution statuses are shown. In this case the information shows that the problem is primal and dual feasible and the solution is optimal. Next the primal and dual objective values are displayed.

`CONSTRAINTS`

Subsequently in the constraint section the following information is listed for each constraint:

`INDEX`

A sequential index assigned to the constraint by MOSEK

`NAME`

The name of the 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 F.1: Status keys.

`AT`

The status of the constraint. In Table F.1 the possible values of the status keys and their interpretation are shown.

`ACTIVITY`

Given the i th constraint on the form

$$l_i^c \leq \sum_{j=1}^{n} a_{ij} x_j \leq u_i^c, \tag{F.7}$$

then activity denote the quantity $\sum_{j=1}^{n} a_{ij} x_j^*$, where x^* is the value for the x solution.

`LOWER LIMIT`

Is the quantity l_i^c (see (F.7)).

`UPPER LIMIT`

Is the quantity u_i^c (see (F.7)).

`DUAL LOWER`

Is the dual multiplier corresponding to the lower limit on the constraint.

`DUAL UPPER`

Is the dual multiplier corresponding to the upper limit on the constraint.

`VARIABLES`

The last section of the solution report lists information for the variables. This information has a similar interpretation as for the constraints. However, the column with the header `[CONIC DUAL]` is only included for problems having one or more conic constraints. This column shows the dual variables corresponding to the conic constraints.

F.7.2 The integer solution file

The integer solution is equivalent to the basic and interior solution files except that no dual information is included.

Appendix G

Problem analyzer examples

This appendix presents a few examples of the output produced by the problem analyzer described in Section 13.1. The first two problems are taken from the MIPLIB 2003 collection, http://miplib.zib.de/.

G.1 air04

```
Analyzing the problem

Constraints               Bounds                    Variables
 fixed   : all             ranged  : all             bin : all

-------------------------------------------------------------------------------

Objective, min cx
   range: min |c|: 31.0000     max |c|: 2258.00
 distrib:        |c|       vars
           [31, 100)        176
        [100, 1e+03)       8084
   [1e+03, 2.26e+03]        644

-------------------------------------------------------------------------------

Constraint matrix A has
       823 rows (constraints)
      8904 columns (variables)
     72965 (0.995703%) nonzero entries (coefficients)

Row nonzeros, A_i
   range: min A_i: 2 (0.0224618%)    max A_i: 368 (4.13297%)
 distrib:         A_i        rows       rows%        acc%
                    2           2        0.24        0.24
               [3, 7]           4        0.49        0.73
              [8, 15]          19        2.31        3.04
             [16, 31]          80        9.72       12.76
             [32, 63]         236       28.68       41.43
            [64, 127]         289       35.12       76.55
```

```
          [128, 255]         186       22.60       99.15
          [256, 368]           7        0.85      100.00

Column nonzeros, A|j
   range: min A|j: 2 (0.243013%)    max A|j: 15 (1.8226%)
 distrib:        A|j        cols       cols%        acc%
                   2         118        1.33        1.33
              [3, 7]        2853       32.04       33.37
             [8, 15]        5933       66.63      100.00

A nonzeros, A(ij)
   range: all |A(ij)| = 1.00000

-------------------------------------------------------------------------------

Constraint bounds, lb <= Ax <= ub
 distrib:        |b|            lbs             ubs
             [1, 10]            823             823

Variable bounds, lb <= x <= ub
 distrib:        |b|            lbs             ubs
                   0           8904
             [1, 10]                           8904

-------------------------------------------------------------------------------
```

G.2 arki001

```
Analyzing the problem

Constraints               Bounds                    Variables
 lower bd:        82       lower bd:        38       cont:       850
 upper bd:       946       fixed   :       353       bin :       415
 fixed   :        20       free    :         1       int :       123
                           ranged  :       996

-------------------------------------------------------------------------------

Objective, min cx
   range: all |c| in {0.00000, 1.00000}
 distrib:        |c|        vars
                   0        1387
                   1           1

-------------------------------------------------------------------------------

Constraint matrix A has
      1048 rows (constraints)
      1388 columns (variables)
     20439 (1.40511%) nonzero entries (coefficients)

Row nonzeros, A_i
   range: min A_i: 1 (0.0720461%)    max A_i: 1046 (75.3602%)
 distrib:        A_i        rows       rows%        acc%
                   1          29        2.77        2.77
```

```
                  2         476       45.42       48.19
             [3, 7]          49        4.68       52.86
            [8, 15]          56        5.34       58.21
           [16, 31]          64        6.11       64.31
           [32, 63]         373       35.59       99.90
       [1024, 1046]           1        0.10      100.00

Column nonzeros, A|j
   range: min A|j: 1 (0.0954198%)    max A|j: 29 (2.76718%)
 distrib:        A|j        cols       cols%        acc%
                  1         381       27.45       27.45
                  2          19        1.37       28.82
             [3, 7]          38        2.74       31.56
            [8, 15]         233       16.79       48.34
           [16, 29]         717       51.66      100.00

A nonzeros, A(ij)
   range: min |A(ij)|: 0.000200000     max |A(ij)|: 2.33067e+07
 distrib:      A(ij)      coeffs
     [0.0002, 0.001)         167
       [0.001, 0.01)        1049
         [0.01, 0.1)        4553
            [0.1, 1)        8840
             [1, 10)        3822
           [10, 100)         630
        [100, 1e+03)         267
      [1e+03, 1e+04)         699
      [1e+04, 1e+05)         291
      [1e+05, 1e+06)          83
      [1e+06, 1e+07)          19
   [1e+07, 2.33e+07]          19

--------------------------------------------------------------------------------

Constraint bounds, lb <= Ax <= ub
 distrib:        |b|            lbs            ubs
           [0.1, 1)                            386
            [1, 10)                             74
          [10, 100)            101             456
        [100, 1000)                             34
      [1000, 10000)                             15
    [100000, 1e+06]              1               1

Variable bounds, lb <= x <= ub
 distrib:        |b|            lbs            ubs
                  0            974             323
      [0.001, 0.01)                             19
           [0.1, 1)            370              57
            [1, 10)             41             704
          [10, 100]              2             246

--------------------------------------------------------------------------------
```

G.3 Problem with both linear and quadratic constraints

```
Analyzing the problem

Constraints               Bounds                    Variables
 lower bd:        40       upper bd:         1       cont: all
 upper bd:       121       fixed   :       204
 fixed   :      5480       free    :      5600
 ranged  :       161       ranged  :        40

-------------------------------------------------------------------------------

Objective, maximize cx
   range: all |c| in {0.00000, 15.4737}
 distrib:        |c|       vars
                   0       5844
             15.4737          1

-------------------------------------------------------------------------------

Constraint matrix A has
      5802 rows (constraints)
      5845 columns (variables)
      6480 (0.0191079%) nonzero entries (coefficients)

Row nonzeros, A_i
   range: min A_i: 0 (0%)    max A_i: 3 (0.0513259%)
 distrib:        A_i       rows       rows%        acc%
                   0         80        1.38        1.38
                   1       5003       86.23       87.61
                   2        680       11.72       99.33
                   3         39        0.67      100.00
0/80 empty rows have quadratic terms

Column nonzeros, A|j
   range: min A|j: 0 (0%)    max A|j: 15 (0.258532%)
 distrib:        A|j       cols       cols%        acc%
                   0        204        3.49        3.49
                   1       5521       94.46       97.95
                   2         40        0.68       98.63
              [3, 7]         40        0.68       99.32
             [8, 15]         40        0.68      100.00
0/204 empty columns correspond to variables used in conic
 and/or quadratic expressions only

A nonzeros, A(ij)
   range: min |A(ij)|: 2.02410e-05     max |A(ij)|: 35.8400
 distrib:      A(ij)      coeffs
  [2.02e-05, 0.0001)          40
     [0.0001, 0.001)         118
       [0.001, 0.01)         305
         [0.01, 0.1)         176
            [0.1, 1)          40
             [1, 10)        5721
          [10, 35.8]          80

-------------------------------------------------------------------------------
```

```
Constraint bounds, lb <= Ax <= ub
 distrib:        |b|           lbs            ubs
                   0          5481           5600
      [1000, 10000)                             1
    [10000, 100000)              2              1
     [1e+06, 1e+07)             78             40
     [1e+08, 1e+09]            120            120

Variable bounds, lb <= x <= ub
 distrib:        |b|           lbs            ubs
                   0           243            203
           [0.1, 1)              1              1
     [1e+06, 1e+07)                            40
     [1e+11, 1e+12]                             1

-------------------------------------------------------------------------------

Quadratic constraints: 121

Gradient nonzeros, Qx
   range: min Qx: 1 (0.0171086%)    max Qx: 2720 (46.5355%)
 distrib:         Qx        cons       cons%        acc%
                   1          40       33.06       33.06
          [64, 127]          80       66.12       99.17
       [2048, 2720]           1        0.83      100.00

-------------------------------------------------------------------------------
```

G.4 Problem with both linear and conic constraints

```
Analyzing the problem

Constraints              Bounds                    Variables
 upper bd:      3600      fixed   :      3601       cont: all
 fixed   :     21760      free    :     28802

-------------------------------------------------------------------------------

Objective, minimize cx
   range: all |c| in {0.00000, 1.00000}
 distrib:        |c|        vars
                   0       32402
                   1           1

-------------------------------------------------------------------------------

Constraint matrix A has
     25360 rows (constraints)
     32403 columns (variables)
     93339 (0.0113587%) nonzero entries (coefficients)

Row nonzeros, A_i
   range: min A_i: 1 (0.00308613%)    max A_i: 8 (0.0246891%)
```

```
 distrib:        A_i       rows       rows%        acc%
                   1       3600       14.20       14.20
                   2      10803       42.60       56.79
              [3, 7]       3995       15.75       72.55
                   8       6962       27.45      100.00

Column nonzeros, A|j
   range: min A|j: 0 (0%)    max A|j: 61 (0.240536%)
 distrib:        A|j       cols       cols%        acc%
                   0       3602       11.12       11.12
                   1      10800       33.33       44.45
                   2       7200       22.22       66.67
              [3, 7]       7279       22.46       89.13
             [8, 15]       3521       10.87      100.00
            [32, 61]          1        0.00      100.00
3600/3602 empty columns correspond to variables used in conic
 and/or quadratic constraints only

A nonzeros, A(ij)
   range: min |A(ij)|: 0.00833333     max |A(ij)|: 1.00000
 distrib:      A(ij)      coeffs
     [0.00833, 0.01)       57280
         [0.01, 0.1)          59
            [0.1, 1]       36000

-------------------------------------------------------------------------------

Constraint bounds, lb <= Ax <= ub
 distrib:        |b|           lbs             ubs
                   0          21760           21760
            [0.1, 1]                           3600

Variable bounds, lb <= x <= ub
 distrib:        |b|           lbs             ubs
             [1, 10]           3601            3601

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------

Rotated quadratic cones: 3600
                 dim       RQCs
                   4       3600
```

Bibliography

[1] Chvátal, V.. Linear programming, 1983. W.H. Freeman and Company

[2] Nazareth, J. L.. Computer Solution of Linear Programs, 1987. Oxford University Press, New York

[3] Bazaraa, M. S., Sherali, H. D. and Shetty, C. M.. Nonlinear programming: Theory and algorithms, 2 edition, 1993. John Wiley and Sons, New York

[4] Williams, H. P.. Model building in mathematical programming, 3 edition, 1993. John Wiley and Sons

[5] Cornuejols, Gerard and Tütüncü, Reha. Optimization methods in finance, 2007. Cambridge University Press, New York

[6] Ronald N. Kahn and Richard C. Grinold. Active portfolio management, 2 edition, 2000. McGraw-Hill, New York

[7] MOSEK ApS. MOSEK Modeling manual, 2012. Last revised January 31 2013. http://docs.mosek.com/generic/modeling-a4.pdf

[8] Andersen, E. D. and Andersen, K. D.. Presolving in linear programming. Math. Programming 2:221-245

[9] Andersen, E. D., Gondzio, J., Mészáros, Cs. and Xu, X.. Implementation of interior point methods for large scale linear programming, Interior-point methods of mathematical programming p. 189-252, 1996. Kluwer Academic Publishers

[10] Erling D. Andersen. The homogeneous and self-dual model and algorithm for linear optimization. Technical report TR-1-2009, 2009. MOSEK ApS. http://www.mosek.com/fileadmin/reports/tech/homolo.pdf

[11] Andersen, E. D. and Ye, Y.. Combining interior-point and pivoting algorithms. Management Sci. December 12:1719-1731

[12] Ahuja, R. K., Magnanti, T. L. and Orlin, J. B.. Network flows, Optimization, vol. 1 p. 211-369, 1989. North Holland, Amsterdam

[13] Andersen, E. D., Roos, C. and Terlaky, T.. On implementing a primal-dual interior-point method for conic quadratic optimization. Math. Programming February 2

[14] Andersen, E. D. and Ye, Y.. A computational study of the homogeneous algorithm for large-scale convex optimization. Computational Optimization and Applications 10:243-269

[15] Andersen, E. D. and Ye, Y.. On a homogeneous algorithm for the monotone complementarity problem. Math. Programming February 2:375-399

[16] Wolsey, L. A.. Integer programming, 1998. John Wiley and Sons

[17] Roos, C., Terlaky, T. and Vial, J. -Ph.. Theory and algorithms for linear optimization: an interior point approach, 1997. John Wiley and Sons, New York

[18] Wallace, S. W.. Decision making under uncertainty: Is sensitivity of any use. Oper. Res. January 1:20-25

[19] G. W. Stewart. Matrix Algorithms. Volume 1: Basic decompositions, 1998. P_SIAM

Index