



MOSEK Optimizer API for Python
Release 9.0.98

MOSEK ApS

15 July 2019

18 Interface changes	421
18.1 Backwards compatibility	421
18.2 Functions	421
18.3 Parameters	423
18.4 Constants	424
18.5 Response Codes	425
Bibliography	427
Symbol Index	428
Index	443

Chapter 1

Introduction

The **MOSEK** Optimization Suite 9.0.98 is a powerful software package capable of solving large-scale optimization problems of the following kind:

- linear,
- conic:
 - conic quadratic (also known as second-order cone),
 - involving the exponential cone,
 - involving the power cone,
 - semidefinite,
- convex quadratic and quadratically constrained,
- integer.

In order to obtain an overview of features in the **MOSEK** Optimization Suite consult the [product introduction](#) guide.

The most widespread class of optimization problems is *linear optimization problems*, where all relations are linear. The tremendous success of both applications and theory of linear optimization can be ascribed to the following factors:

- The required data are simple, i.e. just matrices and vectors.
- Convexity is guaranteed since the problem is convex by construction.
- Linear functions are trivially differentiable.
- There exist very efficient algorithms and software for solving linear problems.
- Duality properties for linear optimization are nice and simple.

Even if the linear optimization model is only an approximation to the true problem at hand, the advantages of linear optimization may outweigh the disadvantages. In some cases, however, the problem formulation is inherently nonlinear and a linear approximation is either intractable or inadequate. *Conic optimization* has proved to be a very expressive and powerful way to introduce nonlinearities, while preserving all the nice properties of linear optimization listed above.

The fundamental expression in linear optimization is a linear expression of the form

$$Ax - b \geq 0.$$

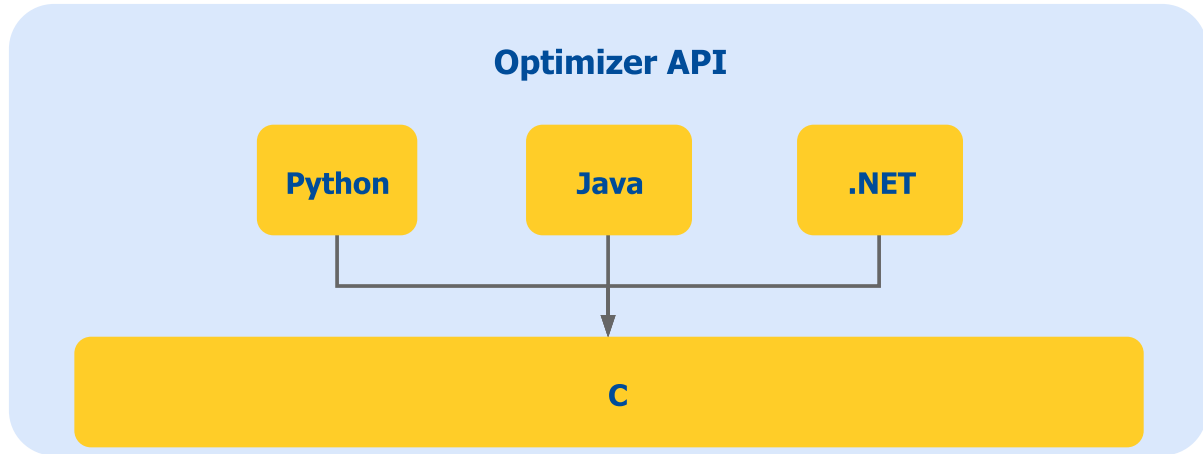
In conic optimization this is replaced with a wider class of constraints

$$Ax - b \in \mathcal{K}$$

where \mathcal{K} is a *convex cone*. For example in 3 dimensions \mathcal{K} may correspond to an ice cream cone. The conic optimizer in **MOSEK** supports a number of different types of cones \mathcal{K} , which allows a surprisingly large number of nonlinear relations to be modeled, as described in the **MOSEK** [Modeling Cookbook](#), while preserving the nice algorithmic and theoretical properties of linear optimization.

1.1 Why the Optimizer API for Python?

The Optimizer API for Python provides an object-oriented interface to the **MOSEK** optimizers. This object oriented design is common to Java, Python and .NET and is based on a thin class-based interface to the native C optimizer API. The overhead introduced by this mapping is minimal.



The Optimizer API for Python can be used with any application running on recent Python 2 and 3 interpreters. It consists of a single **mosek** package which can be used in Python scripts and interactive shells making it suited for fast prototyping and inspection of models.

The Optimizer API for Python provides access to:

- Linear Optimization (LO)
- Conic Quadratic (Second-Order Cone) Optimization (CQO, SOCO)
- Power Cone Optimization
- Conic Exponential Optimization (CEO)
- Convex Quadratic and Quadratically Constrained Optimization (QO, QCQO)
- Semidefinite Optimization (SDO)
- Mixed-Integer Optimization (MIO)

as well as to additional functions for

- problem analysis,
- sensitivity analysis,
- infeasibility diagnostics,
- BLAS/LAPACK linear algebra routines.

Chapter 2

Contact Information

Phone	+45 7174 9373	
Website	mosek.com	
Email		
	sales@mosek.com	Sales, pricing, and licensing
	support@mosek.com	Technical support, questions and bug reports
	info@mosek.com	Everything else.
Mailing Address		
	MOSEK ApS	
	Fruebjergvej 3	
	Symbion Science Park, Box 16	
	2100 Copenhagen O	
	Denmark	

You can get in touch with **MOSEK** using popular social media as well:

Blogger	https://blog.mosek.com/
Google Group	https://groups.google.com/forum/#!forum/mosek
Twitter	https://twitter.com/mosektw
Google+	https://plus.google.com/+Mosek/posts
Linkedin	https://www.linkedin.com/company/mosek-aps

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


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

Zstandard

MOSEK includes the *Zstandard* library developed by Facebook obtained from [github/zstd](https://github.com/facebook/zstd). The license agreement for *Zstandard* is shown in [Listing 3.3](#).

Listing 3.3: *Zstandard* license.

```
BSD License

For Zstandard software

Copyright (c) 2016-present, Facebook, Inc. All rights reserved.

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

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

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

* Neither the name Facebook nor the names of its contributors may be used to
  endorse or promote products derived from this software without specific
  prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```


Retrieving the solutions

When the model is set up, the optimizer is invoked with the call to `Task.optimize`. When the optimization is over, the user can check the results and retrieve numerical values. See further details in [Sec. 7](#).

We refer also to [Sec. 7](#) for information about more advanced mechanisms of interacting with the solver.

Source code example

Below is the most basic code sample that defines and solves a trivial optimization problem

$$\begin{array}{ll}\text{minimize} & x \\ \text{subject to} & 2.0 \leq x \leq 3.0.\end{array}$$

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

Listing 5.1: “Hello World!” in MOSEK

```
from mosek import *;

x = [ 0.0 ]

with Env() as env:                                # Create Environment
    with env.Task(0, 1) as task:                    # Create Task
        task.appendvars(1)                          # 1 variable x
        task.putcj(0, 1.0)                         # c_0 = 1.0
        task.putvarbound(0, boundkey.ra, 2.0, 3.0) # 2.0 <= x <= 3.0
        task.putobjsense(objsense.minimize)         # minimize

        task.optimize()                             # Optimize

        task.getxx(soltype.itr, x)                  # Get solution
        print("Solution x = {}".format(x[0]))        # Print solution
```

Chapter 6

Optimization Tutorials

In this section we demonstrate how to set up basic types of optimization problems. Each short tutorial contains a working example of formulating problems, defining variables and constraints and retrieving solutions.

6.1 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

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

subject to the linear constraints

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

and the bounds

$$l_j^x \leq x_j \leq u_j^x, \quad j = 0, \dots, n-1.$$

The problem description consists of the following elements:

- m and n — the number of constraints and variables, respectively,
- x — the variable vector of length n ,
- c — the coefficient vector of length n

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

- c^f — fixed term in the objective,
- A — an $m \times n$ matrix of coefficients

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

Source code

The complete source code `lo1.py` of this example appears below. See also `lo2.py` for a version where the A matrix is entered row-wise.

Listing 6.1: Linear optimization example.

```
import sys
import mosek

# 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
            bkc = [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 = [[0, 1],
                    [0, 1, 2],
                    [0, 1],
                    [1, 2]]
            aval = [[3.0, 2.0],
                    [1.0, 1.0, 2.0],
                    [2.0, 3.0],
                    [1.0, 3.0]]

            numvar = len(bkc)
```

(continues on next page)

```

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, bkc[j], blx[j], bux[j])

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

# 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):
    xx = [0.] * numvar
    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):
    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.Error as e:
    print("ERROR: %s" % str(e.errno))
    if e.msg is not None:

```

(continues on next page)


```

# Create a task
with env.Task() as 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]
    numvar = 3
    bkc = [mosek.boundkey.lo] * numvar
    blx = [0.0] * numvar
    bux = [inf] * numvar
    c = [0.0, -1.0, 0.0]
    asub = [[0], [0], [0]]
    aval = [[1.0], [1.0], [1.0]]

    numvar = len(bkc)
    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, bkc[j], blx[j], bux[j])
        # Input column j of A
        task.putacol(j,
                     # Variable (column) index.
                     # Row index of non-zeros in column j.
                     asub[j],
                     aval[j]) # Non-zero Values of column j.
    for i in range(numcon):
        task.putconbound(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)

    # Input the objective sense (minimize/maximize)
    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 = [0.] * numvar

```



```

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)

```

While `Task.putqconk` adds quadratic terms to a specific constraint, it is also possible to input all quadratic terms in one chunk using the `Task.putqcon` function.

Source code

Listing 6.3: Implementation of the quadratically constrained problem (6.5).

```

import sys
import mosek

# Since the actual 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 a MOSEK environment
    with mosek.Env() as env:
        # Attach a printer to the environment
        env.set_Stream(mosek.streamtype.log, streamprinter)

        # Create a task
        with env.Task(0, 0) as task:
            # 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]

            bkc = [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 = [[0], [0], [0]]
            aval = [[1.0], [1.0], [1.0]]

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

```

(continues on next page)

```

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

for i in range(numcon):
    task.putconbound(i, bkg[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 = [0.] * numvar
task.getxx(mosek.soltype.itr,

```



```

sys.stdout.write(text)
sys.stdout.flush()

def main():
    # Make a MOSEK environment
    with mosek.Env() as env:
        # Attach a printer to the environment
        env.set_Stream(mosek.streamtype.log, streamprinter)

        # Create a task
        with env.Task(0, 0) as task:
            # 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]
            bkc = [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 = [[0], [0], [0]]
            aval = [[1.0], [1.0], [2.0]]

            numvar = len(bkc)
            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.putvarbound(j, bkc[j], blx[j], bux[j])

            for j in range(len(aval)):
                # Input column j of A
                task.putacol(j,
                             # Variable (column) index.
                             # Row index of non-zeros in column j.
                             asub[j],
                             # Non-zero Values of column j.
                             aval[j])
            for i in range(numcon):
                task.putconbound(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 = [0.] * numvar
task.getxx(mosek.soltype.itr,
        xx)

if solsta == mosek.solsta.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 mosek.solsta.unknown:
    print("Unknown solution status")
else:
    print("Other solution status")

# call the main function
try:
    main()
except mosek.MosekException as e:
    print("ERROR: %s" % str(e.errno))
    print("\t%s" % e.msg)
    sys.exit(1)
except:
    import traceback
    traceback.print_exc()
    sys.exit(1)

```

6.4 Power Cone Optimization

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

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

where x^t is a subset of the problem variables and \mathcal{K}_t is a convex cone. Since the set \mathbb{R}^n of real numbers is also a convex cone, we can simply write a compound conic constraint $x \in \mathcal{K}$ where $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_l$ is a product of smaller cones and x is the full problem variable.

MOSEK can solve conic optimization problems of the form

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

The first argument selects the type of power cone, that is `conetype.ppow`. The second argument is the cone parameter α . The remaining arguments list the variables which form the cone. Variants of this method are available to append multiple cones at a time.

The code below produces the answer of (6.7) which is

[0.06389298 0.78308564 2.30604283]

Source code

Listing 6.5: Source code solving problem (6.7).

```
import sys
import mosek

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

def main():

    # Only a symbolic constant
    inf = 0.0

    # Make a MOSEK environment
    with mosek.Env() as env:
        # Attach a printer to the environment
        env.set_Stream(mosek.streamtype.log, streamprinter)

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

            csub = [3, 4, 0]
            cval = [1.0, 1.0, -1.0]
            asub = [0, 1, 2]
            aval = [1.0, 1.0, 0.5]
            numvar, numcon = 6, 1

            # 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)

            # Set up the linear part of the problem
            task.putclist(csub, cval)
            task.putarow(0, asub, aval)
            task.putvarboundslice(0, numvar, [mosek.boundkey.fr] * numvar, [inf] * numvar,
↳[inf] * numvar)
            task.putvarbound(5, mosek.boundkey.fx, 1.0, 1.0) # x_5 = 1
            task.putconbound(0, mosek.boundkey.fx, 2.0, 2.0)

            # Input the cones
            task.appendcone(mosek.conetype.ppow, 0.2, [0, 1, 3])
            task.appendcone(mosek.conetype.ppow, 0.4, [2, 5, 4])

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

(continues on next page)

```

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

# Output a solution
xx = [0.] * numvar
task.getxx(mosek.soltype.itr,
           xx)

if solsta == mosek.solsta.optimal:
    print("Optimal solution: %s" % xx[0:3])
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 mosek.solsta.unknown:
    print("Unknown solution status")
else:
    print("Other solution status")

# call the main function
try:
    main()
except mosek.MosekException 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)

```

6.5 Conic Exponential Optimization

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

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

where x^t is a subset of the problem variables and \mathcal{K}_t is a convex cone. Since the set \mathbb{R}^n of real numbers is also a convex cone, we can simply write a compound conic constraint $x \in \mathcal{K}$ where $\mathcal{K} = \mathcal{K}_1 \times \cdots \times \mathcal{K}_l$ is a product of smaller cones and x is the full problem variable.

MOSEK can solve conic optimization problems of the form

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

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

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

Source code

Listing 6.6: Source code solving problem (6.9).

```
import sys
import mosek

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

def main():

    # Only a symbolic constant
    inf = 0.0

    # Make a MOSEK environment
    with mosek.Env() as env:
        # Attach a printer to the environment
        env.set_Stream(mosek.streamtype.log, streamprinter)

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

            c = [1.0, 1.0, 0.0]
            a = [1.0, 1.0, 1.0]
            numvar, numcon = 3, 1

            # 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)

            # Set up the linear part of the problem
            task.putcslice(0, numvar, c)
            task.putarow(0, [0, 1, 2], a)
            task.putvarboundslice(0, numvar, [mosek.boundkey.fr] * numvar, [inf] * numvar,
↪ [inf] * numvar)
            task.putconbound(0, mosek.boundkey.fx, 1.0, 1.0)

            # Input the cones
            task.appendcone(mosek.conetype.pexp,
                           0.0,
                           [0, 1, 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)
```

(continues on next page)


```

buc = [1.0, 0.5]

# Below is the sparse representation of the A
# matrix stored by row.
asub = [[0],
        [1, 2]]
aval = [[1.0],
        [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 = [[0, 1, 2],
         [0, 1, 2, 1, 2, 2]]
baraj = [[0, 1, 2],
         [0, 0, 0, 1, 1, 2]]
baraval = [[1.0, 1.0, 1.0],
           [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,
                 asub[i],
                 aval[i])
    # Constraint (row) index.
    # Column index of non-zeros in constraint i.
    # Non-zero values of row i.

# Add the quadratic cone constraint
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)

# 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):
    xx = [0.] * numvar
    task.getxx(mosek.soltype.itr, xx)

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

    print("Optimal solution:\nxx=%s\nbarx=%s" % (xx, barx))
elif (solsta == mosek.solsta.dual_infeas_cer or
      solsta == mosek.solsta.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.MosekException 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)

```



```

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

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

c = [1.0, 0.64]

asub = [[0, 1], [0, 1]]
aval = [[50.0, 3.0], [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.
                 # Row index of non-zeros in column j.
                 asub[j],
                 # Non-zero Values of column j.
                 aval[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])

# Set max solution time
task.putdparam(mosek.dparam.mio_max_time, 60.0);

# 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)

# Output a solution

```



```

blc = [-inf, -inf, -inf]
buc = [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 = [0.0, 0.0, 0.0]
bux = [+inf, +inf, +inf]
# Objective coefficients
csub = [0, 1, 2]
cval = [1.5, 2.5, 3.0]
# We input the A matrix column-wise
# asub contains row indexes
asub = [0, 1, 2,
        0, 1, 2,
        0, 1, 2]
# acof contains coefficients
acof = [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 = [0, 3, 6]
aptre = [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 = [0.] * numvar
task.getsolutionslice(mosek.soltype.bas,

```


Listing 6.42: Calling the parallel optimizer.

```
# Example of how to use ``paropt``.
# Optimizes tasks whose names were read from command line.
def main(argv):
    n = len(argv) - 1
    tasks = []

    with mosek.Env() as env:
        for i in range(n):
            t = mosek.Task(env, 0, 0)
            t.readdata(argv[i+1])
            # Each task will be single-threaded
            t.putintparam(mosek.iparam.intpnt_multi_thread, mosek.onoffkey.off)
            tasks.append(t)

    res, trm = paropt(tasks)

    for i in range(n):
        print("Task {0}  res {1}  trm {2}  obj_val {3}  time {4}".format(
            i,
            res[i],
            trm[i],
            tasks[i].getdouinf(mosek.dinfitem.intpnt_primal_obj),
            tasks[i].getdouinf(mosek.dinfitem.optimizer_time)))
```

Another, slightly more advanced application of the parallel optimizer is presented in [Sec. 11.3](#). For a more in-depth treatment see the following sections:

- *Case studies* for more advanced and complicated optimization examples.
- *Problem Formulation and Solutions* for formal mathematical formulations of problems **MOSEK** can solve, dual problems and infeasibility certificates.

Table 7.3: Integer problems (solution status for integer solution, others undefined)

Outcome	Problem status	Solution status
Integer optimal	<i>prosta.prim_feas</i>	<i>solsta.integer_optimal</i>
Infeasible	<i>prosta.prim_infeas</i>	<i>solsta.unknown</i>
Integer feasible point	<i>prosta.prim_feas</i>	<i>solsta.prim_feas</i>
No conclusion	<i>prosta.unknown</i>	<i>solsta.unknown</i>

7.1.4 Retrieving solution values

After the meaning and quality of the solution (or certificate) have been established, we can query for the actual numerical values. They can be accessed using:

- *Task.getprimalobj*, *Task.getdualobj* — the primal and dual objective value.
- *Task.getxx* — solution values for the variables.
- *Task.getsolution* — a full solution with primal and dual values

and many more specialized methods, see the *API reference*.

7.1.5 Source code example

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

Listing 7.1: Sample framework for checking optimization result.

```
import mosek
import sys

# A log message
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main(args):
    filename = args[0] if len(args) >= 1 else "../data/cqo1.mps"

    try:
        # Create environment and task
        with mosek.Env() as env:
            with env.Task(0, 0) as task:
                # (Optional) set a log stream
                # task.set_Stream(mosek.streamtype.log, streamprinter)

                # (Optional) uncomment to see what happens when solution status is unknown
                # task.putintparam(mosek.iparam.intpnt_max_iterations, 1)

                # In this example we read data from a file
                task.readdata(filename)

                # Optimize
                trmcode = task.optimize()
                task.solutionsummary(mosek.streamtype.log)

                # We expect solution status OPTIMAL
                solsta = task.getsolsta(mosek.soltype.itr)

                if solsta == mosek.solsta.optimal:
                    # Optimal solution. Fetch and print it.
```

(continues on next page)

Optimizer errors and warnings

The optimizer may also produce warning messages. They indicate non-critical but important events, that will not prevent solver execution, but may be an indication that something in the optimization problem might be improved. Warning messages are normally printed to a log stream (see [Sec. 7.3](#)). A typical warning is, for example:

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

Warnings can also be suppressed by setting the `iparam.max_num_warnings` parameter to zero, if they are well-understood.

Error and solution status handling example

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

Listing 7.2: Sample framework for checking optimization result.

```
import mosek
import sys

# A log message
def streamprinter(text):
    sys.stdout.write(text)
    sys.stdout.flush()

def main(args):
    filename = args[0] if len(args) >= 1 else "../data/cqo1.mps"

    try:
        # Create environment and task
        with mosek.Env() as env:
            with env.Task(0, 0) as task:
                # (Optional) set a log stream
                # task.set_Stream(mosek.streamtype.log, streamprinter)

                # (Optional) uncomment to see what happens when solution status is unknown
                # task.putintparam(mosek.iparam.intpnt_max_iterations, 1)

                # In this example we read data from a file
                task.readdata(filename)

                # Optimize
                trmcode = task.optimize()
                task.solutionsummary(mosek.streamtype.log)

                # We expect solution status OPTIMAL
                solsta = task.getsolsta(mosek.soltype.itr)

                if solsta == mosek.solsta.optimal:
                    # Optimal solution. Fetch and print it.
                    print("An optimal interior-point solution is located.")
                    numvar = task.getnumvar()
                    xx = [ 0.0 ] * numvar
                    task.getxx(mosek.soltype.itr, xx)
                    for i in range(numvar):
                        print("x[{0}] = {1}".format(i, xx[i]))

                elif solsta == mosek.solsta.dual_infeas_cer:
                    print("Dual infeasibility certificate found.")
```

(continues on next page)

Listing 7.5: An example of a data callback function.

```
def makeUserCallback(maxtime, task):
    xx = numpy.zeros(task.getnumvar())    # Space for integer solutions

    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.new_int_mio:
            print("New integer solution has been located.")
            task.getxx(soltype.itg, xx)
            print(xx)
            print("Obj.: %f" % douinf[dinfitem.mio_obj_int])
        else:
            pass

        if opttime >= maxtime:
            # mosek is spending too much time. Terminate it.
            print("Terminating.")
            return 1
```

(continues on next page)


```

# line argument (received in `argv`)
task.readdata(inputfile)

# Solve the problem remotely
task.optimizermt(host, port)

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

```

7.7.2 Asynchronous Remote Optimization

In asynchronous mode the client sends a job to the remote server and the execution of the client code continues. In particular, it is the client's responsibility to periodically check the optimization status and, when ready, fetch the results. The client can also interrupt optimization. The most relevant methods are:

- *Task.asyncoptimize* : Offload the optimization task to a solver server.
- *Task.asyncpoll* : Request information about the status of the remote job.
- *Task.asyncgetresult* : Request the results from a completed remote job.
- *Task.asyncstop* : Terminate a remote job.

Source code example

In the example below the program enters in a polling loop that regularly checks whether the result of the optimization is available.

Listing 7.8: Using the OptServer in asynchronous mode.

```

import mosek
import sys
import time

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

if len(sys.argv) != 5:
    print("Missing argument, syntax is:")
    print("  opt-server-async inputfile host port numpolls")
else:
    filename = sys.argv[1]
    host = sys.argv[2]
    port = sys.argv[3]
    numpolls = int(sys.argv[4])
    token = None

    with mosek.Env() as env:
        with env.Task(0, 0) as task:
            print("reading task from file")
            task.readdata(filename)

            print("Solve the problem remotely (async)")
            token = task.asyncoptimize(host, port)

    print("Task token: %s" % token)

```

(continues on next page)

```

with env.Task(0, 0) as task:

    task.readdata(filename)

    task.set_Stream(mosek.streamtype.log, streamprinter)

    i = 0

    while i < numpolls:

        time.sleep(0.1)

        print("poll %d..." % i)
        respavailable, res, trm = task.asyncpoll(host,
                                                port,
                                                token)

        print("done!")

        if respavailable:
            print("solution available!")

            respavailable, res, trm = task.asyncgetresult(host,
                                                        port,
                                                        token)

            task.solutionsummary(mosek.streamtype.log)
            break

        i = i + 1

    if i == numpolls:
        print("max number of polls reached, stopping host.")
        task.asyncstop(host, port, token)

```



```

# 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 (9.4) 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

```



```

        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.
    task.deletesolution(mosek.soltype.bas);

    task.putskcslice(mosek.soltype.bas, 0, numvar, skc);
    task.putskxslice(mosek.soltype.bas, 0, numvar, skx);

    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]

            setup(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")

```



```

env.gemv(mosek.transpose.no, m, n, alpha, A, x, beta, z)
print("\ngemv results is ")
print_matrix(z, 1, len(z))

env.gemm(mosek.transpose.no, mosek.transpose.no,
         m, n, k, alpha, A, B, beta, C)
print("\ngemm results is ")
print_matrix(C, m, n)

env.syrk(mosek.uplo.lo, mosek.transpose.no, m, k, alpha, A, beta, D)
print("\nsyrk results is ")
print_matrix(D, m, m)

# LAPACK routines

env.potrf(mosek.uplo.lo, m, Q)
print("\npotrf results is ")
print_matrix(Q, m, m)

env.syeig(mosek.uplo.lo, m, Q, v)
print("\nsyeig results is ")
print_matrix(v, 1, m)

env.syevd(mosek.uplo.lo, m, Q, v)
print("\nsyevd results is ")
print('v: ')
print_matrix(v, 1, m)
print('Q: ')
print_matrix(Q, m, m)

print("Exiting...")

```

9.3 Computing a Sparse Cholesky Factorization

Given a positive semidefinite symmetric (PSD) matrix

$$A \in \mathbb{R}^{n \times n}$$

it is well known there exists a matrix L such that

$$A = LL^T.$$

If the matrix L is lower triangular then it is called a *Cholesky factorization*. Given A is positive definite (nonsingular) then L is also nonsingular. A Cholesky factorization is useful for many reasons:

- A system of linear equations $Ax = b$ can be solved by first solving the lower triangular system $Ly = b$ followed by the upper triangular system $L^T x = y$.
- A quadratic term $x^T A x$ in a constraint or objective can be replaced with $y^T y$ for $y = L^T x$, potentially leading to a more robust formulation (see [\[And13\]](#)).

Therefore, **MOSEK** provides a function that can compute a Cholesky factorization of a PSD matrix. In addition a function for solving linear systems with a nonsingular lower or upper triangular matrix is available.

In practice A may be very large with n is in the range of millions. However, then A is typically sparse which means that most of the elements in A are zero, and sparsity can be exploited to reduce the cost

(continued from previous page)

```
- 2.2e+01 x0000_x0 - 1.45e+01 x0001_x1 + 1.2e+01 x0002_x2 + x0003
+ 1e+00
[/objective]

[constraints]
[con c0000] 3.605551275463989e+00 x0000_x0 - 5.547001962252291e-01 x0002_x2 + 3.
↪ 328201177351375e+00 x0001_x1 - x0006 = 0e+00 [/con]
[con c0001] 3.419401657060442e+00 x0002_x2 + 2.294598480395823e+00 x0001_x1 - x0007 = 0e+00 ↪
↪ [/con]
[con c0002] 8.111071056538127e-01 x0001_x1 - x0008 = 0e+00 [/con]
[con c0003] - x0003 + x0004 = 0e+00 [/con]
[/constraints]

[bounds]
[b] -1e+00      <= x0000_x0,x0001_x1,x0002_x2 <= 1e+00 [/b]
[b]              x0003,x0004 free [/b]
[b]              x0005 = 1e+00 [/b]
[b]              x0006,x0007,x0008 free [/b]
[cone rquad k0000] x0004, x0005, x0006, x0007, x0008 [/cone]
[/bounds]
```

We can clearly see that constraints c0000, c0001 and c0002 represent the original linear constraints as in (9.11), while c0003 corresponds to (9.10). The cone roots are x0005 and x0004.

Chapter 11

Case Studies

In this section we present some case studies in which the Optimizer API for Python is used to solve real-life applications. These examples involve some more advanced modeling skills and possibly some input data. The user is strongly recommended to first read the basic tutorials of [Sec. 6](#) before going through these advanced case studies.

- *Portfolio Optimization*
 - **Keywords:** Markowitz model, variance, risk, efficient frontier, transaction cost, market impact cost
 - **Type:** Conic Quadratic, Power Cone, Mixed-Integer Optimization
- *Logistic regression*
 - **Keywords:** machine learning, logistic regression, classifier, log-sum-exp, softplus, regularization
 - **Type:** Exponential Cone, Quadratic Cone
- *Concurrent Optimizer*
 - **Keywords:** Concurrent optimization
 - **Type:** Linear Optimization, Mixed-Integer Optimization

11.1 Portfolio Optimization

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

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


```

# 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(2 + 2 * n)

offsetx = 0    # Offset of variable x into the API variable.
offsets = n    # Offset of variable s into the API variable.
offsett = n + 1 # Offset of variable t into the API variable.
offsetu = 2*n + 1 # Offset of variable u 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.putvarboundsliceconst(offsetx, offsetx + n, mosek.boundkey.lo, 0.0, inf)

for j in range(0, n):
    task.putvarname(offsetx + j, "x[%d]" % (1 + j))

# s variable.
task.putvarbound(offsets + 0, mosek.boundkey.fr, -inf, inf)
task.putvarname(offsets + 0, "s")

# u variable.
task.putvarbound(offsetu + 0, mosek.boundkey.fx, 0.5, 0.5)
task.putvarname(offsetu + 0, "u")

# t variables.
task.putaijlist(range(1, n + 1), range(offsett +
                                       0, offsett + n), n * [-1.0])
task.putvarboundsliceconst(offsett, offsett + n, mosek.boundkey.fr, -inf, inf)
for j in range(0, n):
    task.putvarname(offsett + j, "t[%d]" % (1 + j))

task.appendcone(mosek.conetype.rquad, 0.0,
                [offsets, offsetu] + list(range(offsett, offsett + n)))
task.putconename(0, "variance")

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)

```


The next step is to consider how the linear constraint matrix A and the remaining data vectors are laid out. Reusing the idea in [Sec. 11.1.1](#) we can write the data in block matrix form and read off all the required coordinates. This extension of the code setting up the constraint $G^T x - t = 0$ from [Sec. 11.1.1](#) is shown below.

Source code example

The example code in [Listing 11.6](#) demonstrates how to implement the model (11.17).

Listing 11.6: Code implementing model (11.17).

```
import mosek

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 + 3 * 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))

            # Offset of variables into the API variable.
            offsetx = 0
```

(continues on next page)

```

offsets = n
offsett = n + 1
offsetc = 2 * n + 1
offsetz = 3 * n + 1
offsetf = 4 * n + 1

# Variables.
task.appendvars(1 + 5 * 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.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))

# 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.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.
task.putvarboundlist(range(offsetf, offsetf + n),
    n * [mosek.boundkey.fx], n * [1.0], n * [1.0])
for j in range(0, n):
    task.putvarname(offsetf + j, "f[%d]" % (1 + j))

# quadratic cone
task.appendcone(mosek.conetype.quad, 0.0, [

```

(continues on next page)

Example code

The following example code demonstrates how to compute an optimal portfolio when transaction setup costs are included. Note that we are now solving a problem with integer variables, and therefore the solution must be retrieved from `soltype.itg` rather than `soltype.itr`.

Listing 11.8: Code solving problem (11.18).

```
import mosek

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
    f = [0.01, 0.01, 0.01]
    g = [0.001, 0.001, 0.001]

    # 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)

            # Total wealth
            U = w + sum(x0)

            # Constraints.
            task.appendcons(1 + 4 * n)
            task.putconbound(0, mosek.boundkey.fx, U, U)
            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 + 4 * n),
                                  n * [mosek.boundkey.up], n * [-inf], n * [0.0])
            for i in range(0, n):
                task.putconname(1 + 3 * n + i, "ind[%d]" % (1 + i))
```

(continues on next page)

(continued from previous page)

```
task.appendcone(mosek.conetype.quad, 0.0, [
    offsets] + list(range(offsett, offsett + n)))
task.putconename(0, "stddev")

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 = [0.] * n
task.getxxslice(mosek.soltype.itg, offsetx + 0, offsetx + n, x)
for j in range(0, n):
    expret += mu[j] * x[j]

stddev = [0.]
task.getxxslice(mosek.soltype.itg, offsets +
    0, offsets + 1, stddev)

print("\nExpected return %e for gamma %e\n" % (expret, stddev[0]))
```

11.1.7 Cardinality constraints

Another method to reduce costs involved with processing transactions is to only change positions in a small number of assets. In other words, at most k of the differences $|\Delta x_j| = |x_j - x_j^0|$ are allowed to be non-zero, where k is (much) smaller than the total number of assets n .

This type of constraint can be again modeled by introducing a binary variable y_j which indicates if $\Delta x_j \neq 0$ and bounding the sum of y_j . The basic Markowitz model then gets updated as follows:

$$\begin{aligned} & \text{maximize} && \mu^T x \\ & \text{subject to} && e^T x = w + e^T x^0, \\ & && [\gamma; G^T x] \in Q^{n+1}, \\ & && U_j y_j \geq |x_j - x_j^0|, \quad j = 1, \dots, n, \\ & && y_j \in \{0, 1\}, \quad j = 1, \dots, n, \\ & && e^T y \leq k, \\ & && x \geq 0, \end{aligned} \tag{11.20}$$

where U_j is some a priori chosen upper bound on the amount of trading in asset j . This guarantees that $|x_j - x_j^0|$ forces $y_j = 1$ and therefore $e^T y$ counts the number of assets in which we trade. In our problem a safe bound for each U_j is the total initial wealth $w + e^T x^0$, however knowing a tighter bound may lead to shorter solution times.


```

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 + 4 * n),
                    n * [mosek.boundkey.up], n * [-inf], n * [0.0])
for i in range(0, n):
    task.putconname(1 + 3 * n + i, "ind[%d]" % (1 + i))

# Offset of variables into the API variable.
offsetx = 0
offsets = n
offsett = n + 1
offsetz = 2 * n + 1
offsety = 3 * n + 1

# Variables.
task.appendvars(1 + 4 * 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))

# 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])

```

(continues on next page)

Listing 11.12: Simple callback function which signals the optimizer to stop.

```
# Defines a Mosek callback function whose only function
# is to indicate if the optimizer should be stopped.
stop = False
firstStop = -1
def cbFun(code):
    return 1 if stop else 0
```

When all remaining tasks respond to the stop signal, response codes and statuses are returned to the caller, together with the index of the task which won the race.

Listing 11.13: A routine for parallel task race.

```
def runTask(num, task, res, trm):
    global stop
    global firstStop
    try:
        trm[num] = task.optimize();
        res[num] = mosek.rescode.ok
    except mosek.MosekException as e:
        trm[num] = mosek.rescode.err_unknown
        res[num] = e.errno
    finally:
        # If this finished with success, inform other tasks to interrupt
        if res[num] == mosek.rescode.ok:
            if not stop:
                firstStop = num
            stop = True

def optimize(tasks):
    n = len(tasks)
    res = [ mosek.rescode.err_unknown ] * n
    trm = [ mosek.rescode.err_unknown ] * n

    # Set a callback function
    for t in tasks:
        t.set_Progress(cbFun)

    # Start parallel optimizations, one per task
    jobs = [ Thread(target=runTask, args=(i, tasks[i], res, trm)) for i in range(n) ]
    for j in jobs:
        j.start()
    for j in jobs:
        j.join()

    # For debugging, print res and trm codes for all optimizers
    for i in range(n):
        print("Optimizer {0}   res {1}   trm {2}".format(i, res[i], trm[i]))

    return firstStop, res, trm
```

11.3.2 Linear optimization

We use the multithreaded setup to run the interior-point and simplex optimizers simultaneously on a linear problem. The next methods simply clones the given task and sets a different optimizer for each. The result is the clone which finished first.

(continued from previous page)

```
for i in range(n):
    if ((res[i] == mosek.rescode.ok) and
        (tasks[i].getsolsta(mosek.soltype.itg) == mosek.solsta.prim_feas or
         tasks[i].getsolsta(mosek.soltype.itg) == mosek.solsta.integer_optimal) and
        ((tasks[i].getprimalobj(mosek.soltype.itg) < bestObj)
         if (sense == mosek.objsense.minimize) else
         (tasks[i].getprimalobj(mosek.soltype.itg) > bestObj))):
        bestObj = tasks[i].getprimalobj(mosek.soltype.itg)
        bestPos = i

if bestPos >= 0:
    return bestPos, tasks[bestPos], trm[bestPos], res[bestPos]

return -1, None, None, None
```

It remains to call the method with a choice of seeds, for example:

Listing 11.17: Calling concurrent integer optimization.

```
seeds = [ 42, 13, 71749373 ]

idx, t, trm, res = optimizeconcurrentMIO(task, seeds)
```


Chapter 13

Optimizers

The most essential part of **MOSEK** are the optimizers:

- *primal simplex* (linear problems),
- *dual simplex* (linear problems),
- *interior-point* (linear, quadratic and conic problems),
- *mixed-integer* (problems with integer variables).

The structure of a successful optimization process is roughly:

- **Presolve**
 1. *Elimination*: Reduce the size of the problem.
 2. *Dualizer*: Choose whether to solve the primal or the dual form of the problem.
 3. *Scaling*: Scale the problem for better numerical stability.
- **Optimization**
 1. *Optimize*: Solve the problem using selected method.
 2. *Terminate*: Stop the optimization when specific termination criteria have been met.
 3. *Report*: Return the solution or an infeasibility certificate.

The preprocessing stage is transparent to the user, but useful to know about for tuning purposes. The purpose of the preprocessing steps is to make the actual optimization more efficient and robust. We discuss the details of the above steps in the following sections.

13.1 Presolve

Before an optimizer actually performs the optimization the problem is preprocessed using the so-called presolve. The purpose of the presolve is to

1. remove redundant constraints,
2. eliminate fixed variables,
3. remove linear dependencies,
4. substitute out (implied) free variables, and
5. reduce the size of the optimization problem in general.

After the presolved problem has been optimized the solution is automatically postsolved so that the returned solution is valid for the original problem. Hence, the presolve is completely transparent. For further details about the presolve phase, please see [\[AA95\]](#) and [\[AGMX96\]](#).

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

(continued from previous page)

Number of branches	: 4425
Number of relaxations solved	: 4410
Number of interior point iterations	: 25
Number of simplex iterations	: 221131

The first lines contain a summary of the problem as seen by the optimizer. This is followed by the iteration log. The columns have the following meaning:

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

Following that a summary of the optimization process is printed.

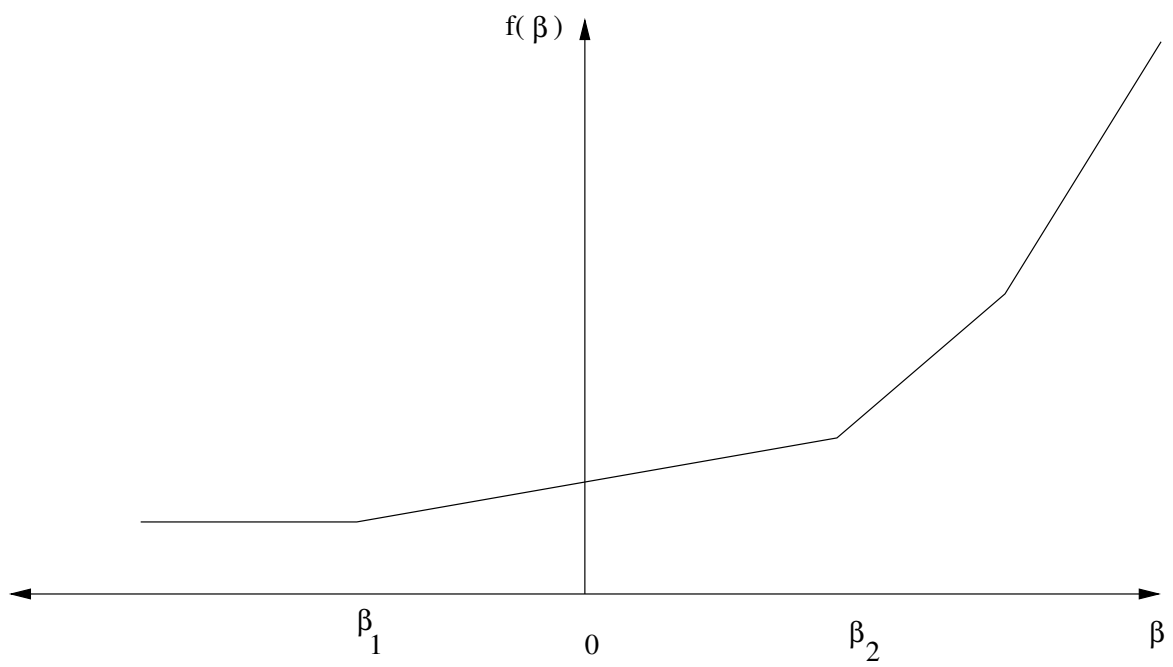


Fig. 14.1: $\beta = 0$ is in the interior of linearity interval.

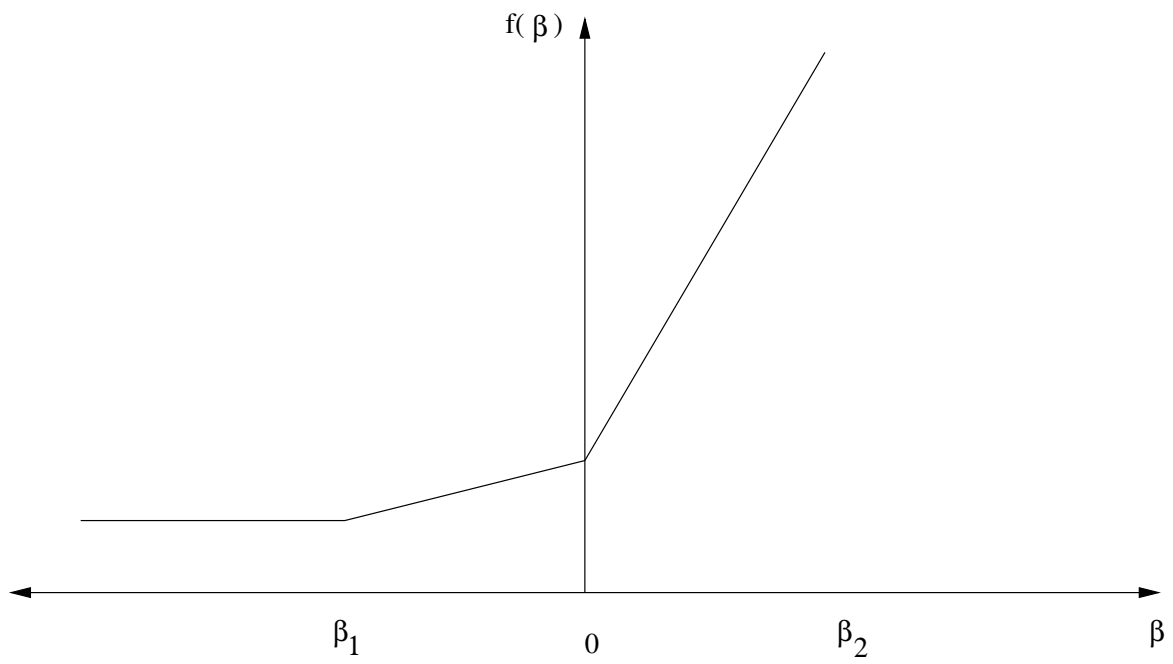


Fig. 14.2: $\beta = 0$ is a breakpoint.


```

        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 = [0., 0.]
rightprice = [0., 0.]
leftrange = [0., 0.]
rightrange = [0., 0.]
subc = [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.MosekException 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 15

API Reference

This section contains the complete reference of the **MOSEK** Optimizer API for Python. It is organized as follows:

- *General API conventions.*
- **Methods:**
 - *Class Env* (The **MOSEK** environment)
 - *Class Task* (An optimization task)
 - *Browse by topic*
- **Optimizer parameters:**
 - *Double, Integer, String*
 - *Full list*
 - *Browse by topic*
- **Optimizer information items:**
 - *Double, Integer, Long*
- *Optimizer response codes*
- *Enumerations*
- *Exceptions*
- *User-defined function types*
- *Nonlinear API (SCopt)*

15.1 API Conventions

15.1.1 Function arguments

Naming Convention

In the definition of the **MOSEK** Optimizer API for Python 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 15.1](#) the variable names used to specify the problem parameters are listed.

Problem data - constraints

- *Task.appendcons* – Appends a number of constraints to the optimization task.
- *Task.getnumcon* – Obtains the number of constraints.
- *Task.putconbound* – Changes the bound for one constraint.
- *Task.putconboundslice* – Changes the bounds for a slice of the constraints.
- *Task.putconname* – Sets the name of a constraint.
- *Task.removecons* – Removes a number of constraints.
- *Infrequent:* *Task.chgconbound*, *Task.generateconnames*, *Task.getconbound*, *Task.getconboundslice*, *Task.getconname*, *Task.getconnameindex*, *Task.getconnamelen*, *Task.getmaxnumcon*, *Task.getnumqconknz*, *Task.getqconk*, *Task.inputdata*, *Task.putconboundlist*, *Task.putconboundlistconst*, *Task.putconboundsliceconst*, *Task.putmaxnumcon*

Problem data - linear part

- *Task.appendcons* – Appends a number of constraints to the optimization task.
- *Task.appendvars* – Appends a number of variables to the optimization task.
- *Task.getnumcon* – Obtains the number of constraints.
- *Task.putacol* – Replaces all elements in one column of the linear constraint matrix.
- *Task.putacolslice* – Replaces all elements in a sequence of columns the linear constraint matrix.
- *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 the linear constraint matrix.
- *Task.putarowslice* – Replaces all elements in several rows the linear constraint matrix.
- *Task.putcfix* – Replaces the fixed term in the objective.
- *Task.putcj* – Modifies one linear coefficient in the objective.
- *Task.putconbound* – Changes the bound for one constraint.
- *Task.putconboundslice* – Changes the bounds for a slice of the constraints.
- *Task.putconname* – Sets the name of a constraint.
- *Task.putcslice* – Modifies a slice of the linear objective coefficients.
- *Task.putobjname* – Assigns a new name to the objective.
- *Task.putobjsense* – Sets the objective sense.
- *Task.putvarbound* – Changes the bounds for one variable.
- *Task.putvarboundslice* – Changes the bounds for a slice of the variables.
- *Task.putvarname* – Sets the name of a variable.
- *Task.removecons* – Removes a number of constraints.
- *Task.removevars* – Removes a number of variables.

- **dim** (**int**) – Dimension of the symmetric matrix that is appended. (input)
- **subi** (**int**[]) – Row subscript in the triplets. (input)
- **subj** (**int**[]) – Column subscripts in the triplets. (input)
- **valij** (**float**[]) – Values of each triplet. (input)

Return **idx** (**int**) – Unique index assigned to the inputted matrix that can be used for later reference.

Groups *Problem data - semidefinite*

Task.appendsparsesymmatlist

```
def appendsparsesymmatlist (dims, nz, subi, subj, valij, idx)
```

MOSEK maintains a storage of symmetric data matrices that is used to build \bar{C} and \bar{A} . The storage can be thought of as a vector of symmetric matrices denoted E . Hence, E_i is a symmetric matrix of certain dimension.

This function appends general sparse symmetric matrixes on triplet form to the vector E of symmetric matrices. 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, 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.

Parameters

- **dims** (**int**[]) – Dimensions of the symmetric matrixes. (input)
- **nz** (**int**[]) – Number of nonzeros for each matrix. (input)
- **subi** (**int**[]) – Row subscript in the triplets. (input)
- **subj** (**int**[]) – Column subscripts in the triplets. (input)
- **valij** (**float**[]) – Values of each triplet. (input)
- **idx** (**int**[]) – Unique index assigned to the inputted matrix that can be used for later reference. (output)

Groups *Problem data - semidefinite*

Task.appendvars

```
def appendvars (num)
```

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.

Parameters **num** (**int**) – Number of variables which should be appended. (input)

Groups *Problem data - linear part, Problem data - variables*

Task.asyncgetresult

```
def asyncgetresult (server, port, token) -> respavailable, resp, trm
```

Request a response from a remote job. If successful, solver response, termination code and solutions are retrieved.

Parameters

- **server** (**str**) – Name or IP address of the solver server. (input)
- **port** (**str**) – Network port of the solver service. (input)
- **token** (**str**) – The task token. (input)

Return

- `respavailable` (int) – Indicates if a remote response is available. If this is not true, `resp` and `trm` should be ignored.
- `resp` (*mosek.rescode*) – Is the response code from the remote solver.
- `trm` (*mosek.rescode*) – Is either *rescode.ok* or a termination response code.

Groups *Remote optimization*

Task.asyncoptimize

```
def asyncoptimize (server, port) -> token
```

Offload the optimization task to a solver server defined by `server:port`. The call will return immediately and not wait for the result.

If the string parameter *sparam.remote_access_token* is not blank, it will be passed to the server as authentication.

Parameters

- `server` (str) – Name or IP address of the solver server (input)
- `port` (str) – Network port of the solver service (input)

Return `token` (str) – Returns the task token

Groups *Remote optimization*

Task.asyncpoll

```
def asyncpoll (server, port, token) -> respavailable, resp, trm
```

Requests information about the status of the remote job.

Parameters

- `server` (str) – Name or IP address of the solver server (input)
- `port` (str) – Network port of the solver service (input)
- `token` (str) – The task token (input)

Return

- `respavailable` (int) – Indicates if a remote response is available. If this is not true, `resp` and `trm` should be ignored.
- `resp` (*mosek.rescode*) – Is the response code from the remote solver.
- `trm` (*mosek.rescode*) – Is either *rescode.ok* or a termination response code.

Groups *Remote optimization*

Task.asyncstop

```
def asyncstop (server, port, token)
```

Request that the job identified by the `token` is terminated.

Parameters

- `server` (str) – Name or IP address of the solver server (input)
- `port` (str) – Network port of the solver service (input)
- `token` (str) – The task token (input)

Groups *Remote optimization*

Task.basiscond

Task.getarowslice

```
def getarowslice (first, last, ptrb, ptre, sub, val)
```

Obtains a sequence of rows from A in sparse format.

Parameters

- **first** (int) – Index of the first row in the sequence. (input)
- **last** (int) – Index of the last row in the sequence **plus one**. (input)
- **ptrb** (int[]) – **ptrb[t]** is an index pointing to the first element in the t -th row obtained. (output)
- **ptre** (int[]) – **ptre[t]** is an index pointing to the last element plus one in the t -th row obtained. (output)
- **sub** (int[]) – Contains the column subscripts. (output)
- **val** (float[]) – Contains the coefficient values. (output)

Groups *Problem data - linear part, Inspecting the task*

Task.getarowslicenumnz

```
def getarowslicenumnz (first, last) -> numnz
```

Obtains the number of non-zeros in a slice of rows of A .

Parameters

- **first** (int) – Index of the first row in the sequence. (input)
- **last** (int) – Index of the last row **plus one** in the sequence. (input)

Return **numnz** (int) – Number of non-zeros in the slice.

Groups *Problem data - linear part, Inspecting the task*

Task.getarowslicetrip

```
def getarowslicetrip (first, last, subi, subj, val)
```

Obtains a sequence of rows from A in sparse triplet format. The function returns the content of all rows whose index i satisfies **first** $\leq i <$ **last**. The triplets corresponding to nonzero entries are stored in the arrays **subi**, **subj** and **val**.

Parameters

- **first** (int) – Index of the first row in the sequence. (input)
- **last** (int) – Index of the last row in the sequence **plus one**. (input)
- **subi** (int[]) – Constraint subscripts. (output)
- **subj** (int[]) – Column subscripts. (output)
- **val** (float[]) – Values. (output)

Groups *Problem data - linear part, Inspecting the task*

Task.getatruncatetol

```
def getatruncatetol (tolzero)
```

Obtains the tolerance value set with *Task.putatruncatetol*.

Parameters **tolzero** (float[]) – All elements $|a_{i,j}|$ less than this tolerance is truncated to zero. (output)

Groups *Parameters, Problem data - linear part*

Parameters

- `subj (int [])` – A list of variable indexes. (input)
- `c (float [])` – Linear terms of the requested list of the objective as a dense vector. (output)

Groups *Inspecting the task, Problem data - linear part*

`Task.getconbound`

```
def getconbound (i) -> bk, bl, bu
```

Obtains bound information for one constraint.

Parameters `i (int)` – Index of the constraint for which the bound information should be obtained. (input)

Return

- `bk (mosek.boundkey)` – Bound keys.
- `bl (float)` – Values for lower bounds.
- `bu (float)` – Values for upper bounds.

Groups *Problem data - linear part, Inspecting the task, Problem data - bounds, Problem data - constraints*

`Task.getconboundslice`

```
def getconboundslice (first, last, bk, bl, bu)
```

Obtains bounds information for a slice of the constraints.

Parameters

- `first (int)` – First index in the sequence. (input)
- `last (int)` – Last index plus 1 in the sequence. (input)
- `bk (mosek.boundkey [])` – Bound keys. (output)
- `bl (float [])` – Values for lower bounds. (output)
- `bu (float [])` – Values for upper bounds. (output)

Groups *Problem data - linear part, Inspecting the task, Problem data - bounds, Problem data - constraints*

`Task.getcone`

```
def getcone (k, submem) -> ct, coneapar, nummem
```

Obtains a cone.

Parameters

- `k (int)` – Index of the cone. (input)
- `submem (int [])` – Variable subscripts of the members in the cone. (output)

Return

- `ct (mosek.conetype)` – Specifies the type of the cone.
- `coneapar (float)` – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0.
- `nummem (int)` – Number of member variables in the cone.

Groups *Inspecting the task, Problem data - cones*

`Task.getconeinfo`

Task.getconnameindex

```
def getconnameindex (somename) -> asgn, index
```

Checks whether the name `somename` has been assigned to any constraint. If so, the index of the constraint is reported.

Parameters `somename` (str) – The name which should be checked. (input)

Return

- `asgn` (int) – Is non-zero if the name `somename` is assigned to some constraint.
- `index` (int) – If the name `somename` is assigned to a constraint, then `index` is the index of the constraint.

Groups *Names, Problem data - linear part, Problem data - constraints, Inspecting the task*

Task.getconnamelen

```
def getconnamelen (i) -> len
```

Obtains the length of the name of a constraint.

Parameters `i` (int) – Index of the constraint. (input)

Return `len` (int) – Returns the length of the indicated name.

Groups *Names, Problem data - linear part, Problem data - constraints, Inspecting the task*

Task.getcslice

```
def getcslice (first, last, c)
```

Obtains a sequence of elements in `c`.

Parameters

- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `c` (float[]) – Linear terms of the requested slice of the objective as a dense vector. The length is `last-first`. (output)

Groups *Inspecting the task, Problem data - linear part*

Task.getdimbarvarj

```
def getdimbarvarj (j) -> dimbarvarj
```

Obtains the dimension of a symmetric matrix variable.

Parameters `j` (int) – Index of the semidefinite variable whose dimension is requested. (input)

Return `dimbarvarj` (int) – The dimension of the j -th semidefinite variable.

Groups *Inspecting the task, Problem data - semidefinite*

Task.getdouinf

```
def getdouinf (whichdinf) -> dvalue
```

Obtains a double information item from the task information database.

Return `maxnumcone (int)` – Number of preallocated conic constraints in the optimization task.

Groups *Inspecting the task, Problem data - cones*

`Task.getmaxnumqnz`

```
def getmaxnumqnz () -> maxnumqnz
```

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 .

Return `maxnumqnz (int)` – Number of non-zero elements preallocated in quadratic coefficient matrices.

Groups *Inspecting the task, Problem data - quadratic part*

`Task.getmaxnumvar`

```
def getmaxnumvar () -> maxnumvar
```

Obtains the number of preallocated variables in the optimization task. When this number of variables is reached **MOSEK** will automatically allocate more space for variables.

Return `maxnumvar (int)` – Number of preallocated variables in the optimization task.

Groups *Inspecting the task, Problem data - linear part, Problem data - variables*

`Task.getmemusage`

```
def getmemusage () -> meminuse, maxmemuse
```

Obtains information about the amount of memory used by a task.

Return

- `meminuse (int)` – Amount of memory currently used by the task.
- `maxmemuse (int)` – Maximum amount of memory used by the task until now.

Groups *System, memory and debugging*

`Task.getnumanz`

```
def getnumanz () -> numanz
```

Obtains the number of non-zeros in A .

Return `numanz (int)` – Number of non-zero elements in the linear constraint matrix.

Groups *Inspecting the task, Problem data - linear part*

`Task.getnumanz64`

```
def getnumanz64 () -> numanz
```

Obtains the number of non-zeros in A .

Return `numanz (int)` – Number of non-zero elements in the linear constraint matrix.

Groups *Inspecting the task, Problem data - linear part*

`Task.getnumbarablocktriplets`

```
def getnumbarblocktriplets () -> num
```

Obtains an upper bound on the number of elements in the block triplet form of \bar{A} .

Return `num (int)` – An upper bound on the number of elements in the block triplet form of \bar{A} .

Groups *Problem data - semidefinite, Inspecting the task*

`Task.getnumbaranz`

```
def getnumbaranz () -> nz
```

Get the number of nonzero elements in \bar{A} .

Return `nz (int)` – The number of nonzero block elements in \bar{A} i.e. the number of \bar{A}_{ij} elements that are nonzero.

Groups *Problem data - semidefinite, Inspecting the task*

`Task.getnumbarcbblocktriplets`

```
def getnumbarcbblocktriplets () -> num
```

Obtains an upper bound on the number of elements in the block triplet form of \bar{C} .

Return `num (int)` – An upper bound on the number of elements in the block triplet form of \bar{C} .

Groups *Problem data - semidefinite, Inspecting the task*

`Task.getnumbarcnz`

```
def getnumbarcnz () -> nz
```

Obtains the number of nonzero elements in \bar{C} .

Return `nz (int)` – The number of nonzeros in \bar{C} i.e. the number of elements \bar{C}_j that are nonzero.

Groups *Problem data - semidefinite, Inspecting the task*

`Task.getnumbarvar`

```
def getnumbarvar () -> numbarvar
```

Obtains the number of semidefinite variables.

Return `numbarvar (int)` – Number of semidefinite variables in the problem.

Groups *Inspecting the task, Problem data - semidefinite*

`Task.getnumcon`

```
def getnumcon () -> numcon
```

Obtains the number of constraints.

Return `numcon (int)` – Number of constraints.

Groups *Problem data - linear part, Problem data - constraints, Inspecting the task*

`Task.getnumcone`

```
def getnumcone () -> numcone
```

Obtains the number of cones.

Return numcone (int) – Number of conic constraints.

Groups *Problem data - cones, Inspecting the task*

Task.getnumconemem

```
def getnumconemem (k) -> nummem
```

Obtains the number of members in a cone.

Parameters k (int) – Index of the cone. (input)

Return nummem (int) – Number of member variables in the cone.

Groups *Problem data - cones, Inspecting the task*

Task.getnumintvar

```
def getnumintvar () -> numintvar
```

Obtains the number of integer-constrained variables.

Return numintvar (int) – Number of integer variables.

Groups *Inspecting the task, Problem data - variables*

Task.getnumparam

```
def getnumparam (partype) -> numparam
```

Obtains the number of parameters of a given type.

Parameters partype (*mosek.parametertype*) – Parameter type. (input)

Return numparam (int) – The number of parameters of type partype.

Groups *Inspecting the task, Parameters*

Task.getnumqconknz

```
def getnumqconknz (k) -> numqcnz
```

Obtains the number of non-zero quadratic terms in a constraint.

Parameters k (int) – Index of the constraint for which the number quadratic terms should be obtained. (input)

Return numqcnz (int) – Number of quadratic terms.

Groups *Inspecting the task, Problem data - constraints, Problem data - quadratic part*

Task.getnumqobjnz

```
def getnumqobjnz () -> numqonz
```

Obtains the number of non-zero quadratic terms in the objective.

Return numqonz (int) – Number of non-zero elements in the quadratic objective terms.

Groups *Inspecting the task, Problem data - quadratic part*

Task.getnumsymmat

```
def getnumsymmat () -> num
```

Obtains the number of symmetric matrices stored in the vector E .

Return num (int) – The number of symmetric sparse matrices.

Groups *Problem data - semidefinite, Inspecting the task*

Task.getnumvar

```
def getnumvar () -> numvar
```

Obtains the number of variables.

Return numvar (int) – Number of variables.

Groups *Inspecting the task, Problem data - variables*

Task.getobjname

```
def getobjname () -> objname
```

Obtains the name assigned to the objective function.

Return objname (str) – Assigned the objective name.

Groups *Inspecting the task, Names*

Task.getobjnamelen

```
def getobjnamelen () -> len
```

Obtains the length of the name assigned to the objective function.

Return len (int) – Assigned the length of the objective name.

Groups *Inspecting the task, Names*

Task.getobjsense

```
def getobjsense () -> sense
```

Gets the objective sense of the task.

Return sense (*mosek.objsense*) – The returned objective sense.

Groups *Problem data - linear part*

Task.getprimalobj

```
def getprimalobj (whichsol) -> primalobj
```

Computes the primal objective value for the desired solution. Note that if the solution is an infeasibility certificate, then the fixed term in the objective is not included.

Parameters whichsol (*mosek.soltype*) – Selects a solution. (input)

Return primalobj (float) – Objective value corresponding to the primal solution.

Groups *Solution information, Solution - primal*

Task.getprimalsolutionnorms

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `skx` (*mosek.stakey*[]) – Status keys for the variables. (output)

Groups *Solution information*

Task.getslc

```
def getslc (whichsol, slc)
```

Obtains the s_l^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `slc` (float[]) – Dual variables corresponding to the lower bounds on the constraints. (output)

Groups *Solution - dual*

Task.getslcslice

```
def getslcslice (whichsol, first, last, slc)
```

Obtains a slice of the s_l^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `slc` (float[]) – Dual variables corresponding to the lower bounds on the constraints. (output)

Groups *Solution - dual*

Task.getslx

```
def getslx (whichsol, slx)
```

Obtains the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `slx` (float[]) – Dual variables corresponding to the lower bounds on the variables. (output)

Groups *Solution - dual*

Task.getslxslice

```
def getslxslice (whichsol, first, last, slx)
```

Obtains a slice of the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `slx` (float[]) – Dual variables corresponding to the lower bounds on the variables. (output)

Groups *Solution - dual*

Task.getsnx

```
def getsnx (whichsol, snx)
```

Obtains the s_n^x vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **snx** (float[]) – Dual variables corresponding to the conic constraints on the variables. (output)

Groups *Solution - dual*

Task.getsnxslice

```
def getsnxslice (whichsol, first, last, snx)
```

Obtains a slice of the s_n^x vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **snx** (float[]) – Dual variables corresponding to the conic constraints on the variables. (output)

Groups *Solution - dual*

Task.getsolsta

```
def getsolsta (whichsol) -> solsta
```

Obtains the solution status.

Parameters **whichsol** (*mosek.soltype*) – Selects a solution. (input)

Return **solsta** (*mosek.solsta*) – Solution status.

Groups *Solution information*

Task.getsolution

```
def getsolution (whichsol, skc, skx, skn, xc, xx, y, slc, suc, slx, sux, snx) -> prosta, ↳ solsta
```

Obtains the complete solution.

Consider the case of linear programming. The primal problem is given by

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

and the corresponding dual problem is

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

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:

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

The mapping between variables and arguments to the function is as follows:

- **xx** : Corresponds to variable x (also denoted x^x).
- **xc** : Corresponds to $x^c := Ax$.
- **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 .
- **snx**: Corresponds to variable s_n^x .

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.

In order to retrieve the primal and dual values of semidefinite variables see [Task.getbarxj](#) and [Task.getbarsj](#).

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **skc** (*mosek.stakey* []) – Status keys for the constraints. (output)
- **skx** (*mosek.stakey* []) – Status keys for the variables. (output)
- **skn** (*mosek.stakey* []) – Status keys for the conic constraints. (output)
- **xc** (float []) – Primal constraint solution. (output)
- **xx** (float []) – Primal variable solution. (output)
- **y** (float []) – Vector of dual variables corresponding to the constraints. (output)
- **slc** (float []) – Dual variables corresponding to the lower bounds on the constraints. (output)
- **suc** (float []) – Dual variables corresponding to the upper bounds on the constraints. (output)
- **slx** (float []) – Dual variables corresponding to the lower bounds on the variables. (output)
- **sux** (float []) – Dual variables corresponding to the upper bounds on the variables. (output)
- **snx** (float []) – Dual variables corresponding to the conic constraints on the variables. (output)

Return

- **prosta** (*mosek.prosta*) – Problem status.

- `solsta` (*mosek.solsta*) – Solution status.

Groups *Solution information, Solution - primal, Solution - dual*

`Task.getsolutioninfo`

```
def getsolutioninfo (whichsol) -> pobj, pviolcon, pviolvar, pviolbarvar, pviolcone,
↳ pviolitg, dobj, dviolcon, dviolvar, dviolbarvar, dviolcone
```

Obtains information about a solution.

Parameters `whichsol` (*mosek.soltype*) – Selects a solution. (input)

Return

- `pobj` (float) – The primal objective value as computed by *Task.getprimalobj*.
- `pviolcon` (float) – Maximal primal violation of the solution associated with the x^c variables where the violations are computed by *Task.getpviolcon*.
- `pviolvar` (float) – Maximal primal violation of the solution for the x variables where the violations are computed by *Task.getpviolvar*.
- `pviolbarvar` (float) – Maximal primal violation of solution for the \bar{X} variables where the violations are computed by *Task.getpviolbarvar*.
- `pviolcone` (float) – Maximal primal violation of solution for the conic constraints where the violations are computed by *Task.getpviolcones*.
- `pviolitg` (float) – Maximal violation in the integer constraints. The violation for an integer 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 basic solutions.
- `dobj` (float) – Dual objective value as computed by *Task.getdualobj*.
- `dviolcon` (float) – Maximal violation of the dual solution associated with the x^c variable as computed by *Task.getdviolcon*.
- `dviolvar` (float) – Maximal violation of the dual solution associated with the x variable as computed by *Task.getdviolvar*.
- `dviolbarvar` (float) – Maximal violation of the dual solution associated with the \bar{S} variable as computed by *Task.getdviolbarvar*.
- `dviolcone` (float) – Maximal violation of the dual solution associated with the dual conic constraints as computed by *Task.getdviolcones*.

Groups *Solution information*

`Task.getsolutionslice`

```
def getsolutionslice (whichsol, solitem, first, last, values)
```

Obtains a slice of one item from the solution. The format of the solution is exactly as in *Task.getsolution*. The parameter `solitem` determines which of the solution vectors should be returned.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `solitem` (*mosek.solitem*) – Which part of the solution is required. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `values` (float[]) – The values in the required sequence are stored sequentially in `values`. (output)

Groups *Solution - primal, Solution - dual, Solution information*

`Task.getsparsesymmat`

```
def getsparsesymmat (idx, subi, subj, valij)
```

Get a single symmetric matrix from the matrix store.

Parameters

- `idx` (`int`) – Index of the matrix to retrieve. (input)
- `subi` (`int[]`) – Row subscripts of the matrix non-zero elements. (output)
- `subj` (`int[]`) – Column subscripts of the matrix non-zero elements. (output)
- `valij` (`float[]`) – Coefficients of the matrix non-zero elements. (output)

Groups *Problem data - semidefinite, Inspecting the task*

`Task.getstrparam`

```
def getstrparam (param) -> len, parvalue
```

Obtains the value of a string parameter.

Parameters `param` (*`mosek.sparam`*) – Which parameter. (input)

Return

- `len` (`int`) – The length of the parameter value.
- `parvalue` (`str`) – Parameter value.

Groups *Names, Parameters*

`Task.getstrparamlen`

```
def getstrparamlen (param) -> len
```

Obtains the length of a string parameter.

Parameters `param` (*`mosek.sparam`*) – Which parameter. (input)

Return `len` (`int`) – The length of the parameter value.

Groups *Names, Parameters*

`Task.getsuc`

```
def getsuc (whichsol, suc)
```

Obtains the s_u^c vector for a solution.

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (output)

Groups *Solution - dual*

`Task.getsucslice`

```
def getsucslice (whichsol, first, last, suc)
```

Obtains a slice of the s_u^c vector for a solution.

Parameters

- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `suc` (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (output)

Groups *Solution - dual*

Task.getsux

```
def getsux (whichsol, sux)
```

Obtains the s_u^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (output)

Groups *Solution - dual*

Task.getsuxslice

```
def getsuxslice (whichsol, first, last, sux)
```

Obtains a slice of the s_u^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `sux` (`float[]`) – Dual variables corresponding to the upper bounds on the variables. (output)

Groups *Solution - dual*

Task.getsymmatinfo

```
def getsymmatinfo (idx) -> dim, nz, type
```

MOSEK maintains a vector denoted by E of symmetric data matrices. This function makes it possible to obtain important information about a single matrix in E .

Parameters `idx` (`int`) – Index of the matrix for which information is requested. (input)

Return

- `dim` (`int`) – Returns the dimension of the requested matrix.
- `nz` (`int`) – Returns the number of non-zeros in the requested matrix.
- `type` (*mosek.symmattype*) – Returns the type of the requested matrix.

Groups *Problem data - semidefinite, Inspecting the task*

Task.gettaskname

```
def gettaskname () -> taskname
```

Obtains the name assigned to the task.

Return `taskname` (`str`) – Returns the task name.

Groups *Names, Inspecting the task*

Task.gettasknamelen

```
def gettasknamelen () -> len
```

Obtains the length the task name.

Return `len` (`int`) – Returns the length of the task name.

Groups *Names, Inspecting the task*

Task.getvarbound

```
def getvarbound (i) -> bk, bl, bu
```

Obtains bound information for one variable.

Parameters *i* (int) – Index of the variable for which the bound information should be obtained. (input)

Return

- *bk* (*mosek.boundkey*) – Bound keys.
- *bl* (float) – Values for lower bounds.
- *bu* (float) – Values for upper bounds.

Groups *Problem data - linear part, Inspecting the task, Problem data - bounds, Problem data - variables*

Task.getvarboundslice

```
def getvarboundslice (first, last, bk, bl, bu)
```

Obtains bounds information for a slice of the variables.

Parameters

- *first* (int) – First index in the sequence. (input)
- *last* (int) – Last index plus 1 in the sequence. (input)
- *bk* (*mosek.boundkey* []) – Bound keys. (output)
- *bl* (float []) – Values for lower bounds. (output)
- *bu* (float []) – Values for upper bounds. (output)

Groups *Problem data - linear part, Inspecting the task, Problem data - bounds, Problem data - variables*

Task.getvarname

```
def getvarname (j) -> name
```

Obtains the name of a variable.

Parameters *j* (int) – Index of a variable. (input)

Return *name* (str) – Returns the required name.

Groups *Names, Problem data - linear part, Problem data - variables, Inspecting the task*

Task.getvarnameindex

```
def getvarnameindex (somename) -> asgn, index
```

Checks whether the name *somename* has been assigned to any variable. If so, the index of the variable is reported.

Parameters *somename* (str) – The name which should be checked. (input)

Return

- *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 index of the variable.

Groups *Names, Problem data - linear part, Problem data - variables, Inspecting the task*

Task.getvarnamelen

```
def getvarnamelen (i) -> len
```

Obtains the length of the name of a variable.

Parameters *i* (int) – Index of a variable. (input)

Return *len* (int) – Returns the length of the indicated name.

Groups *Names, Problem data - linear part, Problem data - variables, Inspecting the task*

Task.getvartype

```
def getvartype (j) -> vartype
```

Gets the variable type of one variable.

Parameters *j* (int) – Index of the variable. (input)

Return *vartype* (*mosek.variabletype*) – Variable type of the *j*-th variable.

Groups *Inspecting the task, Problem data - variables*

Task.getvartypelist

```
def getvartypelist (subj, vartype)
```

Obtains the variable type of one or more variables. Upon return *vartype*[*k*] is the variable type of variable *subj*[*k*].

Parameters

- *subj* (int[]) – A list of variable indexes. (input)
- *vartype* (*mosek.variabletype*[]) – The variables types corresponding to the variables specified by *subj*. (output)

Groups *Inspecting the task, Problem data - variables*

Task.getxc

```
def getxc (whichsol, xc)
```

Obtains the x^c vector for a solution.

Parameters

- *whichsol* (*mosek.soltype*) – Selects a solution. (input)
- *xc* (float[]) – Primal constraint solution. (output)

Groups *Solution - primal*

Task.getxcslice

```
def getxcslice (whichsol, first, last, xc)
```

Obtains a slice of the x^c vector for a solution.

Parameters

- *whichsol* (*mosek.soltype*) – Selects a solution. (input)
- *first* (int) – First index in the sequence. (input)
- *last* (int) – Last index plus 1 in the sequence. (input)
- *xc* (float[]) – Primal constraint solution. (output)

Groups *Solution - primal*

Task.getxx

```
def getxx (whichsol, xx)
```

Obtains the x^x vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **xx** (float[]) – Primal variable solution. (output)

Groups *Solution - primal*

Task.getxxslice

```
def getxxslice (whichsol, first, last, xx)
```

Obtains a slice of the x^x vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **xx** (float[]) – Primal variable solution. (output)

Groups *Solution - primal*

Task.gety

```
def gety (whichsol, y)
```

Obtains the y vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **y** (float[]) – Vector of dual variables corresponding to the constraints. (output)

Groups *Solution - dual*

Task.getyslice

```
def getyslice (whichsol, first, last, y)
```

Obtains a slice of the y vector for a solution.

Parameters

- **whichsol** (*mosek.soltype*) – Selects a solution. (input)
- **first** (int) – First index in the sequence. (input)
- **last** (int) – Last index plus 1 in the sequence. (input)
- **y** (float[]) – Vector of dual variables corresponding to the constraints. (output)

Groups *Solution - dual*

Task.initbasissolve

```
def initbasissolve (basis)
```

Prepare a task for use with the *Task.solvewithbasis* function.

This function should be called


```
def isintparname (parname) -> param
```

Checks whether `parname` is a valid integer parameter name.

Parameters `parname (str)` – Parameter name. (input)

Return `param (mosek.iparam)` – Returns the parameter corresponding to the name, if one exists.

Groups *Parameters, Names*

`Task.isstrparname`

```
def isstrparname (parname) -> param
```

Checks whether `parname` is a valid string parameter name.

Parameters `parname (str)` – Parameter name. (input)

Return `param (mosek.sparam)` – Returns the parameter corresponding to the name, if one exists.

Groups *Parameters, Names*

`Task.linkfiletostream`

```
def linkfiletostream (whichstream, filename, append)
```

Directs all output from a task stream `whichstream` to a file `filename`.

Parameters

- `whichstream (mosek.streamtype)` – Index of the stream. (input)
- `filename (str)` – A valid file name. (input)
- `append (int)` – If this argument is 0 the output file will be overwritten, otherwise it will be appended to. (input)

Groups *Logging*

`Task.onesolutionsummary`

```
def onesolutionsummary (whichstream, whichsol)
```

Prints a short summary of a specified solution.

Parameters

- `whichstream (mosek.streamtype)` – Index of the stream. (input)
- `whichsol (mosek.soltype)` – Selects a solution. (input)

Groups *Logging, Solution information*

`Task.optimize`

```
def optimize () -> trmcode
```

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`.

Return `trmcode (mosek.rescode)` – Is either `rescode.ok` or a termination response code.

Groups *Optimization*

`Task.optimizermt`

```
def optimizermt (server, port) -> trmcode
```

Offload the optimization task to a solver server defined by `server:port`. The call will block until a result is available or the connection closes.

If the string parameter `sparam.remote_access_token` is not blank, it will be passed to the server as authentication.

Parameters

- `server` (`str`) – Name or IP address of the solver server. (input)
- `port` (`str`) – Network port of the solver server. (input)

Return `trmcode` (`mosek.rescode`) – Is either `rescode.ok` or a termination response code.

Groups *Remote optimization*

`Task.optimizersummary`

```
def optimizersummary (whichstream)
```

Prints a short summary with optimizer statistics from last optimization.

Parameters `whichstream` (`mosek.streamtype`) – Index of the stream. (input)

Groups *Logging*

`Task.primalrepair`

```
def primalrepair (wlc, wuc, wlx, wux)
```

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 weighted sum of relaxations to the bounds on the constraints and variables. Observe the function only repairs the problem but does not solve it. If an optimal solution is required the problem should be optimized after the repair.

The function is applicable to linear and conic problems possibly with integer variables.

Observe that when computing the minimal weighted relaxation the termination tolerance specified by the parameters of the task is employed. For instance the parameter `iparam.mio_mode` can be used to make **MOSEK** ignore the integer constraints during the repair which usually leads to a much faster repair. However, the drawback is of course that the repaired problem may not have an integer feasible solution.

Note the function modifies the task in place. If this is not desired, then apply the function to a cloned task.

Parameters

- `wlc` (`float[]`) – $(w_l^c)_i$ is the weight associated with relaxing the lower bound on constraint i . If the weight is negative, then the lower bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)
- `wuc` (`float[]`) – $(w_u^c)_i$ is the weight associated with relaxing the upper bound on constraint i . If the weight is negative, then the upper bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)
- `wlx` (`float[]`) – $(w_l^x)_j$ is the weight associated with relaxing the lower bound on variable j . If the weight is negative, then the lower bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)

Parameters

- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `bkc` (`mosek.boundkey`) – New bound key for all constraints in the slice. (input)
- `blc` (`float`) – New lower bound for all constraints in the slice. (input)
- `buc` (`float`) – New upper bound for all constraints in the slice. (input)

Groups *Problem data - linear part, Problem data - constraints, Problem data - bounds*

`Task.putcone`

```
def putcone (k, ct, coneapar, submem)
```

Replaces a conic constraint.

Parameters

- `k` (`int`) – Index of the cone. (input)
- `ct` (`mosek.conetype`) – Specifies the type of the cone. (input)
- `coneapar` (`float`) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (input)
- `submem` (`int[]`) – Variable subscripts of the members in the cone. (input)

Groups *Problem data - cones*

`Task.putconename`

```
def putconename (j, name)
```

Sets the name of a cone.

Parameters

- `j` (`int`) – Index of the cone. (input)
- `name` (`str`) – The name of the cone. (input)

Groups *Names, Problem data - cones*

`Task.putconname`

```
def putconname (i, name)
```

Sets the name of a constraint.

Parameters

- `i` (`int`) – Index of the constraint. (input)
- `name` (`str`) – The name of the constraint. (input)

Groups *Names, Problem data - constraints, Problem data - linear part*

`Task.putconsolutioni`

```
def putconsolutioni (i, whichsol, sk, x, sl, su)
```

Sets the primal and dual solution information for a single constraint.

Parameters

- `i` (`int`) – Index of the constraint. (input)
- `whichsol` (`mosek.soltype`) – Selects a solution. (input)
- `sk` (`mosek.stakey`) – Status key of the constraint. (input)
- `x` (`float`) – Primal solution value of the constraint. (input)

It is not mandatory to call this function, since **MOSEK** will reallocate internal structures whenever it is necessary.

The function call has no effect if both `maxnumcon` and `maxnumvar` are zero.

Parameters `maxnumanz (int)` – Number of preallocated non-zeros in A . (input)

Groups *Environment and task management, Problem data - semidefinite*

`Task.putmaxnumbarvar`

```
def putmaxnumbarvar (maxnumbarvar)
```

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. It only gives a hint about the amount of data to preallocate for efficiency reasons.

Please note that `maxnumbarvar` must be larger than the current number of symmetric matrix variables in the task.

Parameters `maxnumbarvar (int)` – Number of preallocated symmetric matrix variables. (input)

Groups *Environment and task management, Problem data - semidefinite*

`Task.putmaxnumcon`

```
def putmaxnumcon (maxnumcon)
```

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.

Parameters `maxnumcon (int)` – Number of preallocated constraints in the optimization task. (input)

Groups *Environment and task management, Problem data - constraints*

`Task.putmaxnumcone`

```
def putmaxnumcone (maxnumcone)
```

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 not 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 conic constraints in the task.

Parameters `maxnumcone (int)` – Number of preallocated conic constraints in the optimization task. (input)

Groups *Environment and task management, Problem data - cones*

`Task.putmaxnumqnz`

```
def putmaxnumqnz (maxnumqnz)
```

Sets the number of preallocated non-zero entries in quadratic terms.

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 not mandatory to call this function, since **MOSEK** will reallocate internal structures whenever it is necessary.

Parameters `maxnumqnz (int)` – Number of non-zero elements preallocated in quadratic coefficient matrices. (input)

Groups *Environment and task management, Problem data - quadratic part*

`Task.putmaxnumvar`

```
def putmaxnumvar (maxnumvar)
```

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 not mandatory to call this function. It only gives a hint about 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.

Parameters `maxnumvar (int)` – Number of preallocated variables in the optimization task. (input)

Groups *Environment and task management, Problem data - variables*

`Task.putnadoupam`

```
def putnadoupam (paramname, parvalue)
```

Sets the value of a named double parameter.

Parameters

- `paramname (str)` – Name of a parameter. (input)
- `parvalue (float)` – Parameter value. (input)

Groups *Parameters*

`Task.putnaintparam`

```
def putnaintparam (paramname, parvalue)
```

Sets the value of a named integer parameter.

Parameters

- `paramname (str)` – Name of a parameter. (input)
- `parvalue (int)` – Parameter value. (input)

Groups *Parameters*

`Task.putnastrparam`

```
def putnastrparam (paramname, parvalue)
```

Sets the value of a named string parameter.

Parameters

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `skx` (*mosek.stakey*[]) – Status keys for the variables. (input)

Groups *Solution information*

`Task.putslc`

```
def putslc (whichsol, slc)
```

Sets the s_l^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `slc` (float[]) – Dual variables corresponding to the lower bounds on the constraints. (input)

Groups *Solution - dual*

`Task.putslcslice`

```
def putslcslice (whichsol, first, last, slc)
```

Sets a slice of the s_l^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `slc` (float[]) – Dual variables corresponding to the lower bounds on the constraints. (input)

Groups *Solution - dual*

`Task.putslx`

```
def putslx (whichsol, slx)
```

Sets the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `slx` (float[]) – Dual variables corresponding to the lower bounds on the variables. (input)

Groups *Solution - dual*

`Task.putslxslice`

```
def putslxslice (whichsol, first, last, slx)
```

Sets a slice of the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)

- `slx (float [])` – Dual variables corresponding to the lower bounds on the variables. (input)

Groups *Solution - dual*

`Task.putsnx`

```
def putsnx (whichsol, sux)
```

Sets the s_n^x vector for a solution.

Parameters

- `whichsol (mosek.soltype)` – Selects a solution. (input)
- `sux (float [])` – Dual variables corresponding to the upper bounds on the variables. (input)

Groups *Solution - dual*

`Task.putsnxslice`

```
def putsnxslice (whichsol, first, last, snx)
```

Sets a slice of the s_n^x vector for a solution.

Parameters

- `whichsol (mosek.soltype)` – Selects a solution. (input)
- `first (int)` – First index in the sequence. (input)
- `last (int)` – Last index plus 1 in the sequence. (input)
- `snx (float [])` – Dual variables corresponding to the conic constraints on the variables. (input)

Groups *Solution - dual*

`Task.putsolution`

```
def putsolution (whichsol, skc, skx, skn, xc, xx, y, slc, suc, slx, sux, snx)
```

Inserts a solution into the task.

Parameters

- `whichsol (mosek.soltype)` – Selects a solution. (input)
- `skc (mosek.stakey [])` – Status keys for the constraints. (input)
- `skx (mosek.stakey [])` – Status keys for the variables. (input)
- `skn (mosek.stakey [])` – Status keys for the conic constraints. (input)
- `xc (float [])` – Primal constraint solution. (input)
- `xx (float [])` – Primal variable solution. (input)
- `y (float [])` – Vector of dual variables corresponding to the constraints. (input)
- `slc (float [])` – Dual variables corresponding to the lower bounds on the constraints. (input)
- `suc (float [])` – Dual variables corresponding to the upper bounds on the constraints. (input)
- `slx (float [])` – Dual variables corresponding to the lower bounds on the variables. (input)
- `sux (float [])` – Dual variables corresponding to the upper bounds on the variables. (input)
- `snx (float [])` – Dual variables corresponding to the conic constraints on the variables. (input)

Groups *Solution information, Solution - primal, Solution - dual*

Task.putsolutionyi

```
def putsolutionyi (i, whichsol, y)
```

Inputs the dual variable of a solution.

Parameters

- **i** (`int`) – Index of the dual variable. (input)
- **whichsol** (`mosek.soltype`) – Selects a solution. (input)
- **y** (`float`) – Solution value of the dual variable. (input)

Groups *Solution information, Solution - dual*

Task.putstrparam

```
def putstrparam (param, parvalue)
```

Sets the value of a string parameter.

Parameters

- **param** (`mosek.sparam`) – Which parameter. (input)
- **parvalue** (`str`) – Parameter value. (input)

Groups *Parameters*

Task.putsuc

```
def putsuc (whichsol, suc)
```

Sets the s_u^c vector for a solution.

Parameters

- **whichsol** (`mosek.soltype`) – Selects a solution. (input)
- **suc** (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (input)

Groups *Solution - dual*

Task.putsucslice

```
def putsucslice (whichsol, first, last, suc)
```

Sets a slice of the s_u^c vector for a solution.

Parameters

- **whichsol** (`mosek.soltype`) – Selects a solution. (input)
- **first** (`int`) – First index in the sequence. (input)
- **last** (`int`) – Last index plus 1 in the sequence. (input)
- **suc** (`float[]`) – Dual variables corresponding to the upper bounds on the constraints. (input)

Groups *Solution - dual*

Task.putsux

```
def putsux (whichsol, sux)
```

Sets the s_u^x vector for a solution.

Parameters

Task.putvarsolutionj

```
def putvarsolutionj (j, whichsol, sk, x, sl, su, sn)
```

Sets the primal and dual solution information for a single variable.

Parameters

- `j` (`int`) – Index of the variable. (input)
- `whichsol` (`mosek.soltype`) – Selects a solution. (input)
- `sk` (`mosek.stakey`) – Status key of the variable. (input)
- `x` (`float`) – Primal solution value of the variable. (input)
- `sl` (`float`) – Solution value of the dual variable associated with the lower bound. (input)
- `su` (`float`) – Solution value of the dual variable associated with the upper bound. (input)
- `sn` (`float`) – Solution value of the dual variable associated with the conic constraint. (input)

Groups *Solution information, Solution - primal, Solution - dual*

Task.putvartype

```
def putvartype (j, vartype)
```

Sets the variable type of one variable.

Parameters

- `j` (`int`) – Index of the variable. (input)
- `vartype` (`mosek.variabletype`) – The new variable type. (input)

Groups *Problem data - variables*

Task.putvartypelist

```
def putvartypelist (subj, vartype)
```

Sets the variable type for one or more variables. If the same index is specified multiple times in `subj` only the last entry takes effect.

Parameters

- `subj` (`int[]`) – A list of variable indexes for which the variable type should be changed. (input)
- `vartype` (`mosek.variabletype[]`) – A list of variable types that should be assigned to the variables specified by `subj`. (input)

Groups *Problem data - variables*

Task.putxc

```
def putxc (whichsol, xc)
```

Sets the x^c vector for a solution.

Parameters

- `whichsol` (`mosek.soltype`) – Selects a solution. (input)
- `xc` (`float[]`) – Primal constraint solution. (output)

Groups *Solution - primal*

Task.putxcslice

```
def putxcslice (whichsol, first, last, xc)
```

Sets a slice of the x^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `xc` (float[]) – Primal constraint solution. (input)

Groups *Solution - primal*

Task.putxx

```
def putxx (whichsol, xx)
```

Sets the x^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `xx` (float[]) – Primal variable solution. (input)

Groups *Solution - primal*

Task.putxxslice

```
def putxxslice (whichsol, first, last, xx)
```

Sets a slice of the x^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `xx` (float[]) – Primal variable solution. (input)

Groups *Solution - primal*

Task.puty

```
def puty (whichsol, y)
```

Sets the y vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `y` (float[]) – Vector of dual variables corresponding to the constraints. (input)

Groups *Solution - primal*

Task.putyslice

```
def putyslice (whichsol, first, last, y)
```

Sets a slice of the y vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)

- **first** (**int**) – First index in the sequence. (input)
- **last** (**int**) – Last index plus 1 in the sequence. (input)
- **y** (**float**[]) – Vector of dual variables corresponding to the constraints. (input)

Groups *Solution - dual*

Task.readdata

```
def readdata (filename)
```

Reads an optimization problem and associated data from a file.

Parameters **filename** (**str**) – A valid file name. (input)

Groups *Input/Output*

Task.readdataformat

```
def readdataformat (filename, format, compress)
```

Reads an optimization problem and associated data from a file.

Parameters

- **filename** (**str**) – A valid file name. (input)
- **format** (*mosek.dataformat*) – File data format. (input)
- **compress** (*mosek.compresstype*) – File compression type. (input)

Groups *Input/Output*

Task.readjsonstring

```
def readjsonstring (data)
```

Load task data from a JSON string, replacing any data that already exists in the task object. All problem data, parameters and other settings are resorted, but if the string contains solutions, the solution status after loading a file is set to unknown, even if it is optimal or otherwise well-defined.

Parameters **data** (**str**) – Problem data in text format. (input)

Groups *Input/Output*

Task.readlpstring

```
def readlpstring (data)
```

Load task data from a string in LP format, replacing any data that already exists in the task object.

Parameters **data** (**str**) – Problem data in text format. (input)

Groups *Input/Output*

Task.readopfstring

```
def readopfstring (data)
```

Load task data from a string in OPF format, replacing any data that already exists in the task object.

Parameters **data** (**str**) – Problem data in text format. (input)

Groups *Input/Output*

Task.readparamfile

```
def readparamfile (filename)
```

Reads **MOSEK** parameters from a file. Data is read from the file `filename` if it is a nonempty string. Otherwise data is read from the file specified by `sparam.param_read_file_name`.

Parameters `filename` (`str`) – A valid file name. (input)

Groups *Input/Output*

Task.readptfstring

```
def readptfstring (data)
```

Load task data from a PTF string, replacing any data that already exists in the task object. All problem data, parameters and other settings are resorted, but if the string contains solutions, the solution status after loading a file is set to unknown, even if it is optimal or otherwise well-defined.

Parameters `data` (`str`) – Problem data in text format. (input)

Groups *Input/Output*

Task.readsolution

```
def readsolution (whichsol, filename)
```

Reads a solution file and inserts it as a specified solution in the task. Data is read from the file `filename` if it is a nonempty string. Otherwise data is read from one of the files specified by `sparam.bas_sol_file_name`, `sparam.itr_sol_file_name` or `sparam.int_sol_file_name` depending on which solution is chosen.

Parameters

- `whichsol` (`mosek.soltype`) – Selects a solution. (input)
- `filename` (`str`) – A valid file name. (input)

Groups *Input/Output*

Task.readsummary

```
def readsummary (whichstream)
```

Prints a short summary of last file that was read.

Parameters `whichstream` (`mosek.streamtype`) – Index of the stream. (input)

Groups *Input/Output, Inspecting the task*

Task.readtask

```
def readtask (filename)
```

Load task data from a file, replacing any data that already exists in the task object. All problem data, parameters and other settings are resorted, but if the file contains solutions, the solution status after loading a file is set to unknown, even if it was optimal or otherwise well-defined when the file was dumped.

See section *The Task Format* for a description of the Task format.

Parameters `filename` (`str`) – A valid file name. (input)

Groups *Input/Output*

Task.removebarvars

```
def removebarvars (subset)
```

The function removes a subset of the symmetric matrices from the optimization task. This implies that the remaining symmetric matrices are renumbered.

Parameters `subset (int [])` – Indexes of symmetric matrices which should be removed. (input)

Groups *Problem data - semidefinite*

Task.removecones

```
def removecones (subset)
```

Removes a number of conic constraints from the problem. This implies that the remaining conic constraints are renumbered. In general, it is much more efficient to remove a cone with a high index than a low index.

Parameters `subset (int [])` – Indexes of cones which should be removed. (input)

Groups *Problem data - cones*

Task.removecons

```
def removecons (subset)
```

The function removes a subset of the constraints from the optimization task. This implies that the remaining constraints are renumbered.

Parameters `subset (int [])` – Indexes of constraints which should be removed. (input)

Groups *Problem data - constraints, Problem data - linear part*

Task.removevars

```
def removevars (subset)
```

The function removes a subset of the variables from the optimization task. This implies that the remaining variables are renumbered.

Parameters `subset (int [])` – Indexes of variables which should be removed. (input)

Groups *Problem data - variables, Problem data - linear part*

Task.resizetask

```
def resizetask (maxnumcon, maxnumvar, maxnumcone, maxnumanz, maxnumqnz)
```

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 it only gives a hint about 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.

Parameters

- `maxnumcon (int)` – New maximum number of constraints. (input)
- `maxnumvar (int)` – New maximum number of variables. (input)
- `maxnumcone (int)` – New maximum number of cones. (input)
- `maxnumanz (int)` – New maximum number of non-zeros in A . (input)

- `maxnumqnz (int)` – New maximum number of non-zeros in all Q matrices. (input)

Groups *Environment and task management*

`Task.sensitivityreport`

```
def sensitivityreport (whichstream)
```

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.

Parameters `whichstream (mosek.streamtype)` – Index of the stream. (input)

Groups *Sensitivity analysis*

`Task.set_InfoCallback`

```
def set_InfoCallback (callback)
```

Receive callbacks with solver status and information during optimization.

For example:

```
task.set_InfoCallback(lambda code,dinf,iinf,liinf: print("Called from: {0}".format(code)))
```

Parameters `callback (callbackfunc)` – The callback function. (input)

`Task.set_Progress`

```
def set_Progress (callback)
```

Receive callbacks about current status of the solver during optimization.

For example:

```
task.set_Progress(lambda code: print("Called from: {0}".format(code)))
```

Parameters `callback (progresscallbackfunc)` – The callback function. (input)

`Task.set_Stream`

```
def set_Stream (whichstream, callback)
```

Directs all output from a task stream to a callback function.

Parameters

- `whichstream (streamtype)` – Index of the stream. (input)
- `callback (streamfunc)` – The callback function. (input)

`Task.setdefault`

```
def setdefaults ()
```

Resets all the parameters to their default values.

Groups *Parameters*

`Task.solutiondef`

Return `numnz` (`int`) – As input it is the number of non-zeros in b . As output it is the number of non-zeros in \overline{X} .

Groups *Solving systems with basis matrix*

`Task.strtoconetype`

```
def strtoconetype (str) -> conetype
```

Obtains cone type code corresponding to a cone type string.

Parameters `str` (`str`) – String corresponding to the cone type code `conetype`. (input)

Return `conetype` (`mosek.conetype`) – The cone type corresponding to the string `str`.

Groups *Names*

`Task.strtosk`

```
def strtosk (str) -> sk
```

Obtains the status key corresponding to an abbreviation string.

Parameters `str` (`str`) – A status key abbreviation string. (input)

Return `sk` (`mosek.stakey`) – Status key corresponding to the string.

Groups *Names*

`Task.toconic`

```
def toconic ()
```

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, if any, into a constraint. 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.

Groups *Problem data - quadratic part*

`Task.updatesolutioninfo`

```
def updatesolutioninfo (whichsol)
```

Update the information items related to the solution.

Parameters `whichsol` (`mosek.soltype`) – Selects a solution. (input)

Groups *Information items and statistics*

`Task.writedata`

```
def writedata (filename)
```

Writes problem data associated with the optimization task to a file in one of the supported formats. See Section *Supported File Formats* for the complete list.

The data file format is determined by the file name extension. To write in 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 automatically generated anonymous names by setting the `iparam.write_generic_names` parameter to `onoffkey.on`.

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`.

Parameters `filename` (str) – A valid file name. (input)

Groups *Input/Output*

`Task.writejsonsol`

```
def writejsonsol (filename)
```

Saves the current solutions and solver information items in a JSON file.

Parameters `filename` (str) – A valid file name. (input)

Groups *Input/Output*

`Task.writeparamfile`

```
def writeparamfile (filename)
```

Writes all the parameters to a parameter file.

Parameters `filename` (str) – A valid file name. (input)

Groups *Input/Output, Parameters*

`Task.writesolution`

```
def writesolution (whichsol, filename)
```

Saves the current basic, interior-point, or integer solution to a file.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `filename` (str) – A valid file name. (input)

Groups *Input/Output*

`Task.writetask`

```
def writetask (filename)
```

Write a binary dump of the task data. This format saves all problem data, coefficients and parameter settings. See section *The Task Format* for a description of the Task format.

Parameters `filename` (str) – A valid file name. (input)

Groups *Input/Output*

15.5 Exceptions

MosekException

Base exception class for all **MOSEK** exceptions.

Error

Exception class used for all error response codes from **MOSEK**.

Implements *MosekException*

15.6 Parameters grouped by topic

Analysis

- *dparam.ana_sol_infeas_tol*
- *iparam.ana_sol_basis*
- *iparam.ana_sol_print_violated*
- *iparam.log_ana_pro*

Basis identification

- *dparam.sim_lu_tol_rel_piv*
- *iparam.bi_clean_optimizer*
- *iparam.bi_ignore_max_iter*
- *iparam.bi_ignore_num_error*
- *iparam.bi_max_iterations*
- *iparam.intpnt_basis*
- *iparam.log_bi*
- *iparam.log_bi_freq*

Conic interior-point method

- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*

Data check

- *dparam.data_sym_mat_tol*
- *dparam.data_sym_mat_tol_huge*
- *dparam.data_sym_mat_tol_large*
- *dparam.data_tol_aij_huge*
- *dparam.data_tol_aij_large*
- *dparam.data_tol_bound_inf*
- *dparam.data_tol_bound_wrn*
- *dparam.data_tol_c_huge*
- *dparam.data_tol_cj_large*
- *dparam.data_tol_qij*
- *dparam.data_tol_x*
- *dparam.semidefinite_tol_approx*
- *iparam.check_convexity*
- *iparam.log_check_convexity*

Data input/output

- *iparam.infeas_report_auto*
- *iparam.log_file*
- *iparam.opf_write_header*
- *iparam.opf_write_hints*
- *iparam.opf_write_line_length*
- *iparam.opf_write_parameters*
- *iparam.opf_write_problem*
- *iparam.opf_write_sol_bas*
- *iparam.opf_write_sol_itg*
- *iparam.opf_write_sol_itr*
- *iparam.opf_write_solutions*
- *iparam.param_read_case_name*
- *iparam.param_read_ign_error*
- *iparam.ptf_write_transform*
- *iparam.read_debug*
- *iparam.read_keep_free_con*
- *iparam.read_lp_drop_new_vars_in_bou*
- *iparam.read_lp_quoted_names*
- *iparam.read_mps_format*

- *iparam.read_mps_width*
- *iparam.read_task_ignore_param*
- *iparam.sol_read_name_width*
- *iparam.sol_read_width*
- *iparam.write_bas_constraints*
- *iparam.write_bas_head*
- *iparam.write_bas_variables*
- *iparam.write_compression*
- *iparam.write_data_param*
- *iparam.write_free_con*
- *iparam.write_generic_names*
- *iparam.write_generic_names_io*
- *iparam.write_ignore_incompatible_items*
- *iparam.write_int_constraints*
- *iparam.write_int_head*
- *iparam.write_int_variables*
- *iparam.write_lp_full_obj*
- *iparam.write_lp_line_width*
- *iparam.write_lp_quoted_names*
- *iparam.write_lp_strict_format*
- *iparam.write_lp_terms_per_line*
- *iparam.write_mps_format*
- *iparam.write_mps_int*
- *iparam.write_precision*
- *iparam.write_sol_barvariables*
- *iparam.write_sol_constraints*
- *iparam.write_sol_head*
- *iparam.write_sol_ignore_invalid_names*
- *iparam.write_sol_variables*
- *iparam.write_task_inc_sol*
- *iparam.write_xml_mode*
- *sparam.bas_sol_file_name*
- *sparam.data_file_name*
- *sparam.debug_file_name*
- *sparam.int_sol_file_name*
- *sparam.itr_sol_file_name*

- *sparam.mio_debug_string*
- *sparam.param_comment_sign*
- *sparam.param_read_file_name*
- *sparam.param_write_file_name*
- *sparam.read_mps_bou_name*
- *sparam.read_mps_obj_name*
- *sparam.read_mps_ran_name*
- *sparam.read_mps_rhs_name*
- *sparam.sensitivity_file_name*
- *sparam.sensitivity_res_file_name*
- *sparam.sol_filter_xc_low*
- *sparam.sol_filter_xc_upr*
- *sparam.sol_filter_xx_low*
- *sparam.sol_filter_xx_upr*
- *sparam.stat_file_name*
- *sparam.stat_key*
- *sparam.stat_name*
- *sparam.write_lp_gen_var_name*

Debugging

- *iparam.auto_sort_a_before_opt*

Dual simplex

- *iparam.sim_dual_crash*
- *iparam.sim_dual_restrict_selection*
- *iparam.sim_dual_selection*

Infeasibility report

- *iparam.infeas_generic_names*
- *iparam.infeas_report_level*
- *iparam.log_infeas_ana*

Interior-point method

- *dparam.check_convexity_rel_tol*
- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*
- *dparam.intpnt_qo_tol_dfeas*
- *dparam.intpnt_qo_tol_infeas*
- *dparam.intpnt_qo_tol_mu_red*
- *dparam.intpnt_qo_tol_near_rel*
- *dparam.intpnt_qo_tol_pfeas*
- *dparam.intpnt_qo_tol_rel_gap*
- *dparam.intpnt_tol_dfeas*
- *dparam.intpnt_tol_dsafe*
- *dparam.intpnt_tol_infeas*
- *dparam.intpnt_tol_mu_red*
- *dparam.intpnt_tol_path*
- *dparam.intpnt_tol_pfeas*
- *dparam.intpnt_tol_psafe*
- *dparam.intpnt_tol_rel_gap*
- *dparam.intpnt_tol_rel_step*
- *dparam.intpnt_tol_step_size*
- *dparam.qcgo_reformulate_rel_drop_tol*
- *iparam.bi_ignore_max_iter*
- *iparam.bi_ignore_num_error*
- *iparam.intpnt_basis*
- *iparam.intpnt_diff_step*
- *iparam.intpnt_hotstart*
- *iparam.intpnt_max_iterations*
- *iparam.intpnt_max_num_cor*
- *iparam.intpnt_max_num_refinement_steps*
- *iparam.intpnt_off_col_trh*
- *iparam.intpnt_order_gp_num_seeds*
- *iparam.intpnt_order_method*

- *iparam.intpnt_purify*
- *iparam.intpnt_regularization_use*
- *iparam.intpnt_scaling*
- *iparam.intpnt_solve_form*
- *iparam.intpnt_starting_point*
- *iparam.log_intpnt*

License manager

- *iparam.cache_license*
- *iparam.license_debug*
- *iparam.license_pause_time*
- *iparam.license_suppress_expire_wrns*
- *iparam.license_trh_expiry_wrn*
- *iparam.license_wait*

Logging

- *iparam.log*
- *iparam.log_ana_pro*
- *iparam.log_bi*
- *iparam.log_bi_freq*
- *iparam.log_cut_second_opt*
- *iparam.log_expand*
- *iparam.log_feas_repair*
- *iparam.log_file*
- *iparam.log_include_summary*
- *iparam.log_infeas_ana*
- *iparam.log_intpnt*
- *iparam.log_local_info*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.log_order*
- *iparam.log_presolve*
- *iparam.log_response*
- *iparam.log_sensitivity*
- *iparam.log_sensitivity_opt*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_storage*

Mixed-integer optimization

- *dparam.mio_max_time*
- *dparam.mio_rel_gap_const*
- *dparam.mio_tol_abs_gap*
- *dparam.mio_tol_abs_relax_int*
- *dparam.mio_tol_feas*
- *dparam.mio_tol_rel_dual_bound_improvement*
- *dparam.mio_tol_rel_gap*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.mio_branch_dir*
- *iparam.mio_conic_outer_approximation*
- *iparam.mio_cut_clique*
- *iparam.mio_cut_cmir*
- *iparam.mio_cut_gmi*
- *iparam.mio_cut_implied_bound*
- *iparam.mio_cut_knapsack_cover*
- *iparam.mio_cut_selection_level*
- *iparam.mio_feaspump_level*
- *iparam.mio_heuristic_level*
- *iparam.mio_max_num_branches*
- *iparam.mio_max_num_relaxs*
- *iparam.mio_max_num_root_cut_rounds*
- *iparam.mio_max_num_solutions*
- *iparam.mio_node_optimizer*
- *iparam.mio_node_selection*
- *iparam.mio_perspective_reformulate*
- *iparam.mio_probing_level*
- *iparam.mio_propagate_objective_constraint*
- *iparam.mio_rins_max_nodes*
- *iparam.mio_root_optimizer*
- *iparam.mio_root_repeat_presolve_level*
- *iparam.mio_seed*
- *iparam.mio_vb_detection_level*

Output information

- *iparam.infeas_report_level*
- *iparam.license_suppress_expire_wrns*
- *iparam.license_trh_expiry_wrn*
- *iparam.log*
- *iparam.log_bi*
- *iparam.log_bi_freq*
- *iparam.log_cut_second_opt*
- *iparam.log_expand*
- *iparam.log_feas_repair*
- *iparam.log_file*
- *iparam.log_include_summary*
- *iparam.log_infeas_ana*
- *iparam.log_intpnt*
- *iparam.log_local_info*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.log_order*
- *iparam.log_response*
- *iparam.log_sensitivity*
- *iparam.log_sensitivity_opt*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_sim_minor*
- *iparam.log_storage*
- *iparam.max_num_warnings*

Overall solver

- *iparam.bi_clean_optimizer*
- *iparam.infeas_prefer_primal*
- *iparam.license_wait*
- *iparam.mio_mode*
- *iparam.optimizer*
- *iparam.presolve_level*
- *iparam.presolve_max_num_reductions*
- *iparam.presolve_use*

- *iparam.primal_repair_optimizer*
- *iparam.sensitivity_all*
- *iparam.sensitivity_optimizer*
- *iparam.sensitivity_type*
- *iparam.solution_callback*

Overall system

- *iparam.auto_update_sol_info*
- *iparam.intpnt_multi_thread*
- *iparam.license_wait*
- *iparam.log_storage*
- *iparam.mt_spincount*
- *iparam.num_threads*
- *iparam.remove_unused_solutions*
- *iparam.timing_level*
- *sparam.remote_access_token*

Presolve

- *dparam.presolve_tol_abs_lindep*
- *dparam.presolve_tol_aij*
- *dparam.presolve_tol_rel_lindep*
- *dparam.presolve_tol_s*
- *dparam.presolve_tol_x*
- *iparam.presolve_eliminator_max_fill*
- *iparam.presolve_eliminator_max_num_tries*
- *iparam.presolve_level*
- *iparam.presolve_lindep_abs_work_trh*
- *iparam.presolve_lindep_rel_work_trh*
- *iparam.presolve_lindep_use*
- *iparam.presolve_max_num_pass*
- *iparam.presolve_max_num_reductions*
- *iparam.presolve_use*

Primal simplex

- *iparam.sim_primal_crash*
- *iparam.sim_primal_restrict_selection*
- *iparam.sim_primal_selection*

Progress callback

- *iparam.solution_callback*

Simplex optimizer

- *dparam.basis_rel_tol_s*
- *dparam.basis_tol_s*
- *dparam.basis_tol_x*
- *dparam.sim_lu_tol_rel_piv*
- *dparam.simplex_abs_tol_piv*
- *iparam.basis_solve_use_plus_one*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_sim_minor*
- *iparam.sensitivity_optimizer*
- *iparam.sim_basis_factor_use*
- *iparam.sim_degen*
- *iparam.sim_dual_phaseone_method*
- *iparam.sim_exploit_dupvec*
- *iparam.sim_hotstart*
- *iparam.sim_hotstart_lu*
- *iparam.sim_max_iterations*
- *iparam.sim_max_num_setbacks*
- *iparam.sim_non_singular*
- *iparam.sim_primal_phaseone_method*
- *iparam.sim_refactor_freq*
- *iparam.sim_reformulation*
- *iparam.sim_save_lu*
- *iparam.sim_scaling*
- *iparam.sim_scaling_method*
- *iparam.sim_seed*
- *iparam.sim_solve_form*
- *iparam.sim_stability_priority*
- *iparam.sim_switch_optimizer*

Solution input/output

- *iparam.infeas_report_auto*
- *iparam.sol_filter_keep_basic*
- *iparam.sol_filter_keep_ranged*
- *iparam.sol_read_name_width*
- *iparam.sol_read_width*
- *iparam.write_bas_constraints*
- *iparam.write_bas_head*
- *iparam.write_bas_variables*
- *iparam.write_int_constraints*
- *iparam.write_int_head*
- *iparam.write_int_variables*
- *iparam.write_sol_barvariables*
- *iparam.write_sol_constraints*
- *iparam.write_sol_head*
- *iparam.write_sol_ignore_invalid_names*
- *iparam.write_sol_variables*
- *sparam.bas_sol_file_name*
- *sparam.int_sol_file_name*
- *sparam.itr_sol_file_name*
- *sparam.sol_filter_xc_low*
- *sparam.sol_filter_xc_upr*
- *sparam.sol_filter_xx_low*
- *sparam.sol_filter_xx_upr*

Termination criteria

- *dparam.basis_rel_tol_s*
- *dparam.basis_tol_s*
- *dparam.basis_tol_x*
- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*
- *dparam.intpnt_qo_tol_dfeas*

- *dparam.intpnt_qo_tol_infeas*
- *dparam.intpnt_qo_tol_mu_red*
- *dparam.intpnt_qo_tol_near_rel*
- *dparam.intpnt_qo_tol_pfeas*
- *dparam.intpnt_qo_tol_rel_gap*
- *dparam.intpnt_tol_dfeas*
- *dparam.intpnt_tol_infeas*
- *dparam.intpnt_tol_mu_red*
- *dparam.intpnt_tol_pfeas*
- *dparam.intpnt_tol_rel_gap*
- *dparam.lower_obj_cut*
- *dparam.lower_obj_cut_finite_trh*
- *dparam.mio_max_time*
- *dparam.mio_rel_gap_const*
- *dparam.mio_tol_rel_gap*
- *dparam.optimizer_max_time*
- *dparam.upper_obj_cut*
- *dparam.upper_obj_cut_finite_trh*
- *iparam.bi_max_iterations*
- *iparam.intpnt_max_iterations*
- *iparam.mio_max_num_branches*
- *iparam.mio_max_num_root_cut_rounds*
- *iparam.mio_max_num_solutions*
- *iparam.sim_max_iterations*

Other

- *iparam.compress_statfile*

15.7 Parameters (alphabetical list sorted by type)

- *Double parameters*
- *Integer parameters*
- *String parameters*

Default 1.0e-8
Accepted [0.0; 1.0]
Example `task.putdouparam(dparam.intpnt_tol_dfeas, 1.0e-8)`
Generic name MSK_DPAR_INTPNT_TOL_DFEAS
Groups *Interior-point method, Termination criteria*

`dparam.intpnt_tol_dsafe`

Controls the initial dual starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Default 1.0
Accepted [1.0e-4; +inf]
Example `task.putdouparam(dparam.intpnt_tol_dsafe, 1.0)`
Generic name MSK_DPAR_INTPNT_TOL_DSAFE
Groups *Interior-point method*

`dparam.intpnt_tol_infeas`

Infeasibility tolerance used by the interior-point optimizer for linear problems. Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default 1.0e-10
Accepted [0.0; 1.0]
Example `task.putdouparam(dparam.intpnt_tol_infeas, 1.0e-10)`
Generic name MSK_DPAR_INTPNT_TOL_INFEAS
Groups *Interior-point method, Termination criteria*

`dparam.intpnt_tol_mu_red`

Relative complementarity gap tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-16
Accepted [0.0; 1.0]
Example `task.putdouparam(dparam.intpnt_tol_mu_red, 1.0e-16)`
Generic name MSK_DPAR_INTPNT_TOL_MU_RED
Groups *Interior-point method, Termination criteria*

`dparam.intpnt_tol_path`

Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central path is followed very closely. On numerically unstable problems it may be worthwhile to increase this parameter.

Default 1.0e-8
Accepted [0.0; 0.9999]
Example `task.putdouparam(dparam.intpnt_tol_path, 1.0e-8)`
Generic name MSK_DPAR_INTPNT_TOL_PATH
Groups *Interior-point method*

`dparam.intpnt_tol_pfeas`

Primal feasibility tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-8
Accepted [0.0; 1.0]
Example `task.putdouparam(dparam.intpnt_tol_pfeas, 1.0e-8)`
Generic name MSK_DPAR_INTPNT_TOL_PFEAS
Groups *Interior-point method, Termination criteria*

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.license_suppress_expire_wrns, onoffkey.off)`
Generic name MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS
Groups *License manager, Output information*

`iparam.license_trh_expiry_wrn`

If a license feature expires in a numbers of days less than the value of this parameter then a warning will be issued.

Default 7
Accepted [0; +inf]
Example `task.putintparam(iparam.license_trh_expiry_wrn, 7)`
Generic name MSK_IPAR_LICENSE_TRH_EXPIRY_WRN
Groups *License manager, Output information*

`iparam.license_wait`

If all licenses are in use **MOSEK** returns with an error code. However, by turning on this parameter **MOSEK** will wait for an available license.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.license_wait, onoffkey.off)`
Generic name MSK_IPAR_LICENSE_WAIT
Groups *Overall solver, Overall system, License manager*

`iparam.log`

Controls the amount of log information. The value 0 implies that all log information is suppressed. A higher level implies that more information is logged.

Please note that if a task is employed to solve a sequence of optimization problems the value of this parameter is reduced by the value of *iparam.log_cut_second_opt* for the second and any subsequent optimizations.

Default 10
Accepted [0; +inf]
Example `task.putintparam(iparam.log, 10)`
See also *iparam.log_cut_second_opt*
Generic name MSK_IPAR_LOG
Groups *Output information, Logging*

`iparam.log_ana_pro`

Controls amount of output from the problem analyzer.

Default 1
Accepted [0; +inf]
Example `task.putintparam(iparam.log_ana_pro, 1)`
Generic name MSK_IPAR_LOG_ANA_PRO
Groups *Analysis, Logging*

`iparam.log_bi`

Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.

Default 1
Accepted [0; +inf]
Example `task.putintparam(iparam.log_bi, 1)`
Generic name MSK_IPAR_LOG_BI

Default 1
Accepted [0; +inf]
Example `task.putintparam(iparam.log_file, 1)`
Generic name MSK_IPAR_LOG_FILE
Groups *Data input/output, Output information, Logging*

`iparam.log_include_summary`

If on, then the solution summary will be printed by *Task.optimize*, so a separate call to *Task.solutionsummary* is not necessary.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.log_include_summary, onoffkey.off)`
Generic name MSK_IPAR_LOG_INCLUDE_SUMMARY
Groups *Output information, Logging*

`iparam.log_infeas_ana`

Controls amount of output printed by the infeasibility analyzer procedures. A higher level implies that more information is logged.

Default 1
Accepted [0; +inf]
Example `task.putintparam(iparam.log_infeas_ana, 1)`
Generic name MSK_IPAR_LOG_INFEAS_ANA
Groups *Infeasibility report, Output information, Logging*

`iparam.log_intpnt`

Controls amount of output printed by the interior-point optimizer. A higher level implies that more information is logged.

Default 1
Accepted [0; +inf]
Example `task.putintparam(iparam.log_intpnt, 1)`
Generic name MSK_IPAR_LOG_INTPNT
Groups *Interior-point method, Output information, Logging*

`iparam.log_local_info`

Controls whether local identifying information like environment variables, filenames, IP addresses etc. are printed to the log.

Note that this will only affect some functions. Some functions that specifically emit system information will not be affected.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.log_local_info, onoffkey.on)`
Generic name MSK_IPAR_LOG_LOCAL_INFO
Groups *Output information, Logging*

`iparam.log_mio`

Controls the log level for the mixed-integer optimizer. A higher level implies that more information is logged.

Default 4
Accepted [0; +inf]
Example `task.putintparam(iparam.log_mio, 4)`
Generic name MSK_IPAR_LOG_MIO
Groups *Mixed-integer optimization, Output information, Logging*

`iparam.log_mio_freq`

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.

Default 10

Accepted [-inf; +inf]

Example `task.putintparam(iparam.log_mio_freq, 10)`

Generic name MSK_IPAR_LOG_MIO_FREQ

Groups *Mixed-integer optimization, Output information, Logging*

`iparam.log_order`

If turned on, then factor lines are added to the log.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_order, 1)`

Generic name MSK_IPAR_LOG_ORDER

Groups *Output information, Logging*

`iparam.log_presolve`

Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_presolve, 1)`

Generic name MSK_IPAR_LOG_PRESOLVE

Groups *Logging*

`iparam.log_response`

Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.

Default 0

Accepted [0; +inf]

Example `task.putintparam(iparam.log_response, 0)`

Generic name MSK_IPAR_LOG_RESPONSE

Groups *Output information, Logging*

`iparam.log_sensitivity`

Controls the amount of logging during the sensitivity analysis.

- 0. Means no logging information is produced.
- 1. Timing information is printed.
- 2. Sensitivity results are printed.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_sensitivity, 1)`

Generic name MSK_IPAR_LOG_SENSITIVITY

Groups *Output information, Logging*

`iparam.log_sensitivity_opt`

Controls the amount of logging from the optimizers employed during the sensitivity analysis. 0 means no logging information is produced.

Default 0

Accepted [0; +inf]

`iparam.mio_cut_selection_level`

Controls how aggressively generated cuts are selected to be included in the relaxation.

- -1. The optimizer chooses the level of cut selection
- 0. Generated cuts less likely to be added to the relaxation
- 1. Cuts are more aggressively selected to be included in the relaxation

Default -1

Accepted [-1; +1]

Example `task.putintparam(iparam.mio_cut_selection_level, -1)`

Generic name MSK_IPAR_MIO_CUT_SELECTION_LEVEL

Groups *Mixed-integer optimization*

`iparam.mio_feaspump_level`

Controls the way the Feasibility Pump heuristic is employed by the mixed-integer optimizer.

- -1. The optimizer chooses how the Feasibility Pump is used
- 0. The Feasibility Pump is disabled
- 1. The Feasibility Pump is enabled with an effort to improve solution quality
- 2. The Feasibility Pump is enabled with an effort to reach feasibility early

Default -1

Accepted [-1; 2]

Example `task.putintparam(iparam.mio_feaspump_level, -1)`

Generic name MSK_IPAR_MIO_FEASPUMP_LEVEL

Groups *Mixed-integer optimization*

`iparam.mio_heuristic_level`

Controls the heuristic employed by the mixed-integer optimizer to locate an initial good integer feasible solution. A value of zero means the heuristic is not used at all. A larger value than 0 means that a gradually more sophisticated heuristic is used which is computationally more expensive. A negative value implies that the optimizer chooses the heuristic. Normally a value around 3 to 5 should be optimal.

Default -1

Accepted [-inf; +inf]

Example `task.putintparam(iparam.mio_heuristic_level, -1)`

Generic name MSK_IPAR_MIO_HEURISTIC_LEVEL

Groups *Mixed-integer optimization*

`iparam.mio_max_num_branches`

Maximum number of branches allowed during the branch and bound search. A negative value means infinite.

Default -1

Accepted [-inf; +inf]

Example `task.putintparam(iparam.mio_max_num_branches, -1)`

Generic name MSK_IPAR_MIO_MAX_NUM_BRANCHES

Groups *Mixed-integer optimization, Termination criteria*

`iparam.mio_max_num_relaxs`

Maximum number of relaxations allowed during the branch and bound search. A negative value means infinite.

Default -1

Accepted [-inf; +inf]

Example `task.putintparam(iparam.mio_max_num_relaxs, -1)`

Generic name MSK_IPAR_MIO_MAX_NUM_RELAXS

Generic name MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE

Groups *Mixed-integer optimization*

`iparam.mio_probing_level`

Controls the amount of probing employed by the mixed-integer optimizer in presolve.

- -1. The optimizer chooses the level of probing employed
- 0. Probing is disabled
- 1. A low amount of probing is employed
- 2. A medium amount of probing is employed
- 3. A high amount of probing is employed

Default -1

Accepted [-1; 3]

Example `task.putintparam(iparam.mio_probing_level, -1)`

Generic name MSK_IPAR_MIO_PROBING_LEVEL

Groups *Mixed-integer optimization*

`iparam.mio_propagate_objective_constraint`

Use objective domain propagation.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.mio_propagate_objective_constraint, onoffkey.off)`

Generic name MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT

Groups *Mixed-integer optimization*

`iparam.mio_rins_max_nodes`

Controls the maximum number of nodes allowed in each call to the RINS heuristic. The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Default -1

Accepted [-1; +inf]

Example `task.putintparam(iparam.mio_rins_max_nodes, -1)`

Generic name MSK_IPAR_MIO_RINS_MAX_NODES

Groups *Mixed-integer optimization*

`iparam.mio_root_optimizer`

Controls which optimizer is employed at the root node in the mixed-integer optimizer.

Default *free*

Accepted *free, intpnt, conic, primal_simplex, dual_simplex, free_simplex, mixed_int* (see *optimizertype*)

Example `task.putintparam(iparam.mio_root_optimizer, optimizertype.free)`

Generic name MSK_IPAR_MIO_ROOT_OPTIMIZER

Groups *Mixed-integer optimization*

`iparam.mio_root_repeat_presolve_level`

Controls whether presolve can be repeated at root node.

- -1. The optimizer chooses whether presolve is repeated
- 0. Never repeat presolve
- 1. Always repeat presolve

Default -1

Accepted [-1; 1]

Example `task.putintparam(iparam.mio_root_repeat_presolve_level, -1)`

Generic name MSK_IPAR_MIO_ROOT_REPEAT_PREOLVE_LEVEL

Groups *Mixed-integer optimization*

iparam.mio_seed

Sets the random seed used for randomization in the mixed integer optimizer. Selecting a different seed can change the path the optimizer takes to the optimal solution.

Default 42

Accepted [0; +inf]

Example task.putintparam(iparam.mio_seed, 42)

Generic name MSK_IPAR_MIO_SEED

Groups *Mixed-integer optimization*

iparam.mio_vb_detection_level

Controls how much effort is put into detecting variable bounds.

- -1. The optimizer chooses
- 0. No variable bounds are detected
- 1. Only detect variable bounds that are directly represented in the problem
- 2. Detect variable bounds in probing

Default -1

Accepted [-1; +2]

Example task.putintparam(iparam.mio_vb_detection_level, -1)

Generic name MSK_IPAR_MIO_VB_DETECTION_LEVEL

Groups *Mixed-integer optimization*

iparam.mt_spincount

Set the number of iterations to spin before sleeping.

Default 0

Accepted [0; 1000000000]

Example task.putintparam(iparam.mt_spincount, 0)

Generic name MSK_IPAR_MT_SPINCOUNT

Groups *Overall system*

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.

If using the conic optimizer, the value of this parameter set at first optimization remains constant through the lifetime of the process. **MOSEK** will allocate a thread pool of given size, and changing the parameter value later will have no effect. It will, however, remain possible to demand single-threaded execution by setting *iparam.intpnt_multi_thread*.

For the mixed-integer optimizer and interior-point linear optimizer there is no such restriction.

Default 0

Accepted [0; +inf]

Example task.putintparam(iparam.num_threads, 0)

Generic name MSK_IPAR_NUM_THREADS

Groups *Overall system*

iparam.opf_write_header

Write a text header with date and **MOSEK** version in an OPF file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example task.putintparam(iparam.opf_write_header, onoffkey.on)

Default 50
Accepted [0; 100]
Example task.putintparam(iparam.sim_primal_restrict_selection, 50)
Generic name MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION
Groups *Primal simplex*

iparam.sim_primal_selection

Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.

Default *free*
Accepted *free, full, ase, devex, se, partial* (see *simseltype*)
Example task.putintparam(iparam.sim_primal_selection, simseltype.free)
Generic name MSK_IPAR_SIM_PRIMAL_SELECTION
Groups *Primal simplex*

iparam.sim_refactor_freq

Controls how frequent the basis is refactorized. The value 0 means that the optimizer determines the best point of refactorization. It is strongly recommended NOT to change this parameter.

Default 0
Accepted [0; +inf]
Example task.putintparam(iparam.sim_refactor_freq, 0)
Generic name MSK_IPAR_SIM_REFACTOR_FREQ
Groups *Simplex optimizer*

iparam.sim_reformulation

Controls if the simplex optimizers are allowed to reformulate the problem.

Default *off*
Accepted *on, off, free, aggressive* (see *simreform*)
Example task.putintparam(iparam.sim_reformulation, simreform.off)
Generic name MSK_IPAR_SIM_REFORMULATION
Groups *Simplex optimizer*

iparam.sim_save_lu

Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example task.putintparam(iparam.sim_save_lu, onoffkey.off)
Generic name MSK_IPAR_SIM_SAVE_LU
Groups *Simplex optimizer*

iparam.sim_scaling

Controls how much effort is used in scaling the problem before a simplex optimizer is used.

Default *free*
Accepted *free, none, moderate, aggressive* (see *scalingtype*)
Example task.putintparam(iparam.sim_scaling, scalingtype.free)
Generic name MSK_IPAR_SIM_SCALING
Groups *Simplex optimizer*

iparam.sim_scaling_method

Controls how the problem is scaled before a simplex optimizer is used.

Default *pow2*
Accepted *pow2, free* (see *scalingmethod*)

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_sol_barvariables, onoffkey.on)`
Generic name MSK_IPAR_WRITE_SOL_BARVARIABLES
Groups *Data input/output, Solution input/output*

`iparam.write_sol_constraints`

Controls whether the constraint section is written to the solution file.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_sol_constraints, onoffkey.on)`
Generic name MSK_IPAR_WRITE_SOL_CONSTRAINTS
Groups *Data input/output, Solution input/output*

`iparam.write_sol_head`

Controls whether the header section is written to the solution file.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_sol_head, onoffkey.on)`
Generic name MSK_IPAR_WRITE_SOL_HEAD
Groups *Data input/output, Solution input/output*

`iparam.write_sol_ignore_invalid_names`

Even if the names are invalid MPS names, then they are employed when writing the solution file.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_sol_ignore_invalid_names, onoffkey.off)`
Generic name MSK_IPAR_WRITE_SOL_IGNORE_INVALID_NAMES
Groups *Data input/output, Solution input/output*

`iparam.write_sol_variables`

Controls whether the variables section is written to the solution file.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_sol_variables, onoffkey.on)`
Generic name MSK_IPAR_WRITE_SOL_VARIABLES
Groups *Data input/output, Solution input/output*

`iparam.write_task_inc_sol`

Controls whether the solutions are stored in the task file too.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_task_inc_sol, onoffkey.on)`
Generic name MSK_IPAR_WRITE_TASK_INC_SOL
Groups *Data input/output*

`iparam.write_xml_mode`

Controls if linear coefficients should be written by row or column when writing in the XML file format.

Default *row*
Accepted *row, col* (see *xmlwriteroutputtype*)
Example `task.putintparam(iparam.write_xml_mode, xmlwriteroutputtype.row)`
Generic name MSK_IPAR_WRITE_XML_MODE
Groups *Data input/output*

rescode.wrn_lp_old_quad_format (80)
 Missing $\sqrt{2}$ after quadratic expressions in bound or objective.

rescode.wrn_lp_drop_variable (85)
 Ignored a variable because the variable was not previously defined. Usually this implies that a variable appears in the bound section but not in the objective or the constraints.

rescode.wrn_nz_in_upr_tri (200)
 Non-zero elements specified in the upper triangle of a matrix were ignored.

rescode.wrn_dropped_nz_qobj (201)
 One or more non-zero elements were dropped in the Q matrix in the objective.

rescode.wrn_ignore_integer (250)
 Ignored integer constraints.

rescode.wrn_no_global_optimizer (251)
 No global optimizer is available.

rescode.wrn_mio_infeasible_final (270)
 The final mixed-integer problem with all the integer variables fixed at their optimal values is infeasible.

rescode.wrn_sol_filter (300)
 Invalid solution filter is specified.

rescode.wrn_undef_sol_file_name (350)
 Undefined name occurred in a solution.

rescode.wrn_sol_file_ignored_con (351)
 One or more lines in the constraint section were ignored when reading a solution file.

rescode.wrn_sol_file_ignored_var (352)
 One or more lines in the variable section were ignored when reading a solution file.

rescode.wrn_too_few_basis_vars (400)
 An incomplete basis has been specified. Too few basis variables are specified.

rescode.wrn_too_many_basis_vars (405)
 A basis with too many variables has been specified.

rescode.wrn_license_expire (500)
 The license expires.

rescode.wrn_license_server (501)
 The license server is not responding.

rescode.wrn_empty_name (502)
 A variable or constraint name is empty. The output file may be invalid.

rescode.wrn_using_generic_names (503)
 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_license_feature_expire (505)
 The license expires.

rescode.wrn_param_name_dou (510)
 The parameter name is not recognized as a double parameter.

rescode.wrn_param_name_int (511)
 The parameter name is not recognized as an integer parameter.

rescode.wrn_param_name_str (512)
 The parameter name is not recognized as a string parameter.

rescode.wrn_param_str_value (515)
 The string is not recognized as a symbolic value for the parameter.

rescode.wrn_param_ignored_cmio (516)
 A parameter was ignored by the conic mixed integer optimizer.

rescode.wrn_zeros_in_sparse_row (705)
 One or more (near) zero elements are specified in a sparse row of a matrix. Since, it is redundant to specify zero elements then it may indicate an error.

rescode.wrn_zeros_in_sparse_col (710)
 One or more (near) zero elements are specified in a sparse column of a matrix. It is redundant to specify zero elements. Hence, it may indicate an error.

rescode.wrn_incomplete_linear_dependency_check (800)
 The linear dependency check(s) is incomplete. Normally this is not an important warning unless

the optimization problem has been formulated with linear dependencies. Linear dependencies may prevent **MOSEK** from solving the problem.

rescode.wrn_eliminator_space (801)
The eliminator is skipped at least once due to lack of space.

rescode.wrn_presolve_outofspace (802)
The presolve is incomplete due to lack of space.

rescode.wrn_write_changed_names (803)
Some names were changed because they were invalid for the output file format.

rescode.wrn_write_discarded_cfix (804)
The fixed objective term could not be converted to a variable and was discarded in the output file.

rescode.wrn_duplicate_constraint_names (850)
Two constraint names are identical.

rescode.wrn_duplicate_variable_names (851)
Two variable names are identical.

rescode.wrn_duplicate_barvariable_names (852)
Two barvariable names are identical.

rescode.wrn_duplicate_cone_names (853)
Two cone names are identical.

rescode.wrn_ana_large_bounds (900)
This warning is issued by the problem analyzer, if one or more constraint or variable bounds are very large. One should consider omitting these bounds entirely by setting them to $+\text{inf}$ or $-\text{inf}$.

rescode.wrn_ana_c_zero (901)
This warning is issued by the problem analyzer, if the coefficients in the linear part of the objective are all zero.

rescode.wrn_ana_empty_cols (902)
This warning is issued by the problem analyzer, if columns, in which all coefficients are zero, are found.

rescode.wrn_ana_close_bounds (903)
This warning is issued by problem analyzer, if ranged constraints or variables with very close upper and lower bounds are detected. One should consider treating such constraints as equalities and such variables as constants.

rescode.wrn_ana_almost_int_bounds (904)
This warning is issued by the problem analyzer if a constraint is bound nearly integral.

rescode.wrn_quad_cones_with_root_fixed_at_zero (930)
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 problem, or to fix all the variables in the cone to 0.

rescode.wrn_rquad_cones_with_root_fixed_at_zero (931)
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 problem, or to fix all the variables in the cone to 0.

rescode.wrn_exp_cones_with_variables_fixed_at_zero (932)
For at least one exponential cone $x \geq y \exp(z/y)$ either the variable x or y 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 problem, or to fix all the variables in the cone to 0.

rescode.wrn_pow_cones_with_root_fixed_at_zero (933)
For at least one power 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 problem, or to fix all the variables in the cone to 0.

rescode.wrn_no_dualizer (950)
No automatic dualizer is available for the specified problem. The primal problem is solved.

rescode.wrn_sym_mat_large (960)
A numerically large value is specified for an $e_{i,j}$ element in E . The parameter `dparam.data_sym_mat_tol_large` controls when an $e_{i,j}$ is considered large.

15.8.3 Errors

`rescode.err_license (1000)`
Invalid license.

`rescode.err_license_expired (1001)`
The license has expired.

`rescode.err_license_version (1002)`
The license is valid for another version of **MOSEK**.

`rescode.err_size_license (1005)`
The problem is bigger than the license.

`rescode.err_prob_license (1006)`
The software is not licensed to solve the problem.

`rescode.err_file_license (1007)`
Invalid license file.

`rescode.err_missing_license_file (1008)`
MOSEK cannot find license file or a token server. See the **MOSEK** licensing manual for details.

`rescode.err_size_license_con (1010)`
The problem has too many constraints to be solved with the available license.

`rescode.err_size_license_var (1011)`
The problem has too many variables to be solved with the available license.

`rescode.err_size_license_intvar (1012)`
The problem contains too many integer variables to be solved with the available license.

`rescode.err_optimizer_license (1013)`
The optimizer required is not licensed.

`rescode.err_flexlm (1014)`
The FLEXlm license manager reported an error.

`rescode.err_license_server (1015)`
The license server is not responding.

`rescode.err_license_max (1016)`
Maximum number of licenses is reached.

`rescode.err_license_moseklm_daemon (1017)`
The MOSEKLM license manager daemon is not up and running.

`rescode.err_license_feature (1018)`
A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.

`rescode.err_platform_not_licensed (1019)`
A requested license feature is not available for the required platform.

`rescode.err_license_cannot_allocate (1020)`
The license system cannot allocate the memory required.

`rescode.err_license_cannot_connect (1021)`
MOSEK cannot connect to the license server. Most likely the license server is not up and running.

`rescode.err_license_invalid_hostid (1025)`
The host ID specified in the license file does not match the host ID of the computer.

`rescode.err_license_server_version (1026)`
The version specified in the checkout request is greater than the highest version number the daemon supports.

`rescode.err_license_no_server_support (1027)`
The license server does not support the requested feature. Possible reasons for this error include:

- The feature has expired.
- The feature's start date is later than today's date.
- The version requested is higher than feature's the highest supported version.
- A corrupted license file.

Try restarting the license and inspect the license server debug file, usually called `lmgrd.log`.

`rescode.err_license_no_server_line (1028)`
There is no **SERVER** line in the license file. All non-zero license count features need at least one **SERVER** line.

rescode.err_older_dll (1035)
The dynamic link library is older than the specified version.

rescode.err_newer_dll (1036)
The dynamic link library is newer than the specified version.

rescode.err_link_file_dll (1040)
A file cannot be linked to a stream in the DLL version.

rescode.err_thread_mutex_init (1045)
Could not initialize a mutex.

rescode.err_thread_mutex_lock (1046)
Could not lock a mutex.

rescode.err_thread_mutex_unlock (1047)
Could not unlock a mutex.

rescode.err_thread_create (1048)
Could not create a thread. This error may occur if a large number of environments are created and not deleted again. In any case it is a good practice to minimize the number of environments created.

rescode.err_thread_cond_init (1049)
Could not initialize a condition.

rescode.err_unknown (1050)
Unknown error.

rescode.err_space (1051)
Out of space.

rescode.err_file_open (1052)
Error while opening a file.

rescode.err_file_read (1053)
File read error.

rescode.err_file_write (1054)
File write error.

rescode.err_data_file_ext (1055)
The data file format cannot be determined from the file name.

rescode.err_invalid_file_name (1056)
An invalid file name has been specified.

rescode.err_invalid_sol_file_name (1057)
An invalid file name has been specified.

rescode.err_end_of_file (1059)
End of file reached.

rescode.err_null_env (1060)
env is a NULL pointer.

rescode.err_null_task (1061)
task is a NULL pointer.

rescode.err_invalid_stream (1062)
An invalid stream is referenced.

rescode.err_no_init_env (1063)
env is not initialized.

rescode.err_invalid_task (1064)
The task is invalid.

rescode.err_null_pointer (1065)
An argument to a function is unexpectedly a NULL pointer.

rescode.err_living_tasks (1066)
All tasks associated with an environment must be deleted before the environment is deleted. There are still some undeleted tasks.

rescode.err_blank_name (1070)
An all blank name has been specified.

rescode.err_dup_name (1071)
The same name was used multiple times for the same problem item type.

rescode.err_format_string (1072)
The name format string is invalid.

`rescode.err_invalid_obj_name (1075)`
 An invalid objective name is specified.

`rescode.err_invalid_con_name (1076)`
 An invalid constraint name is used.

`rescode.err_invalid_var_name (1077)`
 An invalid variable name is used.

`rescode.err_invalid_cone_name (1078)`
 An invalid cone name is used.

`rescode.err_invalid_barvar_name (1079)`
 An invalid symmetric matrix variable name is used.

`rescode.err_space_leaking (1080)`
MOSEK is leaking memory. This can be due to either an incorrect use of **MOSEK** or a bug.

`rescode.err_space_no_info (1081)`
 No available information about the space usage.

`rescode.err_read_format (1090)`
 The specified format cannot be read.

`rescode.err_mps_file (1100)`
 An error occurred while reading an MPS file.

`rescode.err_mps_inv_field (1101)`
 A field in the MPS file is invalid. Probably it is too wide.

`rescode.err_mps_inv_marker (1102)`
 An invalid marker has been specified in the MPS file.

`rescode.err_mps_null_con_name (1103)`
 An empty constraint name is used in an MPS file.

`rescode.err_mps_null_var_name (1104)`
 An empty variable name is used in an MPS file.

`rescode.err_mps_undef_con_name (1105)`
 An undefined constraint name occurred in an MPS file.

`rescode.err_mps_undef_var_name (1106)`
 An undefined variable name occurred in an MPS file.

`rescode.err_mps_inv_con_key (1107)`
 An invalid constraint key occurred in an MPS file.

`rescode.err_mps_inv_bound_key (1108)`
 An invalid bound key occurred in an MPS file.

`rescode.err_mps_inv_sec_name (1109)`
 An invalid section name occurred in an MPS file.

`rescode.err_mps_no_objective (1110)`
 No objective is defined in an MPS file.

`rescode.err_mps_splitting_var (1111)`
 All elements in a column of the A matrix must be specified consecutively. Hence, it is illegal to specify non-zero elements in A for variable 1, then for variable 2 and then variable 1 again.

`rescode.err_mps_mul_con_name (1112)`
 A constraint name was specified multiple times in the ROWS section.

`rescode.err_mps_mul_qsec (1113)`
 Multiple QSECTIONs are specified for a constraint in the MPS data file.

`rescode.err_mps_mul_qobj (1114)`
 The Q term in the objective is specified multiple times in the MPS data file.

`rescode.err_mps_inv_sec_order (1115)`
 The sections in the MPS data file are not in the correct order.

`rescode.err_mps_mul_csec (1116)`
 Multiple CSECTIONs are given the same name.

`rescode.err_mps_cone_type (1117)`
 Invalid cone type specified in a CSECTION.

`rescode.err_mps_cone_overlap (1118)`
 A variable is specified to be a member of several cones.

`rescode.err_mps_cone_repeat (1119)`
 A variable is repeated within the CSECTION.

`rescode.err_mps_non_symmetric_q (1120)`
 A non symmetric matrix has been speciefied.

`rescode.err_mps_duplicate_q_element (1121)`
 Duplicate elements is specified in a Q matrix.

`rescode.err_mps_invalid_objsense (1122)`
 An invalid objective sense is specified.

`rescode.err_mps_tab_in_field2 (1125)`
 A tab char occurred in field 2.

`rescode.err_mps_tab_in_field3 (1126)`
 A tab char occurred in field 3.

`rescode.err_mps_tab_in_field5 (1127)`
 A tab char occurred in field 5.

`rescode.err_mps_invalid_obj_name (1128)`
 An invalid objective name is specified.

`rescode.err_lp_incompatible (1150)`
 The problem cannot be written to an LP formatted file.

`rescode.err_lp_empty (1151)`
 The problem cannot be written to an LP formatted file.

`rescode.err_lp_dup_slack_name (1152)`
 The name of the slack variable added to a ranged constraint already exists.

`rescode.err_write_mps_invalid_name (1153)`
 An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.

`rescode.err_lp_invalid_var_name (1154)`
 A variable name is invalid when used in an LP formatted file.

`rescode.err_lp_free_constraint (1155)`
 Free constraints cannot be written in LP file format.

`rescode.err_write_opf_invalid_var_name (1156)`
 Empty variable names cannot be written to OPF files.

`rescode.err_lp_file_format (1157)`
 Syntax error in an LP file.

`rescode.err_write_lp_format (1158)`
 Problem cannot be written as an LP file.

`rescode.err_read_lp_missing_end_tag (1159)`
 Syntax error in LP file. Possibly missing End tag.

`rescode.err_lp_format (1160)`
 Syntax error in an LP file.

`rescode.err_write_lp_non_unique_name (1161)`
 An auto-generated name is not unique.

`rescode.err_read_lp_nonexisting_name (1162)`
 A variable never occurred in objective or constraints.

`rescode.err_lp_write_conic_problem (1163)`
 The problem contains cones that cannot be written to an LP formatted file.

`rescode.err_lp_write_geco_problem (1164)`
 The problem contains general convex terms that cannot be written to an LP formatted file.

`rescode.err_writing_file (1166)`
 An error occurred while writing file

`rescode.err_ptf_format (1167)`
 Syntax error in an PTF file

`rescode.err_opf_format (1168)`
 Syntax error in an OPF file

`rescode.err_opf_new_variable (1169)`
 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_invalid_name_in_sol_file (1170)`
 An invalid name occurred in a solution file.

`rescode.err_lp_invalid_con_name (1171)`
 A constraint name is invalid when used in an LP formatted file.

rescode.err_opf_premature_eof (1172)
 Premature end of file in an OPF file.

rescode.err_json_syntax (1175)
 Syntax error in an JSON data

rescode.err_json_string (1176)
 Error in JSON string.

rescode.err_json_number_overflow (1177)
 Invalid number entry - wrong type or value overflow.

rescode.err_json_format (1178)
 Error in an JSON Task file

rescode.err_json_data (1179)
 Inconsistent data in JSON Task file

rescode.err_json_missing_data (1180)
 Missing data section in JSON task file.

rescode.err_argument_lenneq (1197)
 Incorrect length of arguments.

rescode.err_argument_type (1198)
 Incorrect argument type.

rescode.err_num_arguments (1199)
 Incorrect number of function arguments.

rescode.err_in_argument (1200)
 A function argument is incorrect.

rescode.err_argument_dimension (1201)
 A function argument is of incorrect dimension.

rescode.err_shape_is_too_large (1202)
 The size of the n-dimensional shape is too large.

rescode.err_index_is_too_small (1203)
 An index in an argument is too small.

rescode.err_index_is_too_large (1204)
 An index in an argument is too large.

rescode.err_param_name (1205)
 The parameter name is not correct.

rescode.err_param_name_dou (1206)
 The parameter name is not correct for a double parameter.

rescode.err_param_name_int (1207)
 The parameter name is not correct for an integer parameter.

rescode.err_param_name_str (1208)
 The parameter name is not correct for a string parameter.

rescode.err_param_index (1210)
 Parameter index is out of range.

rescode.err_param_is_too_large (1215)
 The parameter value is too large.

rescode.err_param_is_too_small (1216)
 The parameter value is too small.

rescode.err_param_value_str (1217)
 The parameter value string is incorrect.

rescode.err_param_type (1218)
 The parameter type is invalid.

rescode.err_inf_dou_index (1219)
 A double information index is out of range for the specified type.

rescode.err_inf_int_index (1220)
 An integer information index is out of range for the specified type.

rescode.err_index_arr_is_too_small (1221)
 An index in an array argument is too small.

rescode.err_index_arr_is_too_large (1222)
 An index in an array argument is too large.

rescode.err_inf_lint_index (1225)
 A long integer information index is out of range for the specified type.

`rescode.err_arg_is_too_small` (1226)
 The value of a argument is too small.

`rescode.err_arg_is_too_large` (1227)
 The value of a argument is too large.

`rescode.err_invalid_whichsol` (1228)
`whichsol` is invalid.

`rescode.err_inf_dou_name` (1230)
 A double information name is invalid.

`rescode.err_inf_int_name` (1231)
 An integer information name is invalid.

`rescode.err_inf_type` (1232)
 The information type is invalid.

`rescode.err_inf_lint_name` (1234)
 A long integer information name is invalid.

`rescode.err_index` (1235)
 An index is out of range.

`rescode.err_whichsol` (1236)
 The solution defined by `whichsol` does not exists.

`rescode.err_solitem` (1237)
 The solution item number `solitem` is invalid. Please note that `solitem.snz` is invalid for the basic solution.

`rescode.err_whichitem_not_allowed` (1238)
`whichitem` is unacceptable.

`rescode.err_maxnumcon` (1240)
 The maximum number of constraints specified is smaller than the number of constraints in the task.

`rescode.err_maxnumvar` (1241)
 The maximum number of variables specified is smaller than the number of variables in the task.

`rescode.err_maxnumbarvar` (1242)
 The maximum number of semidefinite variables specified is smaller than the number of semidefinite variables in the task.

`rescode.err_maxnumqnz` (1243)
 The maximum number of non-zeros specified for the Q matrices is smaller than the number of non-zeros in the current Q matrices.

`rescode.err_too_small_max_num_nz` (1245)
 The maximum number of non-zeros specified is too small.

`rescode.err_invalid_idx` (1246)
 A specified index is invalid.

`rescode.err_invalid_max_num` (1247)
 A specified index is invalid.

`rescode.err_numconlim` (1250)
 Maximum number of constraints limit is exceeded.

`rescode.err_numvarlim` (1251)
 Maximum number of variables limit is exceeded.

`rescode.err_too_small_maxnumanz` (1252)
 The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A .

`rescode.err_inv_aptre` (1253)
`aptrb[j]` is strictly smaller than `aptrb[j]` for some j .

`rescode.err_mul_a_element` (1254)
 An element in A is defined multiple times.

`rescode.err_inv_bk` (1255)
 Invalid bound key.

`rescode.err_inv_bkc` (1256)
 Invalid bound key is specified for a constraint.

`rescode.err_inv_bkx` (1257)
 An invalid bound key is specified for a variable.

`rescode.err_inv_var_type` (1258)
 An invalid variable type is specified for a variable.

`rescode.err_solver_prodtype` (1259)
 Problem type does not match the chosen optimizer.

`rescode.err_objective_range` (1260)
 Empty objective range.

`rescode.err_undef_solution` (1265)
MOSEK has the following solution types:

- an interior-point solution,
- a basic solution,
- and an integer solution.

Each optimizer may set one or more of these solutions; e.g by default a successful optimization with the interior-point optimizer defines the interior-point solution and, for linear problems, also the basic solution. This error occurs when asking for a solution or for information about a solution that is not defined.

`rescode.err_basis` (1266)
 An invalid basis is specified. Either too many or too few basis variables are specified.

`rescode.err_inv_skc` (1267)
 Invalid value in `skc`.

`rescode.err_inv_skc` (1268)
 Invalid value in `skx`.

`rescode.err_inv_skn` (1274)
 Invalid value in `skn`.

`rescode.err_inv_sk_str` (1269)
 Invalid status key string encountered.

`rescode.err_inv_sk` (1270)
 Invalid status key code.

`rescode.err_inv_cone_type_str` (1271)
 Invalid cone type string encountered.

`rescode.err_inv_cone_type` (1272)
 Invalid cone type code is encountered.

`rescode.err_invalid_surplus` (1275)
 Invalid surplus.

`rescode.err_inv_name_item` (1280)
 An invalid name item code is used.

`rescode.err_pro_item` (1281)
 An invalid problem is used.

`rescode.err_invalid_format_type` (1283)
 Invalid format type.

`rescode.err_firsti` (1285)
 Invalid `firsti`.

`rescode.err_lasti` (1286)
 Invalid `lasti`.

`rescode.err_firstj` (1287)
 Invalid `firstj`.

`rescode.err_lastj` (1288)
 Invalid `lastj`.

`rescode.err_max_len_is_too_small` (1289)
 A maximum length that is too small has been specified.

`rescode.err_nonlinear_equality` (1290)
 The model contains a nonlinear equality which defines a nonconvex set.

`rescode.err_nonconvex` (1291)
 The optimization problem is nonconvex.

`rescode.err_nonlinear_ranged` (1292)
 Nonlinear constraints with finite lower and upper bound always define a nonconvex feasible set.

`rescode.err_con_q_not_psd (1293)`
The quadratic constraint matrix is not positive semidefinite as expected for a constraint with finite upper bound. This results in a nonconvex problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.

`rescode.err_con_q_not_nsd (1294)`
The quadratic constraint matrix is not negative semidefinite as expected for a constraint with finite lower bound. This results in a nonconvex problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.

`rescode.err_obj_q_not_psd (1295)`
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_obj_q_not_nsd (1296)`
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_argument_perm_array (1299)`
An invalid permutation array is specified.

`rescode.err_cone_index (1300)`
An index of a non-existing cone has been specified.

`rescode.err_cone_size (1301)`
A cone with incorrect number of members is specified.

`rescode.err_cone_overlap (1302)`
One or more of the variables in the cone to be added is already member of another cone. Now assume the variable is x_j then add a new variable say x_k and the constraint

$$x_j = x_k$$

and then let x_k be member of the cone to be appended.

`rescode.err_cone_rep_var (1303)`
A variable is included multiple times in the cone.

`rescode.err_maxnumcone (1304)`
The value specified for `maxnumcone` is too small.

`rescode.err_cone_type (1305)`
Invalid cone type specified.

`rescode.err_cone_type_str (1306)`
Invalid cone type specified.

`rescode.err_cone_overlap_append (1307)`
The cone to be appended has one variable which is already member of another cone.

`rescode.err_remove_cone_variable (1310)`
A variable cannot be removed because it will make a cone invalid.

`rescode.err_appending_too_big_cone (1311)`
Trying to append a too big cone.

`rescode.err_cone_parameter (1320)`
An invalid cone parameter.

`rescode.err_sol_file_invalid_number (1350)`
An invalid number is specified in a solution file.

`rescode.err_huge_c (1375)`
A huge value in absolute size is specified for one c_j .

`rescode.err_huge_aij (1380)`
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_duplicate_aij (1385)`
An element in the A matrix is specified twice.

`rescode.err_lower_bound_is_a_nan (1390)`
The lower bound specified is not a number (nan).

`rescode.err_upper_bound_is_a_nan (1391)`
The upper bound specified is not a number (nan).

`rescode.err_infinite_bound` (1400)
 A numerically huge bound value is specified.

`rescode.err_inv_qobj_subi` (1401)
 Invalid value in `qosubi`.

`rescode.err_inv_qobj_subj` (1402)
 Invalid value in `qosubj`.

`rescode.err_inv_qobj_val` (1403)
 Invalid value in `qoval`.

`rescode.err_inv_qcon_subk` (1404)
 Invalid value in `qcsubk`.

`rescode.err_inv_qcon_subi` (1405)
 Invalid value in `qcsubi`.

`rescode.err_inv_qcon_subj` (1406)
 Invalid value in `qcsbj`.

`rescode.err_inv_qcon_val` (1407)
 Invalid value in `qcval`.

`rescode.err_qcon_subi_too_small` (1408)
 Invalid value in `qcsubi`.

`rescode.err_qcon_subi_too_large` (1409)
 Invalid value in `qcsubi`.

`rescode.err_qobj_upper_triangle` (1415)
 An element in the upper triangle of Q^o is specified. Only elements in the lower triangle should be specified.

`rescode.err_qcon_upper_triangle` (1417)
 An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.

`rescode.err_fixed_bound_values` (1420)
 A fixed constraint/variable has been specified using the bound keys but the numerical value of the lower and upper bound is different.

`rescode.err_too_small_a_truncation_value` (1421)
 A too small value for the A truncation value is specified.

`rescode.err_invalid_objective_sense` (1445)
 An invalid objective sense is specified.

`rescode.err_undefined_objective_sense` (1446)
 The objective sense has not been specified before the optimization.

`rescode.err_y_is_undefined` (1449)
 The solution item y is undefined.

`rescode.err_nan_in_double_data` (1450)
 An invalid floating point value was used in some double data.

`rescode.err_nan_in_blc` (1461)
 l^c contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_buc` (1462)
 u^c contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_c` (1470)
 c contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_blx` (1471)
 l^x contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_bux` (1472)
 u^x contains an invalid floating point value, i.e. a NaN.

`rescode.err_invalid_aij` (1473)
 $a_{i,j}$ contains an invalid floating point value, i.e. a NaN or an infinite value.

`rescode.err_sym_mat_invalid` (1480)
 A symmetric matrix contains an invalid floating point value, i.e. a NaN or an infinite value.

`rescode.err_sym_mat_huge` (1482)
 A symmetric matrix contains a huge value in absolute size. The parameter `dparam.data_sym_mat_tol_huge` controls when an $e_{i,j}$ is considered huge.

`rescode.err_inv_problem` (1500)
 Invalid problem type. Probably a nonconvex problem has been specified.

`rescode.err_mixed_conic_and_nl (1501)`
 The problem contains nonlinear terms conic constraints. The requested operation cannot be applied to this type of problem.

`rescode.err_global_inv_conic_problem (1503)`
 The global optimizer can only be applied to problems without semidefinite variables.

`rescode.err_inv_optimizer (1550)`
 An invalid optimizer has been chosen for the problem.

`rescode.err_mio_no_optimizer (1551)`
 No optimizer is available for the current class of integer optimization problems.

`rescode.err_no_optimizer_var_type (1552)`
 No optimizer is available for this class of optimization problems.

`rescode.err_final_solution (1560)`
 An error occurred during the solution finalization.

`rescode.err_first (1570)`
 Invalid first.

`rescode.err_last (1571)`
 Invalid index last. A given index was out of expected range.

`rescode.err_slice_size (1572)`
 Invalid slice size specified.

`rescode.err_negative_surplus (1573)`
 Negative surplus.

`rescode.err_negative_append (1578)`
 Cannot append a negative number.

`rescode.err_postsolve (1580)`
 An error occurred during the postsolve. Please contact **MOSEK** support.

`rescode.err_overflow (1590)`
 A computation produced an overflow i.e. a very large number.

`rescode.err_no_basis_sol (1600)`
 No basic solution is defined.

`rescode.err_basis_factor (1610)`
 The factorization of the basis is invalid.

`rescode.err_basis_singular (1615)`
 The basis is singular and hence cannot be factored.

`rescode.err_factor (1650)`
 An error occurred while factorizing a matrix.

`rescode.err_feasrepair_cannot_relax (1700)`
 An optimization problem cannot be relaxed.

`rescode.err_feasrepair_solving_relaxed (1701)`
 The relaxed problem could not be solved to optimality. Please consult the log file for further details.

`rescode.err_feasrepair_inconsistent_bound (1702)`
 The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.

`rescode.err_repair_invalid_problem (1710)`
 The feasibility repair does not support the specified problem type.

`rescode.err_repair_optimization_failed (1711)`
 Computation the optimal relaxation failed. The cause may have been numerical problems.

`rescode.err_name_max_len (1750)`
 A name is longer than the buffer that is supposed to hold it.

`rescode.err_name_is_null (1760)`
 The name buffer is a NULL pointer.

`rescode.err_invalid_compression (1800)`
 Invalid compression type.

`rescode.err_invalid_iomode (1801)`
 Invalid io mode.

`rescode.err_no_primal_infeas_cer (2000)`
 A certificate of primal infeasibility is not available.

`rescode.err_no_dual_infeas_cer (2001)`
 A certificate of infeasibility is not available.

```

rescode.err_no_solution_in_callback (2500)
    The required solution is not available.
rescode.err_inv_marki (2501)
    Invalid value in marki.
rescode.err_inv_markj (2502)
    Invalid value in markj.
rescode.err_inv_numi (2503)
    Invalid numi.
rescode.err_inv_numj (2504)
    Invalid numj.
rescode.err_task_incompatible (2560)
    The Task file is incompatible with this platform. This results from reading a file on a 32 bit
    platform generated on a 64 bit platform.
rescode.err_task_invalid (2561)
    The Task file is invalid.
rescode.err_task_write (2562)
    Failed to write the task file.
rescode.err_lu_max_num_tries (2800)
    Could not compute the LU factors of the matrix within the maximum number of allowed tries.
rescode.err_invalid_utf8 (2900)
    An invalid UTF8 string is encountered.
rescode.err_invalid_wchar (2901)
    An invalid wchar string is encountered.
rescode.err_no_dual_for_itg_sol (2950)
    No dual information is available for the integer solution.
rescode.err_no_snx_for_bas_sol (2953)
     $s_n^x$  is not available for the basis solution.
rescode.err_internal (3000)
    An internal error occurred. Please report this problem.
rescode.err_api_array_too_small (3001)
    An input array was too short.
rescode.err_api_cb_connect (3002)
    Failed to connect a callback object.
rescode.err_api_fatal_error (3005)
    An internal error occurred in the API. Please report this problem.
rescode.err_api_internal (3999)
    An internal fatal error occurred in an interface function.
rescode.err_sen_format (3050)
    Syntax error in sensitivity analysis file.
rescode.err_sen_undef_name (3051)
    An undefined name was encountered in the sensitivity analysis file.
rescode.err_sen_index_range (3052)
    Index out of range in the sensitivity analysis file.
rescode.err_sen_bound_invalid_up (3053)
    Analysis of upper bound requested for an index, where no upper bound exists.
rescode.err_sen_bound_invalid_lo (3054)
    Analysis of lower bound requested for an index, where no lower bound exists.
rescode.err_sen_index_invalid (3055)
    Invalid range given in the sensitivity file.
rescode.err_sen_invalid_regexp (3056)
    Syntax error in regexp or regexp longer than 1024.
rescode.err_sen_solution_status (3057)
    No optimal solution found to the original problem given for sensitivity analysis.
rescode.err_sen_numerical (3058)
    Numerical difficulties encountered performing the sensitivity analysis.
rescode.err_sen_unhandled_problem_type (3080)
    Sensitivity analysis cannot be performed for the specified problem. Sensitivity analysis is only
    possible for linear problems.

```

`rescode.err_unb_step_size (3100)`
 A step size in an optimizer was unexpectedly unbounded. For instance, if the step-size becomes unbounded in phase 1 of the simplex algorithm then an error occurs. Normally this will happen only if the problem is badly formulated. Please contact **MOSEK** support if this error occurs.

`rescode.err_identical_tasks (3101)`
 Some tasks related to this function call were identical. Unique tasks were expected.

`rescode.err_ad_invalid_codelist (3102)`
 The code list data was invalid.

`rescode.err_internal_test_failed (3500)`
 An internal unit test function failed.

`rescode.err_xml_invalid_problem_type (3600)`
 The problem type is not supported by the XML format.

`rescode.err_invalid_ampl_stub (3700)`
 Invalid AMPL stub.

`rescode.err_int64_to_int32_cast (3800)`
 A 64 bit integer could not be cast to a 32 bit integer.

`rescode.err_size_license_numcores (3900)`
 The computer contains more cpu cores than the license allows for.

`rescode.err_infeas_undefined (3910)`
 The requested value is not defined for this solution type.

`rescode.err_no_barx_for_solution (3915)`
 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_bars_for_solution (3916)`
 There is no \bar{s} available for the solution specified. In particular note there are no \bar{s} defined for the basic and integer solutions.

`rescode.err_bar_var_dim (3920)`
 The dimension of a symmetric matrix variable has to be greater than 0.

`rescode.err_sym_mat_invalid_row_index (3940)`
 A row index specified for sparse symmetric matrix is invalid.

`rescode.err_sym_mat_invalid_col_index (3941)`
 A column index specified for sparse symmetric matrix is invalid.

`rescode.err_sym_mat_not_lower_tringular (3942)`
 Only the lower triangular part of sparse symmetric matrix should be specified.

`rescode.err_sym_mat_invalid_value (3943)`
 The numerical value specified in a sparse symmetric matrix is not a floating point value.

`rescode.err_sym_mat_duplicate (3944)`
 A value in a symmetric matrix as been specified more than once.

`rescode.err_invalid_sym_mat_dim (3950)`
 A sparse symmetric matrix of invalid dimension is specified.

`rescode.err_invalid_file_format_for_sym_mat (4000)`
 The file format does not support a problem with symmetric matrix variables.

`rescode.err_invalid_file_format_for_cfix (4001)`
 The file format does not support a problem with nonzero fixed term in c.

`rescode.err_invalid_file_format_for_ranged_constraints (4002)`
 The file format does not support a problem with ranged constraints.

`rescode.err_invalid_file_format_for_free_constraints (4003)`
 The file format does not support a problem with free constraints.

`rescode.err_invalid_file_format_for_cones (4005)`
 The file format does not support a problem with conic constraints.

`rescode.err_invalid_file_format_for_nonlinear (4010)`
 The file format does not support a problem with nonlinear terms.

`rescode.err_duplicate_constraint_names (4500)`
 Two constraint names are identical.

`rescode.err_duplicate_variable_names (4501)`
 Two variable names are identical.

`rescode.err_duplicate_barvariable_names (4502)`
 Two barvariable names are identical.

`rescode.err_duplicate_cone_names (4503)`
 Two cone names are identical.

`rescode.err_non_unique_array (5000)`
 An array does not contain unique elements.

`rescode.err_argument_is_too_large (5005)`
 The value of a function argument is too large.

`rescode.err_mio_internal (5010)`
 A fatal error occurred in the mixed integer optimizer. Please contact **MOSEK** support.

`rescode.err_invalid_problem_type (6000)`
 An invalid problem type.

`rescode.err_unhandled_solution_status (6010)`
 Unhandled solution status.

`rescode.err_upper_triangle (6020)`
 An element in the upper triangle of a lower triangular matrix is specified.

`rescode.err_lau_singular_matrix (7000)`
 A matrix is singular.

`rescode.err_lau_not_positive_definite (7001)`
 A matrix is not positive definite.

`rescode.err_lau_invalid_lower_triangular_matrix (7002)`
 An invalid lower triangular matrix.

`rescode.err_lau_unknown (7005)`
 An unknown error.

`rescode.err_lau_arg_m (7010)`
 Invalid argument m.

`rescode.err_lau_arg_n (7011)`
 Invalid argument n.

`rescode.err_lau_arg_k (7012)`
 Invalid argument k.

`rescode.err_lau_arg_transa (7015)`
 Invalid argument transa.

`rescode.err_lau_arg_transb (7016)`
 Invalid argument transb.

`rescode.err_lau_arg_uplo (7017)`
 Invalid argument uplo.

`rescode.err_lau_arg_trans (7018)`
 Invalid argument trans.

`rescode.err_lau_invalid_sparse_symmetric_matrix (7019)`
 An invalid sparse symmetric matrix is specified. Note only the lower triangular part with no duplicates is specified.

`rescode.err_cbf_parse (7100)`
 An error occurred while parsing an CBF file.

`rescode.err_cbf_obj_sense (7101)`
 An invalid objective sense is specified.

`rescode.err_cbf_no_variables (7102)`
 No variables are specified.

`rescode.err_cbf_too_many_constraints (7103)`
 Too many constraints specified.

`rescode.err_cbf_too_many_variables (7104)`
 Too many variables specified.

`rescode.err_cbf_no_version_specified (7105)`
 No version specified.

`rescode.err_cbf_syntax (7106)`
 Invalid syntax.

`rescode.err_cbf_duplicate_obj (7107)`
 Duplicate OBJ keyword.

`rescode.err_cbf_duplicate_con (7108)`
 Duplicate CON keyword.

```

rescode.err_cbf_duplicate_var (7109)
    Duplicate VAR keyword.
rescode.err_cbf_duplicate_int (7110)
    Duplicate INT keyword.
rescode.err_cbf_invalid_var_type (7111)
    Invalid variable type.
rescode.err_cbf_invalid_con_type (7112)
    Invalid constraint type.
rescode.err_cbf_invalid_domain_dimension (7113)
    Invalid domain dimension.
rescode.err_cbf_duplicate_objcoord (7114)
    Duplicate index in OBJCOORD.
rescode.err_cbf_duplicate_bcoord (7115)
    Duplicate index in BCOORD.
rescode.err_cbf_duplicate_acoord (7116)
    Duplicate index in ACOORD.
rescode.err_cbf_too_few_variables (7117)
    Too few variables defined.
rescode.err_cbf_too_few_constraints (7118)
    Too few constraints defined.
rescode.err_cbf_too_few_ints (7119)
    Too few ints are specified.
rescode.err_cbf_too_many_ints (7120)
    Too many ints are specified.
rescode.err_cbf_invalid_int_index (7121)
    Invalid INT index.
rescode.err_cbf_unsupported (7122)
    Unsupported feature is present.
rescode.err_cbf_duplicate_psdvar (7123)
    Duplicate PSDVAR keyword.
rescode.err_cbf_invalid_psdvar_dimension (7124)
    Invalid PSDVAR dimension.
rescode.err_cbf_too_few_psdvar (7125)
    Too few variables defined.
rescode.err_cbf_invalid_exp_dimension (7126)
    Invalid dimension of a exponential cone.
rescode.err_cbf_duplicate_pow_cones (7130)
    Multiple POWCONES specified.
rescode.err_cbf_duplicate_pow_star_cones (7131)
    Multiple POW*CONES specified.
rescode.err_cbf_invalid_power (7132)
    Invalid power specified.
rescode.err_cbf_power_cone_is_too_long (7133)
    Power cone is too long.
rescode.err_cbf_invalid_power_cone_index (7134)
    Invalid power cone index.
rescode.err_cbf_invalid_power_star_cone_index (7135)
    Invalid power star cone index.
rescode.err_cbf_unhandled_power_cone_type (7136)
    An unhandled power cone type.
rescode.err_cbf_unhandled_power_star_cone_type (7137)
    An unhandled power star cone type.
rescode.err_cbf_power_cone_mismatch (7138)
    The power cone does not match with it definition.
rescode.err_cbf_power_star_cone_mismatch (7139)
    The power star cone does not match with it definition.
rescode.err_cbf_invalid_number_of_cones (7740)
    Invalid number of cones.

```


mark
 Mark

mark.lo
 The lower bound is selected for sensitivity analysis.

mark.up
 The upper bound is selected for sensitivity analysis.

simdegen
 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.

transpose
 Transposed matrix.

transpose.no
 No transpose is applied.

transpose.yes
 A transpose is applied.

uplo
 Triangular part of a symmetric matrix.

uplo.lo
 Lower part.

uplo.up
 Upper part.

simreform
 Problem reformulation.

simreform.on
 Allow the simplex optimizer to reformulate the problem.

simreform.off
 Disallow 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.

simdupvec
 Exploit duplicate columns.

simdupvec.on
 Allow the simplex optimizer to exploit duplicated columns.

simdupvec.off
 Disallow the simplex optimizer to exploit duplicated columns.

simdupvec.free
 The simplex optimizer can choose freely.

simhotstart
 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.

`intpnthotstart`
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.

`purify`
Solution purification employed optimizer.

`purify.none`
The optimizer performs no solution purification.

`purify.primal`
The optimizer purifies the primal solution.

`purify.dual`
The optimizer purifies the dual solution.

`purify.primal_dual`
The optimizer purifies both the primal and dual solution.

`purify.auto`
TBD

`callbackcode`
Progress callback codes

`callbackcode.begin_bi`
The basis identification procedure has been started.

`callbackcode.begin_conic`
The callback function is called when the conic optimizer is started.

`callbackcode.begin_dual_bi`
The callback 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 callback function is called when the dual BI phase is started.

`callbackcode.begin_dual_simplex`
The callback function is called when the dual simplex optimizer started.

`callbackcode.begin_dual_simplex_bi`
The callback 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 callback function is called when the infeasibility analyzer is started.

`callbackcode.begin_intpnt`
The callback function is called when the interior-point optimizer is started.

`callbackcode.begin_license_wait`
Begin waiting for license.

`callbackcode.begin_mio`
The callback function is called when the mixed-integer optimizer is started.

`callbackcode.begin_optimizer`
The callback function is called when the optimizer is started.

`callbackcode.begin_presolve`
The callback function is called when the presolve is started.

`callbackcode.begin_primal_bi`
The callback function is called from within the basis identification procedure when the primal 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 callback function is called when the primal BI setup is started.

`callbackcode.begin_primal_simplex`
The callback function is called when the primal simplex optimizer is started.

`callbackcode.begin_primal_simplex_bi`
The callback function is called from within the basis identification procedure when the primal simplex clean-up phase is started.

`callbackcode.begin_qcqp_reformulate`
Begin QCQP reformulation.

`callbackcode.begin_read`
MOSEK has started reading a problem file.

`callbackcode.begin_root_cutgen`
The callback function is called when root cut generation is started.

`callbackcode.begin_simplex`
The callback function is called when the simplex optimizer is started.

`callbackcode.begin_simplex_bi`
The callback function is called from within the basis identification procedure when the simplex clean-up phase is started.

`callbackcode.begin_to_conic`
Begin conic reformulation.

`callbackcode.begin_write`
MOSEK has started writing a problem file.

`callbackcode.conic`
The callback function is called from within the conic optimizer after the information database has been updated.

`callbackcode.dual_simplex`
The callback function is called from within the dual simplex optimizer.

`callbackcode.end_bi`
The callback function is called when the basis identification procedure is terminated.

`callbackcode.end_conic`
The callback function is called when the conic optimizer is terminated.

`callbackcode.end_dual_bi`
The callback 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 callback function is called when the dual BI phase is terminated.

`callbackcode.end_dual_simplex`
The callback function is called when the dual simplex optimizer is terminated.

`callbackcode.end_dual_simplex_bi`
The callback 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 callback function is called when the infeasibility analyzer is terminated.

`callbackcode.end_intpnt`
The callback function is called when the interior-point optimizer is terminated.

`callbackcode.end_license_wait`
End waiting for license.

`callbackcode.end_mio`
The callback function is called when the mixed-integer optimizer is terminated.

`callbackcode.end_optimizer`
The callback function is called when the optimizer is terminated.

`callbackcode.end_presolve`
The callback function is called when the presolve is completed.

`callbackcode.end_primal_bi`
The callback function is called from within the basis identification procedure when the primal 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 callback function is called when the primal BI setup is terminated.

`callbackcode.end_primal_simplex`
The callback function is called when the primal simplex optimizer is terminated.

`callbackcode.end_primal_simplex_bi`
The callback function is called from within the basis identification procedure when the primal clean-up phase is terminated.

`callbackcode.end_qcqp_reformulate`
End QCQP reformulation.

`callbackcode.end_read`
MOSEK has finished reading a problem file.

`callbackcode.end_root_cutgen`
The callback function is called when root cut generation is terminated.

`callbackcode.end_simplex`
The callback function is called when the simplex optimizer is terminated.

`callbackcode.end_simplex_bi`
The callback function is called from within the basis identification procedure when the simplex clean-up phase is terminated.

`callbackcode.end_to_conic`
End conic reformulation.

`callbackcode.end_write`
MOSEK has finished writing a problem file.

`callbackcode.im_bi`
The callback function is called from within the basis identification procedure at an intermediate point.

`callbackcode.im_conic`
The callback function is called at an intermediate stage within the conic optimizer where the information database has not been updated.

`callbackcode.im_dual_bi`
The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

`callbackcode.im_dual_sensitivity`
The callback function is called at an intermediate stage of the dual sensitivity analysis.

`callbackcode.im_dual_simplex`
The callback function is called at an intermediate point in the dual simplex optimizer.

`callbackcode.im_full_convexity_check`
The callback function is called at an intermediate stage of the full convexity check.

`callbackcode.im_intpnt`
The callback 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 callback function is called from within the LU factorization procedure at an intermediate point.

`callbackcode.im_mio`
The callback function is called at an intermediate point in the mixed-integer optimizer.

`callbackcode.im_mio_dual_simplex`
The callback function is called at an intermediate point in the mixed-integer optimizer while running the dual simplex optimizer.

`callbackcode.im_mio_intpnt`
The callback function is called at an intermediate point in the mixed-integer optimizer while running the interior-point optimizer.

`callbackcode.im_mio_primal_simplex`
The callback function is called at an intermediate point in the mixed-integer optimizer while running the primal simplex optimizer.

`callbackcode.im_order`
The callback function is called from within the matrix ordering procedure at an intermediate point.

`callbackcode.im_presolve`
The callback function is called from within the presolve procedure at an intermediate stage.

`callbackcode.im_primal_bi`
The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

`callbackcode.im_primal_sensitivity`
The callback function is called at an intermediate stage of the primal sensitivity analysis.

`callbackcode.im_primal_simplex`
The callback function is called at an intermediate point in the primal simplex optimizer.

`callbackcode.im_qo_reformulate`
The callback function is called at an intermediate stage of the conic quadratic reformulation.

`callbackcode.im_read`
Intermediate stage in reading.

`callbackcode.im_root_cutgen`
The callback is called from within root cut generation at an intermediate stage.

`callbackcode.im_simplex`
The callback function is called from within the simplex optimizer at an intermediate point.

`callbackcode.im_simplex_bi`
The callback function is called from within the basis identification procedure at an intermediate point in the simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.intpnt`
The callback function is called from within the interior-point optimizer after the information database has been updated.

`callbackcode.new_int_mio`
The callback function is called after a new integer solution has been located by the mixed-integer optimizer.

`callbackcode.primal_simplex`
The callback function is called from within the primal simplex optimizer.

`callbackcode.read_opf`
The callback function is called from the OPF reader.

`callbackcode.read_opf_section`
A chunk of Q non-zeros has been read from a problem file.

`callbackcode.solving_remote`
The callback function is called while the task is being solved on a remote server.

`callbackcode.update_dual_bi`
The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

`callbackcode.update_dual_simplex`
The callback function is called in the dual simplex optimizer.

`callbackcode.update_dual_simplex_bi`
The callback function is called from within the basis identification procedure at an intermediate point in the dual simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.update_presolve`
The callback function is called from within the presolve procedure.

`callbackcode.update_primal_bi`
The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

`callbackcode.update_primal_simplex`
The callback function is called in the primal simplex optimizer.

`callbackcode.update_primal_simplex_bi`
The callback function is called from within the basis identification procedure at an intermediate point in the primal simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.write_opf`
The callback function is called from the OPF writer.

`checkconvexitytype`
Types of convexity checks.

`checkconvexitytype.none`
No convexity check.

`checkconvexitytype.simple`
Perform simple and fast convexity check.

`checkconvexitytype.full`
Perform a full convexity check.

`compresstype`
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.

`compresstype.zstd`
The type of compression used is zstd compatible.

`conetype`
Cone types

`conetype.quad`
The cone is a quadratic cone.

`conetype.rquad`
The cone is a rotated quadratic cone.

`conetype.pexp`
A primal exponential cone.

`conetype.dexp`
A dual exponential cone.

`conetype.ppow`
A primal power cone.

`conetype.dpow`
A dual power cone.

`conetype.zero`
The zero cone.

`nametype`
Name types

`nametype.gen`
General names. However, no duplicate and blank names are allowed.

`nametype.mps`
MPS type names.

`nametype.lp`
LP type names.

`scopr`
SCopt operator types

`scopr.ent`
Entropy

`scopr.exp`
Exponential

`scopr.log`
Logarithm

`scopr.pow`
Power

`scopr.sqrt`
Square root

`symmattype`
Cone types

`symmattype.sparse`
Sparse symmetric matrix.

`dataformat`
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.free_mps`
The data a free MPS formatted file.

`dataformat.task`
Generic task dump file.

`dataformat.ptf`
(P)retty (T)ext (F)format.

`dataformat.cb`
Conic benchmark format,

`dataformat.json_task`
JSON based task format.

`dinfitem`
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_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.intpnt_dual_feas`
Dual feasibility measure reported by the interior-point optimizer. (For the interior-point optimizer this measure is not directly related to the original problem because a homogeneous model is employed.)

`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`
A measure of optimality of the solution. It should converge to +1 if the problem has a primal-dual optimal solution, and converge to -1 if the problem is (strictly) primal or dual infeasible. If the measure converges to another constant, or fails to settle, the problem is usually ill-posed.

`dinfitem.intpnt_order_time`
Order time (in seconds).

`dinfitem.intpnt_primal_feas`
Primal feasibility measure reported by the interior-point optimizer. (For the interior-point optimizer this measure is not directly related to the original problem because a homogeneous model is employed).

`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.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.qcqp_reformulate_max_perturbation`
Maximum absolute diagonal perturbation occurring during the QCQP reformulation.

`dinfitem.qcqp_reformulate_time`
Time spent with conic quadratic reformulation.

`dinfitem.qcqp_reformulate_worst_cholesky_column_scaling`
Worst Cholesky column scaling.

`dinfitem.qcqp_reformulate_worst_cholesky_diag_scaling`
Worst Cholesky diagonal scaling.

`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_obj`
Objective value reported by the simplex optimizer.

`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. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solution_info*.

`dinfitem.sol_bas_dviolcon`
Maximal dual bound violation for x^c in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solution_info*.

`dinfitem.sol_bas_dviolvar`
Maximal dual bound violation for x^x in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solution_info*.

`dinfitem.sol_bas_nrm_barx`
Infinity norm of \bar{X} in the basic solution.

`dinfitem.sol_bas_nrm_slc`
Infinity norm of s_l^c in the basic solution.

`dinfitem.sol_bas_nrm_slx`
Infinity norm of s_l^x in the basic solution.

`dinfitem.sol_bas_nrm_suc`
Infinity norm of s_u^c in the basic solution.

`dinfitem.sol_bas_nrm_sux`
Infinity norm of s_u^X in the basic solution.

`dinfitem.sol_bas_nrm_xc`
Infinity norm of x^c in the basic solution.

`dinfitem.sol_bas_nrm_xx`
Infinity norm of x^x in the basic solution.

`dinfitem.sol_bas_nrm_y`
Infinity norm of y in the basic solution.

`dinfitem.sol_bas_primal_obj`
Primal objective value of the basic solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_bas_pviolcon`
Maximal primal bound violation for x^c in the basic solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_bas_pviolvar`
Maximal primal bound violation for x^x in the basic solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_itg_nrm_barx`
Infinity norm of \bar{X} in the integer solution.

`dinfitem.sol_itg_nrm_xc`
Infinity norm of x^c in the integer solution.

`dinfitem.sol_itg_nrm_xx`
Infinity norm of x^x in the integer solution.

`dinfitem.sol_itg_primal_obj`
Primal objective value of the integer solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_itg_pviolbarvar`
Maximal primal bound violation for \bar{X} in the integer solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_itg_pviolcon`
Maximal primal bound violation for x^c in the integer solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_itg_pviolcones`
Maximal primal violation for primal conic constraints in the integer solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_itg_pviolitg`
Maximal violation for the integer constraints in the integer solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_itg_pviolvar`
Maximal primal bound violation for x^x in the integer solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_itr_dual_obj`
Dual objective value of the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_itr_dviolbarvar`
Maximal dual bound violation for \bar{X} in the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_itr_dviolcon`
Maximal dual bound violation for x^c in the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_itr_dviolcones`
Maximal dual violation for dual conic constraints in the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_itr_dviolvar`
Maximal dual bound violation for x^x in the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solutioninfo`.

`dinfitem.sol_itr_nrm_bars`
Infinity norm of \bar{S} in the interior-point solution.

dinfitem.sol_itr_nrm_barx
 Infinity norm of \bar{X} in the interior-point solution.

dinfitem.sol_itr_nrm_slc
 Infinity norm of s_l^c in the interior-point solution.

dinfitem.sol_itr_nrm_slx
 Infinity norm of s_l^x in the interior-point solution.

dinfitem.sol_itr_nrm_snx
 Infinity norm of s_n^x in the interior-point solution.

dinfitem.sol_itr_nrm_suc
 Infinity norm of s_u^c in the interior-point solution.

dinfitem.sol_itr_nrm_sux
 Infinity norm of s_u^X in the interior-point solution.

dinfitem.sol_itr_nrm_xc
 Infinity norm of x^c in the interior-point solution.

dinfitem.sol_itr_nrm_xx
 Infinity norm of x^x in the interior-point solution.

dinfitem.sol_itr_nrm_y
 Infinity norm of y in the interior-point solution.

dinfitem.sol_itr_primal_obj
 Primal objective value of the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solution_info*.

dinfitem.sol_itr_pviolbarvar
 Maximal primal bound violation for \bar{X} in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solution_info*.

dinfitem.sol_itr_pviolcon
 Maximal primal bound violation for x^c in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solution_info*.

dinfitem.sol_itr_pviolcones
 Maximal primal violation for primal conic constraints in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solution_info*.

dinfitem.sol_itr_pviolvar
 Maximal primal bound violation for x^x in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solution_info*.

dinfitem.to_conic_time
 Time spent in the last to conic reformulation.

feature
 License feature

feature.pts
 Base system.

feature.pton
 Conic extension.

liinfitem
 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_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_anz`
Number of non-zero entries in the constraint matrix of the problem to be solved by the mixed-integer optimizer.

`liinfitem.mio_intpnt_iter`
Number of interior-point iterations performed by the mixed-integer optimizer.

`liinfitem.mio_presolved_anz`
Number of non-zero entries in the constraint matrix of the problem after the mixed-integer optimizer's presolve.

`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.

`iinfitem`
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 *Task.analyzeproblem*.

`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_absgap_satisfied`
Non-zero if absolute gap is within tolerances.

`iinfitem.mio_clique_table_size`
Size of the clique table.

`iinfitem.mio_construct_solution`
This item informs if **MOSEK** constructed an initial integer feasible solution.

- -1: tried, but failed,
- 0: no partial solution supplied by the user,
- 1: constructed feasible solution.

`iinfitem.mio_node_depth`
Depth of the last node solved.

`iinfitem.mio_num_active_nodes`
Number of active branch and bound nodes.

`iinfitem.mio_num_branch`
Number of branches performed during the optimization.

`iinfitem.mio_num_clique_cuts`
Number of clique cuts.

`iinfitem.mio_num_cmir_cuts`
Number of Complemented Mixed Integer Rounding (CMIR) cuts.

`iinfitem.mio_num_gomory_cuts`
Number of Gomory cuts.

`iinfitem.mio_num_implied_bound_cuts`
Number of implied bound cuts.

`iinfitem.mio_num_int_solutions`
Number of integer feasible solutions that have been found.

`iinfitem.mio_num_knapsack_cover_cuts`
Number of clique cuts.

`iinfitem.mio_num_relax`
Number of relaxations solved during the optimization.

`iinfitem.mio_num_repeated_presolve`
Number of times presolve was repeated at root.

`iinfitem.mio_numbin`
Number of binary variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numbinconevar`
Number of binary cone variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numcon`
Number of constraints in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numcone`
Number of cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numconevar`
Number of cone variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numcont`
Number of continuous variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numcontconevar`
Number of continuous cone variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numdexpcones`
Number of dual exponential cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numdpowcones`
Number of dual power cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numint`
Number of integer variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numintconevar`
Number of integer cone variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numexpcones`
Number of primal exponential cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numppowcones`
Number of primal power cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numqcones`
Number of quadratic cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numrqcones`
Number of rotated quadratic cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numvar`
Number of variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_obj_bound_defined`
Non-zero if a valid objective bound has been found, otherwise zero.

`iinfitem.mio_presolved_numbin`
Number of binary variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numbinconevar`
Number of binary cone variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numcon`
Number of constraints in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numcone`
Number of cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numconevar`
Number of cone variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numcont`
Number of continuous variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numcontconevar`
Number of continuous cone variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numdexpcones`
Number of dual exponential cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numdpowcones`
Number of dual power cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numint`
Number of integer variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numintconevar`
Number of integer cone variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numpexpcones`
Number of primal exponential cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numppowcones`
Number of primal power cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numqcones`
Number of quadratic cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numrqcones`
Number of rotated quadratic cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numvar`
Number of variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_relgap_satisfied`
Non-zero if relative gap is within tolerances.

`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 response code returned by optimize.

`iinfitem.purify_dual_success`
Is nonzero if the dual solution is purified.

`iinfitem.purify_primal_success`
Is nonzero if the primal solution is purified.

`iinfitem.rd_numbarvar`
Number of symmetric 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 matrices read.

`iinfitem.rd_numvar`
Number of variables read.

`iinfitem.rd_prototype`
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_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_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_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_realloc`
Number of times the storage for storing A has been changed. A large value may indicate that memory fragmentation may occur.

`inftype`
Information item types

`inftype.dou_type`
Is a double information type.

`inftype.int_type`
Is an integer.

`inftype.lint_type`
Is a long integer.

`iomode`
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 write.

branchdir
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.

branchdir.near
Branch in direction nearest to selected fractional variable.

branchdir.far
Branch in direction farthest from selected fractional variable.

branchdir.root_lp
Chose direction based on root lp value of selected variable.

branchdir.guided
Branch in direction of current incumbent.

branchdir.pseudocost
Branch based on the pseudocost of the variable.

miocontsoltype
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.

miomode
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.

mionodeseltype
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.pseudo
The optimizer employs selects the node based on a pseudo cost estimate.

mpsformat
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.

mpsformat.cplex
The CPLEX compatible version of the MPS format is employed.

objsense
Objective sense types

objsense.minimize
The problem should be minimized.

objsense.maximize
The problem should be maximized.

onoffkey
On/off

onoffkey.on
Switch the option on.

onoffkey.off
Switch the option off.

optimizertype
Optimizer types

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.primal_simplex
The primal simplex optimizer is used.

orderingtype
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 graph partitioning based ordering.

orderingtype.force_graphpar
Always use the graph partitioning based ordering even if it is worse than the approximate minimum local fill ordering.

`orderingtype.none`
No ordering is used.

`presolvemode`
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.

`parametertype`
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.

`problemitem`
Problem data items

`problemitem.var`
Item is a variable.

`problemitem.con`
Item is a constraint.

`problemitem.cone`
Item is a cone.

`problemtyp`
Problem types

`problemtyp.lo`
The problem is a linear optimization problem.

`problemtyp.qo`
The problem is a quadratic optimization problem.

`problemtyp.qcqo`
The problem is a quadratically constrained optimization problem.

`problemtyp.conic`
A conic optimization.

`problemtyp.mixed`
General nonlinear constraints and conic constraints. This combination can not be solved by **MOSEK**.

`prosta`
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.prim_infeas_or_unbounded
The problem is either primal infeasible or unbounded. This may occur for mixed-integer problems.

xmlwriteroutputtype
XML writer output mode

xmlwriteroutputtype.row
Write in row order.

xmlwriteroutputtype.col
Write in column order.

rescodetype
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.

scalingtype
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.

scalingmethod
Scaling method

scalingmethod.pow2
Scales only with power of 2 leaving the mantissa untouched.

scalingmethod.free
The optimizer chooses the scaling heuristic.

sensitivitytype
Sensitivity types

sensitivitytype.basis
Basis sensitivity analysis is performed.

simseltype
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.

`solitem`
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.

`solsta`
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.prim_illposed_cer`
The solution is a certificate that the primal problem is illposed.

solsta.dual_illposed_cer
The solution is a certificate that the dual problem is illposed.

solsta.integer_optimal
The primal solution is integer optimal.

soltype
Solution types

soltype.bas
The basic solution.

soltype.itr
The interior solution.

soltype.itg
The integer solution.

solveform
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.

stakey
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.

startpointtype
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 chosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should be employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

streamtype
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.

`value`
 Integer values

`value.max_str_len`
 Maximum string length allowed in **MOSEK**.

`value.license_buffer_length`
 The length of a license key buffer.

`variabletype`
 Variable types

`variabletype.type_cont`
 Is a continuous variable.

`variabletype.type_int`
 Is an integer variable.

15.10 Function Types

`callbackfunc`

```
def callbackfunc (code, dinf, iinf, liinf) -> stop
```

The progress and information callback function is a user-defined function which will be called by **MOSEK** occasionally during the optimization process. In particular, the callback function is called at the beginning of each iteration in the interior-point optimizer. For the simplex optimizers `iparam.log_sim_freq` controls how frequently the callback is called.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function. The only exception is the possibility to retrieve an integer solution, see *Progress and data callback*.

Parameters

- `code` (`callbackcode`) – Callback code indicating current operation of the solver. (input)
- `dinf` (`float[]`) – Array of double information items. (input)
- `iinf` (`int[]`) – Array of integer information items. (input)
- `liinf` (`int[]`) – Array of long integer information items. (input)

Return `stop` (`int`) – Non-zero if the optimizer should be terminated; zero otherwise.

`progresscallbackfunc`

```
def progresscallbackfunc (code) -> stop
```

The progress callback function is a user-defined function which will be called by **MOSEK** occasionally during the optimization process. In particular, the callback function is called at the beginning of each iteration in the interior-point optimizer. For the simplex optimizers `iparam.log_sim_freq` controls how frequently the callback is called.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function. If the progress callback function returns a non-zero value, the optimization process is terminated.

Parameters `code` (*mosek.callbackcode*) – Callback code indicating the current status of the solver. (input)

Return `stop` (int) – Non-zero if the optimizer should be terminated; zero otherwise.

`streamfunc`

```
def streamfunc (msg)
```

The message-stream callback function is a user-defined function which can be linked to any of the **MOSEK** streams. Doing so, the function is called whenever **MOSEK** sends a message to the stream.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function.

Parameters `msg` (str) – A string containing the message. (input)

15.11 Nonlinear interfaces (obsolete)

Important: This is a legacy document for users familiar with SCoPt, DGopt, EXPopt, mskenopt, mskscopt and mskgpopt from previous versions of **MOSEK**. These interfaces have now been removed. We assume familiarity with documentation included in version 8. All problems expressible with this interface can (and should) be reformulated using the exponential cone and power cones.

New users should formulate problems involving powers, logarithms and exponentials directly in conic form.

Conversion tutorial

We recommend converting all nonlinear problems using SCoPt, DGopt, EXPopt, mskenopt, mskscopt and mskgpopt into conic form. Depending on the values of f, g, h either the epigraph or hypograph of a SCoPt function if convex, and a bounding variable can be introduced following the basic rules below. We assume all variables are within safe bounds where the SCoPt operators are defined and convex. We also assume $f > 0$.

A more comprehensive modeling guide for these types of problems can be found in the **MOSEK Modeling Cookbook**.

Powers

Consider $f(x + h)^g$. This can be reformulated using the power cone.

- If $g > 1$ then $t \geq f(x + h)^g$ is equivalent to $(t/f)^{1/g} \geq |x + h|$, that is $(t/f, 1, x + h) \in \mathcal{P}_3^{1/g, 1-1/g}$.
- If $0 < g < 1$ then $|t| \leq f(x + h)^g$ is equivalent to $(x + h, 1, t/f) \in \mathcal{P}_3^{g, 1-g}$.
- If $g < 0$ then $t \geq f(x + h)^g$ is equivalent to $(t/f)(x + h)^{-g} \geq 1$, that is $(t/f, x + h, 1) \in \mathcal{P}_3^{1/(1-g), -g/(1-g)}$.

Logarithm

The bound $t \leq f \log(gx + h)$ is equivalent to $(gx + h, 1, t/f) \in K_{\text{exp}}$.

Entropy

The bound $t \geq fx \log x$ is equivalent to $(1, x, -t/f) \in K_{\text{exp}}$.

Exponential

The bound $t \geq f \exp(gx + h)$ is equivalent to $(t/f, 1, gx + h) \in K_{\text{exp}}$.

Exponential optimization (EXPopt), Geometric programming (mskgpopt)

For a basic tutorial in geometric programming (GP) see [Sec. 6.8](#).

An exponential optimization problem in standard form consists of constraints of the type:

$$t \geq \log \left(\sum_i \exp(a_i^T x + b_i) \right).$$

This log-sum-exp bound is equivalent to

$$\sum_i \exp(a_i^T x + b_i - t) \leq 1$$

and requires bounding each exponential function as explained above.

Dual geometric optimization (DGopt)

The objective function of a dual geometric problem involves maximizing expressions of the form

$$x \log \frac{c}{x} \quad \text{and} \quad x_i \log \frac{e^T x}{x_i},$$

which can be achieved using bounds $t \leq x \log \frac{y}{x}$, that is $(t, x, y) \in K_{\text{exp}}$.

Chapter 16

Supported File Formats

MOSEK supports a range of problem and solution formats listed in [Table 16.1](#) and [Table 16.2](#). The **Task format** is **MOSEK**'s native binary format and it supports all features that **MOSEK** supports. The **OPF format** is **MOSEK**'s human-readable alternative that supports nearly all features (everything except semidefinite problems). In general, text formats are significantly slower to read, but can be examined and edited directly in any text editor.

Problem formats

Table 16.1: List of supported file formats for optimization problems. The column *Conic* refers to conic problems involving the quadratic, rotated quadratic, power or exponential cone. The last two columns indicate if the format supports solutions and optimizer parameters.

Format Type	Ext.	Binary/Text	LP	QO	Conic	SDP	Sol	Param
<i>LP</i>	lp	plain text	X	X				
<i>MPS</i>	mps	plain text	X	X	X			
<i>OPF</i>	opf	plain text	X	X	X		X	X
<i>PTF</i>	ptf	plain text	X	X	X	X	X	
<i>CBF</i>	cbf	plain text	X		X	X		
<i>Task format</i>	task	binary	X	X	X	X	X	X
<i>Jtask format</i>	jtask	text	X	X	X	X	X	X

Solution formats

Table 16.2: List of supported solution formats.

Format Type	Ext.	Binary/Text	Description
<i>SOL</i>	sol	plain text	Interior Solution
	bas	plain text	Basic Solution
	int	plain text	Integer
<i>Jsol format</i>	jsol	text	Solution

Compression

MOSEK supports GZIP and Zstandard compression. Problem files with extension `.gz` (for GZIP) and `.zst` (for Zstandard) are assumed to be compressed when read, and are automatically compressed when written. For example, a file called

problem.mps.gz

will be considered as a GZIP compressed MPS file.

16.1 The LP File Format

MOSEK supports the LP file format with some extensions. The LP format is not a completely well-defined standard and hence different optimization packages may interpret the same LP file in slightly different ways. **MOSEK** tries to emulate as closely as possible CPLEX's behavior, but tries to stay backward compatible.

The LP file format can specify problems of the form

$$\begin{array}{ll} \text{minimize/maximize} & c^T x + \frac{1}{2} q^o(x) \\ \text{subject to} & l^c \leq Ax + \frac{1}{2} q(x) \leq u^c, \\ & l^x \leq x \leq u^x, \\ & x_{\mathcal{J}} \text{ integer,} \end{array}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear term in the objective.
- $q^o : \mathbb{R}^n \rightarrow \mathbb{R}$ 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

$$Q^i = (Q^i)^T.$$

- $\mathcal{J} \subseteq \{1, 2, \dots, n\}$ is an index set of the integer constrained variables.

16.1.1 File Sections

An LP formatted file contains a number of sections specifying the objective, constraints, variable bounds, and variable types. The section keywords may be any mix of upper and lower case letters.

Objective Function

The first section beginning with one of the keywords

```
max
maximum
maximize
min
minimum
minimize
```

defines the objective sense and the objective function, i.e.

$$c^T x + \frac{1}{2} x^T Q^o x.$$

The objective may be given a name by writing

```
myname:
```

before the expressions. 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 (`[]/2`) and are either squared or multiplied as in the examples

```
x1^2
```

and

```
x1 * x2
```

There may be zero or more pairs of brackets containing quadratic expressions.

An example of an objective section is

```
minimize
myobj: 4 x1 + x2 - 0.1 x3 + [ x1^2 + 2.1 x1 * x2 ]/2
```

Please note that the quadratic expressions are multiplied with $\frac{1}{2}$, so that the above expression means

$$\text{minimize } 4x_1 + x_2 - 0.1 \cdot x_3 + \frac{1}{2}(x_1^2 + 2.1 \cdot x_1 \cdot x_2)$$

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.

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 per line, but **MOSEK** supports defining ranged constraints by using double-colon (`::`) instead of a single-colon (`:`) after the constraint name, i.e.

$$-5 \leq x_1 + x_2 \leq 5 \tag{16.1}$$

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 an equality with a slack variable. For example the expression (16.1) may be written as

$$x_1 + x_2 - sl_1 = 0, \quad -5 \leq sl_1 \leq 5.$$

Bounds

Bounds on the variables can be specified in the bound section beginning with one of the keywords

```
bound
bounds
```

The bounds section is optional but should, if present, follow the **subject to** section. All variables listed in the bounds section must occur in either the objective or a constraint.

The default lower and upper bounds are 0 and $+\infty$. A variable may be declared free with the keyword **free**, which means that the lower bound is $-\infty$ and the upper bound is $+\infty$. Furthermore it may be assigned a finite lower and upper bound. The bound definitions for a given variable may be written in one or two lines, and bounds can be any number or $\pm\infty$ (written as **+inf/-inf/+infinity/-infinity**) as in the example

```
bounds
x1 free
x2 <= 5
0.1 <= x2
x3 = 42
2 <= x4 < +inf
```

Variable Types

The final two sections are optional and must begin with one of the keywords

```
bin
binaries
binary
```

and

```
gen
general
```

Under **general** all integer variables are listed, and under **binary** all binary (integer variables with bounds 0 and 1) are listed:

```
general
x1 x2
binary
x3 x4
```

Again, all variables listed in the binary or general sections must occur in either the objective or a constraint.

Terminating Section

Finally, an LP formatted file must be terminated with the keyword

```
end
```

16.1.2 LP File Examples

Linear example `lo1.lp`

```
\ File: lo1.lp
maximize
obj: 3 x1 + x2 + 5 x3 + x4
subject to
c1: 3 x1 + x2 + 2 x3 = 30
c2: 2 x1 + x2 + 3 x3 + x4 >= 15
c3: 2 x2 + 3 x4 <= 25
bounds
0 <= x1 <= +infinity
0 <= x2 <= 10
0 <= x3 <= +infinity
0 <= x4 <= +infinity
end
```

Mixed integer example `mil01.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
```

16.1.3 LP Format peculiarities

Comments

Anything on a line after a `\` is ignored and is treated as a comment.

Names

A name for an objective, a constraint or a variable may contain the letters `a-z`, `A-Z`, the digits `0-9` and the characters

```
!"#$%&()/,.;?@_`'|~
```

The first character in a name must not be a number, a period or the letter `e` or `E`. Keywords must not be used as names.

MOSEK accepts any character as valid for names, except `\0`. A name that is not allowed in LP file will be changed and a warning will be issued.

The algorithm for making names LP valid works as follows: The name is interpreted as an `utf-8` string. For a Unicode character `c`:

- If `c==_` (underscore), the output is `__` (two underscores).
- If `c` is a valid LP name character, the output is just `c`.
- If `c` is another character in the ASCII range, the output is `_XX`, where `XX` is the hexadecimal code for the character.
- If `c` is a character in the range `127-65535`, the output is `_uXXXX`, where `XXXX` is the hexadecimal code for the character.

- If `c` is a character above 65535, the output is `_XXXXXXXX`, where `XXXXXXXX` is the hexadecimal code for the character.

Invalid `utf-8` substrings are escaped as `_XX'`, and if a name starts with a period, `e` or `E`, that character is escaped as `_XX`.

Variable Bounds

Specifying several upper or lower bounds on one variable is possible but **MOSEK** uses only the tightest bounds. If a variable is fixed (with `=`), then it is considered the tightest bound.

MOSEK Extensions to the LP Format

Some optimization software packages employ a more strict definition of the LP format than 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.

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).

If an LP formatted file created by **MOSEK** should satisfy 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 lose their uniqueness and change the problem.

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.

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 set the parameter `iparam.write_lp_strict_format` to `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 set the parameter `iparam.write_generic_names` to `onoffkey.on` which will cause all names to be renamed systematically in the output file.

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` 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.
- `iparam.write_lp_terms_per_line` 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.

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).

16.2 The MPS File Format

MOSEK supports the standard MPS format with some extensions. For a detailed description of the MPS format see the book by Nazareth [Naz87].

16.2.1 MPS File Structure

The version of the MPS format supported by **MOSEK** allows specification of an optimization problem of the form

$$\begin{aligned} & \text{maximize/minimize} && c^T x + q_0(x) \\ & l^c \leq && Ax + q(x) \leq u^c, \\ & l^x \leq && x \leq u^x, \\ & && x \in \mathcal{K}, \\ & && x_{\mathcal{J}} \text{ integer}, \end{aligned} \tag{16.2}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \rightarrow \mathbb{R}$ is a vector of quadratic functions. Hence,

$$q_i(x) = \frac{1}{2} x^T Q^i x$$

where it is assumed that $Q^i = (Q^i)^T$. Please note the explicit $\frac{1}{2}$ in the quadratic term and that Q^i is required to be symmetric. The same applies to q_0 .

- \mathcal{K} is a convex cone.
- $\mathcal{J} \subseteq \{1, 2, \dots, n\}$ is an index set of the integer-constrained variables.
- c is the vector of objective coefficients.

An MPS file with one row and one column can be illustrated like this:

```
*          1          2          3          4          5          6
*23456789012345678901234567890123456789012345678901234567890
NAME          [name]
OBJSENSE
    [objsense]
OBJNAME          [objname]
ROWS
    ?  [cname1]
COLUMNS
    [vname1]  [cname1]  [value1]          [cname2]  [value2]
RHS
    [name]    [cname1]  [value1]          [cname2]  [value2]
RANGES
    [name]    [cname1]  [value1]          [cname2]  [value2]
QSECTION
    [vname1]  [vname2]  [value1]          [vname3]  [value2]
QMATRIX
    [vname1]  [vname2]  [value1]
```

(continues on next page)

```

QUADOBJ
  [vname1]  [vname2]  [value1]
QCMATRIX   [cname1]
  [vname1]  [vname2]  [value1]
BOUNDS
  ?? [name]  [vname1]  [value1]
CSECTION   [kname1]  [value1]      [ktype]
  [vname1]
ENDATA

```

Here the names in capitals are keywords of the MPS format and names in brackets are custom defined names or values. A couple of notes on the structure:

- Fields: All items surrounded by brackets appear in *fields*. The fields named “valueN” are numerical values. Hence, they must have the format

```
[+|-]XXXXXXX.XXXXXX[e|E][+|-]XXX]
```

where

```
X = [0|1|2|3|4|5|6|7|8|9].
```

- Sections: The MPS file consists of several sections where the names in capitals indicate the beginning of a new section. For example, COLUMNS denotes the beginning of the columns section.
- Comments: Lines starting with an * are comment lines and are ignored by **MOSEK**.
- Keys: The question marks represent keys to be specified later.
- Extensions: The sections QSECTION and CSECTION are specific **MOSEK** extensions of the MPS format. The sections QMATRIX, QUADOBJ and QCMATRIX are included for sake of compatibility with other vendors extensions to the MPS format.
- The standard MPS format is a fixed format, i.e. everything in the MPS file must be within certain fixed positions. **MOSEK** also supports a *free format*. See [Sec. 16.2.5](#) for details.

Linear example lo1.mps

A concrete example of a MPS file is presented below:

```

* File: lo1.mps
NAME          lo1
OBJSENSE
  MAX
ROWS
  N  obj
  E  c1
  G  c2
  L  c3
COLUMNS
  x1      obj      3
  x1      c1       3
  x1      c2       2
  x2      obj      1
  x2      c1       1
  x2      c2       1
  x2      c3       2
  x3      obj      5
  x3      c1       2
  x3      c2       3
  x4      obj      1
  x4      c2       1

```

(continues on next page)

(continued from previous page)

x4	c3	3
RHS		
rhs	c1	30
rhs	c2	15
rhs	c3	25
RANGES		
BOUNDS		
UP bound	x2	10
ENDATA		

Subsequently each individual section in the MPS format is discussed.

NAME (optional)

In this section a name ([name]) is assigned to the problem.

OBJSENSE (optional)

This is an optional section that can be used to specify the sense of the objective function. The OBJSENSE section contains one line at most which can be one of the following:

```
MIN
MINIMIZE
MAX
MAXIMIZE
```

It should be obvious what the implication is of each of these four lines.

OBJNAME (optional)

This is an optional section that can be used to specify the name of the row that is used as objective function. objname should be a valid row name.

ROWS

A record in the ROWS section has the form

```
? [cname1]
```

where the requirements for the fields are as follows:

Field	Starting Position	Max Width	required	Description
?	2	1	Yes	Constraint key
[cname1]	5	8	Yes	Constraint name

Hence, in this section each constraint is assigned a unique name denoted by [cname1]. Please note that [cname1] starts in position 5 and the field can be at most 8 characters wide. An initial key ? must be present to specify the type of the constraint. The key can have values E, G, L, or N with the following interpretation:

Constraint type	l_i^c	u_i^c
E (equal)	finite	$= l_i^c$
G (greater)	finite	∞
L (lower)	$-\infty$	finite
N (none)	$-\infty$	∞

In the MPS format the objective vector is not specified explicitly, but one of the constraints having the key N will be used as the objective vector c . In general, if multiple N type constraints are specified, then the first will be used as the objective vector c , unless something else was specified in the section OBJNAME.

COLUMNS

In this section the elements of A are specified using one or more records having the form:

[vname1]	[cname1]	[value1]	[cname2]	[value2]
----------	----------	----------	----------	----------

where the requirements for each field are as follows:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

Hence, a record specifies one or two elements a_{ij} of A using the principle that [vname1] and [cname1] determines j and i respectively. Please note that [cname1] must be a constraint name specified in the ROWS section. Finally, [value1] denotes the numerical value of a_{ij} . Another optional element is specified by [cname2], and [value2] for the variable specified by [vname1]. Some important comments are:

- All elements belonging to one variable must be grouped together.
- Zero elements of A should not be specified.
- At least one element for each variable should be specified.

RHS (optional)

A record in this section has the format

[name]	[cname1]	[value1]	[cname2]	[value2]
--------	----------	----------	----------	----------

where the requirements for each field are as follows:

Field	Starting Position	Max Width	required	Description
[name]	5	8	Yes	Name of the RHS vector
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

The interpretation of a record is that [name] is the name of the RHS vector to be specified. In general, several vectors can be specified. [cname1] denotes a constraint name previously specified in the ROWS section. Now, assume that this name has been assigned to the i -th constraint and v_1 denotes the value specified by [value1], then the interpretation of v_1 is:

Constraint	l_i^c	u_i^c
E	v_1	v_1
G	v_1	
L		v_1
N		

An optional second element is specified by [cname2] and [value2] and is interpreted in the same way. Please note that it is not necessary to specify zero elements, because elements are assumed to be zero.

RANGES (optional)

A record in this section has the form

[name]	[cname1]	[value1]	[cname2]	[value2]
--------	----------	----------	----------	----------

where the requirements for each fields are as follows:

Field	Starting Position	Max Width	required	Description
[name]	5	8	Yes	Name of the RANGE vector
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

The records in this section are used to modify the bound vectors for the constraints, i.e. the values in l^c and u^c . A record has the following interpretation: [name] is the name of the RANGE vector and [cname1] is a valid constraint name. Assume that [cname1] is assigned to the i -th constraint and let v_1 be the value specified by [value1], then a record has the interpretation:

Constraint type	Sign of v_1	l_i^c	u_i^c
E	—	$u_i^c + v_1$	
E	+		$l_i^c + v_1$
G	— or +		$l_i^c + v_1 $
L	— or +	$u_i^c - v_1 $	
N			

Another constraint bound can optionally be modified using [cname2] and [value2] the same way.

QSECTION (optional)

Within the QSECTION the label [cname1] must be a constraint name previously specified in the ROWS section. The label [cname1] denotes the constraint to which the quadratic terms belong. A record in the QSECTION has the form

[vname1]	[vname2]	[value1]	[vname3]	[value2]
----------	----------	----------	----------	----------

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[vname2]	15	8	Yes	Variable name
[value1]	25	12	Yes	Numerical value
[vname3]	40	8	No	Variable name
[value2]	50	12	No	Numerical value

A record specifies one or two elements in the lower triangular part of the Q^i matrix where [cname1] specifies the i . Hence, if the names [vname1] and [vname2] have been assigned to the k -th and j -th variable, then Q_{kj}^i 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

$$\begin{aligned}
 &\text{minimize} && -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) \\
 &\text{subject to} && x_1 + x_2 + x_3 \geq 1, \\
 &&& x \geq 0
 \end{aligned}$$

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

(continues on next page)

(continued from previous page)

x3	c1	1.0
RHS		
rhs	c1	1.0
QSECTION	obj	
x1	x1	2.0
x1	x3	-1.0
x2	x2	0.2
x3	x3	2.0
ENDATA		

Regarding the QSECTIONS please note that:

- Only one QSECTION is allowed for each constraint.
- The QSECTIONS can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- All entries specified in a QSECTION are assumed to belong to the lower triangular part of the quadratic term of Q .

QMATRIX/QUADOBJ (optional)

The QMATRIX and QUADOBJ sections allow to define the quadratic term of the objective function. They differ in how the quadratic term of the objective function is stored:

- QMATRIX stores all the nonzeros coefficients, without taking advantage of the symmetry of the Q matrix.
- QUADOBJ stores the upper diagonal nonzero elements of the Q matrix.

A record in both sections has the form:

[vname1]	[vname2]	[value1]
----------	----------	----------

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[vname2]	15	8	Yes	Variable name
[value1]	25	12	Yes	Numerical value

A record specifies one elements of the Q matrix in the objective function. Hence, if the names [vname1] and [vname2] have been assigned to the k -th and j -th variable, then Q_{kj} is assigned the value given by [value1]. Note that a line must appear for each off-diagonal coefficient if using a QMATRIX section, while only one entry is required in a QUADOBJ section. The quadratic part of the objective function will be evaluated as $1/2x^T Qx$.

The example

$$\begin{aligned}
 &\text{minimize} && -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) \\
 &\text{subject to} && x_1 + x_2 + x_3 \geq 1, \\
 &&& x \geq 0
 \end{aligned}$$

has the following MPS file representation using QMATRIX

* File: qo1_matrix.mps		
NAME	qo1_qmatrix	
ROWS		
N obj		
G c1		
COLUMNS		
x1	c1	1.0
x2	obj	-1.0

(continues on next page)

(continued from previous page)

	x2	c1	1.0
	x3	c1	1.0
RHS			
	rhs	c1	1.0
QMATRIX			
	x1	x1	2.0
	x1	x3	-1.0
	x3	x1	-1.0
	x2	x2	0.2
	x3	x3	2.0
ENDATA			

or the following using QUADOBJ

* File:	qo1_quadobj.mps
NAME	qo1_quadobj
ROWS	
	N obj
	G c1
COLUMNS	
	x1 c1 1.0
	x2 obj -1.0
	x2 c1 1.0
	x3 c1 1.0
RHS	
	rhs c1 1.0
QUADOBJ	
	x1 x1 2.0
	x1 x3 -1.0
	x2 x2 0.2
	x3 x3 2.0
ENDATA	

Please also note that:

- A QMATRIX/QUADOBJ section can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QMATRIX/QUADOBJ section must already be specified in the COLUMNS section.

QCMATRIX (optional)

A QCMATRIX section allows to specify the quadratic part of a given constraint. Within the QCMATRIX the label [cname1] must be a constraint name previously specified in the ROWS section. The label [cname1] denotes the constraint to which the quadratic term belongs. A record in the QSECTION has the form

[vname1]	[vname2]	[value1]
----------	----------	----------

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[vname2]	15	8	Yes	Variable name
[value1]	25	12	Yes	Numerical value

A record specifies an entry of the Q^i matrix where [cname1] specifies the i . Hence, if the names [vname1] and [vname2] have been assigned to the k -th and j -th variable, then Q_{kj}^i is assigned the value given by [value1]. Moreover, the quadratic term is represented as $1/2x^T Qx$.

The example

$$\begin{array}{ll}
\text{minimize} & x_2 \\
\text{subject to} & x_1 + x_2 + x_3 \geq 1, \\
& \frac{1}{2}(-2x_1x_3 + 0.2x_2^2 + 2x_3^2) \leq 10, \\
& x \geq 0
\end{array}$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
  N  obj
  G  c1
  L  q1
COLUMNS
  x1      c1      1.0
  x2      obj     -1.0
  x2      c1      1.0
  x3      c1      1.0
RHS
  rhs     c1      1.0
  rhs     q1     10.0
QCMATRIX  q1
  x1      x1      2.0
  x1      x3     -1.0
  x3      x1     -1.0
  x2      x2      0.2
  x3      x3      2.0
ENDATA
```

Regarding the QCMATRIXs please note that:

- Only one QCMATRIX is allowed for each constraint.
- The QCMATRIXs can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- QCMATRIX does not exploit the symmetry of Q : an off-diagonal entry (i, j) should appear twice.

BOUNDS (optional)

In the BOUNDS section changes to the default bounds vectors l^x and u^x are specified. The default bounds vectors are $l^x = 0$ and $u^x = \infty$. Moreover, it is possible to specify several sets of bound vectors. A record in this section has the form

```
?? [name]      [vname1]      [value1]
```

where the requirements for each field are:

Field	Starting Position	Max Width	Required	Description
??	2	2	Yes	Bound key
[name]	5	8	Yes	Name of the BOUNDS vector
[vname1]	15	8	Yes	Variable name
[value1]	25	12	No	Numerical value

Hence, a record in the BOUNDS section has the following interpretation: [name] is the name of the bound vector and [vname1] is the name of the variable for which the bounds are modified by the record. ?? and [value1] are used to modify the bound vectors according to the following table:

??	l_j^x	u_j^x	Made integer (added to \mathcal{J})
FR	$-\infty$	∞	No
FX	v_1	v_1	No
LO	v_1	unchanged	No
MI	$-\infty$	unchanged	No
PL	unchanged	∞	No
UP	unchanged	v_1	No
BV	0	1	Yes
LI	$\lceil v_1 \rceil$	unchanged	Yes
UI	unchanged	$\lfloor v_1 \rfloor$	Yes

Here v_1 is the value specified by `[value1]`.

CSECTION (optional)

The purpose of the CSECTION is to specify the conic constraint

$$x \in \mathcal{K}$$

in (16.2). It is assumed that \mathcal{K} satisfies the following requirements. Let

$$x^t \in \mathbb{R}^{n^t}, \quad t = 1, \dots, k$$

be vectors comprised of parts of the decision variables x so that each decision variable is a member of exactly **one** vector x^t , for example

$$x^1 = \begin{bmatrix} x_1 \\ x_4 \\ x_7 \end{bmatrix} \quad \text{and} \quad x^2 = \begin{bmatrix} x_6 \\ x_5 \\ x_3 \\ x_2 \end{bmatrix}.$$

Next define

$$\mathcal{K} := \{x \in \mathbb{R}^n : x^t \in \mathcal{K}_t, \quad t = 1, \dots, k\}$$

where \mathcal{K}_t must have one of the following forms:

- \mathbb{R} set:

$$\mathcal{K}_t = \mathbb{R}^{n^t}.$$

- Zero cone:

$$\mathcal{K}_t = \{0\} \subseteq \mathbb{R}^{n^t}. \quad (16.3)$$

- Quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \quad (16.4)$$

- Rotated quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2, \quad x_1, x_2 \geq 0 \right\}. \quad (16.5)$$

- Primal exponential cone:

$$\mathcal{K}_t = \{x \in \mathbb{R}^3 : x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0\}. \quad (16.6)$$

- Primal power cone (with parameter $0 < \alpha < 1$):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1^\alpha x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \quad (16.7)$$

- Dual exponential cone:

$$\mathcal{K}_t = \{x \in \mathbb{R}^3 : x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0\}. \quad (16.8)$$

- Dual power cone (with parameter $0 < \alpha < 1$):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : \left(\frac{x_1}{\alpha} \right)^\alpha \left(\frac{x_2}{1-\alpha} \right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \quad (16.9)$$

In general, membership in the \mathbb{R} set is not specified. If a variable is not a member of any other cone then it is assumed to be a member of the \mathbb{R} cone.

Next, let us study an example. Assume that the power cone

$$x_4^{1/3} x_5^{2/3} \geq |x_8|$$

and the rotated quadratic cone

$$2x_3x_7 \geq x_1^2 + x_0^2, \quad x_3, x_7 \geq 0,$$

should be specified in the MPS file. One CSECTION is required for each cone and they are specified as follows:

*	1	2	3	4	5	6
*23456789012345678901234567890123456789012345678901234567890						
CSECTION	konea	3e-1		PPOW		
x4						
x5						
x8						
CSECTION	koneb	0.0		RQUAD		
x7						
x3						
x1						
x0						

In general, a CSECTION header has the format

CSECTION	[kname1]	[value1]	[ktype]
----------	----------	----------	---------

where the requirements for each field are as follows:

Field	Starting Position	Max Width	Required	Description
[kname1]	15	8	Yes	Name of the cone
[value1]	25	12	No	Cone parameter
[ktype]	40		Yes	Type of the cone.

The possible cone type keys are:

[ktype]	Members	[value1]	Interpretation.
ZERO	≥ 0	unused	Zero cone (16.3).
QUAD	≥ 1	unused	Quadratic cone (16.4).
RQUAD	≥ 2	unused	Rotated quadratic cone (16.5).
PEXP	3	unused	Primal exponential cone (16.6).
PPOW	≥ 2	α	Primal power cone (16.7).
DEXP	3	unused	Dual exponential cone (16.8).
DPOW	≥ 2	α	Dual power cone (16.9).

A record in the CSECTION has the format

[vname1]

where the requirements for each field are

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	A valid variable name

A variable must occur in at most one CSECTION.

ENDATA

This keyword denotes the end of the MPS file.

16.2.2 Integer Variables

Using special bound keys in the **BOUNDS** section it is possible to specify that some or all of the variables should be integer-constrained i.e. be members of \mathcal{J} . However, an alternative method is available. This method is available only for backward compatibility and we recommend that it is not used. This method requires that markers are placed in the **COLUMNS** section as in the example:

```
COLUMNS
x1      obj      -10.0      c1      0.7
x1      c2        0.5      c3      1.0
x1      c4        0.1
* Start of integer-constrained variables.
MARK000 'MARKER'          'INTORG'
x2      obj      -9.0      c1      1.0
x2      c2        0.8333333333 c3      0.66666667
x2      c4        0.25
x3      obj      1.0      c6      2.0
MARK001 'MARKER'          'INTEND'
* End of integer-constrained variables.
```

Please note that special marker lines are used to indicate the start and the end of the integer variables. Furthermore be aware of the following

- All variables between the markers are assigned a default lower bound of 0 and a default upper bound of 1. **This may not be what is intended.** If it is not intended, the correct bounds should be defined in the **BOUNDS** section of the MPS formatted file.
- **MOSEK** ignores field 1, i.e. MARK0001 and MARK001, however, other optimization systems require them.
- Field 2, i.e. **MARKER**, must be specified including the single quotes. This implies that no row can be assigned the name **MARKER**.
- Field 3 is ignored and should be left blank.
- Field 4, i.e. **INTORG** and **INTEND**, must be specified.
- It is possible to specify several such integer marker sections within the **COLUMNS** section.

16.2.3 General Limitations

- An MPS file should be an ASCII file.

16.2.4 Interpretation of the MPS Format

Several issues related to the MPS format are not well-defined by the industry standard. However, **MOSEK** uses the following interpretation:

- If a matrix element in the **COLUMNS** section is specified multiple times, then the multiple entries are added together.
- If a matrix element in a **QSECTION** section is specified multiple times, then the multiple entries are added together.

16.2.5 The Free MPS Format

MOSEK supports a free format variation of the MPS format. The free format is similar to the MPS file format but less restrictive, e.g. it allows longer names. However, a name must not contain any blanks.

Moreover, by default a line in the MPS file must not contain more than 1024 characters. By modifying the parameter `iparam.read_mps_width` an arbitrary large line width will be accepted.

The free MPS format is default. To change to the strict and other formats use the parameter `iparam.read_mps_format`.

16.3 The OPF Format

The *Optimization Problem Format (OPF)* is an alternative to LP and MPS files for specifying optimization problems. It is row-oriented, inspired by the CPLEX LP format.

Apart from containing objective, constraints, bounds etc. it may contain complete or partial solutions, comments and extra information relevant for solving the problem. It is designed to be easily read and modified by hand and to be forward compatible with possible future extensions.

Intended use

The OPF file format is meant to replace several other files:

- The LP file format: Any problem that can be written as an LP file can be written as an OPF file too; furthermore it naturally accommodates ranged constraints and variables as well as arbitrary characters in names, fixed expressions in the objective, empty constraints, and conic constraints.
- Parameter files: It is possible to specify integer, double and string parameters along with the problem (or in a separate OPF file).
- Solution files: It is possible to store a full or a partial solution in an OPF file and later reload it.

16.3.1 The File Format

The format uses tags to structure data. A simple example with the basic sections may look like this:

```
[comment]
This is a comment. You may write almost anything here...
[/comment]

# This is a single-line comment.

[objective min 'myobj']
x + 3 y + x^2 + 3 y^2 + z + 1
[/objective]

[constraints]
[con 'con01'] 4 <= x + y  [/con]
[/constraints]

[bounds]
[b] -10 <= x,y <= 10  [/b]

[cone quad] x,y,z [/cone]
[/bounds]
```

A scope is opened by a tag of the form `[tag]` and closed by a tag of the form `[/tag]`. An opening tag may accept a list of unnamed and named arguments, for examples:

```
[tag value] tag with one unnamed argument [/tag]
[tag arg=value] tag with one named argument [/tag]
```

Unnamed arguments are identified by their order, while named arguments may appear in any order, but never before an unnamed argument. The `value` can be a quoted, single-quoted or double-quoted text string, i.e.

```
[tag 'value']      single-quoted value [/tag]
[tag arg='value']  single-quoted value [/tag]
[tag "value"]      double-quoted value [/tag]
[tag arg="value"]  double-quoted value [/tag]
```

16.3.2 Sections

The recognized tags are

`[comment]`

A comment section. This can contain *almost* any text: Between single quotes (') or double quotes (") any text may appear. Outside quotes the markup characters ([and]) must be prefixed by backslashes. Both single and double quotes may appear alone or inside a pair of quotes if it is prefixed by a backslash.

`[objective]`

The objective function: This accepts one or two parameters, where the first one (in the above example `min`) is either `min` or `max` (regardless of case) and defines the objective sense, and the second one (above `myobj`), if present, is the objective name. The section may contain linear and quadratic expressions.

If several objectives are specified, all but the last are ignored.

`[constraints]`

This does not directly contain any data, but may contain subsections `con` defining a linear constraint.

`[con]`

Defines a single constraint; if an argument is present (`[con NAME]`) this is used as the name of the constraint, otherwise it is given a null-name. The section contains a constraint definition written as linear and quadratic expressions with a lower bound, an upper bound, with both or with an equality. Examples:

```
[constraints]
[con 'con1'] 0 <= x + y      [/con]
[con 'con2'] 0 >= x + y      [/con]
[con 'con3'] 0 <= x + y <= 10 [/con]
[con 'con4']      x + y = 10 [/con]
[/constraints]
```

Constraint names are unique. If a constraint is specified which has the same name as a previously defined constraint, the new constraint replaces the existing one.

`[bounds]`

This does not directly contain any data, but may contain subsections `b` (linear bounds on variables) and `cone` (cones).

`[b]`

Bound definition on one or several variables separated by comma (,). An upper or lower bound on a variable replaces any earlier defined bound on that variable. If only one bound (upper or lower) is given only this bound is replaced. This means that upper and lower bounds can be specified separately. So the OPF bound definition:

```
[b] x,y >= -10 [/b]
[b] x,y <= 10  [/b]
```

results in the bound $-10 \leq x, y \leq 10$.

[cone]

Specifies a cone. A cone is defined as a sequence of variables which belong to a single unique cone. The supported cone types are:

- **quad**: a quadratic cone of n variables x_1, \dots, x_n defines a constraint of the form

$$x_1^2 \geq \sum_{i=2}^n x_i^2, \quad x_1 \geq 0.$$

- **rquad**: a rotated quadratic cone of n variables x_1, \dots, x_n defines a constraint of the form

$$2x_1x_2 \geq \sum_{i=3}^n x_i^2, \quad x_1, x_2 \geq 0.$$

- **pexp**: primal exponential cone of 3 variables x_1, x_2, x_3 defines a constraint of the form

$$x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0.$$

- **ppow** with parameter $0 < \alpha < 1$: primal power cone of n variables x_1, \dots, x_n defines a constraint of the form

$$x_1^\alpha x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^n x_j^2}, \quad x_1, x_2 \geq 0.$$

- **dexp**: dual exponential cone of 3 variables x_1, x_2, x_3 defines a constraint of the form

$$x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0.$$

- **dpow** with parameter $0 < \alpha < 1$: dual power cone of n variables x_1, \dots, x_n defines a constraint of the form

$$\left(\frac{x_1}{\alpha}\right)^\alpha \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^n x_j^2}, \quad x_1, x_2 \geq 0.$$

- **zero**: zero cone of n variables x_1, \dots, x_n defines a constraint of the form

$$x_1 = \dots = x_n = 0$$

A [bounds]-section example:

```
[bounds]
[b]  0 <= x,y <= 10  [/b] # ranged bound
[b]  10 >= x,y >=  0  [/b] # ranged bound
[b]  0 <= x,y <= inf  [/b] # using inf
[b]      x,y free    [/b] # free variables
# Let (x,y,z,w) belong to the cone K
[cone rquad] x,y,z,w [/cone] # rotated quadratic cone
[cone ppow '3e-01' 'a'] x1, x2, x3 [/cone] # power cone with alpha=1/3 and name 'a'
[/bounds]
```

By default all variables are free.

[variables]

This defines an ordering of variables as they should appear in the problem. This is simply a space-separated list of variable names.

[integer]

This contains a space-separated list of variables and defines the constraint that the listed variables must be integer-valued.

[hints]

This may contain only non-essential data; for example estimates of the number of variables, constraints and non-zeros. Placed before all other sections containing data this may reduce the time spent reading the file.

In the `hints` section, any subsection which is not recognized by **MOSEK** is simply ignored. In this section a hint is defined as follows:

```
[hint ITEM] value [/hint]
```

The hints recognized by **MOSEK** are:

- `numvar` (number of variables),
- `numcon` (number of linear/quadratic constraints),
- `numanz` (number of linear non-zeros in constraints),
- `numqnz` (number of quadratic non-zeros in constraints).

[solutions]

This section can contain a set of full or partial solutions to a problem. Each solution must be specified using a `[solution]`-section, i.e.

```
[solutions]
[solution]...[/solution] #solution 1
[solution]...[/solution] #solution 2
#other solutions....
[solution]...[/solution] #solution n
[/solutions]
```

The syntax of a `[solution]`-section is the following:

```
[solution SOLTYPE status=STATUS]...[/solution]
```

where `SOLTYPE` is one of the strings

- `interior`, a non-basic solution,
- `basic`, a basic solution,
- `integer`, an integer solution,

and `STATUS` is one of the strings

- `UNKNOWN`,
- `OPTIMAL`,
- `INTEGER_OPTIMAL`,
- `PRIM_FEAS`,
- `DUAL_FEAS`,
- `PRIM_AND_DUAL_FEAS`,
- `NEAR_OPTIMAL`,
- `NEAR_PRIM_FEAS`,

- NEAR_DUAL_FEAS,
- NEAR_PRIM_AND_DUAL_FEAS,
- PRIM_INFEAS_CER,
- DUAL_INFEAS_CER,
- NEAR_PRIM_INFEAS_CER,
- NEAR_DUAL_INFEAS_CER,
- NEAR_INTEGER_OPTIMAL.

Most of these values are irrelevant for input solutions; when constructing a solution for simplex hot-start or an initial solution for a mixed integer problem the safe setting is `UNKNOWN`.

A `[solution]`-section contains `[con]` and `[var]` sections. Each `[con]` and `[var]` section defines solution information for a single variable or constraint, specified as list of `KEYWORD/value` pairs, in any order, written as

```
KEYWORD=value
```

Allowed keywords are as follows:

- `sk`. The status of the item, where the `value` is one of the following strings:
 - `LOW`, the item is on its lower bound.
 - `UPR`, the item is on its upper bound.
 - `FIX`, it is a fixed item.
 - `BAS`, the item is in the basis.
 - `SUPBAS`, the item is super basic.
 - `UNK`, the status is unknown.
 - `INF`, the item is outside its bounds (infeasible).
- `lv1` Defines the level of the item.
- `s1` 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`, `lv1`, `s1` and `su`. Items `s1` and `su` are not required for `integer` solutions.

A `[con]` section should always contain `sk`, `lv1`, `s1`, `su` and `y`.

An example of a solution section

```
[solution basic status=UNKNOWN]
[var x0] sk=LOW    lv1=5.0      [/var]
[var x1] sk=UPR    lv1=10.0     [/var]
[var x2] sk=SUPBAS lv1=2.0    s1=1.5 su=0.0 [/var]

[con c0] sk=LOW    lv1=3.0 y=0.0 [/con]
[con c0] sk=UPR    lv1=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.

16.3.3 Numbers

Numbers, when used for parameter values or coefficients, are written in the usual way by the `printf` function. That is, they may be prefixed by a sign (+ or -) and may contain an integer part, decimal part and an exponent. The decimal point is always . (a dot). Some examples are

```
1
1.0
.0
1.
1e10
1e+10
1e-10
```

Some *invalid* examples are

```
e10 # invalid, must contain either integer or decimal part
. # invalid
.e10 # invalid
```

More formally, the following standard regular expression describes numbers as used:

```
[+|-]?([0-9]+|[.][0-9]*|.[0-9]+)([eE][+|-]?[0-9]+)?
```

16.3.4 Names

Variable names, constraint names and objective name may contain arbitrary characters, which in some cases must be enclosed by quotes (single or double) that in turn must be preceded by a backslash. Unquoted names must begin with a letter (a-z or A-Z) and contain only the following characters: the letters a-z and A-Z, the digits 0-9, braces { and } and underscore (_).

Some examples of legal names:

```
an_unquoted_name
another_name{123}
'single quoted name'
"double quoted name"
"name with \"quote\" in it"
"name with []s in it"
```

16.3.5 Parameters Section

In the `vendor` section solver parameters are defined inside the `parameters` subsection. Each parameter is written as

```
[p PARAMETER_NAME] value [/p]
```

where `PARAMETER_NAME` is replaced by a **MOSEK** parameter name, usually of the form `MSK_IPAR_...`, `MSK_DPAR_...` or `MSK_SPAR_...`, and the `value` is replaced by the value of that parameter; both integer values and named values may be used. Some simple examples are

```
[vendor mosek]
[parameters]
[p MSK_IPAR_OPF_MAX_TERMS_PER_LINE] 10 [/p]
[p MSK_IPAR_OPF_WRITE_PARAMETERS] MSK_ON [/p]
[p MSK_DPAR_DATA_TOL_BOUND_INF] 1.0e18 [/p]
[/parameters]
[/vendor]
```

16.3.6 Writing OPF Files from MOSEK

To write an OPF file then make sure the file extension is `.opf`.

Then modify the following parameters to define what the file should contain:

<i>iparam.opf_write_sol_bas</i>	Include basic solution, if defined.
<i>iparam.opf_write_sol_itg</i>	Include integer solution, if defined.
<i>iparam.opf_write_sol_itr</i>	Include interior solution, if defined.
<i>iparam.opf_write_solutions</i>	Include solutions if they are defined. If this is off, no solutions are included.
<i>iparam.opf_write_header</i>	Include a small header with comments.
<i>iparam.opf_write_problem</i>	Include the problem itself — objective, constraints and bounds.
<i>iparam.opf_write_parameters</i>	Include all parameter settings.
<i>iparam.opf_write_hints</i>	Include hints about the size of the problem.

16.3.7 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

Linear Example lo1.opf

Consider the example:

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

having the bounds

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

In the OPF format the example is displayed as shown in [Listing 16.1](#).

Listing 16.1: Example of an OPF file for a linear problem.

```

[comment]
  The lo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 4 [/hint]
  [hint NUMCON] 3 [/hint]
  [hint NUMANZ] 9 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4
[/variables]

[objective maximize 'obj']
  3 x1 + x2 + 5 x3 + x4
[/objective]

[constraints]
  [con 'c1'] 3 x1 +   x2 + 2 x3           = 30 [/con]
  [con 'c2'] 2 x1 +   x2 + 3 x3 +   x4 >= 15 [/con]
  [con 'c3']      2 x2           + 3 x4 <= 25 [/con]
[/constraints]

```

(continues on next page)

```
[bounds]
  [b] 0 <= * [/b]
  [b] 0 <= x2 <= 10 [/b]
[/bounds]
```

Quadratic Example qo1.opf

An example of a quadratic optimization problem is

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

This can be formulated in `opf` as shown below.

Listing 16.2: Example of an OPF file for a quadratic problem.

```
[comment]
  The qo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 3 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
  [hint NUMQNZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3
[/variables]

[objective minimize 'obj']
  # The quadratic terms are often written with a factor of 1/2 as here,
  # but this is not required.

  - x2 + 0.5 ( 2.0 x1 ^ 2 - 2.0 x3 * x1 + 0.2 x2 ^ 2 + 2.0 x3 ^ 2 )
[/objective]

[constraints]
  [con 'c1'] 1.0 <= x1 + x2 + x3 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]
```

Conic Quadratic Example cqo1.opf

Consider the example:

$$\begin{aligned} & \text{minimize} && x_3 + x_4 + x_5 \\ & \text{subject to} && x_0 + x_1 + 2x_2 = 1, \\ & && x_0, x_1, x_2 \geq 0, \\ & && x_3 \geq \sqrt{x_0^2 + x_1^2}, \\ & && 2x_4x_5 \geq x_2^2. \end{aligned}$$

Please note that the type of the cones is defined by the parameter to `[cone ...]`; the content of the cone-section is the names of variables that belong to the cone. The resulting OPF file is in [Listing 16.3](#).

Listing 16.3: Example of an OPF file for a conic quadratic problem.

```
[comment]
  The cqo1 example in OPF format.
[/comment]

[hints]
  [hint NUMVAR] 6 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4 x5 x6
[/variables]

[objective minimize 'obj']
  x4 + x5 + x6
[/objective]

[constraints]
  [con 'c1'] x1 + x2 + 2e+00 x3 = 1e+00 [/con]
[/constraints]

[bounds]
  # We let all variables default to the positive orthant
  [b] 0 <= * [/b]

  # ...and change those that differ from the default
  [b] x4,x5,x6 free [/b]

  # Define quadratic cone:  $x_4 \geq \sqrt{x_1^2 + x_2^2}$ 
  [cone quad 'k1'] x4, x1, x2 [/cone]

  # Define rotated quadratic cone:  $2 x_5 x_6 \geq x_3^2$ 
  [cone rquad 'k2'] x5, x6, x3 [/cone]
[/bounds]
```

Mixed Integer Example milo1.opf

Consider the mixed integer problem:

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

This can be implemented in OPF with the file in [Listing 16.4](#).

Listing 16.4: Example of an OPF file for a mixed-integer linear problem.

```
[comment]
  The milo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 2 [/hint]
  [hint NUMCON] 2 [/hint]
  [hint NUMANZ] 4 [/hint]
[/hints]
```

(continues on next page)

```

[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]

```

16.4 The CBF Format

This document constitutes the technical reference manual of the *Conic Benchmark Format* with file extension: `.cbf` or `.CBF`. It unifies linear, second-order cone (also known as conic quadratic) and semidefinite optimization with mixed-integer variables. The format has been designed with benchmark libraries in mind, and therefore focuses on compact and easily parsable representations. The problem structure is separated from the problem data, and the format moreover facilitates benchmarking of hotstart capability through sequences of changes.

16.4.1 How Instances Are Specified

This section defines the spectrum of conic optimization problems that can be formulated in terms of the keywords of the CBF format.

In the CBF format, conic optimization problems are considered in the following form:

$$\begin{aligned}
 & \min / \max && g^{obj} \\
 \text{s.t.} &&& g_i \in \mathcal{K}_i, \quad i \in \mathcal{I}, \\
 &&& G_i \in \mathcal{K}_i, \quad i \in \mathcal{I}^{PSD}, \\
 &&& x_j \in \mathcal{K}_j, \quad j \in \mathcal{J}, \\
 &&& \overline{X}_j \in \mathcal{K}_j, \quad j \in \mathcal{J}^{PSD}.
 \end{aligned} \tag{16.10}$$

- **Variables** are either scalar variables, x_j for $j \in \mathcal{J}$, or variables, \overline{X}_j for $j \in \mathcal{J}^{PSD}$. Scalar variables can also be declared as integer.
- **Constraints** are affine expressions of the variables, either scalar-valued g_i for $i \in \mathcal{I}$, or matrix-valued G_i for $i \in \mathcal{I}^{PSD}$

$$\begin{aligned}
 g_i &= \sum_{j \in \mathcal{J}^{PSD}} \langle F_{ij}, X_j \rangle + \sum_{j \in \mathcal{J}} a_{ij} x_j + b_i, \\
 G_i &= \sum_{j \in \mathcal{J}} x_j H_{ij} + D_i.
 \end{aligned}$$

- The **objective function** is a scalar-valued affine expression of the variables, either to be minimized or maximized. We refer to this expression as g^{obj}

$$g^{obj} = \sum_{j \in \mathcal{J}^{PSD}} \langle F_j^{obj}, X_j \rangle + \sum_{j \in \mathcal{J}} a_j^{obj} x_j + b^{obj}.$$

CBF format can represent the following cones \mathcal{K} :

- **Free domain** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n\}, \text{ for } n \geq 1.$$

- **Positive orthant** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j \geq 0 \text{ for } j = 1, \dots, n\}, \text{ for } n \geq 1.$$

- **Negative orthant** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j \leq 0 \text{ for } j = 1, \dots, n\}, \text{ for } n \geq 1.$$

- **Fixpoint zero** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j = 0 \text{ for } j = 1, \dots, n\}, \text{ for } n \geq 1.$$

- **Quadratic cone** - A cone in the second-order cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R}^{n-1}, p^2 \geq x^T x, p \geq 0 \right\}, \text{ for } n \geq 2.$$

- **Rotated quadratic cone** - A cone in the second-order cone family defined by

$$\left\{ \begin{pmatrix} p \\ q \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}^{n-2}, 2pq \geq x^T x, p \geq 0, q \geq 0 \right\}, \text{ for } n \geq 3.$$

16.4.2 The Structure of CBF Files

This section defines how information is written in the CBF format, without being specific about the type of information being communicated.

All information items belong to exactly one of the three groups of information. These information groups, and the order they must appear in, are:

1. File format.
2. Problem structure.
3. Problem data.

The first group, file format, provides information on how to interpret the file. The second group, problem structure, provides the information needed to deduce the type and size of the problem instance. Finally, the third group, problem data, specifies the coefficients and constants of the problem instance.

Information items

The format is composed as a list of information items. The first line of an information item is the **KEYWORD**, revealing the type of information provided. The second line - of some keywords only - is the **HEADER**, typically revealing the size of information that follows. The remaining lines are the **BODY** holding the actual information to be specified.

KEYWORD BODY
KEYWORD HEADER BODY

The **KEYWORD** determines how each line in the **HEADER** and **BODY** is structured. Moreover, the number of lines in the **BODY** follows either from the **KEYWORD**, the **HEADER**, or from another information item required to precede it.

Embedded hotstart-sequences

A sequence of problem instances, based on the same problem structure, is within a single file. This is facilitated via the **CHANGE** within the problem data information group, as a separator between the information items of each instance. The information items following a **CHANGE** keyword are appending to, or changing (e.g., setting coefficients back to their default value of zero), the problem data of the preceding instance.

The sequence is intended for benchmarking of hotstart capability, where the solvers can reuse their internal state and solution (subject to the achieved accuracy) as warmpoint for the succeeding instance. Whenever this feature is unsupported or undesired, the keyword **CHANGE** should be interpreted as the end of file.

File encoding and line width restrictions

The format is based on the US-ASCII printable character set with two extensions as listed below. Note, by definition, that none of these extensions can be misinterpreted as printable US-ASCII characters:

- A line feed marks the end of a line, carriage returns are ignored.
- Comment-lines may contain unicode characters in UTF-8 encoding.

The line width is restricted to 512 bytes, with 3 bytes reserved for the potential carriage return, line feed and null-terminator.

Integers and floating point numbers must follow the ISO C decimal string representation in the standard C locale. The format does not impose restrictions on the magnitude of, or number of significant digits in numeric data, but the use of 64-bit integers and 64-bit IEEE 754 floating point numbers should be sufficient to avoid loss of precision.

Comment-line and whitespace rules

The format allows single-line comments respecting the following rule:

- Lines having first byte equal to '#' (US-ASCII 35) are comments, and should be ignored. Comments are only allowed between information items.

Given that a line is not a comment-line, whitespace characters should be handled according to the following rules:

- Leading and trailing whitespace characters should be ignored.
 - The separator between multiple pieces of information on one line, is either one or more whitespace characters.
- Lines containing only whitespace characters are empty, and should be ignored. Empty lines are only allowed between information items.

16.4.3 Problem Specification

The problem structure

The problem structure defines the objective sense, whether it is minimization and maximization. It also defines the index sets, \mathcal{J} , \mathcal{J}^{PSD} , \mathcal{I} and \mathcal{I}^{PSD} , which are all numbered from zero, $\{0, 1, \dots\}$, and empty until explicitly constructed.

- **Scalar variables** are constructed in vectors restricted to a conic domain, such as $(x_0, x_1) \in \mathbb{R}_+^2$, $(x_2, x_3, x_4) \in \mathcal{Q}^3$, etc. In terms of the Cartesian product, this generalizes to

$$x \in \mathcal{K}_1^{n_1} \times \mathcal{K}_2^{n_2} \times \dots \times \mathcal{K}_k^{n_k}$$

which in the CBF format becomes:

```
VAR
n k
K1 n1
K2 n2
...
Kk nk
```

where $\sum_i n_i = n$ is the total number of scalar variables. The list of supported cones is found in [Table 16.3](#). Integrality of scalar variables can be specified afterwards.

- **PSD variables** are constructed one-by-one. That is, $X_j \succeq \mathbf{0}^{n_j \times n_j}$ for $j \in \mathcal{J}^{PSD}$, constructs a matrix-valued variable of size $n_j \times n_j$ restricted to be symmetric positive semidefinite. In the CBF format, this list of constructions becomes:

```
PSDVAR
N
n1
n2
...
nN
```

where N is the total number of PSD variables.

- **Scalar constraints** are constructed in vectors restricted to a conic domain, such as $(g_0, g_1) \in \mathbb{R}_+^2$, $(g_2, g_3, g_4) \in \mathcal{Q}^3$, etc. In terms of the Cartesian product, this generalizes to

$$g \in \mathcal{K}_1^{m_1} \times \mathcal{K}_2^{m_2} \times \dots \times \mathcal{K}_k^{m_k}$$

which in the CBF format becomes:

```
CON
m k
K1 m1
K2 m2
..
Kk mk
```

where $\sum_i m_i = m$ is the total number of scalar constraints. The list of supported cones is found in [Table 16.3](#).

- **PSD constraints** are constructed one-by-one. That is, $G_i \succeq \mathbf{0}^{m_i \times m_i}$ for $i \in \mathcal{I}^{PSD}$, constructs a matrix-valued affine expressions of size $m_i \times m_i$ restricted to be symmetric positive semidefinite. In the CBF format, this list of constructions becomes

```
PSDCON
M
m1
m2
..
mM
```

where M is the total number of PSD constraints.

With the objective sense, variables (with integer indications) and constraints, the definitions of the many affine expressions follow in problem data.

Problem data

The problem data defines the coefficients and constants of the affine expressions of the problem instance. These are considered zero until explicitly defined, implying that instances with no keywords from this information group are, in fact, valid. Duplicating or conflicting information is a failure to comply with the standard. Consequently, two coefficients written to the same position in a matrix (or to transposed positions in a symmetric matrix) is an error.

The affine expressions of the objective, g^{obj} , of the scalar constraints, g_i , and of the PSD constraints, G_i , are defined separately. The following notation uses the standard trace inner product for matrices, $\langle X, Y \rangle = \sum_{i,j} X_{ij}Y_{ij}$.

- The affine expression of the objective is defined as

$$g^{obj} = \sum_{j \in \mathcal{J}^{PSD}} \langle F_j^{obj}, X_j \rangle + \sum_{j \in \mathcal{J}} a_j^{obj} x_j + b^{obj},$$

in terms of the symmetric matrices, F_j^{obj} , and scalars, a_j^{obj} and b^{obj} .

- The affine expressions of the scalar constraints are defined, for $i \in \mathcal{I}$, as

$$g_i = \sum_{j \in \mathcal{J}^{PSD}} \langle F_{ij}, X_j \rangle + \sum_{j \in \mathcal{J}} a_{ij} x_j + b_i,$$

in terms of the symmetric matrices, F_{ij} , and scalars, a_{ij} and b_i .

- The affine expressions of the PSD constraints are defined, for $i \in \mathcal{I}^{PSD}$, as

$$G_i = \sum_{j \in \mathcal{J}} x_j H_{ij} + D_i,$$

in terms of the symmetric matrices, H_{ij} and D_i .

List of cones

The format uses an explicit syntax for symmetric positive semidefinite cones as shown above. For scalar variables and constraints, constructed in vectors, the supported conic domains and their minimum sizes are given as follows.

Table 16.3: Cones available in the CBF format

Name	CBF keyword	Cone family
Free domain	F	linear
Positive orthant	L+	linear
Negative orthant	L-	linear
Fixpoint zero	L=	linear
Quadratic cone	Q	second-order
Rotated quadratic cone	QR	second-order

16.4.4 File Format Keywords

VER

Description: The version of the Conic Benchmark Format used to write the file.

HEADER: None

BODY: One line formatted as:

INT

This is the version number.
Must appear exactly once in a file, as the first keyword.

OBJSENSE

Description: Define the objective sense.

HEADER: None

BODY: One line formatted as:

STR

having MIN indicates minimize, and MAX indicates maximize. Capital letters are required.
Must appear exactly once in a file.

PSDVAR

Description: Construct the PSD variables.

HEADER: One line formatted as:

INT

This is the number of PSD variables in the problem.
BODY: A list of lines formatted as:

INT

This indicates the number of rows (equal to the number of columns) in the matrix-valued PSD variable. The number of lines should match the number stated in the header.

VAR

Description: Construct the scalar variables.

HEADER: One line formatted as:

INT INT

This is the number of scalar variables, followed by the number of conic domains they are restricted to.

BODY: A list of lines formatted as:

STR INT

This indicates the cone name (see [Table 16.3](#)), and the number of scalar variables restricted to this cone. These numbers should add up to the number of scalar variables stated first in the header. The number of lines should match the second number stated in the header.

INT

Description: Declare integer requirements on a selected subset of scalar variables.

HEADER: one line formatted as:

INT

This is the number of integer scalar variables in the problem.
BODY: a list of lines formatted as:

INT

This indicates the scalar variable index $j \in \mathcal{J}$. The number of lines should match the number stated in the header.

Can only be used after the keyword VAR.

PSDCON

Description: Construct the PSD constraints.

HEADER: One line formatted as:

INT

This is the number of PSD constraints in the problem.

BODY: A list of lines formatted as:

INT

This indicates the number of rows (equal to the number of columns) in the matrix-valued affine expression of the PSD constraint. The number of lines should match the number stated in the header.

Can only be used after these keywords: PSDVAR, VAR.

CON

Description: Construct the scalar constraints.

HEADER: One line formatted as:

INT INT

This is the number of scalar constraints, followed by the number of conic domains they restrict to.

BODY: A list of lines formatted as:

STR INT

This indicates the cone name (see Table 16.3), and the number of affine expressions restricted to this cone. These numbers should add up to the number of scalar constraints stated first in the header. The number of lines should match the second number stated in the header.

Can only be used after these keywords: PSDVAR, VAR

OBJFCOORD

Description: Input sparse coordinates (quadruplets) to define the symmetric matrices F_j^{obj} , as used in the objective.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT INT INT REAL

This indicates the PSD variable index $j \in \mathcal{J}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

OBJACOORD

Description: Input sparse coordinates (pairs) to define the scalars, a_j^{obj} , as used in the objective.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT REAL

This indicates the scalar variable index $j \in \mathcal{J}$ and the coefficient value. The number of lines should match the number stated in the header.

OBJCOORD

Description: Input the scalar, b^{obj} , as used in the objective.

HEADER: None.

BODY: One line formatted as:

REAL

This indicates the coefficient value.

FCOORD

Description: Input sparse coordinates (quintuplets) to define the symmetric matrices, F_{ij} , as used in the scalar constraints.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT INT INT INT REAL

This indicates the scalar constraint index $i \in \mathcal{I}$, the PSD variable index $j \in \mathcal{J}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

ACCOORD

Description: Input sparse coordinates (triplets) to define the scalars, a_{ij} , as used in the scalar constraints.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT INT REAL

This indicates the scalar constraint index $i \in \mathcal{I}$, the scalar variable index $j \in \mathcal{J}$ and the coefficient value. The number of lines should match the number stated in the header.

BCOORD

Description: Input sparse coordinates (pairs) to define the scalars, b_i , as used in the scalar constraints.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT REAL

This indicates the scalar constraint index $i \in \mathcal{I}$ and the coefficient value. The number of lines should match the number stated in the header.

HCOORD

Description: Input sparse coordinates (quintuplets) to define the symmetric matrices, H_{ij} , as used in the PSD constraints.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as

INT INT INT INT REAL

This indicates the PSD constraint index $i \in \mathcal{I}^{PSD}$, the scalar variable index $j \in \mathcal{J}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

DCOORD

Description: Input sparse coordinates (quadruplets) to define the symmetric matrices, D_i , as used in the PSD constraints.

HEADER: One line formatted as

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT INT INT REAL

This indicates the PSD constraint index $i \in \mathcal{I}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

CHANGE

Start of a new instance specification based on changes to the previous. Can be interpreted as the end of file when the hotstart-sequence is unsupported or undesired.

BODY: None

Header: None

16.4.5 CBF Format Examples

Minimal Working Example

The conic optimization problem (16.11), has three variables in a quadratic cone - first one is integer - and an affine expression in domain 0 (equality constraint).

$$\begin{aligned} &\text{minimize} && 5.1 x_0 \\ &\text{subject to} && 6.2 x_1 + 7.3 x_2 - 8.4 \in \{0\} \\ &&& x \in \mathcal{Q}^3, x_0 \in \mathbb{Z}. \end{aligned} \tag{16.11}$$

Its formulation in the Conic Benchmark Format begins with the version of the CBF format used, to safeguard against later revisions.

VER
1

Next follows the problem structure, consisting of the objective sense, the number and domain of variables, the indices of integer variables, and the number and domain of scalar-valued affine expressions (i.e., the equality constraint).

OBJSENSE
MIN
VAR
3 1
Q 3
INT
1
0

(continues on next page)

(continued from previous page)

```

CON
1 1
L= 1

```

Finally follows the problem data, consisting of the coefficients of the objective, the coefficients of the constraints, and the constant terms of the constraints. All data is specified on a sparse coordinate form.

```

OBJCOORD
1
0 5.1

ACCOORD
2
0 1 6.2
0 2 7.3

BCCOORD
1
0 -8.4

```

This concludes the example.

Mixing Linear, Second-order and Semidefinite Cones

The conic optimization problem (16.12), has a semidefinite cone, a quadratic cone over unordered subindices, and two equality constraints.

$$\begin{aligned}
 & \text{minimize} && \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, X_1 \right\rangle + x_1 \\
 & \text{subject to} && \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 &= 1.0, \\
 & && \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, X_1 \right\rangle + x_0 + x_2 &= 0.5, \\
 & && x_1 \geq \sqrt{x_0^2 + x_2^2}, \\
 & && X_1 \succeq \mathbf{0}.
 \end{aligned} \tag{16.12}$$

The equality constraints are easily rewritten to the conic form, $(g_0, g_1) \in \{0\}^2$, by moving constants such that the right-hand-side becomes zero. The quadratic cone does not fit under the VAR keyword in this variable permutation. Instead, it takes a scalar constraint $(g_2, g_3, g_4) = (x_1, x_0, x_2) \in \mathcal{Q}^3$, with scalar variables constructed as $(x_0, x_1, x_2) \in \mathbb{R}^3$. Its formulation in the CBF format is reported in the following list

```

# File written using this version of the Conic Benchmark Format:
# | Version 1.
VER
1

# The sense of the objective is:
# | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
# | Three times three.
PSDVAR
1
3

```

(continues on next page)

```

# Three scalar variables in this one conic domain:
#   | Three are free.
VAR
3 1
F 3

# Five scalar constraints with affine expressions in two conic domains:
#   | Two are fixed to zero.
#   | Three are in conic quadratic domain.
CON
5 2
L= 2
Q 3

# Five coordinates in  $F^{\text{obj}}_j$  coefficients:
#   |  $F^{\text{obj}}[0][0,0] = 2.0$ 
#   |  $F^{\text{obj}}[0][1,0] = 1.0$ 
#   | and more...
OBJFCOORD
5
0 0 0 2.0
0 1 0 1.0
0 1 1 2.0
0 2 1 1.0
0 2 2 2.0

# One coordinate in  $a^{\text{obj}}_j$  coefficients:
#   |  $a^{\text{obj}}[1] = 1.0$ 
OBJACOORD
1
1 1.0

# Nine coordinates in  $F_{ij}$  coefficients:
#   |  $F[0,0][0,0] = 1.0$ 
#   |  $F[0,0][1,1] = 1.0$ 
#   | and more...
FCOORD
9
0 0 0 0 1.0
0 0 1 1 1.0
0 0 2 2 1.0
1 0 0 0 1.0
1 0 1 0 1.0
1 0 2 0 1.0
1 0 1 1 1.0
1 0 2 1 1.0
1 0 2 2 1.0

# Six coordinates in  $a_{ij}$  coefficients:
#   |  $a[0,1] = 1.0$ 
#   |  $a[1,0] = 1.0$ 
#   | and more...
ACOORD
6
0 1 1.0
1 0 1.0
1 2 1.0
2 1 1.0
3 0 1.0
4 2 1.0

```

(continued from previous page)

```
# Two coordinates in b_i coefficients:
#      | b[0] = -1.0
#      | b[1] = -0.5
BCOORD
2
0 -1.0
1 -0.5
```

Mixing Semidefinite Variables and Linear Matrix Inequalities

The standard forms in semidefinite optimization are usually based either on semidefinite variables or linear matrix inequalities. In the CBF format, both forms are supported and can even be mixed as shown in.

$$\begin{aligned} & \text{minimize} && \left\langle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 + x_2 + 1 \\ & \text{subject to} && \left\langle \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, X_1 \right\rangle - x_1 - x_2 \geq 0.0, \\ & && x_1 \begin{bmatrix} 0 & 1 \\ 1 & 3 \end{bmatrix} + x_2 \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \succeq \mathbf{0}, \\ & && X_1 \succeq \mathbf{0}. \end{aligned} \tag{16.13}$$

Its formulation in the CBF format is written in what follows

```
# File written using this version of the Conic Benchmark Format:
#      | Version 1.
VER
1

# The sense of the objective is:
#      | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
#      | Two times two.
PSDVAR
1
2

# Two scalar variables in this one conic domain:
#      | Two are free.
VAR
2 1
F 2

# One PSD constraint of this size:
#      | Two times two.
PSDCON
1
2

# One scalar constraint with an affine expression in this one conic domain:
#      | One is greater than or equal to zero.
CON
1 1
L+ 1

# Two coordinates in F^{obj}_j coefficients:
```

(continues on next page)

```

#      | F^{obj}[0][0,0] = 1.0
#      | F^{obj}[0][1,1] = 1.0
OBJFCOORD
2
0 0 0 1.0
0 1 1 1.0

# Two coordinates in a^{obj}_j coefficients:
#      | a^{obj}[0] = 1.0
#      | a^{obj}[1] = 1.0
OBJACOORD
2
0 1.0
1 1.0

# One coordinate in b^{obj} coefficient:
#      | b^{obj} = 1.0
OBJBCOORD
1.0

# One coordinate in F_ij coefficients:
#      | F[0,0][1,0] = 1.0
FCOORD
1
0 0 1 0 1.0

# Two coordinates in a_ij coefficients:
#      | a[0,0] = -1.0
#      | a[0,1] = -1.0
ACCOORD
2
0 0 -1.0
0 1 -1.0

# Four coordinates in H_ij coefficients:
#      | H[0,0][1,0] = 1.0
#      | H[0,0][1,1] = 3.0
#      | and more...
HCOORD
4
0 0 1 0 1.0
0 0 1 1 3.0
0 1 0 0 3.0
0 1 1 0 1.0

# Two coordinates in D_i coefficients:
#      | D[0][0,0] = -1.0
#      | D[0][1,1] = -1.0
DCCOORD
2
0 0 0 -1.0
0 1 1 -1.0

```

Optimization Over a Sequence of Objectives

The linear optimization problem (16.14), is defined for a sequence of objectives such that hotstarting from one to the next might be advantages.

$$\begin{aligned}
 & \text{maximize}_k && g_k^{obj} \\
 & \text{subject to} && 50x_0 + 31 \leq 250, \\
 & && 3x_0 - 2x_1 \geq -4, \\
 & && x \in \mathbb{R}_+^2,
 \end{aligned} \tag{16.14}$$

given,

1. $g_0^{obj} = x_0 + 0.64x_1$.
2. $g_1^{obj} = 1.11x_0 + 0.76x_1$.
3. $g_2^{obj} = 1.11x_0 + 0.85x_1$.

Its formulation in the CBF format is reported in [Listing 16.5](#).

Listing 16.5: Problem (16.14) in CBF format.

```
# File written using this version of the Conic Benchmark Format:
#   | Version 1.
VER
1

# The sense of the objective is:
#   | Maximize.
OBJSENSE
MAX

# Two scalar variables in this one conic domain:
#   | Two are nonnegative.
VAR
2 1
L+ 2

# Two scalar constraints with affine expressions in these two conic domains:
#   | One is in the nonpositive domain.
#   | One is in the nonnegative domain.
CON
2 2
L- 1
L+ 1

# Two coordinates in a^{obj}_j coefficients:
#   | a^{obj}[0] = 1.0
#   | a^{obj}[1] = 0.64
OBJACCOORD
2
0 1.0
1 0.64

# Four coordinates in a_ij coefficients:
#   | a[0,0] = 50.0
#   | a[1,0] = 3.0
#   | and more...
ACCOORD
4
0 0 50.0
1 0 3.0
0 1 31.0
1 1 -2.0

# Two coordinates in b_i coefficients:
#   | b[0] = -250.0
#   | b[1] = 4.0
BCCOORD
2
0 -250.0
1 4.0
```

(continues on next page)

```
# New problem instance defined in terms of changes.
CHANGE

# Two coordinate changes in a^{obj}_j coefficients. Now it is:
#   | a^{obj}[0] = 1.11
#   | a^{obj}[1] = 0.76
OBJCOORD
2
0 1.11
1 0.76

# New problem instance defined in terms of changes.
CHANGE

# One coordinate change in a^{obj}_j coefficients. Now it is:
#   | a^{obj}[0] = 1.11
#   | a^{obj}[1] = 0.85
OBJCOORD
1
1 0.85
```

16.5 The PTF Format

The PTF format is a new human-readable, natural text format. Its features and structure are similar to the *OPF* format, with the difference that the PTF format **does** support semidefinite terms.

16.5.1 The overall format

The format is indentation based, where each section is started by a head line and followed by a section body with deeper indentation than the head line. For example:

```
Header line
  Body line 1
  Body line 1
  Body line 1
```

Section can also be nested:

```
Header line A
  Body line in A
  Header line A.1
    Body line in A.1
    Body line in A.1
  Body line in A
```

The indentation of blank lines is ignored, so a subsection can contain a blank line with no indentation. The character # defines a line comment and anything between the # character and the end of the line is ignored.

In a PTF file, the first section must be a **Task** section. The order of the remaining section is arbitrary, and sections may occur multiple times or not at all.

MOSEK will ignore any top-level section it does not recognize.

Names

In the description of the format we use following definitions for name strings:

```
NAME: PLAIN_NAME | QUOTED_NAME
PLAIN_NAME: [a-zA-Z_] [a-zA-Z0-9_-.!|]
QUOTED_NAME: '"' ( [^'\\r\n] | "\\" ( [\\rn] | "x" [0-9a-fA-F] [0-9a-fA-F] ) ) * '"'
```

Expressions

An expression is a sum of terms. A term is either a linear term (a coefficient and a variable name, where the coefficient can be left out if it is 1.0), or a matrix inner product.

An expression:

```
EXPR: EMPTY | [+ -]? TERM ( [+ -] TERM )*  
TERM: LINEAR_TERM | MATRIX_TERM
```

A linear term

```
LINEAR_TERM: FLOAT? NAME
```

A matrix term

```
MATRIX_TERM: "<" FLOAT? NAME ( [+ -] FLOAT? NAME)* ";" NAME ">"
```

Here the right-hand name is the name of a (semidefinite) matrix variable, and the left-hand side is a sum of symmetric matrixes. The actual matrixes are defined in a separate section.

Expressions can span multiple lines by giving subsequent lines a deeper indentation.

For example following two section are equivalent:

```
# Everything on one line:  
x1 + x2 + x3 + x4  
  
# Split into multiple lines:  
x1  
  + x2  
  + x3  
  + x4
```

16.5.2 Task section

The first section of the file must be a **Task**. The text in this section is not used and may contain comments, or meta-information from the writer or about the content.

Format:

```
Task NAME  
  Anything goes here...
```

NAME is a the task name.

16.5.3 Objective section

The **Objective** section defines the objective name, sense and function. The format:

```
"Objective" NAME?  
( "Minimize" | "Maximize" ) EXPR
```

For example:

```
Objective 'obj'  
  Minimize x1 + 0.2 x2 + < M1 ; X1 >
```

16.5.4 Constraints section

The constraints section defines a series of constraints. A constraint defines a term $A \cdot x + b \in K$. For linear constraints A is just one row, while for conic constraints it can be multiple rows. If a constraint spans multiple rows these can either be written inline separated by semi-colons, or each expression in a separate sub-section.

Simple linear constraints:

```
"Constraints"
NAME? "[" [-+] (FLOAT | "Inf") (";" [-+] (FLOAT | "Inf"))? "]" EXPR
```

If the brackets contain two values, they are used as upper and lower bounds. If they contain one value the constraint is an equality.

For example:

```
Constraints
'c1' [0;10] x1 + x2 + x3
[0] x1 + x2 + x3
```

Constraint blocks put the expression either in a subsection or inline. The cone type (domain) is written in the brackets, and **MOSEK** currently supports following types:

- SOC(N) Second order cone of dimension N
- RSOC(N) Rotated second order cone of dimension N
- PSD(N) Symmetric positive semidefinite cone of dimension N. This contains $N*(N+1)/2$ elements.
- PEXP Primal exponential cone of dimension 3
- DEXP Dual exponential cone of dimension 3
- PPOW(N,P) Primal power cone of dimension N with parameter P
- DPOW(N,P) Dual power cone of dimension N with parameter P
- ZERO(N) The zero-cone of dimension N.

```
"Constraints"
NAME? "[" DOMAIN "]" EXPR_LIST
```

For example:

```
Constraints
'K1' [SOC(3)] x1 + x2 ; x2 + x3 ; x3 + x1
'K2' [RSOC(3)]
    x1 + x2
    x2 + x3
    x3 + x1
```

16.5.5 Variables section

Any variable used in an expression must be defined in a variable section. The variable section defines each variable domain.

```
"Variables"
NAME "[" [-+] (FLOAT | "Inf") (";" [-+] (FLOAT | "Inf"))? "]"
NAME "[" DOMAIN "]" NAMES
```

For example, a linear variable

```
Variables
x1 [0;Inf]
```

As with constraints, members of a conic domain can be listed either inline or in a subsection:

```
Variables
k1 [SOC(3)] x1 ; x2 ; x3
k2 [RSOC(3)]
    x1
    x2
    x3
```

16.5.6 Integer section

This section contains a list of variables that are integral. For example:

```
Integer
  x1 x2 x3
```

16.5.7 SymmetricMatrixes section

This section defines the symmetric matrixes used for matrix coefficients in matrix inner product terms. The section lists named matrixes, each with a size and a number of non-zeros. Only non-zeros in the lower triangular part should be defined.

```
"SymmetricMatrixes"
  NAME "SYMMAT" "(" INT ")" ( "(" INT "," INT "," FLOAT ")" ) *
  ...
```

For example:

```
SymmetricMatrixes
M1 SYMMAT(3) (0,0,1.0) (1,1,2.0) (2,1,0.5)
M2 SYMMAT(3)
  (0,0,1.0)
  (1,1,2.0)
  (2,1,0.5)
```

16.5.8 Solutions section

Each subsection defines a solution. A solution defines for each constraint and for each variable exactly one primal value and either one (for conic domains) or two (for linear domains) dual values. The values follow the same logic as in the **MOSEK C API**. A primal and a dual solution status defines the meaning of the values primal and dual (solution, certificate, unknown, etc.)

The format is this:

```
"Solutions"
  "Solution" WHICH SOL
  "ProblemStatus" PROSTA PROSTA?
  "SolutionStatus" SOLSTA SOLSTA?
  "Objective" FLOAT FLOAT
  "Variables"
    # Linear variable status: level, slx, sux
    NAME "[" STATUS "]" FLOAT (FLOAT FLOAT)?
    # Conic variable status: level, snx
    NAME
      "[" STATUS "]" FLOAT FLOAT?
    ...
  "Constraints"
    # Linear variable status: level, slx, sux
    NAME "[" STATUS "]" FLOAT (FLOAT FLOAT)?
    # Conic variable status: level, snx
    NAME
      "[" STATUS "]" FLOAT FLOAT?
    ...
```

Following values for WHICH SOL are supported:

- **interior** Interior solution, the result of an interior-point solver.
- **basic** Basic solution, as produced by a simplex solver.
- **integer** Integer solution, the solution to a mixed-integer problem. This does not define a dual solution.

Following values for PROSTA are supported:

- **unknown** The problem status is unknown
- **feasible** The problem has been proven feasible
- **infeasible** The problem has been proven infeasible
- **illposed** The problem has been proved to be ill posed
- **infeasible_or_unbounded** The problem is infeasible or unbounded

Following values for **SOLSTA** are supported:

- **unknown** The solution status is unknown
- **feasible** The solution is feasible
- **optimal** The solution is optimal
- **infeas_cert** The solution is a certificate of infeasibility
- **illposed_cert** The solution is a certificate of illposedness

Following values for **STATUS** are supported:

- **unknown** The value is unknown
- **super_basic** The value is super basic
- **at_lower** The value is basic and at its lower bound
- **at_upper** The value is basic and at its upper bound
- **fixed** The value is basic fixed
- **infinite** The value is at infinity

16.6 The Task Format

The Task format is **MOSEK**'s native binary format. It contains a complete image of a **MOSEK** task, i.e.

- Problem data: Linear, conic, semidefinite and quadratic data
- Problem item names: Variable names, constraints names, cone names etc.
- Parameter settings
- Solutions

There are a few things to be aware of:

- Status of a solution read from a file will *always* be unknown.
- Parameter settings in a task file *always override* any parameters set on the command line or in a parameter file.

The format is based on the *TAR* (USTar) file format. This means that the individual pieces of data in a **.task** file can be examined by unpacking it as a *TAR* file. Please note that the inverse may not work: Creating a file using *TAR* will most probably not create a valid **MOSEK** Task file since the order of the entries is important.

16.7 The JSON Format

MOSEK provides the possibility to read/write problems in valid JSON format.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

The official JSON website <http://www.json.org> provides plenty of information along with the format definition.

MOSEK defines two JSON-like formats:

- *jtask*
- *jsol*

Despite being text-based human-readable formats, *jtask* and *jsol* files will include no indentation and no new-lines, in order to keep the files as compact as possible. We therefore strongly advise to use JSON viewer tools to inspect *jtask* and *jsol* files.

16.7.1 *jtask* format

It stores a problem instance. The *jtask* format contains the same information as a *task format*. Even though a *jtask* file is human-readable, we do not recommend users to create it by hand, but to rely on **MOSEK**.

16.7.2 *jsol* format

It stores a problem solution. The *jsol* format contains all solutions and information items.

You can write a *jsol* file using `Task.writejsonsol`. You **can not** read a *jsol* file into **MOSEK**.

16.7.3 A *jtask* example

In [Listing 16.6](#) we present a file in the *jtask* format that corresponds to the sample problem from `1o1.lp`. The listing has been formatted for readability.

Listing 16.6: A formatted *jtask* file for the `1o1.lp` example.

```
{
  "$schema": "http://mosek.com/json/schema#",
  "Task/INFO": {
    "taskname": "1o1",
    "numvar": 4,
    "numcon": 3,
    "numcone": 0,
    "numbarvar": 0,
    "numanz": 9,
    "numsymmat": 0,
    "mosekver": [
      8,
      0,
      0,
      9
    ]
  },
  "Task/data": {
    "var": {
      "name": [
        "x1",
        "x2",
```

(continues on next page)

```

        "x3",
        "x4"
    ],
    "bk": [
        "lo",
        "ra",
        "lo",
        "lo"
    ],
    "b1": [
        0.0,
        0.0,
        0.0,
        0.0
    ],
    "bu": [
        1e+30,
        1e+1,
        1e+30,
        1e+30
    ],
    "type": [
        "cont",
        "cont",
        "cont",
        "cont"
    ]
    ],
    "con": {
        "name": [
            "c1",
            "c2",
            "c3"
        ],
        "bk": [
            "fx",
            "lo",
            "up"
        ],
        "b1": [
            3e+1,
            1.5e+1,
            -1e+30
        ],
        "bu": [
            3e+1,
            1e+30,
            2.5e+1
        ]
    },
    "objective": {
        "sense": "max",
        "name": "obj",
        "c": {
            "subj": [
                0,
                1,
                2,
                3
            ],
            "val": [

```

```

        3e+0,
        1e+0,
        5e+0,
        1e+0
    ]
},
    "cfix":0.0
},
    "A":{
        "subi":[
            0,
            0,
            0,
            1,
            1,
            1,
            1,
            2,
            2
        ],
        "subj":[
            0,
            1,
            2,
            0,
            1,
            2,
            3,
            1,
            3
        ],
        "val":[
            3e+0,
            1e+0,
            2e+0,
            2e+0,
            1e+0,
            3e+0,
            1e+0,
            2e+0,
            3e+0
        ]
    }
},
    "Task/parameters":{
        "iparam":{
            "ANA_SOL_BASIS":"ON",
            "ANA_SOL_PRINT_VIOLATED":"OFF",
            "AUTO_SORT_A_BEFORE_OPT":"OFF",
            "AUTO_UPDATE_SOL_INFO":"OFF",
            "BASIS_SOLVE_USE_PLUS_ONE":"OFF",
            "BI_CLEAN_OPTIMIZER":"OPTIMIZER_FREE",
            "BI_IGNORE_MAX_ITER":"OFF",
            "BI_IGNORE_NUM_ERROR":"OFF",
            "BI_MAX_ITERATIONS":1000000,
            "CACHE_LICENSE":"ON",
            "CHECK_CONVEXITY":"CHECK_CONVEXITY_FULL",
            "COMPRESS_STATFILE":"ON",
            "CONCURRENT_NUM_OPTIMIZERS":2,
            "CONCURRENT_PRIORITY_DUAL_SIMPLEX":2,
            "CONCURRENT_PRIORITY_FREE_SIMPLEX":3,

```

```

"CONCURRENT_PRIORITY_INTPNT":4,
"CONCURRENT_PRIORITY_PRIMAL_SIMPLEX":1,
"FEASREPAIR_OPTIMIZE":"FEASREPAIR_OPTIMIZE_NONE",
"INFEAS_GENERIC_NAMES":"OFF",
"INFEAS_PREFER_PRIMAL":"ON",
"INFEAS_REPORT_AUTO":"OFF",
"INFEAS_REPORT_LEVEL":1,
"INTPNT_BASIS":"BI_ALWAYS",
"INTPNT_DIFF_STEP":"ON",
"INTPNT_FACTOR_DEBUG_LVL":0,
"INTPNT_FACTOR_METHOD":0,
"INTPNT_HOTSTART":"INTPNT_HOTSTART_NONE",
"INTPNT_MAX_ITERATIONS":400,
"INTPNT_MAX_NUM_COR":-1,
"INTPNT_MAX_NUM_REFINEMENT_STEPS":-1,
"INTPNT_OFF_COL_TRH":40,
"INTPNT_ORDER_METHOD":"ORDER_METHOD_FREE",
"INTPNT_REGULARIZATION_USE":"ON",
"INTPNT_SCALING":"SCALING_FREE",
"INTPNT_SOLVE_FORM":"SOLVE_FREE",
"INTPNT_STARTING_POINT":"STARTING_POINT_FREE",
"LIC_TRH_EXPIRY_WRN":7,
"LICENSE_DEBUG":"OFF",
"LICENSE_PAUSE_TIME":0,
"LICENSE_SUPPRESS_EXPIRE_WRNS":"OFF",
"LICENSE_WAIT":"OFF",
"LOG":10,
"LOG_ANA_PRO":1,
"LOG_BI":4,
"LOG_BI_FREQ":2500,
"LOG_CHECK_CONVEXITY":0,
"LOG_CONCURRENT":1,
"LOG_CUT_SECOND_OPT":1,
"LOG_EXPAND":0,
"LOG_FACTOR":1,
"LOG_FEAS_REPAIR":1,
"LOG_FILE":1,
"LOG_HEAD":1,
"LOG_INFEAS_ANA":1,
"LOG_INTPNT":4,
"LOG_MIO":4,
"LOG_MIO_FREQ":1000,
"LOG_OPTIMIZER":1,
"LOG_ORDER":1,
"LOG_PRESOLVE":1,
"LOG_RESPONSE":0,
"LOG_SENSITIVITY":1,
"LOG_SENSITIVITY_OPT":0,
"LOG_SIM":4,
"LOG_SIM_FREQ":1000,
"LOG_SIM_MINOR":1,
"LOG_STORAGE":1,
"MAX_NUM_WARNINGS":10,
"MIO_BRANCH_DIR":"BRANCH_DIR_FREE",
"MIO_CONSTRUCT_SOL":"OFF",
"MIO_CUT_CLIQUE":"ON",
"MIO_CUT_CMIR":"ON",
"MIO_CUT_GMI":"ON",
"MIO_CUT_KNAPSACK_COVER":"OFF",
"MIO_HEURISTIC_LEVEL":-1,
"MIO_MAX_NUM_BRANCHES":-1,

```

```

"MIO_MAX_NUM_RELAXS":-1,
"MIO_MAX_NUM_SOLUTIONS":-1,
"MIO_MODE":"MIO_MODE_SATISFIED",
"MIO_MT_USER_CB":"ON",
"MIO_NODE_OPTIMIZER":"OPTIMIZER_FREE",
"MIO_NODE_SELECTION":"MIO_NODE_SELECTION_FREE",
"MIO_PERSPECTIVE_REFORMULATE":"ON",
"MIO_PROBING_LEVEL":-1,
"MIO_RINS_MAX_NODES":-1,
"MIO_ROOT_OPTIMIZER":"OPTIMIZER_FREE",
"MIO_ROOT_REPEAT_PRESOLVE_LEVEL":-1,
"MT_SPINCOUNT":0,
"NUM_THREADS":0,
"OPF_MAX_TERMS_PER_LINE":5,
"OPF_WRITE_HEADER":"ON",
"OPF_WRITE_HINTS":"ON",
"OPF_WRITE_PARAMETERS":"OFF",
"OPF_WRITE_PROBLEM":"ON",
"OPF_WRITE_SOL_BAS":"ON",
"OPF_WRITE_SOL_ITG":"ON",
"OPF_WRITE_SOL_ITR":"ON",
"OPF_WRITE_SOLUTIONS":"OFF",
"OPTIMIZER":"OPTIMIZER_FREE",
"PARAM_READ_CASE_NAME":"ON",
"PARAM_READ_IGN_ERROR":"OFF",
"PRESOLVE_ELIMINATOR_MAX_FILL":-1,
"PRESOLVE_ELIMINATOR_MAX_NUM_TRIES":-1,
"PRESOLVE_LEVEL":-1,
"PRESOLVE_LINDEP_ABS_WORK_TRH":100,
"PRESOLVE_LINDEP_REL_WORK_TRH":100,
"PRESOLVE_LINDEP_USE":"ON",
"PRESOLVE_MAX_NUM_REDUCTIONS":-1,
"PRESOLVE_USE":"PRESOLVE_MODE_FREE",
"PRIMAL_REPAIR_OPTIMIZER":"OPTIMIZER_FREE",
"QO_SEPARABLE_REFORMULATION":"OFF",
"READ_DATA_COMPRESSED":"COMPRESS_FREE",
"READ_DATA_FORMAT":"DATA_FORMAT_EXTENSION",
"READ_DEBUG":"OFF",
"READ_KEEP_FREE_CON":"OFF",
"READ_LP_DROP_NEW_VARS_IN_BOU":"OFF",
"READ_LP_QUOTED_NAMES":"ON",
"READ_MPS_FORMAT":"MPS_FORMAT_FREE",
"READ_MPS_WIDTH":1024,
"READ_TASK_IGNORE_PARAM":"OFF",
"SENSITIVITY_ALL":"OFF",
"SENSITIVITY_OPTIMIZER":"OPTIMIZER_FREE_SIMPLEX",
"SENSITIVITY_TYPE":"SENSITIVITY_TYPE_BASIS",
"SIM_BASIS_FACTOR_USE":"ON",
"SIM_DEGEN":"SIM_DEGEN_FREE",
"SIM_DUAL_CRASH":90,
"SIM_DUAL_PHASEONE_METHOD":0,
"SIM_DUAL_RESTRICT_SELECTION":50,
"SIM_DUAL_SELECTION":"SIM_SELECTION_FREE",
"SIM_EXPLOIT_DUPVEC":"SIM_EXPLOIT_DUPVEC_OFF",
"SIM_HOTSTART":"SIM_HOTSTART_FREE",
"SIM_HOTSTART_LU":"ON",
"SIM_INTEGER":0,
"SIM_MAX_ITERATIONS":10000000,
"SIM_MAX_NUM_SETBACKS":250,
"SIM_NON_SINGULAR":"ON",
"SIM_PRIMAL_CRASH":90,

```

```

    "SIM_PRIMAL_PHASEONE_METHOD":0,
    "SIM_PRIMAL_RESTRICT_SELECTION":50,
    "SIM_PRIMAL_SELECTION":"SIM_SELECTION_FREE",
    "SIM_REFACTOR_FREQ":0,
    "SIM_REFORMULATION":"SIM_REFORMULATION_OFF",
    "SIM_SAVE_LU":"OFF",
    "SIM_SCALING":"SCALING_FREE",
    "SIM_SCALING_METHOD":"SCALING_METHOD_POW2",
    "SIM_SOLVE_FORM":"SOLVE_FREE",
    "SIM_STABILITY_PRIORITY":50,
    "SIM_SWITCH_OPTIMIZER":"OFF",
    "SOL_FILTER_KEEP_BASIC":"OFF",
    "SOL_FILTER_KEEP_RANGED":"OFF",
    "SOL_READ_NAME_WIDTH":-1,
    "SOL_READ_WIDTH":1024,
    "SOLUTION_CALLBACK":"OFF",
    "TIMING_LEVEL":1,
    "WRITE_BAS_CONSTRAINTS":"ON",
    "WRITE_BAS_HEAD":"ON",
    "WRITE_BAS_VARIABLES":"ON",
    "WRITE_DATA_COMPRESSED":0,
    "WRITE_DATA_FORMAT":"DATA_FORMAT_EXTENSION",
    "WRITE_DATA_PARAM":"OFF",
    "WRITE_FREE_CON":"OFF",
    "WRITE_GENERIC_NAMES":"OFF",
    "WRITE_GENERIC_NAMES_IO":1,
    "WRITE_IGNORE_INCOMPATIBLE_CONIC_ITEMS":"OFF",
    "WRITE_IGNORE_INCOMPATIBLE_ITEMS":"OFF",
    "WRITE_IGNORE_INCOMPATIBLE_NL_ITEMS":"OFF",
    "WRITE_IGNORE_INCOMPATIBLE_PSD_ITEMS":"OFF",
    "WRITE_INT_CONSTRAINTS":"ON",
    "WRITE_INT_HEAD":"ON",
    "WRITE_INT_VARIABLES":"ON",
    "WRITE_LP_FULL_OBJ":"ON",
    "WRITE_LP_LINE_WIDTH":80,
    "WRITE_LP_QUOTED_NAMES":"ON",
    "WRITE_LP_STRICT_FORMAT":"OFF",
    "WRITE_LP_TERMS_PER_LINE":10,
    "WRITE_MPS_FORMAT":"MPS_FORMAT_FREE",
    "WRITE_MPS_INT":"ON",
    "WRITE_PRECISION":15,
    "WRITE_SOL_BARVARIABLES":"ON",
    "WRITE_SOL_CONSTRAINTS":"ON",
    "WRITE_SOL_HEAD":"ON",
    "WRITE_SOL_IGNORE_INVALID_NAMES":"OFF",
    "WRITE_SOL_VARIABLES":"ON",
    "WRITE_TASK_INC_SOL":"ON",
    "WRITE_XML_MODE":"WRITE_XML_MODE_ROW"
},
"dparam":{
    "ANA_SOL_INFEAS_TOL":1e-6,
    "BASIS_REL_TOL_S":1e-12,
    "BASIS_TOL_S":1e-6,
    "BASIS_TOL_X":1e-6,
    "CHECK_CONVEXITY_REL_TOL":1e-10,
    "DATA_TOL_AIJ":1e-12,
    "DATA_TOL_AIJ_HUGE":1e+20,
    "DATA_TOL_AIJ_LARGE":1e+10,
    "DATA_TOL_BOUND_INF":1e+16,
    "DATA_TOL_BOUND_WRN":1e+8,
    "DATA_TOL_C_HUGE":1e+16,

```

```

"DATA_TOL_CJ_LARGE":1e+8,
"DATA_TOL_QIJ":1e-16,
"DATA_TOL_X":1e-8,
"FEASREPAIR_TOL":1e-10,
"INTPNT_CO_TOL_DFEAS":1e-8,
"INTPNT_CO_TOL_INFEAS":1e-10,
"INTPNT_CO_TOL_MU_RED":1e-8,
"INTPNT_CO_TOL_NEAR_REL":1e+3,
"INTPNT_CO_TOL_PFEAS":1e-8,
"INTPNT_CO_TOL_REL_GAP":1e-7,
"INTPNT_NL_MERIT_BAL":1e-4,
"INTPNT_NL_TOL_DFEAS":1e-8,
"INTPNT_NL_TOL_MU_RED":1e-12,
"INTPNT_NL_TOL_NEAR_REL":1e+3,
"INTPNT_NL_TOL_PFEAS":1e-8,
"INTPNT_NL_TOL_REL_GAP":1e-6,
"INTPNT_NL_TOL_REL_STEP":9.95e-1,
"INTPNT_QO_TOL_DFEAS":1e-8,
"INTPNT_QO_TOL_INFEAS":1e-10,
"INTPNT_QO_TOL_MU_RED":1e-8,
"INTPNT_QO_TOL_NEAR_REL":1e+3,
"INTPNT_QO_TOL_PFEAS":1e-8,
"INTPNT_QO_TOL_REL_GAP":1e-8,
"INTPNT_TOL_DFEAS":1e-8,
"INTPNT_TOL_DSAFE":1e+0,
"INTPNT_TOL_INFEAS":1e-10,
"INTPNT_TOL_MU_RED":1e-16,
"INTPNT_TOL_PATH":1e-8,
"INTPNT_TOL_PFEAS":1e-8,
"INTPNT_TOL_PSAFE":1e+0,
"INTPNT_TOL_REL_GAP":1e-8,
"INTPNT_TOL_REL_STEP":9.999e-1,
"INTPNT_TOL_STEP_SIZE":1e-6,
"LOWER_OBJ_CUT":-1e+30,
"LOWER_OBJ_CUT_FINITE_TRH":-5e+29,
"MIO_DISABLE_TERM_TIME":-1e+0,
"MIO_MAX_TIME":-1e+0,
"MIO_MAX_TIME_APRX_OPT":6e+1,
"MIO_NEAR_TOL_ABS_GAP":0.0,
"MIO_NEAR_TOL_REL_GAP":1e-3,
"MIO_REL_GAP_CONST":1e-10,
"MIO_TOL_ABS_GAP":0.0,
"MIO_TOL_ABS_RELAX_INT":1e-5,
"MIO_TOL_FEAS":1e-6,
"MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT":0.0,
"MIO_TOL_REL_GAP":1e-4,
"MIO_TOL_X":1e-6,
"OPTIMIZER_MAX_TIME":-1e+0,
"PRESOLVE_TOL_ABS_LINDEP":1e-6,
"PRESOLVE_TOL_AIJ":1e-12,
"PRESOLVE_TOL_REL_LINDEP":1e-10,
"PRESOLVE_TOL_S":1e-8,
"PRESOLVE_TOL_X":1e-8,
"QCQO_REFORMULATE_REL_DROP_TOL":1e-15,
"SEMIDEFINITE_TOL_APPROX":1e-10,
"SIM_LU_TOL_REL_PIV":1e-2,
"SIMPLEX_ABS_TOL_PIV":1e-7,
"UPPER_OBJ_CUT":1e+30,
"UPPER_OBJ_CUT_FINITE_TRH":5e+29
},
"sparam":{

```

```

        "BAS_SOL_FILE_NAME": "",
        "DATA_FILE_NAME": "examples/tools/data/lo1.mps",
        "DEBUG_FILE_NAME": "",
        "INT_SOL_FILE_NAME": "",
        "ITR_SOL_FILE_NAME": "",
        "MIO_DEBUG_STRING": "",
        "PARAM_COMMENT_SIGN": "%%",
        "PARAM_READ_FILE_NAME": "",
        "PARAM_WRITE_FILE_NAME": "",
        "READ_MPS_BOU_NAME": "",
        "READ_MPS_OBJ_NAME": "",
        "READ_MPS_RAN_NAME": "",
        "READ_MPS_RHS_NAME": "",
        "SENSITIVITY_FILE_NAME": "",
        "SENSITIVITY_RES_FILE_NAME": "",
        "SOL_FILTER_XC_LOW": "",
        "SOL_FILTER_XC_UPR": "",
        "SOL_FILTER_XX_LOW": "",
        "SOL_FILTER_XX_UPR": "",
        "STAT_FILE_NAME": "",
        "STAT_KEY": "",
        "STAT_NAME": "",
        "WRITE_LP_GEN_VAR_NAME": "XMSKGEN"
    }
}
}

```

16.8 The Solution File Format

MOSEK provides several solution files depending on the problem type and the optimizer used:

- *basis solution file* (extension `.bas`) if the problem is optimized using the simplex optimizer or basis identification is performed,
- *interior solution file* (extension `.sol`) if a problem is optimized using the interior-point optimizer and no basis identification is required,
- *integer solution file* (extension `.int`) if the problem contains integer constrained variables.

All solution files have the format:

NAME		: <problem name>					
PROBLEM STATUS		: <status of the problem>					
SOLUTION STATUS		: <status of the solution>					
OBJECTIVE NAME		: <name of the objective function>					
PRIMAL OBJECTIVE		: <primal objective value corresponding to the solution>					
DUAL OBJECTIVE		: <dual objective value corresponding to the solution>					
CONSTRAINTS							
INDEX	NAME	AT ACTIVITY	LOWER LIMIT	UPPER LIMIT	DUAL LOWER	DUAL UPPER	
?	<name>	?? <a value>	<a value>	<a value>	<a value>	<a value>	
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 status are shown. Next the primal and dual objective values are displayed.

- **CONSTRAINTS** For each constraint i of the form

$$l_i^c \leq \sum_{j=1}^n a_{ij}x_j \leq u_i^c, \quad (16.15)$$

the following information is listed:

- **INDEX**: A sequential index assigned to the constraint by **MOSEK**
- **NAME**: The name of the constraint assigned by the user.
- **AT**: The status of the constraint. In Table 16.4 the possible values of the status keys and their interpretation are shown.

Table 16.4: Status keys.

Status key	Interpretation
UN	Unknown status
BS	Is basic
SB	Is superbasic
LL	Is at the lower limit (bound)
UL	Is at the upper limit (bound)
EQ	Lower limit is identical to upper limit
**	Is infeasible i.e. the lower limit is greater than the upper limit.

- **ACTIVITY**: the quantity $\sum_{j=1}^n a_{ij}x_j^*$, where x^* is the value of the primal solution.
- **LOWER LIMIT**: the quantity l_i^c (see (16.15).)
- **UPPER LIMIT**: the quantity u_i^c (see (16.15).)
- **DUAL LOWER**: the dual multiplier corresponding to the lower limit on the constraint.
- **DUAL UPPER**: the dual multiplier corresponding to the upper limit on the constraint.

- **VARIABLES** The last section of the solution report lists information about the variables. This information has a similar interpretation as for the constraints. However, the column with the header **CONIC DUAL** is included for problems having one or more conic constraints. This column shows the dual variables corresponding to the conic constraints.

Example: `1o1.sol`

In Listing 16.7 we show the solution file for the `1o1.opf` problem.

Listing 16.7: An example of `.sol` file.

NAME	:				
PROBLEM STATUS	:	PRIMAL_AND_DUAL_FEASIBLE			
SOLUTION STATUS	:	OPTIMAL			
OBJECTIVE NAME	:	obj			
PRIMAL OBJECTIVE	:	8.33333333e+01			
DUAL OBJECTIVE	:	8.33333332e+01			
CONSTRAINTS					
INDEX	NAME	AT	ACTIVITY	LOWER LIMIT	UPPER LIMIT
↔	DUAL LOWER		DUAL UPPER		
0	c1	EQ	3.00000000000000e+01	3.00000000e+01	3.00000000e+01
↔	00000000000000e+00		-2.49999999741653e+00		-0.
1	c2	SB	5.33333333049187e+01	1.50000000e+01	NONE
↔	09159033069640e-10		-0.00000000000000e+00		2.
2	c3	UL	2.49999999842049e+01	NONE	2.50000000e+01
↔	00000000000000e+00		-3.33333332895108e-01		-0.
VARIABLES					
INDEX	NAME	AT	ACTIVITY	LOWER LIMIT	UPPER LIMIT
↔	DUAL LOWER		DUAL UPPER		
0	x1	LL	1.67020427038537e-09	0.00000000e+00	NONE
↔	49999999528054e+00		-0.00000000000000e+00		-4.

(continues on next page)

(continued from previous page)

1	x2	LL 2.93510446211883e-09	0.00000000e+00	1.00000000e+01	-2.
↪	16666666494915e+00	6.20868657679896e-10			
2	x3	SB 1.49999999899424e+01	0.00000000e+00	NONE	-8.
↪	79123177245553e-10	-0.00000000000000e+00			
3	x4	SB 8.33333332273115e+00	0.00000000e+00	NONE	-1.
↪	69795978848200e-09	-0.00000000000000e+00			

Chapter 17

List of examples

List of examples shipped in the distribution of Optimizer API for Python:

Table 17.1: List of distributed examples

File	Description
blas_lapack.py	Demonstrates the MOSEK interface to BLAS/LAPACK linear algebra routines
callback.py	An example of data/progress callback
ceo1.py	A simple conic exponential problem
concurrent.py	Implementation of a concurrent optimizer for linear and mixed-integer problems
cqo1.py	A simple conic quadratic problem
feasrepairx1.py	A simple example of how to repair an infeasible problem
gp1.py	A simple geometric program (GP) in conic form
lo1.py	A simple linear problem
lo2.py	A simple linear problem
logistic.py	Implements logistic regression and simple log-sum-exp (CEO)
mico1.py	A simple mixed-integer conic problem
milol.py	A simple mixed-integer linear problem
miointsol.py	A simple mixed-integer linear problem with an initial guess
modelLib.py	Library of implementations of basic functions
opt_server_async.py	Uses MOSEK OptServer to solve an optimization problem asynchronously
opt_server_sync.py	Uses MOSEK OptServer to solve an optimization problem synchronously
parallel.py	Demonstrates parallel optimization
parameters.py	Shows how to set optimizer parameters and read information items
portfolio_1_basic.py	Portfolio optimization - basic Markowitz model
portfolio_2_frontier.py	Portfolio optimization - efficient frontier
portfolio_3_impact.py	Portfolio optimization - market impact costs
portfolio_4_transaction.py	Portfolio optimization - transaction costs
portfolio_5_cardinality.py	Portfolio optimization - cardinality constraints
pow1.py	A simple power cone problem
qcqo1.py	A simple quadratically constrained quadratic problem
qo1.py	A simple quadratic problem
reoptimization.py	Demonstrate how to modify and re-optimize a linear problem
response.py	Demonstrates proper response handling

Continued on next page

Table 17.1 – continued from previous page

File	Description
<code>sdo1.py</code>	A simple semidefinite optimization problem
<code>sensitivity.py</code>	Sensitivity analysis performed on a small linear problem
<code>simple.py</code>	A simple I/O example: read problem from a file, solve and write solutions
<code>solutionquality.py</code>	Demonstrates how to examine the quality of a solution
<code>solvebasis.py</code>	Demonstrates solving a linear system with the basis matrix
<code>solvelinear.py</code>	Demonstrates solving a general linear system
<code>sparsecholesky.py</code>	Shows how to find a Cholesky factorization of a sparse matrix

Additional examples can be found on the **MOSEK** website and in other **MOSEK** publications.

Chapter 18

Interface changes

The section shows interface-specific changes to the **MOSEK** Optimizer API for Python in version 9.0. See the [release notes](#) for general changes and new features of the **MOSEK** Optimization Suite.

18.1 Backwards compatibility

- **Parameters.** Users who set parameters to tune the performance and numerical properties of the solver (termination criteria, tolerances, solving primal or dual, presolve etc.) are recommended to reevaluate such tuning. It may be that other, or default, parameter settings will be more beneficial in the current version. The hints in [Sec. 8](#) may be useful for some cases.
- All functions using the enum `accmode` were removed. Use corresponding separate functions for manipulating variables and constraints. For example, instead of

```
task.putbound(accmode.var, ...);  
task.putbound(accmode.con, ...);
```

use

```
task.putvarbound(...);  
task.putconbound(...);
```

and so on.

- Removed all Near problem and solution statuses i.e. `solsta.near_optimal`, `solsta.near_prim_infeas_cer`, etc. See [Sec. 13.3.3](#).
- All functions related to the general nonlinear optimizer and Scopt have been removed. See [Sec. 15.11](#).

18.2 Functions

Added

- *Env.setupthreads*
- *Task.appendsparsesymmatlist*
- *Task.generateconenames*
- *Task.generateconnames*
- *Task.generatevarnames*
- *Task.getacolslice*
- *Task.getacolsllicenumz*

- *Task.getarowslice*
- *Task.getarowslicenumz*
- *Task.getatruncatetol*
- *Task.getbarsslice*
- *Task.getbarxslice*
- *Task.getclist*
- *Task.getskn*
- *Task.putatruncatetol*
- *Task.putbaraijlist*
- *Task.putbararowlist*
- *Task.putconboundlistconst*
- *Task.putconboundsliceconst*
- *Task.putconsolutioni*
- *Task.putvarboundlistconst*
- *Task.putvarboundsliceconst*
- *Task.putvarsolutionj*
- *Task.readjsonstring*
- *Task.readlpstring*
- *Task.readopfstring*
- *Task.readptfstring*

Removed

- *Task.checkconvexity*
- *Task.chgbound*
- *Task.getaslice*
- *Task.getaslicenumz*
- *Task.getbound*
- *Task.getboundslice*
- *Task.getsolutioni*
- *Task.printdata*
- *Task.putbound*
- *Task.putboundlist*
- *Task.putboundslice*
- *Task.putsolutioni*

18.3 Parameters

Added

- *iparam.intpnt_order_gp_num_seeds*
- *iparam.intpnt_purify*
- *iparam.log_include_summary*
- *iparam.log_local_info*
- *iparam.mio_conic_outer_approximation*
- *iparam.mio_feaspump_level*
- *iparam.mio_max_num_root_cut_rounds*
- *iparam.mio_propagate_objective_constraint*
- *iparam.mio_seed*
- *iparam.opf_write_line_length*
- *iparam.presolve_max_num_pass*
- *iparam.ptf_write_transform*
- *iparam.sim_seed*
- *iparam.write_compression*

Removed

- *dparam.data_tol_aij*
- *dparam.intpnt_nl_merit_bal*
- *dparam.intpnt_nl_tol_dfeas*
- *dparam.intpnt_nl_tol_mu_red*
- *dparam.intpnt_nl_tol_near_rel*
- *dparam.intpnt_nl_tol_pfeas*
- *dparam.intpnt_nl_tol_rel_gap*
- *dparam.intpnt_nl_tol_rel_step*
- *dparam.mio_disable_term_time*
- *dparam.mio_near_tol_abs_gap*
- *dparam.mio_near_tol_rel_gap*
- *iparam.mio_construct_sol*
- *iparam.mio_mt_user_cb*
- *iparam.opf_max_terms_per_line*
- *iparam.read_data_compressed*
- *iparam.read_data_format*
- *iparam.write_data_compressed*
- *iparam.write_data_format*

18.4 Constants

Added

- *compresstype.zstd*
- *conetype.dexp*
- *conetype.dpow*
- *conetype.pexp*
- *conetype.ppow*
- *conetype.zero*
- *dataformat.ptf*
- *iinfitem.mio_numbin*
- *iinfitem.mio_numbinconevar*
- *iinfitem.mio_numcone*
- *iinfitem.mio_numconevar*
- *iinfitem.mio_numcont*
- *iinfitem.mio_numcontconevar*
- *iinfitem.mio_numdexpcones*
- *iinfitem.mio_numdpowcones*
- *iinfitem.mio_numintconevar*
- *iinfitem.mio_numpepcones*
- *iinfitem.mio_numppowcones*
- *iinfitem.mio_numqcones*
- *iinfitem.mio_numrqcones*
- *iinfitem.mio_presolved_numbinconevar*
- *iinfitem.mio_presolved_numcone*
- *iinfitem.mio_presolved_numconevar*
- *iinfitem.mio_presolved_numcontconevar*
- *iinfitem.mio_presolved_numdexpcones*
- *iinfitem.mio_presolved_numdpowcones*
- *iinfitem.mio_presolved_numintconevar*
- *iinfitem.mio_presolved_numpepcones*
- *iinfitem.mio_presolved_numppowcones*
- *iinfitem.mio_presolved_numqcones*
- *iinfitem.mio_presolved_numrqcones*
- *iinfitem.purify_dual_success*
- *iinfitem.purify_primal_success*
- *linfitem.mio_anz*

Removed

- `constant.dataformat.xml`
- `constant.dinfitem.mio_heuristic_time`
- `constant.dinfitem.mio_optimizer_time`
- `constant.iinfitem.mio_construct_num_roundings`
- `constant.iinfitem.mio_initial_solution`
- `constant.iinfitem.mio_near_absgap_satisfied`
- `constant.iinfitem.mio_near_relgap_satisfied`
- `constant.liinfitem.mio_sim_maxiter_setbacks`
- `constant.mionodeseltype.hybrid`
- `constant.mionodeseltype.worst`
- `constant.problemtype.geco`
- `constant.prosta.near_dual_feas`
- `constant.prosta.near_prim_and_dual_feas`
- `constant.prosta.near_prim_feas`
- `constant.sensitivitytype.optimal_partition`
- `constant.solsta.near_dual_feas`
- `constant.solsta.near_dual_infeas_cer`
- `constant.solsta.near_integer_optimal`
- `constant.solsta.near_optimal`
- `constant.solsta.near_prim_and_dual_feas`
- `constant.solsta.near_prim_feas`
- `constant.solsta.near_prim_infeas_cer`

18.5 Response Codes

Added

- *`rescode.err_appending_too_big_cone`*
- *`rescode.err_cbf_duplicate_pow_cones`*
- *`rescode.err_cbf_duplicate_pow_star_cones`*
- *`rescode.err_cbf_invalid_dimension_of_cones`*
- *`rescode.err_cbf_invalid_exp_dimension`*
- *`rescode.err_cbf_invalid_number_of_cones`*
- *`rescode.err_cbf_invalid_power`*
- *`rescode.err_cbf_invalid_power_cone_index`*

- *rescode.err_cbf_invalid_power_star_cone_index*
- *rescode.err_cbf_power_cone_is_too_long*
- *rescode.err_cbf_power_cone_mismatch*
- *rescode.err_cbf_power_star_cone_mismatch*
- *rescode.err_cbf_unhandled_power_cone_type*
- *rescode.err_cbf_unhandled_power_star_cone_type*
- *rescode.err_cone_parameter*
- *rescode.err_format_string*
- *rescode.err_invalid_file_format_for_cfix*
- *rescode.err_invalid_file_format_for_free_constraints*
- *rescode.err_invalid_file_format_for_nonlinear*
- *rescode.err_invalid_file_format_for_ranged_constraints*
- *rescode.err_num_arguments*
- *rescode.err_ptf_format*
- *rescode.err_shape_is_too_large*
- *rescode.err_slice_size*
- *rescode.err_too_small_a_truncation_value*
- *rescode.wrn_exp_cones_with_variables_fixed_at_zero*
- *rescode.wrn_pow_cones_with_root_fixed_at_zero*

Removed

- *rescode.err_cannot_clone_nl*
- *rescode.err_cannot_handle_nl*
- *rescode.err_invalid_accmode*
- *rescode.err_invalid_file_format_for_general_nl*
- *rescode.err_nonlinear_functions_not_allowed*
- *rescode.err_nr_arguments*
- *rescode.err_open_dl*
- *rescode.err_user_func_ret*
- *rescode.err_user_func_ret_data*
- *rescode.err_user_nlo_eval*
- *rescode.err_user_nlo_eval_hessubi*
- *rescode.err_user_nlo_eval_hessubj*
- *rescode.err_user_nlo_func*
- *rescode.trm_mio_near_abs_gap*
- *rescode.trm_mio_near_rel_gap*
- *rescode.wrn_construct_invalid_sol_itg*
- *rescode.wrn_construct_no_sol_itg*
- *rescode.wrn_construct_solution_infeas*
- *rescode.wrn_no_nonlinear_function_write*

Bibliography

- [AA95] E. D. Andersen and K. D. Andersen. Presolving in linear programming. *Math. Programming*, 71(2):221–245, 1995.
- [AGMX96] E. D. Andersen, J. Gondzio, Cs. Mészáros, and X. Xu. Implementation of interior point methods for large scale linear programming. In T. Terlaky, editor, *Interior-point methods of mathematical programming*, pages 189–252. Kluwer Academic Publishers, 1996.
- [ART03] E. D. Andersen, C. Roos, and T. Terlaky. On implementing a primal-dual interior-point method for conic quadratic optimization. *Math. Programming*, February 2003.
- [AY96] E. D. Andersen and Y. Ye. Combining interior-point and pivoting algorithms. *Management Sci.*, 42(12):1719–1731, December 1996.
- [And09] Erling D. Andersen. The homogeneous and self-dual model and algorithm for linear optimization. Technical Report TR-1-2009, MOSEK ApS, 2009. URL: <http://docs.mosek.com/whitepapers/homolo.pdf>.
- [And13] Erling D. Andersen. On formulating quadratic functions in optimization models. Technical Report TR-1-2013, MOSEK ApS, 2013. Last revised 23-feb-2016. URL: <http://docs.mosek.com/whitepapers/qmodel.pdf>.
- [BKVH07] S. Boyd, S.J. Kim, L. Vandenbergh, and A. Hassibi. A Tutorial on Geometric Programming. *Optimization and Engineering*, 8(1):67–127, 2007. Available at http://www.stanford.edu/~boyd/gp_tutorial.html.
- [Chv83] V. Chvátal. *Linear programming*. W.H. Freeman and Company, 1983.
- [GK00] Richard C. Grinold and Ronald N. Kahn. *Active portfolio management*. McGraw-Hill, New York, 2 edition, 2000.
- [Naz87] J. L. Nazareth. *Computer Solution of Linear Programs*. Oxford University Press, New York, 1987.
- [RTV97] C. Roos, T. Terlaky, and J. -Ph. Vial. *Theory and algorithms for linear optimization: an interior point approach*. John Wiley and Sons, New York, 1997.
- [Ste98] G. W. Stewart. *Matrix Algorithms. Volume 1: Basic decompositions*. SIAM, 1998.
- [Wal00] S. W. Wallace. Decision making under uncertainty: is sensitivity of any use. *Oper. Res.*, 48(1):20–25, January 2000.
- [Wol98] L. A. Wolsey. *Integer programming*. John Wiley and Sons, 1998.

Symbol Index

Classes

Env, 182
Env.Task, 182
Env.syrk, 189
Env.syevd, 189
Env.syeig, 189
Env.sparsetriangularsolvedense, 188
Env.setupthreads, 188
Env.set_Stream, 188
Env.putlicensewait, 188
Env.putlicensepath, 188
Env.putlicensedebug, 187
Env.putlicensecode, 187
Env.potrf, 187
Env.linkfiletostream, 187
Env.licensecleanup, 187
Env.getversion, 186
Env.getcodedesc, 186
Env.gemv, 186
Env.gemm, 185
Env.Env, 182
Env.echointro, 185
Env.dot, 184
Env.computesparsecholesky, 183
Env.checkoutlicense, 183
Env.checkinlicense, 183
Env.checkinall, 183
Env.axy, 182
Env.__del__, 182
Task, 190
Task.writetask, 262
Task.writesolution, 262
Task.writeparamfile, 262
Task.writejsonsol, 262
Task.writedata, 261
Task.updatesolutioninfo, 261
Task.toconic, 261
Task.Task, 190
Task.strtosk, 261
Task.strtoconetype, 261
Task.solvewithbasis, 260
Task.solutionsummary, 260
Task.solutiondef, 259
Task.setdefaults, 259
Task.set_Stream, 259
Task.set_Progress, 259
Task.set_InfoCallback, 259
Task.sensitivityreport, 259
Task.resizetask, 258
Task.removevars, 258
Task.removecons, 258
Task.removecones, 258
Task.removebarvars, 257
Task.readtask, 257
Task.readsummary, 257
Task.readsolution, 257
Task.readptfstring, 257
Task.readparamfile, 256
Task.readopfstring, 256
Task.readlpstring, 256
Task.readjsonstring, 256
Task.readdataformat, 256
Task.readdata, 256
Task.putyslice, 255
Task.puty, 255
Task.putxxslice, 255
Task.putxx, 255
Task.putxcslice, 254
Task.putxc, 254
Task.putvartypelist, 254
Task.putvartype, 254
Task.putvarsolutionj, 253
Task.putvarname, 253
Task.putvarboundsliceconst, 253
Task.putvarboundslice, 253
Task.putvarboundlistconst, 253
Task.putvarboundlist, 252
Task.putvarbound, 252
Task.puttaskname, 252
Task.putsukslice, 252
Task.putsuks, 251
Task.putsukslice, 251
Task.putsuc, 251
Task.putstrparam, 251
Task.putsolutionyi, 251
Task.putsolution, 250
Task.putsnxslice, 250
Task.putsnx, 250
Task.putslxslice, 249
Task.putslx, 249
Task.putslcslice, 249
Task.putslc, 249
Task.putskxslice, 248
Task.putskx, 248
Task.putskcslice, 248
Task.putskc, 248
Task.putqobjij, 247

Task.putqobj, 247
 Task.putqconk, 247
 Task.putqcon, 246
 Task.putparam, 246
 Task.putobjsense, 246
 Task.putobjname, 246
 Task.putnastrparam, 245
 Task.putnaintparam, 245
 Task.putnadoupparam, 245
 Task.putmaxnumvar, 245
 Task.putmaxnumqnz, 244
 Task.putmaxnumcone, 244
 Task.putmaxnumcon, 244
 Task.putmaxnumbarvar, 244
 Task.putmaxnumanz, 243
 Task.putintparam, 243
 Task.putdoupparam, 243
 Task.putcslice, 243
 Task.putconsolutioni, 242
 Task.putconname, 242
 Task.putconename, 242
 Task.putcone, 242
 Task.putconboundsliceconst, 241
 Task.putconboundslice, 241
 Task.putconboundlistconst, 241
 Task.putconboundlist, 241
 Task.putconbound, 240
 Task.putclist, 240
 Task.putcj, 240
 Task.putcfix, 240
 Task.putbarxj, 239
 Task.putbarvarname, 239
 Task.putbarsj, 239
 Task.putbarcj, 239
 Task.putbarcblocktriplet, 238
 Task.putbararowlist, 238
 Task.putbaraijlist, 238
 Task.putbaraij, 237
 Task.putbarablocktriplet, 237
 Task.putatruncatetol, 237
 Task.putarowslice, 236
 Task.putarowlist, 236
 Task.putarow, 236
 Task.putaijlist, 236
 Task.putaij, 235
 Task.putacolslice, 235
 Task.putacollist, 235
 Task.putacol, 234
 Task.primalsensitivity, 234
 Task.primalrepair, 233
 Task.optimizersummary, 233
 Task.optimizermt, 232
 Task.optimize, 232
 Task.onesolutionsummary, 232
 Task.linkfiletostream, 232
 Task.isstrparname, 232
 Task.isintparname, 231
 Task.isdoupname, 231
 Task.inputdata, 231
 Task.initbasissolve, 230
 Task.getyslice, 230
 Task.gety, 230
 Task.getxxslice, 230
 Task.getxx, 230
 Task.getxcslice, 229
 Task.getxc, 229
 Task.getvartypelist, 229
 Task.getvartype, 229
 Task.getvarnamelen, 229
 Task.getvarnameindex, 228
 Task.getvarname, 228
 Task.getvarboundslice, 228
 Task.getvarbound, 228
 Task.gettasknamelen, 227
 Task.gettaskname, 227
 Task.getsymmatinfo, 227
 Task.getsuxslice, 227
 Task.getsux, 227
 Task.getsucslice, 226
 Task.getsuc, 226
 Task.getstrparamlen, 226
 Task.getstrparam, 226
 Task.getsparsesymmat, 225
 Task.getsolutionslice, 225
 Task.getsolutioninfo, 225
 Task.getsolution, 223
 Task.getsolsta, 223
 Task.getsnxslice, 223
 Task.getsnx, 223
 Task.getslxslice, 222
 Task.getslx, 222
 Task.getslcslice, 222
 Task.getslc, 222
 Task.getskxslice, 221
 Task.getskx, 221
 Task.getskn, 221
 Task.getskcslice, 221
 Task.getskc, 221
 Task.getreducedcosts, 220
 Task.getqobjij, 220
 Task.getqobj, 220
 Task.getqconk, 220
 Task.getpviolvar, 219
 Task.getpviolcones, 219
 Task.getpviolcon, 218
 Task.getpviolbarvar, 218
 Task.getprosta, 218
 Task.getprobtype, 218
 Task.getprimalsolutionnorms, 217
 Task.getprimalobj, 217
 Task.getobjsense, 217
 Task.getobjnamelen, 217
 Task.getobjname, 217
 Task.getnumvar, 217
 Task.getnumsymmat, 216
 Task.getnumqobjnz, 216

- Task.getnumqconknz, 216
- Task.getnumparam, 216
- Task.getnumintvar, 216
- Task.getnumconemem, 216
- Task.getnumcone, 215
- Task.getnumcon, 215
- Task.getnumbarvar, 215
- Task.getnumbarcnz, 215
- Task.getnumbarblocktriplets, 215
- Task.getnumbaranz, 215
- Task.getnumbarablocktriplets, 214
- Task.getnumanz64, 214
- Task.getnumanz, 214
- Task.getmemusage, 214
- Task.getmaxnumvar, 214
- Task.getmaxnumqnz, 214
- Task.getmaxnumcone, 213
- Task.getmaxnumcon, 213
- Task.getmaxnumbarvar, 213
- Task.getmaxnumanz, 213
- Task.getlintinf, 213
- Task.getlenbarvarj, 212
- Task.getintparam, 212
- Task.getintinf, 212
- Task.getinfeasiblesubproblem, 212
- Task.getdviolvar, 211
- Task.getdviolcones, 211
- Task.getdviolcon, 211
- Task.getdviolbarvar, 210
- Task.getdualsolutionnorms, 210
- Task.getdualobj, 210
- Task.getdoupparam, 210
- Task.getdouinf, 209
- Task.getdimbarvarj, 209
- Task.getcslice, 209
- Task.getconnamelen, 209
- Task.getconnameindex, 208
- Task.getconname, 208
- Task.getconenamelen, 208
- Task.getconenameindex, 208
- Task.getconename, 208
- Task.getconeinfo, 207
- Task.getcone, 207
- Task.getconboundslice, 207
- Task.getconbound, 207
- Task.getclist, 206
- Task.getcj, 206
- Task.getcfix, 206
- Task.getc, 206
- Task.getbarxslice, 206
- Task.getbarxj, 205
- Task.getbarvarnamelen, 205
- Task.getbarvarnameindex, 205
- Task.getbarvarname, 205
- Task.getbarsslice, 204
- Task.getbarsj, 204
- Task.getbarcsparsity, 204
- Task.getbarcidxj, 204

- Task.getbarcidxinfo, 204
- Task.getbarcidx, 203
- Task.getbarcblocktriplet, 203
- Task.getbarasparsity, 203
- Task.getbaraidxinfo, 203
- Task.getbaraidxj, 202
- Task.getbaraidx, 202
- Task.getbarablocktriplet, 201
- Task.getatruncatetol, 201
- Task.getarowslicetrip, 201
- Task.getarowslicenumnz, 201
- Task.getarowslice, 201
- Task.getarownumnz, 200
- Task.getarow, 200
- Task.getapiecenumnz, 200
- Task.getaij, 200
- Task.getacolslicetrip, 199
- Task.getacolslicenumnz, 199
- Task.getacolslice, 199
- Task.getacolnumnz, 199
- Task.getacol, 198
- Task.generatevarnames, 198
- Task.generateconnames, 198
- Task.generateconenames, 198
- Task.dualsensitivity, 197
- Task.deletesolution, 197
- Task.commitchanges, 197
- Task.chgvarbound, 196
- Task.chgconbound, 196
- Task.checkmem, 196
- Task.basiscond, 195
- Task.asyncstop, 195
- Task.asyncpoll, 195
- Task.asyncoptimize, 195
- Task.asyncgetresult, 194
- Task.appendvars, 194
- Task.appendsparsesymmatlist, 194
- Task.appendsparsesymmat, 193
- Task.appendcons, 193
- Task.appendconesseq, 193
- Task.appendconeseq, 192
- Task.appendcone, 191
- Task.appendbarvars, 191
- Task.analyzesolution, 191
- Task.analyzeproblem, 191
- Task.analyzenames, 191
- Task.__del__, 190

Enumerations

- basindtype, 337
- basindtype.reserved, 337
- basindtype.no_error, 337
- basindtype.never, 337
- basindtype.if_feasible, 337
- basindtype.always, 337
- boundkey, 337
- boundkey.up, 337
- boundkey.ra, 337

boundkey.lo, 337
 boundkey.fx, 337
 boundkey.fr, 337
 branchdir, 355
 branchdir.up, 355
 branchdir.root_lp, 355
 branchdir.pseudocost, 355
 branchdir.near, 355
 branchdir.guided, 355
 branchdir.free, 355
 branchdir.far, 355
 branchdir.down, 355
 callbackcode, 339
 callbackcode.write_opf, 343
 callbackcode.update_primal_simplex_bi, 343
 callbackcode.update_primal_simplex, 343
 callbackcode.update_primal_bi, 343
 callbackcode.update_presolve, 343
 callbackcode.update_dual_simplex_bi, 343
 callbackcode.update_dual_simplex, 343
 callbackcode.update_dual_bi, 343
 callbackcode.solving_remote, 343
 callbackcode.read_opf_section, 343
 callbackcode.read_opf, 343
 callbackcode.primal_simplex, 343
 callbackcode.new_int_mio, 343
 callbackcode.intpnt, 343
 callbackcode.im_simplex_bi, 342
 callbackcode.im_simplex, 342
 callbackcode.im_root_cutgen, 342
 callbackcode.im_read, 342
 callbackcode.im_qo_reformulate, 342
 callbackcode.im_primal_simplex, 342
 callbackcode.im_primal_sensitivity, 342
 callbackcode.im_primal_bi, 342
 callbackcode.im_presolve, 342
 callbackcode.im_order, 342
 callbackcode.im_mio_primal_simplex, 342
 callbackcode.im_mio_intpnt, 342
 callbackcode.im_mio_dual_simplex, 342
 callbackcode.im_mio, 342
 callbackcode.im_lu, 342
 callbackcode.im_license_wait, 342
 callbackcode.im_intpnt, 342
 callbackcode.im_full_convexity_check, 342
 callbackcode.im_dual_simplex, 342
 callbackcode.im_dual_sensitivity, 342
 callbackcode.im_dual_bi, 342
 callbackcode.im_conic, 341
 callbackcode.im_bi, 341
 callbackcode.end_write, 341
 callbackcode.end_to_conic, 341
 callbackcode.end_simplex_bi, 341
 callbackcode.end_simplex, 341
 callbackcode.end_root_cutgen, 341
 callbackcode.end_read, 341
 callbackcode.end_qcqp_reformulate, 341
 callbackcode.end_primal_simplex_bi, 341
 callbackcode.end_primal_simplex, 341
 callbackcode.end_primal_setup_bi, 341
 callbackcode.end_primal_sensitivity, 341
 callbackcode.end_primal_repair, 341
 callbackcode.end_primal_bi, 341
 callbackcode.end_presolve, 341
 callbackcode.end_optimizer, 341
 callbackcode.end_mio, 341
 callbackcode.end_license_wait, 341
 callbackcode.end_intpnt, 341
 callbackcode.end_infeas_ana, 341
 callbackcode.end_full_convexity_check, 341
 callbackcode.end_dual_simplex_bi, 341
 callbackcode.end_dual_simplex, 340
 callbackcode.end_dual_setup_bi, 340
 callbackcode.end_dual_sensitivity, 340
 callbackcode.end_dual_bi, 340
 callbackcode.end_conic, 340
 callbackcode.end_bi, 340
 callbackcode.dual_simplex, 340
 callbackcode.conic, 340
 callbackcode.begin_write, 340
 callbackcode.begin_to_conic, 340
 callbackcode.begin_simplex_bi, 340
 callbackcode.begin_simplex, 340
 callbackcode.begin_root_cutgen, 340
 callbackcode.begin_read, 340
 callbackcode.begin_qcqp_reformulate, 340
 callbackcode.begin_primal_simplex_bi, 340
 callbackcode.begin_primal_simplex, 340
 callbackcode.begin_primal_setup_bi, 340
 callbackcode.begin_primal_sensitivity, 340
 callbackcode.begin_primal_repair, 340
 callbackcode.begin_primal_bi, 340
 callbackcode.begin_presolve, 340
 callbackcode.begin_optimizer, 340
 callbackcode.begin_mio, 340
 callbackcode.begin_license_wait, 339
 callbackcode.begin_intpnt, 339
 callbackcode.begin_infeas_ana, 339
 callbackcode.begin_full_convexity_check, 339
 callbackcode.begin_dual_simplex_bi, 339
 callbackcode.begin_dual_simplex, 339
 callbackcode.begin_dual_setup_bi, 339
 callbackcode.begin_dual_sensitivity, 339
 callbackcode.begin_dual_bi, 339
 callbackcode.begin_conic, 339
 callbackcode.begin_bi, 339
 checkconvexitytype, 343
 checkconvexitytype.simple, 343
 checkconvexitytype.none, 343
 checkconvexitytype.full, 343
 compresstype, 343
 compresstype.zstd, 344
 compresstype.none, 343
 compresstype.gzip, 344
 compresstype.free, 343

- conetype, 344
- conetype.zero, 344
- conetype.rquad, 344
- conetype.quad, 344
- conetype.ppow, 344
- conetype.pexp, 344
- conetype.dpow, 344
- conetype.dexp, 344
- dataformat, 344
- dataformat.task, 345
- dataformat.ptf, 345
- dataformat.op, 345
- dataformat.mps, 344
- dataformat.lp, 344
- dataformat.json_task, 345
- dataformat.free_mps, 345
- dataformat.extension, 344
- dataformat.cb, 345
- dinfitem, 345
- dinfitem.to_conic_time, 349
- dinfitem.sol_itr_pviolvar, 349
- dinfitem.sol_itr_pviolcones, 349
- dinfitem.sol_itr_pviolcon, 349
- dinfitem.sol_itr_pviolbarvar, 349
- dinfitem.sol_itr_primal_obj, 349
- dinfitem.sol_itr_nrm_y, 349
- dinfitem.sol_itr_nrm_xx, 349
- dinfitem.sol_itr_nrm_xc, 349
- dinfitem.sol_itr_nrm_sux, 349
- dinfitem.sol_itr_nrm_suc, 349
- dinfitem.sol_itr_nrm_snx, 349
- dinfitem.sol_itr_nrm_slx, 349
- dinfitem.sol_itr_nrm_slc, 349
- dinfitem.sol_itr_nrm_barx, 348
- dinfitem.sol_itr_nrm_bars, 348
- dinfitem.sol_itr_dviolvar, 348
- dinfitem.sol_itr_dviolcones, 348
- dinfitem.sol_itr_dviolcon, 348
- dinfitem.sol_itr_dviolbarvar, 348
- dinfitem.sol_itr_dual_obj, 348
- dinfitem.sol_itg_pviolvar, 348
- dinfitem.sol_itg_pviolitg, 348
- dinfitem.sol_itg_pviolcones, 348
- dinfitem.sol_itg_pviolcon, 348
- dinfitem.sol_itg_pviolbarvar, 348
- dinfitem.sol_itg_primal_obj, 348
- dinfitem.sol_itg_nrm_xx, 348
- dinfitem.sol_itg_nrm_xc, 348
- dinfitem.sol_itg_nrm_barx, 348
- dinfitem.sol_bas_pviolvar, 348
- dinfitem.sol_bas_pviolcon, 348
- dinfitem.sol_bas_primal_obj, 348
- dinfitem.sol_bas_nrm_y, 347
- dinfitem.sol_bas_nrm_xx, 347
- dinfitem.sol_bas_nrm_xc, 347
- dinfitem.sol_bas_nrm_sux, 347
- dinfitem.sol_bas_nrm_suc, 347
- dinfitem.sol_bas_nrm_slx, 347
- dinfitem.sol_bas_nrm_slc, 347
- dinfitem.sol_bas_nrm_barx, 347
- dinfitem.sol_bas_dviolvar, 347
- dinfitem.sol_bas_dviolcon, 347
- dinfitem.sol_bas_dual_obj, 347
- dinfitem.sim_time, 347
- dinfitem.sim_primal_time, 347
- dinfitem.sim_obj, 347
- dinfitem.sim_feas, 347
- dinfitem.sim_dual_time, 347
- dinfitem.rd_time, 347
- dinfitem.qcqp_reformulate_worst_cholesky_diag_scaling, 347
- dinfitem.qcqp_reformulate_worst_cholesky_column_scaling, 347
- dinfitem.qcqp_reformulate_time, 347
- dinfitem.qcqp_reformulate_max_perturbation, 347
- dinfitem.primal_repair_penalty_obj, 347
- dinfitem.presolve_time, 347
- dinfitem.presolve_lindep_time, 347
- dinfitem.presolve_eli_time, 346
- dinfitem.optimizer_time, 346
- dinfitem.mio_user_obj_cut, 346
- dinfitem.mio_time, 346
- dinfitem.mio_root_presolve_time, 346
- dinfitem.mio_root_optimizer_time, 346
- dinfitem.mio_root_cutgen_time, 346
- dinfitem.mio_probing_time, 346
- dinfitem.mio_obj_rel_gap, 346
- dinfitem.mio_obj_int, 346
- dinfitem.mio_obj_bound, 346
- dinfitem.mio_obj_abs_gap, 346
- dinfitem.mio_knapsack_cover_separation_time, 346
- dinfitem.mio_implied_bound_time, 346
- dinfitem.mio_gmi_separation_time, 346
- dinfitem.mio_dual_bound_after_presolve, 346
- dinfitem.mio_construct_solution_obj, 346
- dinfitem.mio_cmir_separation_time, 346
- dinfitem.mio_clique_separation_time, 346
- dinfitem.intpnt_time, 345
- dinfitem.intpnt_primal_obj, 345
- dinfitem.intpnt_primal_feas, 345
- dinfitem.intpnt_order_time, 345
- dinfitem.intpnt_opt_status, 345
- dinfitem.intpnt_factor_num_flops, 345
- dinfitem.intpnt_dual_obj, 345
- dinfitem.intpnt_dual_feas, 345
- dinfitem.bi_time, 345
- dinfitem.bi_primal_time, 345
- dinfitem.bi_dual_time, 345
- dinfitem.bi_clean_time, 345
- dinfitem.bi_clean_primal_time, 345
- dinfitem.bi_clean_dual_time, 345
- dparam, 275
- feature, 349
- feature.pts, 349

feature.pton, 349
 iinfitem, 350
 iinfitem.sto_num_a_realloc, 354
 iinfitem.sol_itr_solsta, 354
 iinfitem.sol_itr_prosta, 354
 iinfitem.sol_itg_solsta, 354
 iinfitem.sol_itg_prosta, 354
 iinfitem.sol_bas_solsta, 354
 iinfitem.sol_bas_prosta, 354
 iinfitem.sim_solve_dual, 354
 iinfitem.sim_primal_iter, 354
 iinfitem.sim_primal_inf_iter, 354
 iinfitem.sim_primal_hotstart_lu, 354
 iinfitem.sim_primal_hotstart, 354
 iinfitem.sim_primal_deg_iter, 354
 iinfitem.sim_numvar, 354
 iinfitem.sim_numcon, 354
 iinfitem.sim_dual_iter, 354
 iinfitem.sim_dual_inf_iter, 353
 iinfitem.sim_dual_hotstart_lu, 353
 iinfitem.sim_dual_hotstart, 353
 iinfitem.sim_dual_deg_iter, 353
 iinfitem.rd_prototype, 353
 iinfitem.rd_numvar, 353
 iinfitem.rd_numq, 353
 iinfitem.rd_numintvar, 353
 iinfitem.rd_numcone, 353
 iinfitem.rd_numcon, 353
 iinfitem.rd_numbarvar, 353
 iinfitem.purify_primal_success, 353
 iinfitem.purify_dual_success, 353
 iinfitem.optimize_response, 353
 iinfitem.opt_numvar, 353
 iinfitem.opt_numcon, 353
 iinfitem.mio_user_obj_cut, 353
 iinfitem.mio_total_num_cuts, 353
 iinfitem.mio_relgap_satisfied, 353
 iinfitem.mio_presolved_numvar, 353
 iinfitem.mio_presolved_numrqcones, 353
 iinfitem.mio_presolved_numqcones, 353
 iinfitem.mio_presolved_numppowcones, 353
 iinfitem.mio_presolved_numpexpcones, 353
 iinfitem.mio_presolved_numintconevar, 352
 iinfitem.mio_presolved_numint, 352
 iinfitem.mio_presolved_numdpowcones, 352
 iinfitem.mio_presolved_numdexpcones, 352
 iinfitem.mio_presolved_numcontconevar, 352
 iinfitem.mio_presolved_numcont, 352
 iinfitem.mio_presolved_numconevar, 352
 iinfitem.mio_presolved_numcone, 352
 iinfitem.mio_presolved_numcon, 352
 iinfitem.mio_presolved_numbinconevar, 352
 iinfitem.mio_presolved_numbin, 352
 iinfitem.mio_obj_bound_defined, 352
 iinfitem.mio_numvar, 352
 iinfitem.mio_numrqcones, 352
 iinfitem.mio_numqcones, 352
 iinfitem.mio_numppowcones, 352
 iinfitem.mio_numpexpcones, 352
 iinfitem.mio_numintconevar, 352
 iinfitem.mio_numint, 352
 iinfitem.mio_numdpowcones, 352
 iinfitem.mio_numdexpcones, 352
 iinfitem.mio_numcontconevar, 352
 iinfitem.mio_numcont, 352
 iinfitem.mio_numconevar, 352
 iinfitem.mio_numcone, 352
 iinfitem.mio_numcon, 351
 iinfitem.mio_numbinconevar, 351
 iinfitem.mio_numbin, 351
 iinfitem.mio_num_repeated_presolve, 351
 iinfitem.mio_num_relax, 351
 iinfitem.mio_num_knapsack_cover_cuts, 351
 iinfitem.mio_num_int_solutions, 351
 iinfitem.mio_num_implied_bound_cuts, 351
 iinfitem.mio_num_gomory_cuts, 351
 iinfitem.mio_num_cmir_cuts, 351
 iinfitem.mio_num_clique_cuts, 351
 iinfitem.mio_num_branch, 351
 iinfitem.mio_num_active_nodes, 351
 iinfitem.mio_node_depth, 351
 iinfitem.mio_construct_solution, 351
 iinfitem.mio_clique_table_size, 351
 iinfitem.mio_absgap_satisfied, 351
 iinfitem.intpnt_solve_dual, 351
 iinfitem.intpnt_num_threads, 351
 iinfitem.intpnt_iter, 351
 iinfitem.intpnt_factor_dim_dense, 351
 iinfitem.ana_pro_num_var_up, 351
 iinfitem.ana_pro_num_var_ra, 351
 iinfitem.ana_pro_num_var_lo, 350
 iinfitem.ana_pro_num_var_int, 350
 iinfitem.ana_pro_num_var_fr, 350
 iinfitem.ana_pro_num_var_eq, 350
 iinfitem.ana_pro_num_var_cont, 350
 iinfitem.ana_pro_num_var_bin, 350
 iinfitem.ana_pro_num_var, 350
 iinfitem.ana_pro_num_con_up, 350
 iinfitem.ana_pro_num_con_ra, 350
 iinfitem.ana_pro_num_con_lo, 350
 iinfitem.ana_pro_num_con_fr, 350
 iinfitem.ana_pro_num_con_eq, 350
 iinfitem.ana_pro_num_con, 350
 inftype, 354
 inftype.lint_type, 354
 inftype.int_type, 354
 inftype.dou_type, 354
 intpnthotstart, 339
 intpnthotstart.primal_dual, 339
 intpnthotstart.primal, 339
 intpnthotstart.none, 339
 intpnthotstart.dual, 339
 iomode, 354
 iomode.write, 354
 iomode.readwrite, 354
 iomode.read, 354

- iparam, 285
- liinfitem, 349
- liinfitem.rd_numqnz, 350
- liinfitem.rd_numanz, 350
- liinfitem.mio_simplex_iter, 350
- liinfitem.mio_presolved_anz, 350
- liinfitem.mio_intpnt_iter, 350
- liinfitem.mio_anz, 350
- liinfitem.intpnt_factor_num_nz, 350
- liinfitem.bi_primal_iter, 350
- liinfitem.bi_dual_iter, 349
- liinfitem.bi_clean_primal_iter, 349
- liinfitem.bi_clean_primal_deg_iter, 349
- liinfitem.bi_clean_dual_iter, 349
- liinfitem.bi_clean_dual_deg_iter, 349
- mark, 337
- mark.up, 338
- mark.lo, 338
- miocontsoltype, 355
- miocontsoltype.root, 355
- miocontsoltype.none, 355
- miocontsoltype.itg_rel, 355
- miocontsoltype.itg, 355
- miomode, 355
- miomode.satisfied, 355
- miomode.ignored, 355
- mionodeseltype, 355
- mionodeseltype.pseudo, 355
- mionodeseltype.free, 355
- mionodeseltype.first, 355
- mionodeseltype.best, 355
- mpsformat, 355
- mpsformat.strict, 356
- mpsformat.relaxed, 356
- mpsformat.free, 356
- mpsformat.cplex, 356
- nametype, 344
- nametype.mps, 344
- nametype.lp, 344
- nametype.gen, 344
- objsense, 356
- objsense.minimize, 356
- objsense.maximize, 356
- onoffkey, 356
- onoffkey.on, 356
- onoffkey.off, 356
- optimizertype, 356
- optimizertype.primal_simplex, 356
- optimizertype.mixed_int, 356
- optimizertype.intpnt, 356
- optimizertype.free_simplex, 356
- optimizertype.free, 356
- optimizertype.dual_simplex, 356
- optimizertype.conic, 356
- orderingtype, 356
- orderingtype.try_graphpar, 356
- orderingtype.none, 357
- orderingtype.free, 356
- orderingtype.force_graphpar, 356
- orderingtype.experimental, 356
- orderingtype.appminloc, 356
- parametertype, 357
- parametertype.str_type, 357
- parametertype.invalid_type, 357
- parametertype.int_type, 357
- parametertype.dou_type, 357
- presolvemode, 357
- presolvemode.on, 357
- presolvemode.off, 357
- presolvemode.free, 357
- problemitem, 357
- problemitem.var, 357
- problemitem.cone, 357
- problemitem.con, 357
- problemtyp, 357
- problemtyp.qo, 357
- problemtyp.qcqp, 357
- problemtyp.mixed, 357
- problemtyp.lo, 357
- problemtyp.conic, 357
- prosta, 357
- prosta.unknown, 357
- prosta.prim_infeas_or_unbounded, 358
- prosta.prim_infeas, 357
- prosta.prim_feas, 357
- prosta.prim_and_dual_infeas, 358
- prosta.prim_and_dual_feas, 357
- prosta.ill_posed, 358
- prosta.dual_infeas, 358
- prosta.dual_feas, 357
- purify, 339
- purify.primal_dual, 339
- purify.primal, 339
- purify.none, 339
- purify.dual, 339
- purify.auto, 339
- rescode, 319
- rescodetype, 358
- rescodetype.wrn, 358
- rescodetype.unk, 358
- rescodetype.trm, 358
- rescodetype.ok, 358
- rescodetype.err, 358
- scalingmethod, 358
- scalingmethod.pow2, 358
- scalingmethod.free, 358
- scalingtype, 358
- scalingtype.none, 358
- scalingtype.moderate, 358
- scalingtype.free, 358
- scalingtype.aggressive, 358
- scopr, 344
- scopr.sqrt, 344
- scopr.pow, 344
- scopr.log, 344
- scopr.exp, 344

- scopr.ent, 344
- sensitivitytype, 358
- sensitivitytype.basis, 358
- simdegen, 338
- simdegen.none, 338
- simdegen.moderate, 338
- simdegen.minimum, 338
- simdegen.free, 338
- simdegen.aggressive, 338
- simdupvec, 338
- simdupvec.on, 338
- simdupvec.off, 338
- simdupvec.free, 338
- simhotstart, 338
- simhotstart.status_keys, 339
- simhotstart.none, 338
- simhotstart.free, 338
- simreform, 338
- simreform.on, 338
- simreform.off, 338
- simreform.free, 338
- simreform.aggressive, 338
- simseltype, 358
- simseltype.se, 359
- simseltype.partial, 359
- simseltype.full, 359
- simseltype.free, 358
- simseltype.devex, 359
- simseltype.ase, 359
- solitem, 359
- solitem.y, 359
- solitem.xx, 359
- solitem.xc, 359
- solitem.sux, 359
- solitem.suc, 359
- solitem.snx, 359
- solitem.slx, 359
- solitem.slc, 359
- solsta, 359
- solsta.unknown, 359
- solsta.prim_infeas_cer, 359
- solsta.prim_illposed_cer, 359
- solsta.prim_feas, 359
- solsta.prim_and_dual_feas, 359
- solsta.optimal, 359
- solsta.integer_optimal, 360
- solsta.dual_infeas_cer, 359
- solsta.dual_illposed_cer, 359
- solsta.dual_feas, 359
- solttype, 360
- solttype.itr, 360
- solttype.itg, 360
- solttype.bas, 360
- solveform, 360
- solveform.primal, 360
- solveform.free, 360
- solveform.dual, 360
- sparam, 316

- stakey, 360
- stakey.upr, 360
- stakey.unk, 360
- stakey.supbas, 360
- stakey.low, 360
- stakey.inf, 360
- stakey.fix, 360
- stakey.bas, 360
- startpointtype, 360
- startpointtype.satisfy_bounds, 360
- startpointtype.guess, 360
- startpointtype.free, 360
- startpointtype.constant, 360
- streamtype, 360
- streamtype.wrn, 361
- streamtype.msg, 361
- streamtype.log, 360
- streamtype.err, 361
- symmatttype, 344
- symmatttype.sparse, 344
- transpose, 338
- transpose.yes, 338
- transpose.no, 338
- uplo, 338
- uplo.up, 338
- uplo.lo, 338
- value, 361
- value.max_str_len, 361
- value.license_buffer_length, 361
- variabletype, 361
- variabletype.type_int, 361
- variabletype.type_cont, 361
- xmlwriteroutputtype, 358
- xmlwriteroutputtype.row, 358
- xmlwriteroutputtype.col, 358

Exceptions

- Error, 263
- MosekException, 263

Parameters

- Double parameters, 275
- dparam.ana_sol_infeas_tol, 275
- dparam.basis_rel_tol_s, 275
- dparam.basis_tol_s, 275
- dparam.basis_tol_x, 275
- dparam.check_convexity_rel_tol, 275
- dparam.data_sym_mat_tol, 275
- dparam.data_sym_mat_tol_huge, 276
- dparam.data_sym_mat_tol_large, 276
- dparam.data_tol_aij_huge, 276
- dparam.data_tol_aij_large, 276
- dparam.data_tol_bound_inf, 276
- dparam.data_tol_bound_wrn, 276
- dparam.data_tol_c_huge, 277
- dparam.data_tol_cj_large, 277
- dparam.data_tol_qij, 277
- dparam.data_tol_x, 277

dparam.intpnt_co_tol_dfeas, 277
dparam.intpnt_co_tol_infeas, 277
dparam.intpnt_co_tol_mu_red, 278
dparam.intpnt_co_tol_near_rel, 278
dparam.intpnt_co_tol_pfeas, 278
dparam.intpnt_co_tol_rel_gap, 278
dparam.intpnt_qo_tol_dfeas, 278
dparam.intpnt_qo_tol_infeas, 279
dparam.intpnt_qo_tol_mu_red, 279
dparam.intpnt_qo_tol_near_rel, 279
dparam.intpnt_qo_tol_pfeas, 279
dparam.intpnt_qo_tol_rel_gap, 279
dparam.intpnt_tol_dfeas, 279
dparam.intpnt_tol_dsafe, 280
dparam.intpnt_tol_infeas, 280
dparam.intpnt_tol_mu_red, 280
dparam.intpnt_tol_path, 280
dparam.intpnt_tol_pfeas, 280
dparam.intpnt_tol_psafe, 280
dparam.intpnt_tol_rel_gap, 281
dparam.intpnt_tol_rel_step, 281
dparam.intpnt_tol_step_size, 281
dparam.lower_obj_cut, 281
dparam.lower_obj_cut_finite_trh, 281
dparam.mio_max_time, 282
dparam.mio_rel_gap_const, 282
dparam.mio_tol_abs_gap, 282
dparam.mio_tol_abs_relax_int, 282
dparam.mio_tol_feas, 282
dparam.mio_tol_rel_dual_bound_improvement, 282
dparam.mio_tol_rel_gap, 283
dparam.optimizer_max_time, 283
dparam.presolve_tol_abs_lindep, 283
dparam.presolve_tol_aij, 283
dparam.presolve_tol_rel_lindep, 283
dparam.presolve_tol_s, 283
dparam.presolve_tol_x, 283
dparam.qcqp_reformulate_rel_drop_tol, 284
dparam.semidefinite_tol_approx, 284
dparam.sim_lu_tol_rel_piv, 284
dparam.simplex_abs_tol_piv, 284
dparam.upper_obj_cut, 284
dparam.upper_obj_cut_finite_trh, 284
Integer parameters, 285
iparam.ana_sol_basis, 285
iparam.ana_sol_print_violated, 285
iparam.auto_sort_a_before_opt, 285
iparam.auto_update_sol_info, 285
iparam.basis_solve_use_plus_one, 285
iparam.bi_clean_optimizer, 286
iparam.bi_ignore_max_iter, 286
iparam.bi_ignore_num_error, 286
iparam.bi_max_iterations, 286
iparam.cache_license, 286
iparam.check_convexity, 287
iparam.compress_statfile, 287
iparam.infeas_generic_names, 287
iparam.infeas_prefer_primal, 287
iparam.infeas_report_auto, 287
iparam.infeas_report_level, 287
iparam.intpnt_basis, 288
iparam.intpnt_diff_step, 288
iparam.intpnt_hotstart, 288
iparam.intpnt_max_iterations, 288
iparam.intpnt_max_num_cor, 288
iparam.intpnt_max_num_refinement_steps, 288
iparam.intpnt_multi_thread, 289
iparam.intpnt_off_col_trh, 289
iparam.intpnt_order_gp_num_seeds, 289
iparam.intpnt_order_method, 289
iparam.intpnt_purify, 289
iparam.intpnt_regularization_use, 289
iparam.intpnt_scaling, 290
iparam.intpnt_solve_form, 290
iparam.intpnt_starting_point, 290
iparam.license_debug, 290
iparam.license_pause_time, 290
iparam.license_suppress_expire_wrns, 290
iparam.license_trh_expiry_wrn, 291
iparam.license_wait, 291
iparam.log, 291
iparam.log_ana_pro, 291
iparam.log_bi, 291
iparam.log_bi_freq, 292
iparam.log_check_convexity, 292
iparam.log_cut_second_opt, 292
iparam.log_expand, 292
iparam.log_feas_repair, 292
iparam.log_file, 292
iparam.log_include_summary, 293
iparam.log_infeas_ana, 293
iparam.log_intpnt, 293
iparam.log_local_info, 293
iparam.log_mio, 293
iparam.log_mio_freq, 293
iparam.log_order, 294
iparam.log_presolve, 294
iparam.log_response, 294
iparam.log_sensitivity, 294
iparam.log_sensitivity_opt, 294
iparam.log_sim, 295
iparam.log_sim_freq, 295
iparam.log_sim_minior, 295
iparam.log_storage, 295
iparam.max_num_warnings, 295
iparam.mio_branch_dir, 295
iparam.mio_conic_outer_approximation, 296
iparam.mio_cut_clique, 296
iparam.mio_cut_cmir, 296
iparam.mio_cut_gmi, 296
iparam.mio_cut_implied_bound, 296
iparam.mio_cut_knapsack_cover, 296
iparam.mio_cut_selection_level, 296
iparam.mio_feaspump_level, 297
iparam.mio_heuristic_level, 297

- iparam.mio_max_num_branches, 297
- iparam.mio_max_num_relaxs, 297
- iparam.mio_max_num_root_cut_rounds, 298
- iparam.mio_max_num_solutions, 298
- iparam.mio_mode, 298
- iparam.mio_node_optimizer, 298
- iparam.mio_node_selection, 298
- iparam.mio_perspective_reformulate, 298
- iparam.mio_probing_level, 299
- iparam.mio_propagate_objective_constraint, 299
- iparam.mio_rins_max_nodes, 299
- iparam.mio_root_optimizer, 299
- iparam.mio_root_repeat_presolve_level, 299
- iparam.mio_seed, 300
- iparam.mio_vb_detection_level, 300
- iparam.mt_spincount, 300
- iparam.num_threads, 300
- iparam.opf_write_header, 301
- iparam.opf_write_hints, 301
- iparam.opf_write_line_length, 301
- iparam.opf_write_parameters, 301
- iparam.opf_write_problem, 301
- iparam.opf_write_sol_bas, 301
- iparam.opf_write_sol_itg, 301
- iparam.opf_write_sol_itr, 301
- iparam.opf_write_solutions, 302
- iparam.optimizer, 302
- iparam.param_read_case_name, 302
- iparam.param_read_ign_error, 302
- iparam.presolve_eliminator_max_fill, 302
- iparam.presolve_eliminator_max_num_tries, 302
- iparam.presolve_level, 303
- iparam.presolve_lindep_abs_work_trh, 303
- iparam.presolve_lindep_rel_work_trh, 303
- iparam.presolve_lindep_use, 303
- iparam.presolve_max_num_pass, 303
- iparam.presolve_max_num_reductions, 303
- iparam.presolve_use, 304
- iparam.primal_repair_optimizer, 304
- iparam.ptf_write_transform, 304
- iparam.read_debug, 304
- iparam.read_keep_free_con, 304
- iparam.read_lp_drop_new_vars_in_bou, 305
- iparam.read_lp_quoted_names, 305
- iparam.read_mps_format, 305
- iparam.read_mps_width, 305
- iparam.read_task_ignore_param, 305
- iparam.remove_unused_solutions, 305
- iparam.sensitivity_all, 306
- iparam.sensitivity_optimizer, 306
- iparam.sensitivity_type, 306
- iparam.sim_basis_factor_use, 306
- iparam.sim_degen, 306
- iparam.sim_dual_crash, 306
- iparam.sim_dual_phaseone_method, 307
- iparam.sim_dual_restrict_selection, 307
- iparam.sim_dual_selection, 307
- iparam.sim_exploit_dupvec, 307
- iparam.sim_hotstart, 307
- iparam.sim_hotstart_lu, 307
- iparam.sim_max_iterations, 308
- iparam.sim_max_num_setbacks, 308
- iparam.sim_non_singular, 308
- iparam.sim_primal_crash, 308
- iparam.sim_primal_phaseone_method, 308
- iparam.sim_primal_restrict_selection, 308
- iparam.sim_primal_selection, 309
- iparam.sim_refactor_freq, 309
- iparam.sim_reformulation, 309
- iparam.sim_save_lu, 309
- iparam.sim_scaling, 309
- iparam.sim_scaling_method, 309
- iparam.sim_seed, 310
- iparam.sim_solve_form, 310
- iparam.sim_stability_priority, 310
- iparam.sim_switch_optimizer, 310
- iparam.sol_filter_keep_basic, 310
- iparam.sol_filter_keep_ranged, 310
- iparam.sol_read_name_width, 311
- iparam.sol_read_width, 311
- iparam.solution_callback, 311
- iparam.timing_level, 311
- iparam.write_bas_constraints, 311
- iparam.write_bas_head, 311
- iparam.write_bas_variables, 312
- iparam.write_compression, 312
- iparam.write_data_param, 312
- iparam.write_free_con, 312
- iparam.write_generic_names, 312
- iparam.write_generic_names_io, 312
- iparam.write_ignore_incompatible_items, 312
- iparam.write_int_constraints, 313
- iparam.write_int_head, 313
- iparam.write_int_variables, 313
- iparam.write_lp_full_obj, 313
- iparam.write_lp_line_width, 313
- iparam.write_lp_quoted_names, 313
- iparam.write_lp_strict_format, 314
- iparam.write_lp_terms_per_line, 314
- iparam.write_mps_format, 314
- iparam.write_mps_int, 314
- iparam.write_precision, 314
- iparam.write_sol_barvariables, 314
- iparam.write_sol_constraints, 315
- iparam.write_sol_head, 315
- iparam.write_sol_ignore_invalid_names, 315
- iparam.write_sol_variables, 315
- iparam.write_task_inc_sol, 315
- iparam.write_xml_mode, 315
- String parameters, 316
- sparam.bas_sol_file_name, 316
- sparam.data_file_name, 316
- sparam.debug_file_name, 316
- sparam.int_sol_file_name, 316

- sparam.itr_sol_file_name, 316
- sparam.mio_debug_string, 316
- sparam.param_comment_sign, 316
- sparam.param_read_file_name, 317
- sparam.param_write_file_name, 317
- sparam.read_mps_bou_name, 317
- sparam.read_mps_obj_name, 317
- sparam.read_mps_ran_name, 317
- sparam.read_mps_rhs_name, 317
- sparam.remote_access_token, 317
- sparam.sensitivity_file_name, 318
- sparam.sensitivity_res_file_name, 318
- sparam.sol_filter_xc_low, 318
- sparam.sol_filter_xc_upr, 318
- sparam.sol_filter_xx_low, 318
- sparam.sol_filter_xx_upr, 318
- sparam.stat_file_name, 318
- sparam.stat_key, 319
- sparam.stat_name, 319
- sparam.write_lp_gen_var_name, 319

Response codes

Termination, 319

rescode.ok, 319

rescode.trm_internal, 320

rescode.trm_internal_stop, 320

rescode.trm_max_iterations, 319

rescode.trm_max_num_setbacks, 320

rescode.trm_max_time, 319

rescode.trm_mio_num_branches, 319

rescode.trm_mio_num_relaxs, 319

rescode.trm_num_max_num_int_solutions, 319

rescode.trm_numerical_problem, 320

rescode.trm_objective_range, 319

rescode.trm_stall, 320

rescode.trm_user_callback, 320

Warnings, 320

rescode.wrn_ana_almost_int_bounds, 322

rescode.wrn_ana_c_zero, 322

rescode.wrn_ana_close_bounds, 322

rescode.wrn_ana_empty_cols, 322

rescode.wrn_ana_large_bounds, 322

rescode.wrn_dropped_nz_qobj, 321

rescode.wrn_duplicate_barvariable_names, 322

rescode.wrn_duplicate_cone_names, 322

rescode.wrn_duplicate_constraint_names, 322

rescode.wrn_duplicate_variable_names, 322

rescode.wrn_eliminator_space, 322

rescode.wrn_empty_name, 321

rescode.wrn_exp_cones_with_variables_fixed_at_zero, 322

rescode.wrn_ignore_integer, 321

rescode.wrn_incomplete_linear_dependency_checks, 321

rescode.wrn_large_aij, 320

rescode.wrn_large_bound, 320

rescode.wrn_large_cj, 320

rescode.wrn_large_con_fx, 320

rescode.wrn_large_lo_bound, 320

rescode.wrn_large_up_bound, 320

rescode.wrn_license_expire, 321

rescode.wrn_license_feature_expire, 321

rescode.wrn_license_server, 321

rescode.wrn_lp_drop_variable, 321

rescode.wrn_lp_old_quad_format, 320

rescode.wrn_mio_infeasible_final, 321

rescode.wrn_mps_split_bou_vector, 320

rescode.wrn_mps_split_ran_vector, 320

rescode.wrn_mps_split_rhs_vector, 320

rescode.wrn_name_max_len, 320

rescode.wrn_no_dualizer, 322

rescode.wrn_no_global_optimizer, 321

rescode.wrn_nz_in_upr_tri, 321

rescode.wrn_open_param_file, 320

rescode.wrn_param_ignored_cmio, 321

rescode.wrn_param_name_dou, 321

rescode.wrn_param_name_int, 321

rescode.wrn_param_name_str, 321

rescode.wrn_param_str_value, 321

rescode.wrn_pow_cones_with_root_fixed_at_zero, 322

rescode.wrn_presolve_outofspace, 322

rescode.wrn_quad_cones_with_root_fixed_at_zero, 322

rescode.wrn_rquad_cones_with_root_fixed_at_zero, 322

rescode.wrn_sol_file_ignored_con, 321

rescode.wrn_sol_file_ignored_var, 321

rescode.wrn_sol_filter, 321

rescode.wrn_spar_max_len, 320

rescode.wrn_sym_mat_large, 322

rescode.wrn_too_few_basis_vars, 321

rescode.wrn_too_many_basis_vars, 321

rescode.wrn_undef_sol_file_name, 321

rescode.wrn_using_generic_names, 321

rescode.wrn_write_changed_names, 322

rescode.wrn_write_discarded_cfix, 322

rescode.wrn_zero_aij, 320

rescode.wrn_zeros_in_sparse_col, 321

rescode.wrn_zeros_in_sparse_row, 321

Errors, 323

rescode.err_ad_invalid_codelist, 334

rescode.err_api_array_too_small, 333

rescode.err_api_cb_connect, 333

rescode.err_api_fatal_error, 333

rescode.err_api_internal, 333

rescode.err_appending_too_big_cone, 330

rescode.err_arg_is_too_large, 328

rescode.err_arg_is_too_small, 327

rescode.err_argument_dimension, 327

rescode.err_argument_is_too_large, 335

rescode.err_argument_lenneq, 327

rescode.err_argument_perm_array, 330

rescode.err_argument_type, 327

rescode.err_bar_var_dim, 334

rescode.err_basis, 329
 rescode.err_basis_factor, 332
 rescode.err_basis_singular, 332
 rescode.err_blank_name, 324
 rescode.err_cbf_duplicate_acoord, 336
 rescode.err_cbf_duplicate_bcoord, 336
 rescode.err_cbf_duplicate_con, 335
 rescode.err_cbf_duplicate_int, 336
 rescode.err_cbf_duplicate_obj, 335
 rescode.err_cbf_duplicate_objacoord, 336
 rescode.err_cbf_duplicate_pow_cones, 336
 rescode.err_cbf_duplicate_pow_star_cones, 336
 rescode.err_cbf_duplicate_psdvar, 336
 rescode.err_cbf_duplicate_var, 335
 rescode.err_cbf_invalid_con_type, 336
 rescode.err_cbf_invalid_dimension_of_cones, 336
 rescode.err_cbf_invalid_domain_dimension, 336
 rescode.err_cbf_invalid_exp_dimension, 336
 rescode.err_cbf_invalid_int_index, 336
 rescode.err_cbf_invalid_number_of_cones, 336
 rescode.err_cbf_invalid_power, 336
 rescode.err_cbf_invalid_power_cone_index, 336
 rescode.err_cbf_invalid_power_star_cone_index, 336
 rescode.err_cbf_invalid_psdvar_dimension, 336
 rescode.err_cbf_invalid_var_type, 336
 rescode.err_cbf_no_variables, 335
 rescode.err_cbf_no_version_specified, 335
 rescode.err_cbf_obj_sense, 335
 rescode.err_cbf_parse, 335
 rescode.err_cbf_power_cone_is_too_long, 336
 rescode.err_cbf_power_cone_mismatch, 336
 rescode.err_cbf_power_star_cone_mismatch, 336
 rescode.err_cbf_syntax, 335
 rescode.err_cbf_too_few_constraints, 336
 rescode.err_cbf_too_few_ints, 336
 rescode.err_cbf_too_few_psdvar, 336
 rescode.err_cbf_too_few_variables, 336
 rescode.err_cbf_too_many_constraints, 335
 rescode.err_cbf_too_many_ints, 336
 rescode.err_cbf_too_many_variables, 335
 rescode.err_cbf_unhandled_power_cone_type, 336
 rescode.err_cbf_unhandled_power_star_cone_type, 336
 rescode.err_cbf_unsupported, 336
 rescode.err_con_q_not_nsd, 330
 rescode.err_con_q_not_psd, 329
 rescode.err_cone_index, 330
 rescode.err_cone_overlap, 330
 rescode.err_cone_overlap_append, 330
 rescode.err_cone_parameter, 330
 rescode.err_cone_rep_var, 330
 rescode.err_cone_size, 330
 rescode.err_cone_type, 330
 rescode.err_cone_type_str, 330
 rescode.err_data_file_ext, 324
 rescode.err_dup_name, 324
 rescode.err_duplicate_aij, 330
 rescode.err_duplicate_barvariable_names, 334
 rescode.err_duplicate_cone_names, 334
 rescode.err_duplicate_constraint_names, 334
 rescode.err_duplicate_variable_names, 334
 rescode.err_end_of_file, 324
 rescode.err_factor, 332
 rescode.err_feasrepair_cannot_relax, 332
 rescode.err_feasrepair_inconsistent_bound, 332
 rescode.err_feasrepair_solving_relaxed, 332
 rescode.err_file_license, 323
 rescode.err_file_open, 324
 rescode.err_file_read, 324
 rescode.err_file_write, 324
 rescode.err_final_solution, 332
 rescode.err_first, 332
 rescode.err_firsti, 329
 rescode.err_firstj, 329
 rescode.err_fixed_bound_values, 331
 rescode.err_flexlm, 323
 rescode.err_format_string, 324
 rescode.err_global_inv_conic_problem, 332
 rescode.err_huge_aij, 330
 rescode.err_huge_c, 330
 rescode.err_identical_tasks, 334
 rescode.err_in_argument, 327
 rescode.err_index, 328
 rescode.err_index_arr_is_too_large, 327
 rescode.err_index_arr_is_too_small, 327
 rescode.err_index_is_too_large, 327
 rescode.err_index_is_too_small, 327
 rescode.err_inf_dou_index, 327
 rescode.err_inf_dou_name, 328
 rescode.err_inf_int_index, 327
 rescode.err_inf_int_name, 328
 rescode.err_inf_lint_index, 327
 rescode.err_inf_lint_name, 328
 rescode.err_inf_type, 328
 rescode.err_infeas_undefined, 334
 rescode.err_infinite_bound, 330
 rescode.err_int64_to_int32_cast, 334
 rescode.err_internal, 333
 rescode.err_internal_test_failed, 334
 rescode.err_inv_aptre, 328
 rescode.err_inv_bk, 328
 rescode.err_inv_bkc, 328
 rescode.err_inv_bkx, 328
 rescode.err_inv_cone_type, 329
 rescode.err_inv_cone_type_str, 329

rescode.err_inv_marki, 333
 rescode.err_inv_markj, 333
 rescode.err_inv_name_item, 329
 rescode.err_inv_numi, 333
 rescode.err_inv_numj, 333
 rescode.err_inv_optimizer, 332
 rescode.err_inv_problem, 331
 rescode.err_inv_qcon_subi, 331
 rescode.err_inv_qcon_subj, 331
 rescode.err_inv_qcon_subk, 331
 rescode.err_inv_qcon_val, 331
 rescode.err_inv_qobj_subi, 331
 rescode.err_inv_qobj_subj, 331
 rescode.err_inv_qobj_val, 331
 rescode.err_inv_sk, 329
 rescode.err_inv_sk_str, 329
 rescode.err_inv_skc, 329
 rescode.err_inv_skn, 329
 rescode.err_inv_skx, 329
 rescode.err_inv_var_type, 328
 rescode.err_invalid_aij, 331
 rescode.err_invalid_ampl_stub, 334
 rescode.err_invalid_barvar_name, 325
 rescode.err_invalid_compression, 332
 rescode.err_invalid_con_name, 325
 rescode.err_invalid_cone_name, 325
 rescode.err_invalid_file_format_for_cfis, 334
 rescode.err_invalid_file_format_for_cones, 334
 rescode.err_invalid_file_format_for_free_constraints, 334
 rescode.err_invalid_file_format_for_nonlinear_constraints, 334
 rescode.err_invalid_file_format_for_ranged_constraints, 334
 rescode.err_invalid_file_format_for_sym_mat, 334
 rescode.err_invalid_file_name, 324
 rescode.err_invalid_format_type, 329
 rescode.err_invalid_idx, 328
 rescode.err_invalid_iomode, 332
 rescode.err_invalid_max_num, 328
 rescode.err_invalid_name_in_sol_file, 326
 rescode.err_invalid_obj_name, 324
 rescode.err_invalid_objective_sense, 331
 rescode.err_invalid_problem_type, 335
 rescode.err_invalid_sol_file_name, 324
 rescode.err_invalid_stream, 324
 rescode.err_invalid_surplus, 329
 rescode.err_invalid_sym_mat_dim, 334
 rescode.err_invalid_task, 324
 rescode.err_invalid_utf8, 333
 rescode.err_invalid_var_name, 325
 rescode.err_invalid_wchar, 333
 rescode.err_invalid_whichsol, 328
 rescode.err_json_data, 327
 rescode.err_json_format, 327
 rescode.err_json_missing_data, 327
 rescode.err_json_number_overflow, 327
 rescode.err_json_string, 327
 rescode.err_json_syntax, 327
 rescode.err_last, 332
 rescode.err_lasti, 329
 rescode.err_lastj, 329
 rescode.err_lau_arg_k, 335
 rescode.err_lau_arg_m, 335
 rescode.err_lau_arg_n, 335
 rescode.err_lau_arg_trans, 335
 rescode.err_lau_arg_transa, 335
 rescode.err_lau_arg_transb, 335
 rescode.err_lau_arg_uplo, 335
 rescode.err_lau_invalid_lower_triangular_matrix, 335
 rescode.err_lau_invalid_sparse_symmetric_matrix, 335
 rescode.err_lau_not_positive_definite, 335
 rescode.err_lau_singular_matrix, 335
 rescode.err_lau_unknown, 335
 rescode.err_license, 323
 rescode.err_license_cannot_allocate, 323
 rescode.err_license_cannot_connect, 323
 rescode.err_license_expired, 323
 rescode.err_license_feature, 323
 rescode.err_license_invalid_hostid, 323
 rescode.err_license_max, 323
 rescode.err_license_moseklm_daemon, 323
 rescode.err_license_no_server_line, 323
 rescode.err_license_no_server_support, 323
 rescode.err_license_server, 323
 rescode.err_license_server_version, 323
 rescode.err_license_version, 323
 rescode.err_link_file_dll, 324
 rescode.err_living_tasks, 324
 rescode.err_lower_bound_is_a_nan, 330
 rescode.err_lp_dup_slack_name, 326
 rescode.err_lp_empty, 326
 rescode.err_lp_file_format, 326
 rescode.err_lp_format, 326
 rescode.err_lp_free_constraint, 326
 rescode.err_lp_incompatible, 326
 rescode.err_lp_invalid_con_name, 326
 rescode.err_lp_invalid_var_name, 326
 rescode.err_lp_write_conic_problem, 326
 rescode.err_lp_write_geco_problem, 326
 rescode.err_lu_max_num_tries, 333
 rescode.err_max_len_is_too_small, 329
 rescode.err_maxnumbarvar, 328
 rescode.err_maxnumcon, 328
 rescode.err_maxnumcone, 330
 rescode.err_maxnumqnz, 328
 rescode.err_maxnumvar, 328
 rescode.err_mio_internal, 335
 rescode.err_mio_invalid_node_optimizer, 337
 rescode.err_mio_invalid_root_optimizer, 337
 rescode.err_mio_no_optimizer, 332

rescode.err_missing_license_file, 323
 rescode.err_mixed_conic_and_nl, 331
 rescode.err_mps_cone_overlap, 325
 rescode.err_mps_cone_repeat, 325
 rescode.err_mps_cone_type, 325
 rescode.err_mps_duplicate_q_element, 326
 rescode.err_mps_file, 325
 rescode.err_mps_inv_bound_key, 325
 rescode.err_mps_inv_con_key, 325
 rescode.err_mps_inv_field, 325
 rescode.err_mps_inv_marker, 325
 rescode.err_mps_inv_sec_name, 325
 rescode.err_mps_inv_sec_order, 325
 rescode.err_mps_invalid_obj_name, 326
 rescode.err_mps_invalid_objsense, 326
 rescode.err_mps_mul_con_name, 325
 rescode.err_mps_mul_csec, 325
 rescode.err_mps_mul_qobj, 325
 rescode.err_mps_mul_qsec, 325
 rescode.err_mps_no_objective, 325
 rescode.err_mps_non_symmetric_q, 325
 rescode.err_mps_null_con_name, 325
 rescode.err_mps_null_var_name, 325
 rescode.err_mps_splitting_var, 325
 rescode.err_mps_tab_in_field2, 326
 rescode.err_mps_tab_in_field3, 326
 rescode.err_mps_tab_in_field5, 326
 rescode.err_mps_undef_con_name, 325
 rescode.err_mps_undef_var_name, 325
 rescode.err_mul_a_element, 328
 rescode.err_name_is_null, 332
 rescode.err_name_max_len, 332
 rescode.err_nan_in_blc, 331
 rescode.err_nan_in_blx, 331
 rescode.err_nan_in_buc, 331
 rescode.err_nan_in_bux, 331
 rescode.err_nan_in_c, 331
 rescode.err_nan_in_double_data, 331
 rescode.err_negative_append, 332
 rescode.err_negative_surplus, 332
 rescode.err_newer_dll, 324
 rescode.err_noBars_for_solution, 334
 rescode.err_no_barx_for_solution, 334
 rescode.err_no_basis_sol, 332
 rescode.err_no_dual_for_itg_sol, 333
 rescode.err_no_dual_infeas_cer, 332
 rescode.err_no_init_env, 324
 rescode.err_no_optimizer_var_type, 332
 rescode.err_no_primal_infeas_cer, 332
 rescode.err_no_snx_for_bas_sol, 333
 rescode.err_no_solution_in_callback, 332
 rescode.err_non_unique_array, 335
 rescode.err_nonconvex, 329
 rescode.err_nonlinear_equality, 329
 rescode.err_nonlinear_ranged, 329
 rescode.err_null_env, 324
 rescode.err_null_pointer, 324
 rescode.err_null_task, 324
 rescode.err_num_arguments, 327
 rescode.err_numconlim, 328
 rescode.err_numvarlim, 328
 rescode.err_obj_q_not_nsd, 330
 rescode.err_obj_q_not_psd, 330
 rescode.err_objective_range, 329
 rescode.err_older_dll, 323
 rescode.err_opf_format, 326
 rescode.err_opf_new_variable, 326
 rescode.err_opf_premature_eof, 326
 rescode.err_optimizer_license, 323
 rescode.err_overflow, 332
 rescode.err_param_index, 327
 rescode.err_param_is_too_large, 327
 rescode.err_param_is_too_small, 327
 rescode.err_param_name, 327
 rescode.err_param_name_dou, 327
 rescode.err_param_name_int, 327
 rescode.err_param_name_str, 327
 rescode.err_param_type, 327
 rescode.err_param_value_str, 327
 rescode.err_platform_not_licensed, 323
 rescode.err_postsolve, 332
 rescode.err_pro_item, 329
 rescode.err_prob_license, 323
 rescode.err_ptf_format, 326
 rescode.err_qcon_subi_too_large, 331
 rescode.err_qcon_subi_too_small, 331
 rescode.err_qcon_upper_triangle, 331
 rescode.err_qobj_upper_triangle, 331
 rescode.err_read_format, 325
 rescode.err_read_lp_missing_end_tag, 326
 rescode.err_read_lp_nonexisting_name, 326
 rescode.err_remove_cone_variable, 330
 rescode.err_repair_invalid_problem, 332
 rescode.err_repair_optimization_failed, 332
 rescode.err_sen_bound_invalid_lo, 333
 rescode.err_sen_bound_invalid_up, 333
 rescode.err_sen_format, 333
 rescode.err_sen_index_invalid, 333
 rescode.err_sen_index_range, 333
 rescode.err_sen_invalid_regexp, 333
 rescode.err_sen_numerical, 333
 rescode.err_sen_solution_status, 333
 rescode.err_sen_undef_name, 333
 rescode.err_sen_unhandled_problem_type, 333
 rescode.err_server_connect, 337
 rescode.err_server_protocol, 337
 rescode.err_server_status, 337
 rescode.err_server_token, 337
 rescode.err_shape_is_too_large, 327
 rescode.err_size_license, 323
 rescode.err_size_license_con, 323
 rescode.err_size_license_intvar, 323
 rescode.err_size_license_numcores, 334
 rescode.err_size_license_var, 323
 rescode.err_slice_size, 332
 rescode.err_sol_file_invalid_number, 330

```

rescode.err_solitem, 328
rescode.err_solver_probtype, 329
rescode.err_space, 324
rescode.err_space_leaking, 325
rescode.err_space_no_info, 325
rescode.err_sym_mat_duplicate, 334
rescode.err_sym_mat_huge, 331
rescode.err_sym_mat_invalid, 331
rescode.err_sym_mat_invalid_col_index, 334
rescode.err_sym_mat_invalid_row_index, 334
rescode.err_sym_mat_invalid_value, 334
rescode.err_sym_mat_not_lower_tringular,
    334
rescode.err_task_incompatible, 333
rescode.err_task_invalid, 333
rescode.err_task_write, 333
rescode.err_thread_cond_init, 324
rescode.err_thread_create, 324
rescode.err_thread_mutex_init, 324
rescode.err_thread_mutex_lock, 324
rescode.err_thread_mutex_unlock, 324
rescode.err_toconic_constr_not_conic, 337
rescode.err_toconic_constr_q_not_psd, 337
rescode.err_toconic_constraint_fx, 337
rescode.err_toconic_constraint_ra, 337
rescode.err_toconic_objective_not_psd, 337
rescode.err_too_small_a_truncation_value,
    331
rescode.err_too_small_max_num_nz, 328
rescode.err_too_small_maxnumanz, 328
rescode.err_unb_step_size, 333
rescode.err_undef_solution, 329
rescode.err_undefined_objective_sense, 331
rescode.err_unhandled_solution_status, 335
rescode.err_unknown, 324
rescode.err_upper_bound_is_a_nan, 330
rescode.err_upper_triangle, 335
rescode.err_whichitem_not_allowed, 328
rescode.err_whichsol, 328
rescode.err_write_lp_format, 326
rescode.err_write_lp_non_unique_name, 326
rescode.err_write_mps_invalid_name, 326
rescode.err_write_opf_invalid_var_name, 326
rescode.err_writing_file, 326
rescode.err_xml_invalid_problem_type, 334
rescode.err_y_is_undefined, 331

```

Index

A

- analysis
 - infeasibility, 158
- attaching
 - streams, 13

B

- basic
 - solution, 55
- basis identification, 82, 146
- basis type
 - sensitivity analysis, 165
- BLAS, 88
- bound
 - constraint, 10, 131, 134
 - linear optimization, 10
 - variable, 10, 131, 134

C

- callback, 64
- cardinality constraints, 48, 121
- CBF format, 390
- ceo1
 - example, 29
- certificate, 56
 - dual, 133, 137
 - primal, 132, 136
- Cholesky factorization, 90, 111
- column ordered
 - matrix format, 173
- complementarity, 132, 136
- concurrent optimizer, 127
- cone
 - dual, 135
 - dual exponential, 28
 - exponential, 28
 - power, 25
 - quadratic, 22
 - rotated quadratic, 22
 - semidefinite, 31
- conic exponential optimization, 28
- conic optimization, 22, 25, 28, 134
 - interior-point, 149
 - termination criteria, 151
- conic problem
 - example, 23, 26, 29
- conic quadratic optimization, 22
- Conic quadratic reformulation, 93
- constraint

- bound, 10, 131, 134
- linear optimization, 10
- matrix, 10, 131, 134
- quadratic, 139
- correlation matrix, 102
- covariance matrix, *see* correlation matrix
- cqo1
 - example, 23
- cut, 154

D

- defining
 - objective, 13
- determinism, 97
- dual
 - certificate, 133, 137
 - cone, 135
 - feasible, 132
 - infeasible, 132, 133, 137
 - problem, 132, 135, 138
 - solution, 57
 - variable, 132, 135
- duality
 - conic, 135
 - linear, 132
 - semidefinite, 138
- dualizer, 142

E

- efficient frontier, 108
- eliminator, 142
- entropy, 46
 - relative, 46
- error
 - optimization, 55
- errors, 58
- example
 - ceo1, 29
 - conic problem, 23, 26, 29
 - cqo1, 23
 - lo1, 13
 - pow1, 26
 - qo1, 16
 - quadratic objective, 16
- exceptions, 58
- exponential, 45
- exponential cone, 28

F

- factor model, 111

- feasible
 - dual, 132
 - primal, 131, 144, 150
 - problem, 131
- format, 61
 - CBF, 390
 - json, 408
 - LP, 364
 - MPS, 369
 - OPF, 381
 - PTF, 404
 - sol, 416
 - task, 408
- full
 - vector format, 172
- G
 - geometric mean, 45
 - geometric programming, 40
 - GP, 40
- H
 - hot-start, 148
- I
 - I/O, 61
- infeasibility, 56, 132, 136
 - analysis, 158
 - linear optimization, 132
 - repair, 158
 - semidefinite, 138
- infeasible
 - dual, 132, 133, 137
 - primal, 131, 132, 136, 144, 151
 - problem, 131, 132, 138
- information item, 63, 64
- installation, 5
 - Conda, 6
 - PIP, 6
 - requirements, 5
 - setup script, 7
 - troubleshooting, 5
- integer
 - optimizer, 153
 - solution, 55
 - variable, 36
- integer feasible
 - solution, 155
- integer optimization, 36, 153
 - cut, 154
 - initial solution, 38
 - objective bound, 154
 - optimality gap, 156
 - parameter, 36
 - relaxation, 154
 - termination criteria, 155
 - tolerance, 155
- integer optimizer

- logging, 156
- interior-point
 - conic optimization, 149
 - linear optimization, 143
 - logging, 147, 153
 - optimizer, 143, 149
 - solution, 55
 - termination criteria, 145, 151

J

- json format, 408

L

- LAPACK, 88
- license, 99
- linear
 - objective, 13
- linear constraint matrix, 10
- linear dependency, 142
- linear optimization, 10, 131
 - bound, 10
 - constraint, 10
 - infeasibility, 132
 - interior-point, 143
 - objective, 10
 - simplex, 148
 - termination criteria, 145, 148
 - variable, 10
- linearity interval, 165
- lo1
 - example, 13
- log-sum-exp, 46, 125
- logarithm, 45
- logging, 60
 - integer optimizer, 156
 - interior-point, 147, 153
 - optimizer, 147, 149, 153
 - simplex, 149
- logistic regression, 124
- LP format, 364

M

- machine learning
 - logistic regression, 124
- market impact cost, 112
- Markowitz
 - model, 101
- Markowitz model, 102
 - portfolio optimization, 101
- matrix
 - constraint, 10, 131, 134
 - semidefinite, 31
 - symmetric, 31
- matrix format
 - column ordered, 173
 - row ordered, 173
 - triplets, 173
- memory management, 97

MIP, *see* integer optimization
 mixed-integer, *see* integer
 mixed-integer optimization, *see* integer optimization
 model
 Markowitz, 101
 portfolio optimization, 101
 modeling
 design, 7
 monomial, 44
 MPS format, 369
 free, 380

N
 near-optimal
 solution, 155
 norm
 1-norm, 44
 2-norm, 44
 p-norm, 45
 numerical issues
 presolve, 142
 scaling, 142
 simplex, 148
 numpy, 98

O
 objective, 131, 134
 defining, 13
 linear, 13
 linear optimization, 10
 objective bound, 154
 OPF format, 381
 optimal
 solution, 56
 optimality gap, 156
 optimization
 conic, 134
 conic quadratic, 134
 error, 55
 linear, 10, 131
 semidefinite, 138
 optimizer
 concurrent, 127
 determinism, 97
 integer, 153
 interior-point, 143, 149
 interrupt, 64
 logging, 147, 149, 153
 parallel, 53
 selection, 142, 143
 simplex, 148

P
 parallel optimization, 53, 127
 parallelization, 97
 parameter, 62
 integer optimization, 36

 simplex, 148
 Pareto optimality, 102
 portfolio optimization
 cardinality constraints, 48, 121
 efficient frontier, 108
 factor model, 111
 market impact cost, 112
 Markowitz model, 102
 model, 101
 Pareto optimality, 102
 slippage cost, 112
 transaction cost, 117
 positive semidefinite, 16
 pow1
 example, 26
 power, 44
 power cone, 25
 power cone optimization, 25
 presolve, 141
 eliminator, 142
 linear dependency check, 142
 numerical issues, 142
 primal
 certificate, 132, 136
 feasible, 131, 144, 150
 infeasible, 131, 132, 136, 144, 151
 problem, 132, 135, 138
 solution, 57, 131
 primal-dual
 problem, 143, 150
 solution, 132
 problem
 dual, 132, 135, 138
 feasible, 131
 infeasible, 131, 132, 138
 load, 61
 primal, 132, 135, 138
 primal-dual, 143, 150
 save, 61
 status, 55
 unbounded, 133, 137
 PTF format, 404

Q
 qo1
 example, 16
 quadratic
 constraint, 139
 quadratic cone, 22
 quadratic objective
 example, 16
 quadratic optimization, 139
 quality
 solution, 156

R
 regression
 logistic, 124

- relaxation, 154
- repair
 - infeasibility, 158
- response code, 58
- rotated quadratic cone, 22
- row ordered
 - matrix format, 173
- S
- scaling, 142
- semicontinuous variable, 47
- semidefinite
 - cone, 31
 - infeasibility, 138
 - matrix, 31
 - variable, 31
- semidefinite optimization, 31, 138
- sensitivity analysis, 163
 - basis type, 165
- setup script, 7
- shadow price, 165
- simplex
 - linear optimization, 148
 - logging, 149
 - numerical issues, 148
 - optimizer, 148
 - parameter, 148
 - termination criteria, 148
- slippage cost, 112
- softplus, 46
- sol format, 416
- solution
 - basic, 55
 - dual, 57
 - file format, 416
 - integer, 55
 - integer feasible, 155
 - interior-point, 55
 - near-optimal, 155
 - optimal, 56
 - primal, 57, 131
 - primal-dual, 132
 - quality, 156
 - retrieve, 55
 - status, 13, 56
- solving linear system, 86
- sparse
 - vector format, 172
- sparse vector, 172
- status
 - problem, 55
 - solution, 13, 56
- streams
 - attaching, 13
- symmetric
 - matrix, 31

T

- task format, 408
- termination, 55
- termination criteria, 64
 - conic optimization, 151
 - integer optimization, 155
 - interior-point, 145, 151
 - linear optimization, 145, 148
 - simplex, 148
 - tolerance, 146, 152, 155
- thread, 97
- time limit, 64
- tolerance
 - integer optimization, 155
 - termination criteria, 146, 152, 155
- transaction cost, 117
- triplets
 - matrix format, 173
- troubleshooting
 - installation, 5

U

- unbounded
 - problem, 133, 137
- user callback, *see* callback

V

- variable, 131, 134
 - bound, 10, 131, 134
 - dual, 132, 135
 - integer, 36
 - linear optimization, 10
 - semicontinuous, 47
 - semidefinite, 31
- vector format
 - full, 172
 - sparse, 172