The `moseklua` manual
Version 7.1 (Revision 45)

info@mosek.com

www.mosek.com

- Published by MOSEK ApS, Denmark.
- Copyright © MOSEK ApS, Denmark. All rights reserved.

Contents

Contact information

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

License agreement

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

Getting support and help

0.1 MOSEK documentation

For an overview of the available MOSEK documentation please see

```
mosek/7/docs/
```

in the distribution.

0.2 Additional reading

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

Another useful resource is "Mathematical Programming Glossary" available at

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

Chapter 1

MOSEK/LUA tutorial

MOSEK includes a LUA 5.2 interpreter called "`moseklua`". This is found in the `bin/` directory in the distribution. To use it, MOSEK must be correctly installed. It includes a relatively complete interface to MOSEK and the default set of LUA libraries.

This section documents only the command line syntax and the MOSEK interface, not LUA in general. LUA documentation can be found here:

LUA 5.2 reference manual

http://www.lua.org/manual/5.2/manual.html

Programming in LUA (first edition)

http://www.lua.org/pil/index.html

The interpreter can be used in various ways: It can be used as an interactive interpreter, to interpret script files or to interpret inline-scripts given on the command line. The intended use for this interface is as a scriptable and flexible command line tool for solving simple tasks and tests.

1.1 Command line usage

The program is located in the MOSEK distribution in the `bin/` directory, and is called `moseklua` or `moseklua.exe`.

To start the interactive interactive interpreter, simply run the program

```
moseklua
```

If the script is passed to the interpreter through `stdin` and interpreted in a non-interactive manner, this can be done by passing a "`-`", e.g. on UNIX:

```
moseklua - < myscript.lua
```

To execute an inline script given on the command line, use the "`-e`" option, e.g.:

```
moseklua -e 'print (string.format ("Hello, MOSEK %d.%d.%d.%d!", getversion()))'
```

Arguments can be passed to the inline script by adding them to the command line. These arguments will be passed in the global variable "`arg`":

```
moseklua -e 'print(string.format("Argument one is %s, and two is %s.",arg[1],arg[2]))' myinputfile.opf myoutputfile.opf
```

This can be used to call the *help* facility. To list all MOSEK functions:

```
moseklua -e 'help()'
```

or to get help on, say `getsolution`:

```
moseklua -e 'help(arg[1])' getsolution
```

To execute a script with arguments, simply run `moseklua` with the script as first argument:

```
moseklua myscript.lua myinputfile.opf myoutputfile.opf
```

1.2 The interactive command line

The interactive interpreter has some limitations: Each line must be a complete valid statement, i.e. loops, functions etc. must be written entirely on one line. For example, a function must be written as

```
function myfunc(a,b) print("A =",a) print("B =",b) return a end
```

but *when interactive, never on multiple lines:*

```
-- INVALID in interactive mode
function myfunc(a,b)
  print("A =",a)
  print("B =",b)
  return a
end
```

1.3 The scripting environment

The scripting environment provides the standard LUA libraries as well as the API to MOSEK. Section 2 contains a full description of the MOSEK API.

1.3.1 Command line arguments

When not in interactive mode, command line arguments can be accessed through the global variable `arg`, e.g. when using the "`-e`" option:

```
moseklua -e 'print(arg[0],arg[1])' Hello MOSEK
> Hello Mosek
```

The `arg` is simply a sequence table, so accessing elements outside it will return a `nil` value:

```
moseklua -e 'print(string.format("Number of arguments: %d",#arg)) print(arg[0],arg[1],arg[2],arg[3])' Hello
MOSEK
> Number of arguments: 2
> Hello Mosek nil
```

1.3.2 The MOSEK API

The API consists of a single global optimization task and associated information. This is not directly available to the user, only through a set of API functions that can be used to modify it and extract information.

All functions are available directly in the primary environment as global values. The `help()` function will provide a list of all functions available, typing

```
help(help)
```

will print help on the help function itself. All functions are described in more detail in section 2

1.4 Simple LUA script

The purpose of the `moseklua` tool is to be able to script small programs that perform simple operations on optimization problems like solve and print information, modify a problem, convert problems from one file format to another or write customized command-line tools to work with other programs.

1.4.1 Format conversion

Below the example `read_write.lua` demonstrates a simple script that takes as input a script, changes a list of parameter settings, (optionally) optimizes the problem, and finally writes the problem to a new file.

[read_write.lua]

```
--
-- Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
--
-- File:      read_write.lua
--
-- Purpose:   Read a problem from the command line, set some parameters and
--            write the solution solve it and write the problem, including
--            solution.
--
-- Usage:
--   lmosek read_write.lua [options] inputfile outputile
-- Options:
--   -d PARNAME PARVALUE   Set parameter value.
--   -x                    Don't solve.
--   -v                    Print some extra info.
-- Example (convert file from task to opf):
--   lmosek read_write.lua -x myfile.task myfile.opf
--


do_optimize = true
verbose     = false
inputfile   = nil
outputfile  = nil
params      = {}
-- Arguments are passed in the 'arg' variable.
```

```
-- Loop over all argments and extract options, input and output file names.
do
   local i = 1
   while i <= #arg do
      if     arg[i] == '-d' then
         parname = arg[i+1]
         parval  = arg[i+2]
         params[parname] = parval
         i = i + 3
      elseif arg[i] == '-x' then
         do_optimize = false
         i = i + 1
      elseif arg[i] == '-v' then
         verbose = false
         i = i + 1
      else
         inputfile = arg[i]
         outputfile = arg[i+1]
         i = #arg+1
      end
   end
end

if inputfile == nil then
   print("Missing input and output files.")
elseif outputfile == nil then
   print("Missing output file.")
else
   readdata(inputfile)

   for parname,parval in pairs(params) do
      -- Call setparam() in protected mode, so if we
      if not pcall(setparam,parname,parval) then
         print(string.format("Warning: Failed to set %s = %s",parname,parval))
      end
   end

   if do_optimize then
      optimize()

      if verbose then
         solutionsummary(streamtype.msg)
      end
   end

   writedata(outputfile)

   print(string.format("Problem written to: %s",outputfile))
end
```

1.4.2 Name stripping

The example `strip.lua` is a script to strip all names from a problem. This can, to some degree, be used to anonymize an optimization problem.

[strip.lua]

```
--
-- Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
--
-- File:      strip.lua
--
-- Purpose:   Read a problem from the command line, strip out all names and write it to a new file.
--
-- Usage:
--   lmosek strip.lua inputfile outputile
-- Example:
--   lmosek strip.lua myfile.opf mystrippedfile.task

readdata(arg[1])


-- clear all variable names
if getnumvar() > 0 then
   for i=1,getnumvar() do
      setvarname(i-1,"")
   end
end

-- clear all constraint names
if getnumcon() > 0 then
   for i=1,getnumcon() do
      setconname(i-1,"")
   end
end

-- clear all cone names
if getnumcone() > 0 then
   for i=1,getnumcone() do
      setconename(i-1,"")
   end
end

-- clear objective and task names
putobjname("")
puttaskname("")

-- clear string parameters
for k,v in pairs(sparam) do
   setparam(v,"")
end

-- write the stripped output file
writedata(arg[2])
```

1.5 Using API functions

It it quite possible to implement complete optimization problems in the MOSEK/LUA interface, but it was not designed to be used for large scale modeling; the facilities for error handling, callbacks and printing are limited compared to the other APIs.

Below is an example of the example `lo1` as it might look in LUA:

[lo1.lua]

```
--
-- Copyright: Copyright (c) MOSEK ApS, Denmark. All rights reserved.
--
-- File:      lo1.lua
--
-- Purpose:   Demonstrates how to input and solve a small
--            optimization problem in Mosek/LUA.
--


local numcon = 3
local numvar = 4
local numanz = 9

    -- Since the value infinity is never used, we define
    -- 'infinity' symbolic purposes only
local infinity = 0.0

local c    = { 3.0, 1.0, 5.0, 1.0 }
local asub = { { 0,1 },
               { 0,1,2 },
               { 0,1 },
               { 1,2 } }
local aval = { { 3.0, 2.0 },
               { 1.0, 1.0, 2.0 },
               { 2.0, 3.0 },
               { 1.0, 3.0 } }
local bkc  = { boundkey.fx,
               boundkey.lo,
               boundkey.up }
local blc  = { 30.0,
               15.0,
             - infinity }
local buc  = { 30.0,
               infinity,
               25.0 }
local bkx  = { boundkey.lo,
               boundkey.ra,
               boundkey.lo,
               boundkey.lo }
local blx  = { 0.0,
               0.0,
               0.0,
               0.0 }
local bux  = { infinity,
               10.0,
               infinity,
               infinity }

-- Append (numcon) empty constraints.
-- The constraints will initially have no bounds.
append(accmode.con,numcon)
-- Append (numvar) variables.
```

```
-- The variables will initially be fixed at zero (x=0).
append(accmode.var,numvar)

-- Optionally add a constant term to the objective.
putcfix(0.0)
for j=1,numvar do
   -- Set the linear term c_j in the objective.
   putcj(j-1,c[j])
   -- Set the bounds on variable j.
   --   blx[j] <= x_j <= bux[j]
   putbound(accmode.var,j-1,bkx[j],blx[j],bux[j])
   -- Input column j of A
   putavec(accmode.var, -- Input columns of A.
           j-1,         -- Variable (column) index.
           asub[j],     -- Row index of non-zeros in column j.
           aval[j])     -- Non-zero Values of column j.
end
-- Set the bounds on constraints.
--   for i=1, ...,numcon : blc[i] <= constraint i <= buc[i] */
for i=1,numcon do
   putbound(accmode.con,i-1,bkc[i],blc[i],buc[i])
end
putobjsense(objsense.maximize)
optimize()
-- Print a summary containing information
--   about the solution for debugging purposes
solutionsummary(streamtype.msg)

-- Get status information about the solution
local prostatus,solstatus = getsolutionstatus(soltype.bas)
local xx = getxx(soltype.bas) -- Basic solution.

print(string.format("Solution status = %s",solstatus))
if     ( solstatus == solsta.optimal or
         solstatus == solsta.near_optimal ) then
   for i=1,#xx do
      print(string.format("xx[%d] = %s",i,xx[i]))
   end
elseif ( solstatus == solsta.prim_infeas_cer or
         solstatus == solsta.dual_infeas_cer or
         solstatus == solsta.neat_prim_infeas_cer or
         solstatus == solsta.near_dual_infeas_cer ) then
   print("Primal or dual infeasibility.")
end
```

Chapter 2

API reference

2.1 analyzenames()

The function analyzes the names and issue an error if a name is invalid. Syntax:

```
analyzenames(whichstream,nametype) -> nil
```

whichstream
: Index of the stream.

nametype
: The type of names e.g. valid in MPS or LP files.

2.2 analyzeproblem()

The function analyzes the data of task and writes out a report. Syntax:

```
analyzeproblem(whichstream) -> nil
```

whichstream
: Index of the stream.

2.3 analyzesolution()

Print information related to the quality of the solution and other solution statistics.

By default this function prints information about the largest infeasibilites in the solution, the primal (and possibly dual) objective value and the solution status.

Following parameters can be used to configure the printed statistics:

- iparam.ana_sol_basis. Enables or disables printing of statistics specific to the basis solution (condition number, number of basic variables etc.). Default is on.
- iparam.ana_sol_print_violated. Enables or disables listing names of all constraints (both primal and dual) which are violated by the solution. Default is off.
- dparam.ana_sol_infeas_tol. The tolerance defining when a constraint is considered violated. If a constraint is violated more than this, it will be listed in the summary.

Syntax:

```
analyzesolution(whichstream,whichsol) -> nil
```

whichstream

Index of the stream.

whichsol

Selects a solution.

2.4 appendbarvars()

Appends a positive semidefinite matrix variable of dimension `dim` to the problem. Syntax:

```
appendbarvars(dim) -> nil
```

dim

Dimension of symmetric matrix variables to be added.

2.5 appendcone()

Appends a new conic constraint to the problem. Hence, add a constraint

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

to the problem where $\mathcal{C}$ is a convex cone. $\hat{x}$ is a subset of the variables which will be specified by the argument `submem`.

Depending on the value of `conetype` this function appends a normal (conetype.quad) or rotated quadratic cone (conetype.rquad). Define

$$\hat{x} = x_{\mathtt{submem}[0]}, \ldots, x_{\mathtt{submem}[\mathtt{nummem}-1]}$$

. Depending on the value of `conetype` this function appends one of the constraints:

- Quadratic cone (`conetype.quad`) :

$$\hat{x}_0 \geq \sqrt{\sum_{i=1}^{i<\texttt{nummem}} \hat{x}_i^2}$$

- Rotated quadratic cone (`conetype.rquad`) :

$$2\hat{x}_0\hat{x}_1 \geq \sum_{i=2}^{i<\texttt{nummem}} \hat{x}_i^2, \quad \hat{x}_0, \hat{x}_1 \geq 0$$

Please note that the sets of variables appearing in different conic constraints must be disjoint.

Syntax:

```
appendcone(conetype,conepar,submem) -> nil
```

conetype

Specifies the type of the cone.

conepar

This argument is currently not used. Can be set to 0.0.

submem

Variable subscripts of the members in the cone.

2.6 appendconeseq()

Appends a new conic constraint to the problem. The function assumes the members of cone are sequential where the first emeber has index `j` and the last `j+nummem-1`. Syntax:

```
appendconeseq(conetype,conepar,nummem,j) -> nil
```

conetype

Specifies the type of the cone.

conepar

This argument is currently not used. Can be set to 0.0.

nummem

Dimension of the conic constraint to be appended.

j

Index of the first variable in the conic constraint.

2.7 appendconesseq()

Appends a number conic constraints to the problem. The kth cone is assumed to be of dimension `nummem[k]`. Moreover, is is asummed that the first variable of the first cone has index j and the index of the variable in each cone are sequential. Finally, it assumed in the second cone is the last index of first cone plus one and so forth. Syntax:

```
appendconesseq(conetype,conepar,nummem,j) -> nil
```

conetype

Specifies the type of the cone.

conepar

This argument is currently not used. Can be set to 0.0.

nummem

Number of member variables in the cone.

j

Index of the first variable in the first cone to be appended.

2.8 appendcons()

Appends a number of constraints to the model. Appended constraints will be declared free. Please note that MOSEK will automatically expand the problem dimension to accommodate the additional constraints. Syntax:

```
appendcons(num) -> nil
```

num

Number of constraints which should be appended.

2.9 appendsparsesymmat()

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

This functionsappends a general sparse symmetric matrix on triplet form to the vector E of symmetric matrixes. The vectors `subi`, `subj`, and `valij` contains the row subscripts, column subscripts and values of each element in the symmetric matrix to be appended. Since the matrix that is appended is symmetric then only the lower triangular part should be specified. Moreover, duplicates are not allowed.

Observe the function reports the index (position) of the appended matrix in E. This index should be used for later references to the appended matrix. Syntax:

```
appendsparsesymmat(dim,subi,subj,valij) -> idx
```

dim

Dimension of the symmetric matrix that is appended.

subi

Row subscript in the triplets.

subj

Column subscripts in the triplets.

valij

Values of each triplet.

idx

Each matrix that is appended to E is assigned a unique index i.e. `idx` that can be used for later reference.

2.10 appendvars()

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. Syntax:

```
appendvars(num) -> nil
```

num

Number of variables which should be appended.

2.11 basiscond()

If a basic solution is available and it defines a nonsingular basis, then this function computes the 1-norm estimate of the basis matrix and an 1-norm estimate for the inverse of the basis matrix. The 1-norm estimates are computed using the method outlined in [5].

By defintion the 1-norm condition number of a matrix B is defined as

$$\kappa_1(B) := \|B\|_1 \left\|B^{-1}\right\|.$$

Moreover, the larger the condition number is the harder it is to solve linear equation systems involving B. Given estimates for $\|B\|_1$ and $\left\|B^{-1}\right\|_1$ it is also possible to estimate $\kappa_1(B)$. Syntax:

```
basiscond() -> nrmbasis,nrminvbasis
```

nrmbasis

An estimate for the 1 norm of the basis.

nrminvbasis

An estimate for the 1 norm of the inverse of the basis.

2.12 chgbound()

Changes a bound for one constraint or variable. If `accmode` equals `accmode.con`, a constraint bound is changed, otherwise a variable bound is changed.

If `lower` is non-zero, then the lower bound is changed as follows:

$$\text{new lower bound} = \begin{cases} -\infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Otherwise if `lower` is zero, then

$$\text{new upper bound} = \begin{cases} \infty, & \texttt{finite} = 0, \\ \texttt{value} & \text{otherwise.} \end{cases}$$

Please note that this function automatically updates the bound key for bound, in particular, if the lower and upper bounds are identical, the bound key is changed to `fixed`. Syntax:

```
chgbound(accmode,i,lower,finite,value) -> nil
```

accmode

Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

i

Index of the constraint or variable for which the bounds should be changed.

lower

If non-zero, then the lower bound is changed, otherwise the upper bound is changed.

finite

If non-zero, then `value` is assumed to be finite.

value

New value for the bound.

2.13 deletesolution()

Undefines a solution and frees the memory it uses. Syntax:

```
deletesolution(whichsol) -> nil
```

whichsol

Selects a solution.

2.14 dualsensitivity()

Calculates sensitivity information for objective coefficients. The indexes of the coefficients to analyze are

$$\{\texttt{subj}[i] | i \in 0, \ldots, \texttt{numj} - 1\}$$

The results are returned so that e.g `leftprice`$[j]$ is the left shadow price of the objective coefficient with index `subj`$[j]$.

The type of sensitivity analysis to perform (basis or optimal partition) is controlled by the parameter `iparam.sensitivity_type`.

Syntax:

```
dualsensitivity(subj) -> leftpricej,rightpricej,leftrangej,rightrangej
```

subj

Index of objective coefficients to analyze.

leftpricej

`leftpricej`$[j]$ is the left shadow price for the coefficients with index `subj`[j].

rightpricej

`rightpricej`$[j]$ is the right shadow price for the coefficients with index `subj`[j].

leftrangej

`leftrangej`$[j]$ is the left range β_1 for the coefficient with index `subj`[j].

rightrangej

`rightrangej`$[j]$ is the right range β_2 for the coefficient with index `subj`[j].

2.15 getacol()

Obtains one column of A in a sparse format. Syntax:

```
getacol(j) -> subj,valj
```

j

Index of the column.

subj

Index of the non-zeros in the column obtained.

valj

Numerical values of the column obtained.

2.16 getacolnumnz()

Obtains the number of non-zero elements in one column of A. Syntax:

```
getacolnumnz(i) -> nzj
```

i

Index of the column.

nzj

Number of non-zeros in the jth row or column of A.

2.17 getaij()

Obtains a single coefficient in A. Syntax:

```
getaij(i,j) -> aij
```

i

Row index of the coefficient to be returned.

j

Column index of the coefficient to be returned.

aij

The required coefficient $a_{i,j}$.

2.18 getapiecenumnz()

Obtains the number non-zeros in a rectangular piece of A, i.e. the number

$$|\{(i,j):\ a_{i,j} \neq 0,\ \texttt{firsti} \leq i \leq \texttt{lasti} - 1,\ \texttt{firstj} \leq j \leq \texttt{lastj} - 1\}|$$

where $|\mathcal{I}|$ means the number of elements in the set $\mathcal{I}$.

This function is not an efficient way to obtain the number of non-zeros in one row or column. In that case use the function `Task.getarownumnz` or `Task.getacolnumnz`. Syntax:

```
getapiecenumnz(firsti,lasti,firstj,lastj) -> numnz
```

firsti

Index of the first row in the rectangular piece.

lasti

Index of the last row plus one in the rectangular piece.

firstj

Index of the first column in the rectangular piece.

lastj

Index of the last column plus one in the rectangular piece.

numnz

Number of non-zero A elements in the rectangular piece.

2.19 getarow()

Obtains one row of A in a sparse format. Syntax:

```
getarow(i) -> subi,vali
```

i

Index of the row or column.

subi

Index of the non-zeros in the row obtained.

vali

Numerical values of the row obtained.

2.20 getarownumnz()

Obtains the number of non-zero elements in one row of A. Syntax:

```
getarownumnz(i) -> nzi
```

i

Index of the row or column.

nzi

Number of non-zeros in the ith row of A.

2.21 getaslice()

Obtains a sequence of rows or columns from A in sparse format. Syntax:

```
getaslice(accmode,first,last) -> ptrb,ptre,sub,val
```

accmode

Defines whether a column slice or a row slice is requested.

first

Index of the first row or column in the sequence.

last

Index of the last row or column in the sequence **plus one**.

ptrb

`ptrb[t]` is an index pointing to the first element in the tth row or column obtained.

ptre

`ptre[t]` is an index pointing to the last element plus one in the tth row or column obtained.

sub

Contains the row or column subscripts.

val

Contains the coefficient values.

2.22 getaslicenumnz()

Obtains the number of non-zeros in a slice of rows or columns of A. Syntax:

```
getaslicenumnz(accmode,first,last) -> numnz
```

accmode

Defines whether non-zeros are counted in a column slice or a row slice.

first

Index of the first row or column in the sequence.

last

Index of the last row or column **plus one** in the sequence.

numnz

Number of non-zeros in the slice.

2.23 getbarablocktriplet()

Obtains $\bar{A}$ in block triplet form. Syntax:

```
getbarablocktriplet() -> num,subi,subj,subk,subl,valijkl
```

num

Number of elements in the block triplet form.

subi

Constraint index.

subj

Symmetric matrix variable index.

subk

Block row index.

subl

Block column index.

valijkl

A list indexes of the elements from symmetric matrix storage that appers in the weighted sum.

2.24 getbaraidx()

Obtains information about an element in $\bar{A}$. Since $\bar{A}$ is a sparse matrix of symmetric matrixes then only the nonzero elements in $\bar{A}$ are stored in order to save space. Now $\bar{A}$ is stored vectorized form i.e. as one long vector. This function makes it possible to obtain information such as the row index and the column index of a particular element of the vectorized form of $\bar{A}$.

Please observe if one element of $\bar{A}$ is inputted multiple times then it may be stored several times in vectorized form. In that case the element with the highest index is the one that is used. Syntax:

```
getbaraidx(idx) -> i,j,num,sub,weights
```

idx

Position of the element in the vectorized form.

i

Row index of the element at position `idx`.

j

Column index of the element at position `idx`.

num

Number of terms in weighted sum that forms the element.

sub

A list indexes of the elements from symmetric matrix storage that appers in the weighted sum.

weights

The weights associated with each term in the weighted sum.

2.25 getbaraidxij()

Obtains information about an element in $\bar{A}$. Since $\bar{A}$ is a sparse matrix of symmetric matrixes only the nonzero elements in $\bar{A}$ are stored in order to save space. Now $\bar{A}$ is stored vectorized form i.e. as one long vector. This function makes it possible to obtain information such as the row index and the column index of a particular element of the vectorized form of $\bar{A}$.

Please note that if one element of $\bar{A}$ is inputted multiple times then it may be stored several times in vectorized form. In that case the element with the highest index is the one that is used. Syntax:

```
getbaraidxij(idx) -> i,j
```

idx

Position of the element in the vectorized form.

i

Row index of the element at position `idx`.

j

Column index of the element at position `idx`.

2.26 getbaraidxinfo()

Each nonzero element in $\bar{A}_{ij}$ is formed as a weighted sum of symmtric matrixes. Using this function the number terms in the weigthted sum can be obtained. See description of `Task.appendsparsesymmat` for details about the weighted sum. Syntax:

```
getbaraidxinfo(idx) -> num
```

idx

The internal position of the element that should be obtained information for.

num

Number of terms in the weighted sum that forms the specified element in $\bar{A}$.

2.27 getbarasparsity()

The matrix $\bar{A}$ is assumed to be a sparse matrix of symmetric matrixes. This implies that many of elements in $\bar{A}$ is likely to be zero matrixes. Therefore, in order to save space only nonzero elementsin $\bar{A}$ are stored on vectorized form. This function is used to obtain the sparsity pattern of $\bar{A}$ and the position of each nonzero element in the vectorized form of $\bar{A}$. Syntax:

```
getbarasparsity() -> numnz,idxij
```

numnz

Number of nonzero elements in $\bar{A}$.

idxij

Position of each nonzero element in the vectorized form of $\bar{A}_{ij}$. Hence, `idxij[k]` is the vector position of the element in row `subi[k]` and column `subj[k]` of $\bar{A}_{ij}$.

2.28 getbarcblocktriplet()

Obtains $\bar{C}$ in block triplet form. Syntax:

```
getbarcblocktriplet() -> num,subj,subk,subl,valijkl
```

num

Number of elements in the block triplet form.

subj

Symmetric matrix variable index.

subk

Block row index.

subl

Block column index.

valijkl

A list indexes of the elements from symmetric matrix storage that appers in the weighted sum.

2.29 getbarcidx()

Obtains information about an element in $\bar{c}$. Syntax:

```
getbarcidx(idx) -> j,num,sub,weights
```

idx

Index of the element that should be obtained information about.

j

Row index in $\bar{c}$.

num

Number of terms in the weighted sum.

sub

Elements appearing the weighted sum.

weights

Weights of terms in the weighted sum.

2.30 getbarcidxinfo()

Obtains information about about the $\bar{c}_{ij}$. Syntax:

```
getbarcidxinfo(idx) -> num
```

idx

Index of element that should be obtained information about. The value is an index of a symmetric sparse variable.

num

Number of terms that appears in weighted that forms the requested element.

2.31 getbarcidxj()

Obtains the row index of an element in $\bar{c}$. Syntax:

```
getbarcidxj(idx) -> j
```

idx

Index of the element that should be obtained information about.

j

Row index in $\bar{c}$.

2.32 getbarcsparsity()

Internally only the nonzero elements of $\bar{c}$ is stored

in a vector. This function returns which elements $\bar{c}$ that are nonzero (in `subj`) and their internal position (in `idx`). Using the position detailed information about each nonzero $\bar{C}_j$ can be obatined using `Task.getbarcidxinfo` and `Task.getbarcidx`. Syntax:

```
getbarcsparsity() -> numnz,idxj
```

numnz

Number of nonzero elements in $\bar{C}$.

idxj

Internal positions of the nonzeros elements in $\bar{c}$.

2.33 getbarsj()

Obtains the dual solution for a semidefinite variable. Syntax:

```
getbarsj(whichsol,j) -> barsj
```

whichsol

Selects a solution.

j

Index of the semidefinite variable.

barsj

Value of $\bar{s}_j$.

2.34 getbarvarname()

Obtains a name of a semidefinite variable. Syntax:

```
getbarvarname(i) -> name
```

i

Index.

name

The requested name is copied to this buffer.

2.35 getbarvarnameindex()

Obtains the index of name of semidefinite variable. Syntax:

```
getbarvarnameindex(somename) -> asgn,index
```

somename

The requested name is copied to this buffer.

asgn

Is non-zero if the name `somename` is assigned to a semidefinite variable.

index

If the name `somename` is assigned to a semidefinite variable, then `index` is the name of the constraint.

2.36 getbarvarnamelen()

Obtains the length of a name of a semidefinite variable. Syntax:

```
getbarvarnamelen(i) -> len
```

i

Index.

len

Returns the length of the indicated name.

2.37 getbarxj()

Obtains the primal solution for a semidefinite variable. Syntax:

```
getbarxj(whichsol,j) -> barxj
```

whichsol

Selects a solution.

j

Index of the semidefinite variable.

barxj

Value of $\bar{X}_j$.

2.38 getbound()

Obtains bound information for one constraint or variable. Syntax:

```
getbound(accmode,i) -> bk,bl,bu
```

accmode

Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

i

Index of the constraint or variable for which the bound information should be obtained.

bk

Bound keys.

bl

Values for lower bounds.

bu

Values for upper bounds.

2.39 getboundslice()

Obtains bounds information for a sequence of variables or constraints. Syntax:

```
getboundslice(accmode,first,last) -> bk,bl,bu
```

accmode

Defines if operations are performed row-wise (constraint-oriented) or column-wise (variable-oriented).

first

First index in the sequence.

last

Last index plus 1 in the sequence.

bk

Bound keys.

bl

Values for lower bounds.

bu

Values for upper bounds.

2.40 getc()

Obtains all objective coefficients c. Syntax:

```
getc() -> c
```

c

Linear terms of the objective as a dense vector. The lengths is the number of variables.

2.41 getcfix()

Obtains the fixed term in the objective. Syntax:

```
getcfix() -> cfix
```

cfix

Fixed term in the objective.

2.42 getcj()

Obtains one coefficient of c. Syntax:

```
getcj(j) -> cj
```

j

Index of the variable for which c coefficient should be obtained.

cj

The value of c_j.

2.43 getconbound()

Obtains bound information for one constraint. Syntax:

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

i

Index of the constraint for which the bound information should be obtained.

bk

Bound keys.

bl

Values for lower bounds.

bu

Values for upper bounds.

2.44 getconboundslice()

Obtains bounds information for a slice of the constraints. Syntax:

```
getconboundslice(first,last) -> bk,bl,bu
```

first

First index in the sequence.

last

Last index plus 1 in the sequence.

bk

Bound keys.

bl

Values for lower bounds.

bu

Values for upper bounds.

2.45 getcone()

Obtains a conic constraint. Syntax:

```
getcone(k) -> conetype,submem
```

k

Index of the cone constraint.

conetype

Specifies the type of the cone.

submem

Variable subscripts of the members in the cone.

2.46 getconeinfo()

Obtains information about a conic constraint. Syntax:

```
getconeinfo(k) -> conetype,conepar,nummem
```

k

Index of the conic constraint.

conetype

Specifies the type of the cone.

conepar

This argument is currently not used. Can be set to 0.0.

nummem

Number of member variables in the cone.

2.47 getconename()

Obtains a name of a cone. Syntax:

```
getconename(i) -> name
```

i

Index.

name

Is assigned the required name.

2.48 getconenameindex()

Checks whether the name `somename` has been assigned to any cone. If it has been assigned to cone, then index of the cone is reported. Syntax:

```
getconenameindex(somename) -> asgn,index
```

somename

The name which should be checked.

asgn

Is non-zero if the name `somename` is assigned to a cone.

index

If the name `somename` is assigned to a cone, then `index` is the name of the cone.

2.49 getconenamelen()

Obtains the length of a name of a cone. Syntax:

```
getconenamelen(i) -> len
```

i

Index.

len

Returns the length of the indicated name.

2.50 getconname()

Obtains a name of a constraint. Syntax:

```
getconname(i) -> name
```

i

Index.

name

Is assigned the required name.

2.51 getconnameindex()

Checks whether the name `somename` has been assigned to any constraint. If it has been assigned to constraint, then index of the constraint is reported. Syntax:

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

somename

The name which should be checked.

asgn

Is non-zero if the name `somename` is assigned to a constraint.

index

If the name `somename` is assigned to a constraint, then `index` is the name of the constraint.

2.52 getconnamelen()

Obtains the length of a name of a constraint variable. Syntax:

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

i

Index.

len

Returns the length of the indicated name.

2.53 getcslice()

Obtains a sequence of elements in c. Syntax:

```
getcslice(first,last) -> c
```

first

First index in the sequence.

last

Last index plus 1 in the sequence.

c

Linear terms of the objective as a dense vector. The lengths is the number of variables.

2.54 getdbi()

Deprecated.

Obtains the dual bound infeasibility. Syntax:

```
getdbi(whichsol,accmode,sub) -> dbi
```

whichsol

Selects a solution.

accmode

If set to `accmode.con` then `sub` contains constraint indexes, otherwise variable indexes.

sub

Indexes of constraints or variables.

dbi

Dual bound infeasibility. If `acmode` is `accmode.con` then

$$\mathtt{dbi}[i] = \max(-(s_l^c)_{\mathtt{sub}[i]}, -(s_u^c)_{\mathtt{sub}[i]}, 0) \ \ \mathtt{for} \ \ i = 0, \ldots, \mathtt{len} - 1$$

else

$$\mathtt{dbi}[i] = \max(-(s_l^x)_{\mathtt{sub}[i]}, -(s_u^x)_{\mathtt{sub}[i]}, 0) \ \ \mathtt{for} \ \ i = 0, \ldots, \mathtt{len} - 1.$$

2.55 getdcni()

Deprecated.

Obtains the dual cone infeasibility. Syntax:

```
getdcni(whichsol,sub) -> dcni
```

whichsol

Selects a solution.

sub

Constraint indexes to calculate equation infeasibility for.

dcni

`dcni[i]` contains dual cone infeasibility for the cone with index `sub[i]`.

2.56 getdeqi()

Deprecated.

Optains the dual equation infeasibility. If `acmode` is `accmode.con` then

$$\texttt{pbi}[i] = \left|(-y + s_l^c - s_u^c)_{\texttt{sub}[i]}\right| \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1$$

If `acmode` is `accmode.var` then

$$\texttt{pbi}[i] = \left|(A^T y + s_l^x - s_u^x - c)_{\texttt{sub}[i]}\right| \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1$$

Syntax:

```
getdeqi(whichsol,accmode,sub,normalize) -> deqi
```

whichsol

Selects a solution.

accmode

If set to `accmode.con` the dual equation infeasibilities corresponding to constraints are retrieved. Otherwise for a variables.

sub

Indexes of constraints or variables.

normalize

If non-zero, normalize with largest absolute value of the input data used to compute the individual infeasibility.

deqi

Dual equation infeasibilities corresponding to constraints or variables.

2.57 getdimbarvarj()

Obtains the dimension of a symmetric matrix variable. Syntax:

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

j

Index of the semidefinite variable whose dimension is requested.

dimbarvarj

The dimension of the j'th semidefinite variable.

2.58 getdouinf()

Obtains a double information item from the task information database. Syntax:

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

whichdinf

A `double` information item. See section `dinfitem` for the possible values.

dvalue

The value of the required double information item.

2.59 getdouparam()

Obtains the value of a double parameter. Syntax:

```
getdouparam(param) -> parvalue
```

param

Which parameter.

parvalue

Parameter value.

2.60 getdualobj()

Computes the dual objective value associated with the solution. Note if the solution is a primal infeasibility certificate, then the fixed term in the objective value is not included. Syntax:

```
getdualobj(whichsol) -> dualobj
```

whichsol

Selects a solution.

dualobj

Objective value corresponding to the dual solution.

2.61 getdviolbarvar()

Let $(\bar{S}_j)^*$ be the value of variable $\bar{S}_j$ for the specified solution. Then the dual violation of the solution associated with variable $\bar{S}_j$ is given by

$$\max(-\lambda_{\min}(\bar{S}_j), 0.0).$$

Both when the solution is a certificate of primal infeasibility or when it is dual feasibible solution the violation should be small. Syntax:

```
getdviolbarvar(whichsol,sub) -> viol
```

whichsol

Selects a solution.

sub

An array of indexes of $\bar{X}$ variables.

viol

`viol[k]` is violation of the solution for the constraint $\bar{S}_{\texttt{sub}[k]} \in \mathcal{S}$.

2.62 getdviolcon()

The violation of the dual solution associated with the i'th constraint is computed as follows

$$\max(\rho((s_l^c)_i^*, (b_l^c)_i), \rho((s_u^c)_i^*, -(b_u^c)_i), |-y_i + (s_l^c)_i^* - (s_u^c)_i^*|)$$

where

$$\rho(x, l) = \begin{cases} -x, & l > -\infty, \\ |x|, & \text{otherwise} \end{cases}$$

Both when the solution is a certificate of primal infeasibility or it is a dual feasibible solution the violation should be small. Syntax:

```
getdviolcon(whichsol,sub) -> viol
```

whichsol

Selects a solution.

sub

An array of indexes of constraints.

viol

`viol[k]` is the violation of dual solution associated with the constraint `sub[k]`.

2.63 getdviolcones()

Let $(s_n^x)^*$ be the value of variable (s_n^x) for the specified solution. For simplicity let us assume that s_n^x is a member of quadratic cone, then the violation is computed as follows

$$\begin{cases} \max(0, \left\|(s_n^x)_{2;n}\right\|^* - (s_n^x)_1^*)/\sqrt{2}, & (s_n^x)^* \geq -\left\|(s_n^x)_{2:n}^*\right\|, \\ \left\|(s_n^x)^*\right\|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of primal infeasibility or when it is a dual feasibible solution the violation should be small.

If s_n^x is a member of a conic rotated cone, then the violation is computed mapping the rotated cone to a standard quadratic cone. Indeed, it is well known that there exists an ortogonal transformation T such that

$$s_n^x \in \mathcal{Q} \Leftrightarrow T s_n^x \in \mathcal{Q}_\nabla.$$

Syntax:

```
getdviolcones(whichsol,sub) -> viol
```

whichsol

Selects a solution.

sub

An array of indexes of dual quadratic cones.

viol

`viol[k]` violation of the solution associated with `sub[k]`'th dual conic quadratic constraint.

2.64 getdviolvar()

The violation fo dual solution associated with the j'th variable is computed as follows

$$\max(\rho((s_l^x)_i^*, (b_l^x)_i), \rho((s_u^x)_i^*, -(b_u^x)_i), |\sum j = 1^{numcon} a_{ij} y_i + (s_l^x)_i^* - (s_u^x)_i^* - \tau c_j|)$$

where

$$\rho(x,l) = \begin{cases} -x, & l > -\infty, \\ |x|, & \text{otherwise} \end{cases}$$

$\tau = 0$ if the the solution is certificate of dual infeasibility and $\tau = 1$ otherwise. The formula for computing the violation is only shown for linear case but is generalized approriately for the more general problems. Syntax:

```
getdviolvar(whichsol,sub) -> viol
```

whichsol

Selects a solution.

sub

An array of indexes of x variables.

viol

`viol[k]` is the maximal violation of the solution for the constraints $(s_l^x)_{\texttt{sub}[k]} \geq 0$ and $(s_u^x)_{\texttt{sub}[k]} \geq 0$.

2.65 getinti()

Deprecated.

Obtains the primal equation infeasibility.

$$\texttt{peqi}[i] = \left|(Ax - x^c)_{\texttt{sub}[i]}\right| \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1.$$

Syntax:

```
getinti(whichsol,sub) -> inti
```

whichsol

Selects a solution.

sub

Variable indexes for which to calculate the integer infeasibility.

inti

`inti[i]` contains integer infeasibility of variable `sub[i]`.

2.66 getintinf()

Obtains an integer information item from the task information database. Syntax:

```
getintinf(whichiinf) -> ivalue
```

whichiinf

Specifies an information item.

ivalue

The value of the required integer information item.

2.67 getintparam()

Obtains the value of an integer parameter. Syntax:

```
getintparam(param) -> parvalue
```

param

Which parameter.

parvalue

Parameter value.

2.68 getlenbarvarj()

Obtains the length of the jth semidefinite variable i.e. the number of elements in the triangular part. Syntax:

```
getlenbarvarj(j) -> lenbarvarj
```

j

Index of the semidefinite variable whose length if requested.

lenbarvarj

Number of scalar elements in the lower triangular part of the semidefinite variable.

2.69 getlintinf()

Obtains an integer information item from the task information database. Syntax:

```
getlintinf(whichliinf) -> ivalue
```

whichliinf

Specifies an information item.

ivalue

The value of the required integer information item.

2.70 getmaxnumanz()

Obtains number of preallocated non-zeros in A. When this number of non-zeros is reached MOSEK will automatically allocate more space for A. Syntax:

```
getmaxnumanz() -> maxnumanz
```

maxnumanz
: Number of preallocated non-zero elements in A.

2.71 getmaxnumbarvar()

Obtains the number of semidefinite variables. Syntax:

```
getmaxnumbarvar() -> maxnumbarvar
```

maxnumbarvar
: Obtains maximum number of semidefinite variable currently allowed.

2.72 getmaxnumcon()

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

```
getmaxnumcon() -> maxnumcon
```

maxnumcon
: Number of preallocated constraints in the optimization task.

2.73 getmaxnumcone()

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

```
getmaxnumcone() -> maxnumcone
```

maxnumcone
: Number of preallocated conic constraints in the optimization task.

2.74 getmaxnumqnz()

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. Syntax:

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

maxnumqnz

Number of non-zero elements preallocated in quadratic coefficient matrixes.

2.75 getmaxnumvar()

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

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

maxnumvar

Number of preallocated variables in the optimization task.

2.76 getmemusage()

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

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

meminuse

Amount of memory currently used by the `task`.

maxmemuse

Maximum amount of memory used by the `task` until now.

2.77 getnumanz()

Obtains the number of non-zeros in A. Syntax:

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

numanz

Number of non-zero elements in A.

2.78 getnumanz64()

Obtains the number of non-zeros in A. Syntax:

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

numanz

Number of non-zero elements in A.

2.79 getnumbarablocktriplets()

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

```
getnumbarablocktriplets() -> num
```

num

Number elements in the block triplet form of $\bar{A}$.

2.80 getnumbaranz()

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

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

nz

The number of nonzero elements in $\bar{A}$ i.e. the number of $\bar{a}_{ij}$ elements that is nonzero.

2.81 getnumbarcblocktriplets()

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

```
getnumbarcblocktriplets() -> num
```

num

An upper bound on the number elements in the block trip let form of $\bar{c}$.

2.82 getnumbarcnz()

Obtains the number of nonzero elements in $\bar{c}$. Syntax:

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

nz

The number of nonzeros in $\bar{c}$ i.e. the number of elements $\bar{c}_j$ that is diffrent from 0.

2.83 getnumbarvar()

Obtains the number of semidefinite variables. Syntax:

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

numbarvar

Number of semidefinite variable in the problem.

2.84 getnumcon()

Obtains the number of constraints. Syntax:

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

numcon

Number of constraints.

2.85 getnumcone()

Obtains the number of cones. Syntax:

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

numcone

Number conic constraints.

2.86 getnumconemem()

Obtains the number of members in a cone. Syntax:

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

k

Index of the cone.

nummem

Number of member variables in the cone.

2.87 getnumintvar()

Obtains the number of integer-constrained variables. Syntax:

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

numintvar

Number of integer variables.

2.88 getnumparam()

Obtains the number of parameters of a given type. Syntax:

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

partype

Parameter type.

numparam

Identical to the number of parameters of the type `partype`.

2.89 getnumqconknz()

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

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

k

Index of the constraint for which the number of non-zero quadratic terms should be obtained.

numqcnz

Number of quadratic terms.

2.90 getnumqconknz64()

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

```
getnumqconknz64(k) -> numqcnz
```

k

Index of the constraint for which the number quadratic terms should be obtained.

numqcnz

Number of quadratic terms.

2.91 getnumqobjnz()

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

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

numqonz

Number of non-zero elements in Q^o.

2.92 getnumsymmat()

Get the number of symmetric matrixes stored in the vector E. Syntax:

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

num

Returns the number of symmetric sparse matrixes.

2.93 getnumvar()

Obtains the number of variables. Syntax:

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

numvar

Number of variables.

2.94 getobjname()

Obtains the name assigned to the objective function. Syntax:

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

objname

Assigned the objective name.

2.95 getobjnamelen()

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

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

len

Assigned the length of the objective name.

2.96 getobjsense()

Gets the objective sense of the task. Syntax:

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

sense

The returned objective sense.

2.97 getparammax()

Obtains the maximum index of a parameter of a given type plus 1. Syntax:

```
getparammax(partype) -> parammax
```

partype

Parameter type.

parammax

2.98 getparamname()

Obtains the name for a parameter `param` of type `partype`. Syntax:

```
getparamname(partype,param) -> parname
```

partype

Parameter type.

param

Which parameter.

parname

Parameter name.

2.99 getpbi()

Deprecated.

Obtains the primal bound infeasibility. If `acmode` is `accmode.con` then

$$\texttt{pbi}[i] = \max(x^c_{\texttt{sub[i]}} - u^c_{\texttt{sub[i]}}, l^c_{\texttt{sub[i]}} - x^c_{\texttt{sub[i]}}, 0) \;\; \texttt{for} \;\; i = 0, \ldots, \texttt{len} - 1$$

If `acmode` is `accmode.var` then

$$\texttt{pbi}[i] = \max(x_{\texttt{sub[i]}} - u^x_{\texttt{sub[i]}}, l^x_{\texttt{sub[i]}} - x_{\texttt{sub[i]}}, 0) \;\; \texttt{for} \;\; i = 0, \ldots, \texttt{len} - 1$$

Syntax:

```
getpbi(whichsol,accmode,sub,normalize) -> pbi
```

whichsol

Selects a solution.

accmode

If set to `accmode.var` return bound infeasibility for x otherwise for x^c.

sub

An array of constraint or variable indexes.

normalize

If non-zero, normalize with largest absolute value of the input data used to compute the individual infeasibility.

pbi

Bound infeasibility for x or x^c.

2.100 getpcni()

Deprectaed. Syntax:

```
getpcni(whichsol,sub) -> pcni
```

whichsol

Selects a solution.

sub

Constraint indexes for which to calculate the equation infeasibility.

pcni

`pcni[i]` contains primal cone infeasibility for the cone with index `sub[i]`.

2.101 getpeqi()

Deprecated.

Obtains the primal equation infeasibility.

$$\texttt{peqi}[i] = \left|(Ax - x^c)_{\texttt{sub}[i]}\right| \quad \texttt{for} \quad i = 0, \ldots, \texttt{len} - 1.$$

Syntax:

```
getpeqi(whichsol,sub,normalize) -> peqi
```

whichsol

Selects a solution.

sub

Constraint indexes for which to calculate the equation infeasibility.

normalize

If non-zero, normalize with largest absolute value of the input data used to compute the individual infeasibility.

peqi

`peqi[i]` contains equation infeasibility of constraint `sub[i]`.

2.102 getprimalobj()

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

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

whichsol

Selects a solution.

primalobj

Objective value corresponding to the primal solution.

2.103 getprobtype()

Obtains the problem type. Syntax:

```
getprobtype() -> probtype
```

probtype

The problem type.

2.104 getprosta()

Obtains the problem status. Syntax:

```
getprosta(whichsol) -> prosta
```

whichsol

Selects a solution.

prosta

Problem status.

2.105 getpviolbarvar()

Let $(\bar{X}_j)^*$ be the value of variable $\bar{X}_j$ for the specified solution. Then the primal violation of the solution associated with variable $\bar{X}_j$ is given by

$$\max(-\lambda_{\min}(\bar{X}_j), 0.0).$$

Syntax:

```
getpviolbarvar(whichsol,sub) -> viol
```

whichsol

Selects a solution.

sub

An array of indexes of $\bar{X}$ variables.

viol

`viol[k]` is how much the solution violate the constraint $\bar{X}_{\texttt{sub}[k]} \in \mathcal{S}^+$.

2.106 getpviolcon()

The primal violation of the solution associated of constraint is computed by

$$\max(l_i^c\tau - (x_i^c)^*), (x_i^c)^*\tau - u_i^c\tau, |\sum_{j=1}^{numvar} a_{ij}x_j^* - x_i^c|)$$

where τ is defined as follows. If the solution is a certificate of dual infeasibility, then $\tau = 0$ and otherwise $\tau = 1$. Both when the solution is a valid certificate of dual infeasibility or when it is primal feasibible solution the violation should be small. The above is only shown for linear case but is appropriately generalized for the other cases. Syntax:

```
getpviolcon(whichsol,sub) -> viol
```

whichsol

Selects a solution.

sub

An array of indexes of constraints.

viol

`viol[k]` associated with the solution for the `sub[k]`'th constraint.

2.107 getpviolcones()

Let x^* be the value of variable x for the specified solution. For simplicity let us assume that x is a member of quadratic cone, then the violation is computed as follows

$$\begin{cases} \max(0, \|x_{2;n}\| - x_1)/\sqrt{2}, & x_1 \geq -\|x_{2:n}\|, \\ \|x\|, & \text{otherwise.} \end{cases}$$

Both when the solution is a certificate of dual infeasibility or when it is a primal feasibible solution the violation should be small.

If x is a member of a conic rotated cone, then the violation is computed mapping the rotated cone to a standard quadratic cone. Indeed, it is well known that there exists an ortogonal transformation T such that

$$x \in \mathcal{Q} \Leftrightarrow Tx \in \mathcal{Q}_{\nabla}.$$

Syntax:

```
getpviolcones(whichsol,sub) -> viol
```

whichsol

Selects a solution.

sub

An array of indexes of quadratic cones.

viol

`viol[k]` violation of the solution associated with `sub[k]`'th conic quadratic constraint.

2.108 getpviolvar()

Let x_j^* be the value of variable x_j for the specified solution. Then the primal violation of the solution associated with variable x_j is given by

$$\max(l_j^x \tau - x_j^*, x_j^* - u_j^x \tau).$$

where τ is defined as follows. If the solution is a certificate of dual infeasibility, then $\tau = 0$ and otherwise $\tau = 1$. Both when the solution is a valid certificate of dual infeasibility or when it is primal feasibible solution the violation should be small. Syntax:

```
getpviolvar(whichsol,sub) -> viol
```

whichsol

Selects a solution.

sub

An array of indexes of x variables.

viol

`viol[k]` is the violation associated the solution for variable x_j.

2.109 getqconk()

Obtains all the quadratic terms in a constraint. The quadratic terms are stored sequentially `qcsubi`, `qcsubj`, and `qcval`. Syntax:

```
getqconk(k) -> qcsubi,qcsubj,qcval
```

k

Which constraint.

qcsubi

i subscripts for q_{ij}^k.

qcsubj

j subscripts for q_{ij}^k.

qcval

Numerical value for q_{ij}^k.

2.110 getqobj()

Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in `qosubi`, `qosubj`, and `qoval`. Syntax:

```
getqobj() -> qosubi,qosubj,qoval
```

qosubi

i subscript for q_{ij}^o.

qosubj

j subscript for q_{ij}^o.

qoval

Numerical value for q_{ij}^o.

2.111 getqobj64()

Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in `qosubi`, `qosubj`, and `qoval`. Syntax:

```
getqobj64() -> qosubi,qosubj,qoval
```

qosubi

i subscript for q^o_{ij}.

qosubj

j subscript for q^o_{ij}.

qoval

Numerical value for q^o_{ij}.

2.112 getqobjij()

Obtains one coefficient q^o_{ij} in the quadratic term of the objective. Syntax:

```
getqobjij(i,j) -> qoij
```

i

Row index of the coefficient.

j

Column index of coefficient.

qoij

The required coefficient.

2.113 getreducedcosts()

Computes the reduced costs for a sequence of variables and return them in the variable `redcosts` i.e.

$$\texttt{redcosts}[j-\texttt{first}] = (s^x_l)_j - (s^x_u)_j,\ j=\texttt{first},\ldots,last. \tag{2.1}$$

Syntax:

```
getreducedcosts(whichsol,first,last) -> redcosts
```

whichsol

Selects a solution.

first

See formula (2.1) for the definition.

last

See formula (2.1) for the definition.

redcosts

The reduced costs in the required sequence of variables are stored sequentially in `redcosts` starting at `redcosts[1]`.

2.114 getskc()

Obtains the status keys for the constraints. Syntax:

```
getskc(whichsol) -> skc
```

whichsol

Selects a solution.

skc

Status keys for the constraints.

2.115 getskcslice()

Obtains the status keys for the constraints. Syntax:

```
getskcslice(whichsol,first,last) -> skc
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

skc

Status keys for the constraints.

2.116 getskx()

Obtains the status keys for the scalar variables. Syntax:

```
getskx(whichsol) -> skx
```

whichsol

Selects a solution.

skx

Status keys for the variables.

2.117 getskxslice()

Obtains the status keys for the variables. Syntax:

```
getskxslice(whichsol,first,last) -> skx
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

skx

Status keys for the variables.

2.118 getslc()

Obtains the s_l^c vector for a solution. Syntax:

```
getslc(whichsol) -> slc
```

whichsol

Selects a solution.

slc

The s_l^c vector.

2.119 getslcslice()

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

```
getslcslice(whichsol,first,last) -> slc
```

whichsol
: Selects a solution.

first
: First index in the sequence.

last
: Last index plus 1 in the sequence.

slc
: Dual variables corresponding to the lower bounds on the constraints (s_l^c).

2.120 getslx()

Obtains the s_l^x vector for a solution. Syntax:

```
getslx(whichsol) -> slx
```

whichsol
: Selects a solution.

slx
: The s_l^x vector.

2.121 getslxslice()

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

```
getslxslice(whichsol,first,last) -> slx
```

whichsol
: Selects a solution.

first
: First index in the sequence.

last
: Last index plus 1 in the sequence.

slx
: Dual variables corresponding to the lower bounds on the variables (s_l^x).

2.122 getsnx()

Obtains the s_n^x vector for a solution. Syntax:

```
getsnx(whichsol) -> snx
```

whichsol

Selects a solution.

snx

The s_n^x vector.

2.123 getsnxslice()

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

```
getsnxslice(whichsol,first,last) -> snx
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

snx

Dual variables corresponding to the conic constraints on the variables (s_n^x).

2.124 getsolsta()

Obtains the solution status. Syntax:

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

whichsol

Selects a solution.

solsta

Solution status.

2.125 getsolution()

Obtains the complete solution.

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

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

and the corresponding dual problem is

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

In this case the mapping between variables and arguments to the function is as follows:

`xx`:

Corresponds to variable x.

`y`:

Corresponds to variable y.

`slc`:

Corresponds to variable s_l^c.

`suc`:

Corresponds to variable s_u^c.

`slx`:

Corresponds to variable s_l^x.

`sux`:

Corresponds to variable s_u^x.

`xc`:

Corresponds to Ax.

The meaning of the values returned by this function depend on the *solution status* returned in the argument `solsta`. The most important possible values of `solsta` are:

`solsta.optimal`

An optimal solution satisfying the optimality criteria for continuous problems is returned.

`solsta.integer_optimal`

An optimal solution satisfying the optimality criteria for integer problems is returned.

`solsta.prim_feas`

A solution satisfying the feasibility criteria.

`solsta.prim_infeas_cer`

A primal certificate of infeasibility is returned.

`solsta.dual_infeas_cer`

A dual certificate of infeasibility is returned.

Syntax:

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

whichsol

Selects a solution.

prosta

Problem status.

solsta

Solution status.

skc

Status keys for the constraints.

skx

Status keys for the variables.

skn

Status keys for the conic constraints.

xc

Primal constraint solution.

xx

Primal variable solution (x).

y

Vector of dual variables corresponding to the constraints.

slc

Dual variables corresponding to the lower bounds on the constraints (s_l^c).

suc

Dual variables corresponding to the upper bounds on the constraints (s_u^c).

slx

Dual variables corresponding to the lower bounds on the variables (s_l^x).

sux

Dual variables corresponding to the upper bounds on the variables (appears as s_u^x).

snx

Dual variables corresponding to the conic constraints on the variables (s_n^x).

2.126 getsolutioni()

Obtains the primal and dual solution information for a single constraint or variable. Syntax:

```
getsolutioni(accmode,i,whichsol) -> sk,x,sl,su,sn
```

accmode

If set to accmode.con the solution information for a constraint is retrieved. Otherwise for a variable.

i

Index of the constraint or variable.

whichsol

Selects a solution.

sk

Status key of the constraint of variable.

x

Solution value of the primal variable.

sl

Solution value of the dual variable associated with the lower bound.

su

Solution value of the dual variable associated with the upper bound.

sn

Solution value of the dual variable associated with the cone constraint.

2.127 getsolutioninf()

Deprecated. Use Task.getsolutioninfo instead. Syntax:

```
getsolutioninf(whichsol) -> prosta,solsta,primalobj,maxpbi,maxpcni,maxpeqi,maxinti,dualobj,maxdbi,maxdcni,maxdeqi
```

whichsol

Selects a solution.

prosta

Problem status.

solsta

Solution status.

primalobj

Value of the primal objective.

$$c^T x + c^f$$

maxpbi

Maximum infeasibility in primal bounds on variables.

$$\max\left\{0, \max_{i\in 1,\ldots,n-1}(x_i - u_i^x), \max_{i\in 1,\ldots,n-1}(l_i^x - x_i), \max_{i\in 1,\ldots,n-1}(x_i^c - u_i^c), \max_{i\in 1,\ldots,n-1}(l_i^c - x_i^c)\right\}$$

maxpcni

Maximum infeasibility in the primal conic constraints.

maxpeqi

Maximum infeasibility in primal equality constraints.

$$\|Ax - x^c\|_\infty$$

maxinti

Maximum infeasibility in integer constraints.

$$\max_{i\in\{0,\ldots,n-1\}}(\min(x_i - \lfloor x_i \rfloor, \lceil x_i \rceil - x_i)).$$

dualobj

Value of the dual objective.

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

maxdbi

Maximum infeasibility in bounds on dual variables.

$$\max\{0, \max_{i\in\{0,\ldots,n-1\}} -(s_l^x)_i, \max_{i\in\{0,\ldots,n-1\}} -(s_u^x)_i, \max_{i\in\{0,\ldots,m-1\}} -(s_l^c)_i, \max_{i\in\{0,\ldots,m-1\}} -(s_u^c)_i\}$$

maxdcni

Maximum infeasibility in the dual conic constraints.

maxdeqi

Maximum infeasibility in the dual equality constraints.

$$\max\left\{ \left\|A^T y + s_l^x - s_u^x - c\right\|_\infty, \left\| -y + s_l^c - s_u^c \right\|_\infty \right\}$$

2.128 getsolutioninfo()

Obtains information about a solution. Syntax:

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

whichsol

Selects a solution.

pobj

The primal objective value as computed by Task.getprimalobj.

pviolcon

Maximal primal violation of the solution associated with the x^c variables where the violations are computed by Task.getpviolcon.

pviolvar

Maximal primal violation of the solution for the x^x variables where the violations are computed by Task.getpviolvar.

pviolbarvar

Maximal primal violation of solution for the $\bar{X}$ variables where the violations are computed by Task.getpviolbarvar.

pviolcone

Maximal primal violation of solution for the conic constraints where the violations are computed by Task.getpviolcones.

pviolitg

Maximal violation in the integer constraints. The violation for an integer constrained variable x_j is given by

$$\min(x_j - \lfloor x_j \rfloor, \lceil x_j \rceil - x_j).$$

This number is always zero for the interior-point and the basic solutions.

dobj

Dual objective value as computed as computed by Task.getdualobj.

dviolcon

Maximal violation of the dual solution associated with the x^c variable as computed by as computed by Task.getdviolcon.

dviolvar

Maximal violation of the dual solution associated with the x variable as computed by as computed by Task.getdviolvar.

dviolbarvar

Maximal violation of the dual solution associated with the $\bar{s}$ variable as computed by as computed by Task.getdviolbarvar.

dviolcone

Maximal violation of the dual solution associated with the dual conic constraints as computed by Task.getdviolcones.

2.129 getsolutionslice()

Obtains a slice of the solution.

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

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

and the corresponding dual problem is

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

The `solitem` argument determines which part of the solution is returned:

solitem.xx:

The variable values return x.

solitem.y:

The variable values return y.

solitem.slc:

The variable values return s_l^c.

solitem.suc:

The variable values return s_u^c.

solitem.slx:

The variable values return s_l^x.

solitem.sux:

The variable values return s_u^x.

A conic optimization problem has the same primal variables as in the linear case. Recall that the dual of a conic optimization problem is given by:

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

This introduces one additional dual variable s_n^x. This variable can be acceded by selecting solitem as solitem.snx.

The meaning of the values returned by this function also depends on the *solution status* which can be obtained with Task.getsolsta. Depending on the solution status value will be:

solsta.optimal

A part of the optimal solution satisfying the optimality criteria for continuous problems.

solsta.integer_optimal

A part of the optimal solution satisfying the optimality criteria for integer problems.

solsta.prim_feas

A part of the solution satisfying the feasibility criteria.

solsta.prim_infeas_cer

A part of the primal certificate of infeasibility.

`solsta.dual_infeas_cer`

A part of the dual certificate of infeasibility.

Syntax:

```
getsolutionslice(whichsol,solitem,first,last) -> values
```

whichsol

Selects a solution.

solitem

Which part of the solution is required.

first

Index of the first value in the slice.

last

Value of the last index+1 in the slice, e.g. if $xx[5, \ldots, 9]$ is required `last` should be 10.

values

The values in the required sequence are stored sequentially in `values` starting at `values[1]`.

2.130 getsparsesymmat()

Get a single symmetric matrix from the matrix store. Syntax:

```
getsparsesymmat(idx) -> subi,subj,valij
```

idx

Index of the matrix to get.

subi

Row subscripts of the matrix non-zero elements.

subj

Column subscripts of the matrix non-zero elements.

valij

Coefficients of the matrix non-zero elements.

2.131 getstrparam()

Obtains the value of a string parameter. Syntax:

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

param

Which parameter.

len

The length of the parameter value.

parvalue

If this is not NULL, the parameter value is stored here.

2.132 getstrparamlen()

Obtains the length of a string parameter. Syntax:

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

param

Which parameter.

len

The length of the parameter value.

2.133 getsuc()

Obtains the s_u^c vector for a solution. Syntax:

```
getsuc(whichsol) -> suc
```

whichsol

Selects a solution.

suc

The s_u^c vector.

2.134 getsucslice()

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

```
getsucslice(whichsol,first,last) -> suc
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

suc

Dual variables corresponding to the upper bounds on the constraints (s_u^c).

2.135 getsux()

Obtains the s_u^x vector for a solution. Syntax:

```
getsux(whichsol) -> sux
```

whichsol

Selects a solution.

sux

The s_u^x vector.

2.136 getsuxslice()

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

```
getsuxslice(whichsol,first,last) -> sux
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

sux

Dual variables corresponding to the upper bounds on the variables (appears as s_u^x).

2.137 getsymbcon()

Obtains the name and corresponding value for the ith symbolic constant. Syntax:

```
getsymbcon(i) -> name,value
```

i

Index.

name

Name of the ith symbolic constant.

value

The corresponding value.

2.138 getsymmatinfo()

MOSEK maintains a vector denoted E of symmetric data matrixes. This function makes it possible to obtain important information about an data matrix in E. Syntax:

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

idx

Index of the matrix that is requested information about.

dim

Returns the dimension of the requested matrix.

nz

Returns the number of non-zeros in the requested matrix.

type

Returns the type of the requested matrix.

2.139 gettaskname()

Obtains the name assigned to the task. Syntax:

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

taskname

Is assigned the task name.

2.140 gettasknamelen()

Obtains the length the task name. Syntax:

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

len

Returns the length of the task name.

2.141 getvarbound()

Obtains bound information for one variable. Syntax:

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

i

Index of the variable for which the bound information should be obtained.

bk

Bound keys.

bl

Values for lower bounds.

bu

Values for upper bounds.

2.142 getvarboundslice()

Obtains bounds information for a slice of the variables. Syntax:

```
getvarboundslice(first,last) -> bk,bl,bu
```

first

First index in the sequence.

last

Last index plus 1 in the sequence.

bk

Bound keys.

bl

Values for lower bounds.

bu

Values for upper bounds.

2.143 getvarbranchdir()

Obtains the branching direction for a given variable j. Syntax:

```
getvarbranchdir(j) -> direction
```

j
: Index of the variable.

direction
: The branching direction assigned to variable j.

2.144 getvarbranchorder()

Obtains the branching priority and direction for a given variable j. Syntax:

```
getvarbranchorder(j) -> priority,direction
```

j
: Index of the variable.

priority
: The branching priority assigned to variable j.

direction
: The preferred branching direction for the j'th variable.

2.145 getvarbranchpri()

Obtains the branching priority for a given variable j. Syntax:

```
getvarbranchpri(j) -> priority
```

j
: Index of the variable.

priority
: The branching priority assigned to variable j.

2.146 getvarname()

Obtains a name of a variable. Syntax:

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

j

Index.

name

Is assigned the required name.

2.147 getvarnameindex()

Checks whether the name `somename` has been assigned to any variable. If it has been assigned to variable, then index of the variable is reported. Syntax:

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

somename

The name which should be checked.

asgn

Is non-zero if the name `somename` is assigned to a variable.

index

If the name `somename` is assigned to a variable, then `index` is the name of the variable.

2.148 getvarnamelen()

Obtains the length of a name of a variable variable. Syntax:

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

i

Index.

len

Returns the length of the indicated name.

2.149 getvartype()

Gets the variable type of one variable. Syntax:

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

j

Index of the variable.

vartype

Variable type of variable `j`.

2.150 getvartypelist()

Obtains the variable type of one or more variables.

Upon return `vartype[k]` is the variable type of variable `subj[k]`. Syntax:

```
getvartypelist(subj) -> vartype
```

subj

A list of variable indexes.

vartype

The variables types corresponding to the variables specified by `subj`.

2.151 getxc()

Obtains the x^c vector for a solution. Syntax:

```
getxc(whichsol) -> xc
```

whichsol

Selects a solution.

xc

The x^c vector.

2.152 getxcslice()

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

```
getxcslice(whichsol,first,last) -> xc
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

xc

Primal constraint solution.

2.153 getxx()

Obtains the x^x vector for a solution. Syntax:

```
getxx(whichsol) -> xx
```

whichsol

Selects a solution.

xx

The x^x vector.

2.154 getxxslice()

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

```
getxxslice(whichsol,first,last) -> xx
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

xx

Primal variable solution (x).

2.155 gety()

Obtains the y vector for a solution. Syntax:

```
gety(whichsol) -> y
```

whichsol

Selects a solution.

y

The y vector.

2.156 getyslice()

Obtains a slice of the y vector for a solution. Syntax:

```
getyslice(whichsol,first,last) -> y
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

y

Vector of dual variables corresponding to the constraints.

2.157 inputdata()

Input the linear part of an optimization problem.

Syntax:

```
inputdata(maxnumcon,maxnumvar,c,cfix,aptrb,aptre,asub,aval,bkc,blc,buc,bkx,blx,bux) -> nil
```

maxnumcon

Number of preallocated constraints in the optimization task.

maxnumvar

Number of preallocated variables in the optimization task.

c

Linear terms of the objective as a dense vector. The lengths is the number of variables.

cfix

Fixed term in the objective.

aptrb

Pointer to the first element in the rows or the columns of A.

aptre

Pointers to the last element + 1 in the rows or the columns of A.

asub

Coefficient subscripts.

aval

Coefficient values.

bkc

Bound keys for the constraints.

blc

Lower bounds for the constraints.

buc

Upper bounds for the constraints.

bkx

Bound keys for the variables.

blx

Lower bounds for the variables.

bux

Upper bounds for the variables.

2.158 isdouparname()

Checks whether `parname` is a valid double parameter name. Syntax:

```
isdouparname(parname) -> param
```

parname

Parameter name.

param

Which parameter.

2.159 isintparname()

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

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

parname

Parameter name.

param

Which parameter.

2.160 isstrparname()

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

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

parname

Parameter name.

param

Which parameter.

2.161 linkfiletostream()

Directs all output from a task stream to a file. Syntax:

```
linkfiletostream(whichstream,filename,append) -> nil
```

whichstream

Index of the stream.

filename

The name of the file where text from the stream defined by `whichstream` is written.

append

If this argument is 0 the output file will be overwritten, otherwise text is append to the output file.

2.162 onesolutionsummary()

Prints a short summary for a specified solution. Syntax:

```
onesolutionsummary(whichstream,whichsol) -> nil
```

whichstream

Index of the stream.

whichsol

Selects a solution.

2.163 optimize()

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.

Syntax:

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

trmcode

Is either rescode.ok or a termination response code.

2.164 optimizersummary()

Prints a short summary with optimizer statistics for last optimization. Syntax:

```
optimizersummary(whichstream) -> nil
```

whichstream

Index of the stream.

2.165 primalrepair()

The function repairs a primal infeasible optimization problem by adjusting the bounds on the constraints and variables where the adjustment is computed as the minimal weigthed sum relaxation to the bounds on the constraints and variables.

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

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

Note the function modifies the bounds on the constraints and variables. If this is not a desired feature, then apply the fucntion to a cloned task. Syntax:

```
primalrepair(wlc,wuc,wlx,wux) -> nil
```

wlc

$(w_l^c)_i$ is the weight associated with relaxing the lower bound on constraint i. If the weigth is negative, then the lower bound is not relaxed. Moreover, if the argument is `nil`, then all the weights are assumed to be 1.

wuc

$(w_u^c)_i$ is the weight associated with relaxing the upper bound on constraint i. If the weigth is negative, then the upper bound is not relaxed. Moreover, if the argument is `nil`, then all the weights are assumed to be 1.

wlx

$(w_l^x)_j$ is the weight associated with relaxing the upper bound on constraint j. If the weigth is negative, then the lower bound is not relaxed. Moreover, if the argument is `nil`, then all the weights are assumed to be 1.

wux

$(w_l^x)_i$ is the weight associated with relaxing the upper bound on variable j. If the weigth is negative, then the upper bound is not relaxed. Moreover, if the argument is `nil`, then all the weights are assumed to be 1.

2.166 primalsensitivity()

Calculates sensitivity information for bounds on variables and constraints.

The constraints for which sensitivity analysis is performed are given by the data structures:

- `subi` Index of constraint to analyze.
- `marki` Indicate for which bound of constraint `subi[i]` sensitivity analysis is performed. If `marki[i] = mark.up` the upper bound of constraint `subi[i]` is analyzed, and if `marki[i] = mark.lo` the lower bound is analyzed. If `subi[i]` is an equality constraint, either `mark.lo` or `mark.up` can be used to select the constraint for sensitivity analysis.

Consider the problem:

$$\begin{array}{lccc} \text{minimize} & x_1 + x_2 & & \\ \text{subject to} -1 \leq & x_1 - x_2 & \leq & 1, \\ & x_1 & = & 0, \\ & x_1 \geq 0, x_2 \geq 0 & & \end{array}$$

Suppose that

- `numi = 1;`
- `subi = [0];`
- `marki = [mark.up]`

then

`leftpricei[0]`, `rightpricei[0]`, `leftrangei[0]` and `rightrangei[0]` will contain the sensitivity information for the upper bound on constraint 0 given by the expression:

$$x_1 - x_2 \leq 1$$

Similarly, the variables for which to perform sensitivity analysis are given by the structures:

- `subj` Index of variables to analyze.
- `markj` Indicate for which bound of variable `subi[j]` sensitivity analysis is performed. If `markj[j] = mark.up` the upper bound of constraint `subi[j]` is analyzed, and if `markj[j] = mark.lo` the lower bound is analyzed. If `subi[j]` is an equality constraint, either `mark.lo` or `mark.up` can be used to select the constraint for sensitivity analysis.

The type of sensitivity analysis to be performed (basis or optimal partition) is controlled by the parameter `iparam.sensitivity_type`. Syntax:

```
primalsensitivity(subi,marki,subj,markj) -> leftpricei,rightpricei,leftrangei,rightrangei,leftpricej,rightpricej,leftrangej,
```

subi

Indexes of bounds on constraints to analyze.

marki

The value of `marki[i]` specifies for which bound (upper or lower) on constraint `subi[i]` sensitivity analysis should be performed.

subj

Indexes of bounds on variables to analyze.

markj

The value of `markj[j]` specifies for which bound (upper or lower) on variable `subj[j]` sensitivity analysis should be performed.

leftpricei

`leftpricei[i]` is the left shadow price for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

rightpricei

`rightpricei[i]` is the right shadow price for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

leftrangei

`leftrangei[i]` is the left range for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

rightrangei

`rightrangei[i]` is the right range for the upper/lower bound (indicated by `marki[i]`) of the constraint with index `subi[i]`.

leftpricej

`leftpricej[j]` is the left shadow price for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]`.

rightpricej

`rightpricej[j]` is the right shadow price for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]` .

leftrangej

`leftrangej[j]` is the left range for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]`.

rightrangej

`rightrangej[j]` is the right range for the upper/lower bound (indicated by `marki[j]`) on variable `subj[j]`.

2.167 printdata()

Prints a part of the problem data to a stream. This function is normally used for debugging purposes only, e.g. to verify that the correct data has been inputted. Syntax:

```
printdata(whichstream,firsti,lasti,firstj,lastj,firstk,lastk,c,qo,a,qc,bc,bx,vartype,cones) -> nil
```

whichstream

Index of the stream.

firsti

Index of first constraint for which data should be printed.

lasti

Index of last constraint plus 1 for which data should be printed.

firstj

Index of first variable for which data should be printed.

lastj

Index of last variable plus 1 for which data should be printed.

firstk

Index of first cone for which data should be printed.

lastk

Index of last cone plus 1 for which data should be printed.

c

If non-zero c is printed.

qo

If non-zero Q^o is printed.

a

If non-zero A is printed.

qc

If non-zero Q^k is printed for the relevant constraints.

bc

If non-zero the constraints bounds are printed.

bx

If non-zero the variable bounds are printed.

vartype

If non-zero the variable types are printed.

cones

If non-zero the conic data is printed.

2.168 probtypetostr()

Obtains a string containing the name of a problem type given. Syntax:

```
probtypetostr(probtype) -> str
```

probtype

Problem type.

str

String corresponding to the problem type key `probtype`.

2.169 putacol()

Replaces all entries in column j of A. Assuming that there are no duplicate subscripts in `subj`, assignment is performed as follows:

$$A_{\texttt{subj}[k],\texttt{j}} = \texttt{valj}[k], \quad k = 0, \ldots, \texttt{nzj} - 1$$

All other entries in column j are set to zero. Syntax:

```
putacol(j,subj,valj) -> nil
```

j

Index of column in A.

subj

Row indexes of non-zero values in column j of A.

valj

New non-zero values of column j in A.

2.170 putacollist()

Replaces all elements in a set of columns of A. The elements are replaced as follows

$$\begin{array}{ll}\texttt{for} & i = 1, \ldots, num \\ & a_{\texttt{asub}[k],\texttt{sub}[i]} = \texttt{aval}[k], \quad k = \texttt{aptrb}[i], \ldots, \texttt{aptre}[i] - 1.\end{array}$$

Syntax:

```
putacollist(sub,ptrb,ptre,asub,aval) -> nil
```

sub

Indexes of columns that should be replaced. `sub` should not contain duplicate values.

ptrb

Array of pointers to the first element in the columns stored in `asub` and `aval`.

ptre

Array of pointers to the last element plus one in the columns stored in `asub` and `aval`.

asub

`asub` contains the new variable indexes.

aval

Coefficient values.

2.171 putaij()

Changes a coefficient in A using the method

$$a_{\texttt{ij}} = \texttt{aij}.$$

Syntax:

```
putaij(i,j,aij) -> nil
```

i

Index of the constraint in which the change should occur.

j

Index of the variable in which the change should occur.

aij

New coefficient for $a_{i,j}$.

2.172 putaijlist()

Changes one or more coefficients in A using the method

$$a_{\texttt{subi[k]},\texttt{subj[k]}} = \texttt{valij[k]}, \quad k = 0, \ldots, \texttt{num} - 1.$$

Syntax:

```
putaijlist(subi,subj,valij) -> nil
```

subi

Constraint indexes in which the change should occur.

subj

Variable indexes in which the change should occur.

valij

New coefficient values for $a_{i,j}$.

2.173 putarow()

Replaces all entries in row i of A. Assuming that there are no duplicate subscripts in `subi`, assignment is performed as follows:

$$A_{\texttt{i},\texttt{subi}[k]} = \texttt{vali}[k], \quad k = 0, \ldots, \texttt{nzi} - 1$$

All other entries in row i are set to zero. Syntax:

```
putarow(i,subi,vali) -> nil
```

i

Index of row in A.

subi

Row indexes of non-zero values in row i of A.

vali

New non-zero values of row i in A.

2.174 putarowlist()

Replaces all elements in a set of rows of A. The elements are replaced as follows

$$\begin{array}{ll} \texttt{for} & i = \texttt{first}, \ldots, \texttt{last} - 1 \\ & a_{\texttt{sub[i]},\texttt{asub}[k]} = \texttt{aval}[k], \quad k = \texttt{aptrb}[i], \ldots, \texttt{aptre}[i] - 1. \end{array}$$

Syntax:

```
putarowlist(sub,ptrb,ptre,asub,aval) -> nil
```

sub

Indexes of rows or columns that should be replaced. `sub` should not contain duplicate values.

ptrb

Array of pointers to the first element in the rows stored in `asub` and `aval`.

ptre

Array of pointers to the last element plus one in the rows stored in `asub` and `aval`.

asub

`asub` contains the new variable indexes.

aval

Coefficient values.

2.175 putbarablocktriplet()

Inputs the $\bar{A}$ in block triplet form. Syntax:

```
putbarablocktriplet(num,subi,subj,subk,subl,valijkl) -> nil
```

num

Number of elements in the block triplet form.

subi

Constraint index.

subj

Symmetric matrix variable index.

subk

Block row index.

subl

Block column index.

valijkl

The numerical value associated with the block triplet.

2.176 putbaraij()

This function puts one element associated with $\bar{X}_j$ in the $\bar{A}$ matrix.

Each element in the $\bar{A}$ matrix is a weighted sum of symmetric matrixes, i.e. $\bar{A}_{ij}$ is a symmetric matrix with dimensions as $\bar{X}_j$. By default all elements in $\bar{A}$ are 0, so only non-zero elements need be added.

Setting the same elements again will overwrite the earlier entry.

The symmetric matrixes themselves are defined separately using the funtion `Task.appendsparsesymmat`. Syntax:

```
putbaraij(i,j,sub,weights) -> nil
```

i

Row index of $\bar{A}$.

j

Column index of $\bar{A}$.

sub

See argument `weights` for an explenation.

weights

`weights[k]` times `sub[k]`'th term of E is added to $\bar{A}_{ij}$.

2.177 putbarcblocktriplet()

Inputs the $\bar{C}$ in block triplet form. Syntax:

```
putbarcblocktriplet(num,subj,subk,subl,valjkl) -> nil
```

num

Number of elements in the block triplet form.

subj

Symmetric matrix variable index.

subk

Block row index.

subl

Block column index.

valjkl

The numerical value associated with the block triplet.

2.178 putbarcj()

This function puts one element associated with $\bar{X}_j$ in the $\bar{c}$ vector.

Each element in the $\bar{c}$ vector is a weighted sum of symmetric matrixes, i.e. $\bar{c}_j$ is a symmetric matrix with dimensions as $\bar{X}_j$. By default all elements in $\bar{c}$ are 0, so only non-zero elements need be added.

Setting the same elements again will overwrite the earlier entry.

The symmetric matrixes themselves are defined separately using the funtion `Task.appendsparsesymmat`. Syntax:

```
putbarcj(j,sub,weights) -> nil
```

j

Index of the element in $\bar{c}$ that should be changed.

sub

`sub` is list of indexes of those symmetric matrixes appearing in sum.

weights

The weights of the terms in the weighted sum that forms $\mathtt{c}_j$.

2.179 putbarsj()

Sets the dual solution for a semidefinite variable. Syntax:

```
putbarsj(whichsol,j,barsj) -> nil
```

whichsol

Selects a solution.

j

Index of the semidefinite variable.

barsj

Value of $\bar{s}_j$.

2.180 putbarvarname()

Puts the name of a semidefinite variable. Syntax:

```
putbarvarname(j,name) -> nil
```

j

Index of the variable.

name

The variable name.

2.181 putbarxj()

Sets the primal solution for a semidefinite variable. Syntax:

```
putbarxj(whichsol,j,barxj) -> nil
```

whichsol

Selects a solution.

j

Index of the semidefinite variable.

barxj

Value of $\bar{X}_j$.

2.182 putbound()

Changes the bounds for either one constraint or one variable.

If the a bound value specified is numerically larger than `dparam.data_tol_bound_inf` it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than `dparam.data_tol_bound_wrn`, a warning will be displayed, but the bound is inputted as specified. Syntax:

```
putbound(accmode,i,bk,bl,bu) -> nil
```

accmode

Defines whether the bound for a constraint or a variable is changed.

i

Index of the constraint or variable.

bk

New bound key.

bl

New lower bound.

bu

New upper bound.

2.183 putboundlist()

Changes the bounds for either some constraints or variables. If multiple bound changes are specified for a constraint or a variable, only the last change takes effect. Syntax:

```
putboundlist(accmode,sub,bk,bl,bu) -> nil
```

accmode

Defines whether bounds for constraints (`accmode.con`) or variables (`accmode.var`) are changed.

sub

Subscripts of the bounds that should be changed.

bk

Constraint or variable index `sub[t]` is assigned the bound key `bk[t]`.

bl

Constraint or variable index `sub[t]` is assigned the lower bound `bl[t]`.

bu

Constraint or variable index `sub[t]` is assigned the upper bound `bu[t]`.

2.184 putboundslice()

Changes the bounds for a sequence of variables or constraints. Syntax:

```
putboundslice(con,first,last,bk,bl,bu) -> nil
```

con

Defines whether bounds for constraints (`accmode.con`) or variables (`accmode.var`) are changed.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

bk

Bound keys.

bl

Values for lower bounds.

bu

Values for upper bounds.

2.185 putcfix()

Replaces the fixed term in the objective by a new one. Syntax:

```
putcfix(cfix) -> nil
```

cfix

Fixed term in the objective.

2.186 putcj()

Modifies one coefficient in the linear objective vector c, i.e.

$$c_{\mathtt{j}} = \mathtt{cj}.$$

Syntax:

```
putcj(j,cj) -> nil
```

j

Index of the variable for which c should be changed.

cj

New value of c_j.

2.187 putclist()

Modifies elements in the linear term c in the objective using the principle

$$c_{\mathtt{subj[t]}} = \mathtt{val}[\mathtt{t}], \quad t = 0, \ldots, \mathtt{num} - 1.$$

If a variable index is specified multiple times in `subj` only the last entry is used. Syntax:

```
putclist(subj,val) -> nil
```

subj

Index of variables for which c should be changed.

val

New numerical values for coefficients in c that should be modified.

2.188 putconbound()

Changes the bounds for one constraint.

If the a bound value specified is numerically larger than `dparam.data_tol_bound_inf` it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than `dparam.data_tol_bound_wrn`, a warning will be displayed, but the bound is inputted as specified. Syntax:

```
putconbound(i,bk,bl,bu) -> nil
```

i

Index of the constraint.

bk

New bound key.

bl

New lower bound.

bu

New upper bound.

2.189 putconboundlist()

Changes the bounds for a list of constraints. If multiple bound changes are specified for a constraint, then only the last change takes effect. Syntax:

```
putconboundlist(sub,bkc,blc,buc) -> nil
```

sub

List constraints indexes.

bkc

New bound keys.

blc

New lower bound values.

buc

New upper bound values.

2.190 putconboundslice()

Changes the bounds for a slice of the constraints. Syntax:

```
putconboundslice(first,last,bk,bl,bu) -> nil
```

first

Index of the first constraint in the slice.

last

Index of the last constraint in the slice plus 1.

bk

New bound keys.

bl

New lower bounds.

bu

New upper bounds.

2.191 putcone()

Replaces a conic constraint. Syntax:

```
putcone(k,conetype,conepar,submem) -> nil
```

k

Index of the cone.

conetype

Specifies the type of the cone.

conepar

This argument is currently not used. Can be set to 0.0.

submem

Variable subscripts of the members in the cone.

2.192 putconename()

Puts the name of a cone. Syntax:

```
putconename(j,name) -> nil
```

j

Index of the variable.

name

The variable name.

2.193 putconname()

Puts the name of a constraint. Syntax:

```
putconname(i,name) -> nil
```

i

Index of the variable.

name

The variable name.

2.194 putcslice()

Modifies a slice in the linear term c in the objective using the principle

$$c_{\mathtt{j}} = \mathtt{slice}[\mathtt{j} - \mathtt{first}], \quad j = first, .., last$$

Syntax:

```
putcslice(first,last,slice) -> nil
```

first

First element in the slice of c.

last

Last element plus 1 of the slice in c to be changed.

slice

New numerical values for coefficients in c that should be modified.

2.195 putdouparam()

Sets the value of a double parameter. Syntax:

```
putdouparam(param,parvalue) -> nil
```

param

Which parameter.

parvalue

Parameter value.

2.196 putintparam()

Sets the value of an integer parameter.

Syntax:

```
putintparam(param,parvalue) -> nil
```

param

Which parameter.

parvalue

Parameter value.

2.197 putmaxnumanz()

MOSEK stores only the non-zero elements in A. Therefore, MOSEK cannot predict how much storage is required to store A. Using this function it is possible to specify the number of non-zeros to preallocate for storing A.

If the number of non-zeros in the problem is known, it is a good idea to set `maxnumanz` slightly larger than this number, otherwise a rough estimate can be used. In general, if A is inputted in many small chunks, setting this value may speed up the the data input phase.

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

```
putmaxnumanz(maxnumanz) -> nil
```

maxnumanz

New size of the storage reserved for storing A.

2.198 putmaxnumbarvar()

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

It is not mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

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

```
putmaxnumbarvar(maxnumbarvar) -> nil
```

maxnumbarvar

The maximum number of semidefinite variables.

2.199 putmaxnumcon()

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. Syntax:

```
putmaxnumcon(maxnumcon) -> nil
```

maxnumcon

Number of preallocated constraints in the optimization task.

2.200 putmaxnumcone()

Sets the number of preallocated conic constraints in the optimization task. When this number of conic constraints is reached MOSEK will automatically allocate more space for conic constraints.

It is never mandatory to call this function, since MOSEK will reallocate any internal structures whenever it is required.

Please note that `maxnumcon` must be larger than the current number of constraints in the task. Syntax:

```
putmaxnumcone(maxnumcone) -> nil
```

maxnumcone

Number of preallocated conic constraints in the optimization task.

2.201 putmaxnumqnz()

MOSEK stores only the non-zero elements in Q. Therefore, MOSEK cannot predict how much storage is required to store Q. Using this function it is possible to specify the number non-zeros to preallocate for storing Q (both objective and constraints).

It may be advantageous to reserve more non-zeros for Q than actually needed since it may improve the internal efficiency of MOSEK, however, it is never worthwhile to specify more than the double of the anticipated number of non-zeros in Q.

It is never mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons. Syntax:

```
putmaxnumqnz(maxnumqnz) -> nil
```

maxnumqnz

Number of non-zero elements preallocated in quadratic coefficient matrixes.

2.202 putmaxnumvar()

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

It is never mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

Please note that `maxnumvar` must be larger than the current number of variables in the task. Syntax:

```
putmaxnumvar(maxnumvar) -> nil
```

maxnumvar

Number of preallocated variables in the optimization task.

2.203 putobjname()

Assigns the name given by `objname` to the objective function. Syntax:

```
putobjname(objname) -> nil
```

objname

Name of the objective.

2.204 putobjsense()

Sets the objective sense of the task. Syntax:

```
putobjsense(sense) -> nil
```

sense

The objective sense of the task. The values `objsense.maximize` and `objsense.minimize` means that the the problem is maximized or minimized respectively.

2.205 putparam()

Checks if a `parname` is valid parameter name. If it is, the parameter is assigned the value specified by `parvalue`. Syntax:

```
putparam(parname,parvalue) -> nil
```

parname

Parameter name.

parvalue

Parameter value.

2.206 putqcon()

Replaces all quadratic entries in the constraints. Consider constraints on the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{numvar} \sum_{j=0}^{numvar} q_{ij}^k x_i x_j + \sum_{j=0}^{numvar} a_{kj} x_j \leq u_k^c, \; k = 0, \ldots, m-1.$$

The function assigns values to q such that:

$$q_{\mathtt{qcsubi[t]},\mathtt{qcsubj[t]}}^{\mathtt{qcsubk[t]}} = \mathtt{qcval}[\mathtt{t}], \; t = 0, \ldots, \mathtt{numqcnz} - 1.$$

and

$$q_{\mathtt{qcsubj[t]},\mathtt{qcsubi[t]}}^{\mathtt{qcsubk[t]}} = \mathtt{qcval}[\mathtt{t}], \; t = 0, \ldots, \mathtt{numqcnz} - 1.$$

Values not assigned are set to zero.

Please note that duplicate entries are added together. Syntax:

```
putqcon(qcsubk,qcsubi,qcsubj,qcval) -> nil
```

qcsubk

k subscripts for q_{ij}^k.

qcsubi

i subscripts for q_{ij}^k.

qcsubj

j subscripts for q_{ij}^k.

qcval

Numerical value for q_{ij}^k.

2.207 putqconk()

Replaces all the quadratic entries in one constraint k of the form:

$$l_k^c \leq \frac{1}{2} \sum_{i=1}^{numvar} \sum_{j=1}^{numvar} q_{ij}^k x_i x_j + \sum_{j=1}^{numvar} a_{kj} x_j \leq u_k^c.$$

It is assumed that Q^k is symmetric, i.e. $q_{ij}^k = q_{ji}^k$, and therefore, only the values of q_{ij}^k for which $i \geq j$ should be inputted to MOSEK. To be precise, MOSEK uses the following procedure

$$\begin{array}{ll}
1. & Q^k = 0 \\
2. & \text{for } t = 1 \text{ to } numqonz \\
3. & \quad q^k_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} = q^k_{\texttt{qcsubi[t]},\texttt{qcsubj[t]}} + \texttt{qcval[t]} \\
3. & \quad q^k_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} = q^k_{\texttt{qcsubj[t]},\texttt{qcsubi[t]}} + \texttt{qcval[t]}
\end{array}$$

Please note that:

- For large problems it is essential for the efficiency that the function `Task.putmaxnumqnz` is employed to specify an appropriate `maxnumqnz`.
- Only the lower triangular part should be specified because Q^k is symmetric. Specifying values for q_{ij}^k where $i < j$ will result in an error.
- Only non-zero elements should be specified.
- The order in which the non-zero elements are specified is insignificant.
- Duplicate elements are added together. Hence, it is recommended not to specify the same element multiple times in `qosubi`, `qosubj`, and `qoval`.

Syntax:

```
putqconk(k,qcsubi,qcsubj,qcval) -> nil
```

k

The constraint in which the new Q elements are inserted.

qcsubi

i subscripts for q_{ij}^k.

qcsubj

j subscripts for q_{ij}^k.

qcval

Numerical value for q_{ij}^k.

2.208 putqobj()

Replaces all the quadratic terms in the objective

$$\frac{1}{2}\sum_{i=1}^{numvar}\sum_{j=1}^{numvar} q^o_{ij}x_i x_j + \sum_{j=1}^{numvar} c_j x_j + c^f.$$

It is assumed that Q^o is symmetric, i.e. $q^o_{ij} = q^o_{ji}$, and therefore, only the values of q^o_{ij} for which $i \geq j$ should be specified. To be precise, MOSEK uses the following procedure

1. $Q^o = 0$
2. for $t = 1$ to $numqonz$
3. $\quad q^o_{\texttt{qosubi[t]},\texttt{qosubj[t]}} = q^o_{\texttt{qosubi[t]},\texttt{qosubj[t]}} + \texttt{qoval[t]}$
3. $\quad q^o_{\texttt{qosubj[t]},\texttt{qosubi[t]}} = q^o_{\texttt{qosubj[t]},\texttt{qosubi[t]}} + \texttt{qoval[t]}$

Please note that:

- Only the lower triangular part should be specified because Q^o is symmetric. Specifying values for q^o_{ij} where $i < j$ will result in an error.
- Only non-zero elements should be specified.
- The order in which the non-zero elements are specified is insignificant.
- Duplicate entries are added to together.

Syntax:

```
putqobj(qosubi,qosubj,qoval) -> nil
```

qosubi

i subscript for q^o_{ij}.

qosubj

j subscript for q^o_{ij}.

qoval

Numerical value for q^o_{ij}.

2.209 putqobjij()

Replaces one coefficient in the quadratic term in the objective. The function performs the assignment

$$q^o_{\texttt{ij}} = \texttt{qoij}.$$

Only the elements in the lower triangular part are accepted. Setting q_{ij} with $j > i$ will cause an error.

Please note that replacing all quadratic element, one at a time, is more computationally expensive than replacing all elements at once. Use Task.putqobj instead whenever possible. Syntax:

```
putqobjij(i,j,qoij) -> nil
```

i

Row index for the coefficient to be replaced.

j

Column index for the coefficient to be replaced.

qoij

The new value for q^o_{ij}.

2.210 putskc()

Sets the status keys for the constraints. Syntax:

```
putskc(whichsol,skc) -> nil
```

whichsol

Selects a solution.

skc

Status keys for the constraints.

2.211 putskcslice()

Sets the status keys for the constraints. Syntax:

```
putskcslice(whichsol,first,last,skc) -> nil
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

skc

Status keys for the constraints.

2.212 putskx()

Sets the status keys for the scalar variables. Syntax:

```
putskx(whichsol,skx) -> nil
```

whichsol

Selects a solution.

skx

Status keys for the variables.

2.213 putskxslice()

Sets the status keys for the variables. Syntax:

```
putskxslice(whichsol,first,last,skx) -> nil
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

skx

Status keys for the variables.

2.214 putslc()

Sets the s_l^c vector for a solution. Syntax:

```
putslc(whichsol,slc) -> nil
```

whichsol

Selects a solution.

slc

The s_l^c vector.

2.215 putslcslice()

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

```
putslcslice(whichsol,first,last,slc) -> nil
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

slc

Dual variables corresponding to the lower bounds on the constraints (s_l^c).

2.216 putslx()

Sets the s_l^x vector for a solution. Syntax:

```
putslx(whichsol,slx) -> nil
```

whichsol

Selects a solution.

slx

The s_l^x vector.

2.217 putslxslice()

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

```
putslxslice(whichsol,first,last,slx) -> nil
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

slx

Dual variables corresponding to the lower bounds on the variables (s_l^x).

2.218 putsnx()

Sets the s_n^x vector for a solution. Syntax:

```
putsnx(whichsol,sux) -> nil
```

whichsol

Selects a solution.

sux

The s_n^x vector.

2.219 putsnxslice()

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

```
putsnxslice(whichsol,first,last,snx) -> nil
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

snx

Dual variables corresponding to the conic constraints on the variables (s_n^x).

2.220 putsolution()

Inserts a solution into the task. Syntax:

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

whichsol

Selects a solution.

skc

Status keys for the constraints.

skx

Status keys for the variables.

skn

Status keys for the conic constraints.

xc

Primal constraint solution.

xx

Primal variable solution (x).

y

Vector of dual variables corresponding to the constraints.

slc

Dual variables corresponding to the lower bounds on the constraints (s_l^c).

suc

Dual variables corresponding to the upper bounds on the constraints (s_u^c).

slx

Dual variables corresponding to the lower bounds on the variables (s_l^x).

sux

Dual variables corresponding to the upper bounds on the variables (appears as s_u^x).

snx

Dual variables corresponding to the conic constraints on the variables (s_n^x).

2.221 putsolutioni()

Sets the primal and dual solution information for a single constraint or variable. Syntax:

```
putsolutioni(accmode,i,whichsol,sk,x,sl,su,sn) -> nil
```

accmode

If set to accmode.con the solution information for a constraint is modified. Otherwise for a variable.

i

Index of the constraint or variable.

whichsol

Selects a solution.

sk

Status key of the constraint or variable.

x

Solution value of the primal constraint or variable.

sl

Solution value of the dual variable associated with the lower bound.

su

Solution value of the dual variable associated with the upper bound.

sn

Solution value of the dual variable associated with the cone constraint.

2.222 putsolutionyi()

Inputs the dual variable of a solution. Syntax:

```
putsolutionyi(i,whichsol,y) -> nil
```

i

Index of the dual variable.

whichsol

Selects a solution.

y

Solution value of the dual variable.

2.223 putstrparam()

Sets the value of a string parameter. Syntax:

```
putstrparam(param,parvalue) -> nil
```

param

Which parameter.

parvalue

Parameter value.

2.224 putsuc()

Sets the s_u^c vector for a solution. Syntax:

```
putsuc(whichsol,suc) -> nil
```

whichsol

Selects a solution.

suc

The s_u^c vector.

2.225 putsucslice()

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

```
putsucslice(whichsol,first,last,suc) -> nil
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

suc

Dual variables corresponding to the upper bounds on the constraints (s_u^c).

2.226 putsux()

Sets the s_u^x vector for a solution. Syntax:

```
putsux(whichsol,sux) -> nil
```

whichsol

Selects a solution.

sux

The s_u^x vector.

2.227 putsuxslice()

Sets a slice of the s_u^x vector for a solution. Syntax:

```
putsuxslice(whichsol,first,last,sux) -> nil
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

sux

Dual variables corresponding to the upper bounds on the variables (appears as s_u^x).

2.228 puttaskname()

Assigns the name `taskname` to the task. Syntax:

```
puttaskname(taskname) -> nil
```

taskname

Name assigned to the task.

2.229 putvarbound()

Changes the bounds for one variable.

If the a bound value specified is numerically larger than `dparam.data_tol_bound_inf` it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than `dparam.data_tol_bound_wrn`, a warning will be displayed, but the bound is inputted as specified. Syntax:

```
putvarbound(j,bk,bl,bu) -> nil
```

j

Index of the variable.

bk

New bound key.

bl

New lower bound.

bu

New upper bound.

2.230 putvarboundlist()

Changes the bounds for one or more variables. If multiple bound changes are specified for a variable, then only the last change takes effect. Syntax:

```
putvarboundlist(sub,bkx,blx,bux) -> nil
```

sub

List of variable indexes.

bkx

New bound keys.

blx

New lower bound values.

bux

New upper bound values.

2.231 putvarboundslice()

Changes the bounds for a slice of the variables. Syntax:

```
putvarboundslice(first,last,bk,bl,bu) -> nil
```

first

Index of the first variable in the slice.

last

Index of the last variable in the slice plus 1.

bk

New bound keys.

bl

New lower bounds.

bu

New upper bounds.

2.232 putvarbranchorder()

The purpose of the function is to assign a branching priority and direction. The higher priority that is assigned to an integer variable the earlier the mixed integer optimizer will branch on the variable. The branching direction controls if the optimizer branches up or down on the variable. Syntax:

```
putvarbranchorder(j,priority,direction) -> nil
```

j

Index of the variable.

priority

The branching priority that should be assigned to variable j.

direction

Specifies the preferred branching direction for variable j.

2.233 putvarname()

Puts the name of a variable. Syntax:

```
putvarname(j,name) -> nil
```

j

Index of the variable.

name

The variable name.

2.234 putvartype()

Sets the variable type of one variable. Syntax:

```
putvartype(j,vartype) -> nil
```

j

Index of the variable.

vartype

The new variable type.

2.235 putvartypelist()

Sets the variable type for one or more variables, i.e. variable number `subj`$[k]$ is assigned the variable type `vartype`$[k]$.

If the same index is specified multiple times in `subj` only the last entry takes effect. Syntax:

```
putvartypelist(subj,vartype) -> nil
```

subj

A list of variable indexes for which the variable type should be changed.

vartype

A list of variable types that should be assigned to the variables specified by `subj`. See section `variabletype` for the possible values of `vartype`.

2.236 putxc()

Sets the x^c vector for a solution. Syntax:

```
putxc(whichsol) -> xc
```

whichsol

Selects a solution.

xc

The x^c vector.

2.237 putxcslice()

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

```
putxcslice(whichsol,first,last,xc) -> nil
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

xc

Primal constraint solution.

2.238 putxx()

Sets the x^x vector for a solution. Syntax:

```
putxx(whichsol,xx) -> nil
```

whichsol

Selects a solution.

xx

The x^x vector.

2.239 putxxslice()

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

```
putxxslice(whichsol,first,last,xx) -> nil
```

whichsol

Selects a solution.

first

First index in the sequence.

last

Last index plus 1 in the sequence.

xx

Primal variable solution (x).

2.240 puty()

Sets the y vector for a solution. Syntax:

```
puty(whichsol,y) -> nil
```

whichsol

Selects a solution.

y

The y vector.

2.241 putyslice()

Sets a slice of the y vector for a solution. Syntax:

```
putyslice(whichsol,first,last,y) -> nil
```

whichsol
: Selects a solution.

first
: First index in the sequence.

last
: Last index plus 1 in the sequence.

y
: Vector of dual variables corresponding to the constraints.

2.242 readbranchpriorities()

Reads branching priority data from a file. Syntax:

```
readbranchpriorities(filename) -> nil
```

filename
: Data is read from the file `filename`.

2.243 readdata()

Reads an optimization problem and associated data from a file. Syntax:

```
readdata(filename) -> nil
```

filename
: Data is read from the file `filename`.

2.244 readdataformat()

Reads an optimization problem and associated data from a file. Syntax:

```
readdataformat(filename,format,compress) -> nil
```

filename
: Data is read from the file `filename`.

format

File data format.

compress

File compression type.

2.245 readsolution()

Reads a solution file and inserts the solution into the solution `whichsol`. Syntax:

```
readsolution(whichsol,filename) -> nil
```

whichsol

Selects a solution.

filename

A valid file name.

2.246 readsummary()

Prints a short summary of last file that was read. Syntax:

```
readsummary(whichstream) -> nil
```

whichstream

Index of the stream.

2.247 readtask()

Load task data from a file, replacing any data that already is in the task object. All problem data are restored, but if the file contains solutions, the solution status after loading a file is still unknown, even if it was optimal or otherwise well-defined when the file was dumped.

See section 4.4 for a description of the Task format. Syntax:

```
readtask(filename) -> nil
```

filename

Input file name.

2.248 removebarvars()

The function removes a subset of the symmetric matrix from the optimization task. This implies that the existing symmetric matrix are renumbered, for instance if constraint 5 is removed then constraint 6 becomes constraint 5 and so forth. Syntax:

```
removebarvars(subset) -> nil
```

subset

Indexes of symmetric matrix which should be removed.

2.249 removecones()

Removes a number conic constraint from the problem. In general, it is much more efficient to remove a cone with a high index than a low index. Syntax:

```
removecones(subset) -> nil
```

subset

Indexes of cones which should be removed.

2.250 removecons()

The function removes a subset of the constraints from the optimization task. This implies that the existing constraints are renumbered, for instance if constraint 5 is removed then constraint 6 becomes constraint 5 and so forth. Syntax:

```
removecons(subset) -> nil
```

subset

Indexes of constraints which should be removed.

2.251 removevars()

The function removes a subset of the variables from the optimization task. This implies that the existing variables are renumbered, for instance if constraint 5 is removed then constraint 6 becomes constraint 5 and so forth. Syntax:

```
removevars(subset) -> nil
```

subset

Indexes of variables which should be removed.

2.252 resizetask()

Sets the amount of preallocated space assigned for each type of data in an optimization task.

It is never mandatory to call this function, since its only function is to give a hint of the amount of data to preallocate for efficiency reasons.

Please note that the procedure is **destructive** in the sense that all existing data stored in the task is destroyed. Syntax:

```
resizetask(maxnumcon,maxnumvar,maxnumcone,maxnumanz,maxnumqnz) -> nil
```

maxnumcon

New maximum number of constraints.

maxnumvar

New maximum number of variables.

maxnumcone

New maximum number of cones.

maxnumanz

New maximum number of non-zeros in A.

maxnumqnz

New maximum number of non-zeros in all Q matrixes.

2.253 sensitivityreport()

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. Syntax:

```
sensitivityreport(whichstream) -> nil
```

whichstream

Index of the stream.

2.254 sktostr()

Obtains an explanatory string corresponding to a status key. Syntax:

```
sktostr(sk) -> str
```

sk

A valid status key.

str

String corresponding to the status key `sk`.

2.255 solstatostr()

Obtains an explanatory string corresponding to a solution status. Syntax:

```
solstatostr(solsta) -> str
```

solsta

Solution status.

str

String corresponding to the solution status `solsta`.

2.256 solutiondef()

Checks whether a solution is defined. Syntax:

```
solutiondef(whichsol) -> isdef
```

whichsol

Selects a solution.

isdef

Is non-zero if the requested solution is defined.

2.257 solutionsummary()

Prints a short summary of the current solutions. Syntax:

```
solutionsummary(whichstream) -> nil
```

whichstream

Index of the stream.

2.258 updatesolutioninfo()

Update the information items related to the solution. Syntax:

```
updatesolutioninfo(whichsol) -> nil
```

whichsol

Selects a solution.

2.259 writebranchpriorities()

Writes branching priority data to a file. Syntax:

```
writebranchpriorities(filename) -> nil
```

filename

Data is written to the file `filename`.

2.260 writedata()

Writes problem data associated with the optimization task to a file in one of four formats:

LP:

A text based row oriented format. File extension `.lp`. See Appendix 4.2.

MPS:

A text based column oriented format. File extension `.mps`. See Appendix 4.1.

OPF:

A text based row oriented format. File extension `.opf`. Supports more problem types than MPS and LP. See Appendix 4.3.

TASK:

A MOSEK specific binary format for fast reading and writing. File extension `.task`.

By default the data file format is determined by the file name extension. This behaviour can be overridden by setting the `iparam.write_data_format` parameter.

MOSEK is able to read and write files in a compressed format (gzip). To write in the compressed format append the extension ".gz". E.g to write a gzip compressed MPS file use the extension `mps.gz`.

Please note that MPS, LP and OPF files require all variables to have unique names. If a task contains no names, it is possible to write the file with automaticly generated anonymous names by setting the `iparam.write_generic_names` parameter to `onoffkey.on`.

Please note that if a general nonlinear function appears in the problem then such function *cannot* be written to file and MOSEK will issue a warning. Syntax:

```
writedata(filename) -> nil
```

filename

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`.

2.261 writeparamfile()

Writes all the parameters to a parameter file. Syntax:

```
writeparamfile(filename) -> nil
```

filename

The name of parameter file.

2.262 writesolution()

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

```
writesolution(whichsol,filename) -> nil
```

whichsol

Selects a solution.

filename

A valid file name.

2.263 writetask()

Write a binary dump of the task data. This format saves all problem data, but not callback-funktions and general non-linear terms.

See section 4.4 for a description of the Task format. Syntax:

```
writetask(filename) -> nil
```

filename

Output file name.

Chapter 3

Symbolic constants and parameters

3.1 accmode

accmode.con ("MSK_ACC_CON")

Access data by rows (constraint oriented)

accmode.var ("MSK_ACC_VAR")

Access data by columns (variable oriented)

3.2 basindtype

basindtype.always ("MSK_BI_ALWAYS")

Basis identification is always performed even if the interior-point optimizer terminates abnormally.

basindtype.if_feasible ("MSK_BI_IF_FEASIBLE")

Basis identification is not performed if the interior-point optimizer terminates with a problem status saying that the problem is primal or dual infeasible.

basindtype.never ("MSK_BI_NEVER")

Never do basis identification.

basindtype.no_error ("MSK_BI_NO_ERROR")

Basis identification is performed if the interior-point optimizer terminates without an error.

basindtype.reservered ("MSK_BI_RESERVERED")

Not currently in use.

3.3 boundkey

boundkey.fr ("MSK_BK_FR")

The constraint or variable is free.

boundkey.fx ("MSK_BK_FX")

The constraint or variable is fixed.

boundkey.lo ("MSK_BK_LO")

The constraint or variable has a finite lower bound and an infinite upper bound.

boundkey.ra ("MSK_BK_RA")

The constraint or variable is ranged.

boundkey.up ("MSK_BK_UP")

The constraint or variable has an infinite lower bound and an finite upper bound.

3.4 branchdir

branchdir.down ("MSK_BRANCH_DIR_DOWN")

The mixed-integer optimizer always chooses the down branch first.

branchdir.free ("MSK_BRANCH_DIR_FREE")

The mixed-integer optimizer decides which branch to choose.

branchdir.up ("MSK_BRANCH_DIR_UP")

The mixed-integer optimizer always chooses the up branch first.

3.5 callbackcode

callbackcode.begin_bi ("MSK_CALLBACK_BEGIN_BI")

The basis identification procedure has been started.

callbackcode.begin_concurrent ("MSK_CALLBACK_BEGIN_CONCURRENT")

Concurrent optimizer is started.

callbackcode.begin_conic ("MSK_CALLBACK_BEGIN_CONIC")

The call-back function is called when the conic optimizer is started.

callbackcode.begin_dual_bi ("MSK_CALLBACK_BEGIN_DUAL_BI")

The call-back function is called from within the basis identification procedure when the dual phase is started.

callbackcode.begin_dual_sensitivity ("MSK_CALLBACK_BEGIN_DUAL_SENSITIVITY")

Dual sensitivity analysis is started.

callbackcode.begin_dual_setup_bi ("MSK_CALLBACK_BEGIN_DUAL_SETUP_BI")

The call-back function is called when the dual BI phase is started.

callbackcode.begin_dual_simplex ("MSK_CALLBACK_BEGIN_DUAL_SIMPLEX")

The call-back function is called when the dual simplex optimizer started.

callbackcode.begin_dual_simplex_bi ("MSK_CALLBACK_BEGIN_DUAL_SIMPLEX_BI")

The call-back function is called from within the basis identification procedure when the dual simplex clean-up phase is started.

callbackcode.begin_full_convexity_check ("MSK_CALLBACK_BEGIN_FULL_CONVEXITY_CHECK")

Begin full convexity check.

callbackcode.begin_infeas_ana ("MSK_CALLBACK_BEGIN_INFEAS_ANA")

The call-back function is called when the infeasibility analyzer is started.

callbackcode.begin_intpnt ("MSK_CALLBACK_BEGIN_INTPNT")

The call-back function is called when the interior-point optimizer is started.

callbackcode.begin_license_wait ("MSK_CALLBACK_BEGIN_LICENSE_WAIT")

Begin waiting for license.

callbackcode.begin_mio ("MSK_CALLBACK_BEGIN_MIO")

The call-back function is called when the mixed-integer optimizer is started.

callbackcode.begin_network_dual_simplex ("MSK_CALLBACK_BEGIN_NETWORK_DUAL_SIMPLEX")

The call-back function is called when the dual network simplex optimizer is started.

callbackcode.begin_network_primal_simplex ("MSK_CALLBACK_BEGIN_NETWORK_PRIMAL_SIMPLEX")

The call-back function is called when the primal network simplex optimizer is started.

callbackcode.begin_network_simplex ("MSK_CALLBACK_BEGIN_NETWORK_SIMPLEX")

The call-back function is called when the simplex network optimizer is started.

callbackcode.begin_nonconvex ("MSK_CALLBACK_BEGIN_NONCONVEX")

The call-back function is called when the nonconvex optimizer is started.

callbackcode.begin_optimizer ("MSK_CALLBACK_BEGIN_OPTIMIZER")

The call-back function is called when the optimizer is started.

callbackcode.begin_presolve ("MSK_CALLBACK_BEGIN_PRESOLVE")

The call-back function is called when the presolve is started.

callbackcode.begin_primal_bi ("MSK_CALLBACK_BEGIN_PRIMAL_BI")

The call-back function is called from within the basis identification procedure when the primal phase is started.

callbackcode.begin_primal_dual_simplex ("MSK_CALLBACK_BEGIN_PRIMAL_DUAL_SIMPLEX")

The call-back function is called when the primal-dual simplex optimizer is started.

callbackcode.begin_primal_dual_simplex_bi ("MSK_CALLBACK_BEGIN_PRIMAL_DUAL_SIMPLEX_BI")

The call-back function is called from within the basis identification procedure when the primal-dual simplex clean-up phase is started.

callbackcode.begin_primal_repair ("MSK_CALLBACK_BEGIN_PRIMAL_REPAIR")

Begin primal feasibility repair.

callbackcode.begin_primal_sensitivity ("MSK_CALLBACK_BEGIN_PRIMAL_SENSITIVITY")

Primal sensitivity analysis is started.

callbackcode.begin_primal_setup_bi ("MSK_CALLBACK_BEGIN_PRIMAL_SETUP_BI")

The call-back function is called when the primal BI setup is started.

callbackcode.begin_primal_simplex ("MSK_CALLBACK_BEGIN_PRIMAL_SIMPLEX")

The call-back function is called when the primal simplex optimizer is started.

callbackcode.begin_primal_simplex_bi ("MSK_CALLBACK_BEGIN_PRIMAL_SIMPLEX_BI")

The call-back function is called from within the basis identification procedure when the primal simplex clean-up phase is started.

callbackcode.begin_qcqo_reformulate ("MSK_CALLBACK_BEGIN_QCQO_REFORMULATE")

Begin QCQO reformulation.

callbackcode.begin_read ("MSK_CALLBACK_BEGIN_READ")

MOSEK has started reading a problem file.

callbackcode.begin_simplex ("MSK_CALLBACK_BEGIN_SIMPLEX")

The call-back function is called when the simplex optimizer is started.

callbackcode.begin_simplex_bi ("MSK_CALLBACK_BEGIN_SIMPLEX_BI")

The call-back function is called from within the basis identification procedure when the simplex clean-up phase is started.

callbackcode.begin_simplex_network_detect ("MSK_CALLBACK_BEGIN_SIMPLEX_NETWORK_DETECT")

The call-back function is called when the network detection procedure is started.

callbackcode.begin_write ("MSK_CALLBACK_BEGIN_WRITE")

MOSEK has started writing a problem file.

callbackcode.conic ("`MSK_CALLBACK_CONIC`")

The call-back function is called from within the conic optimizer after the information database has been updated.

callbackcode.dual_simplex ("`MSK_CALLBACK_DUAL_SIMPLEX`")

The call-back function is called from within the dual simplex optimizer.

callbackcode.end_bi ("`MSK_CALLBACK_END_BI`")

The call-back function is called when the basis identification procedure is terminated.

callbackcode.end_concurrent ("`MSK_CALLBACK_END_CONCURRENT`")

Concurrent optimizer is terminated.

callbackcode.end_conic ("`MSK_CALLBACK_END_CONIC`")

The call-back function is called when the conic optimizer is terminated.

callbackcode.end_dual_bi ("`MSK_CALLBACK_END_DUAL_BI`")

The call-back function is called from within the basis identification procedure when the dual phase is terminated.

callbackcode.end_dual_sensitivity ("`MSK_CALLBACK_END_DUAL_SENSITIVITY`")

Dual sensitivity analysis is terminated.

callbackcode.end_dual_setup_bi ("`MSK_CALLBACK_END_DUAL_SETUP_BI`")

The call-back function is called when the dual BI phase is terminated.

callbackcode.end_dual_simplex ("`MSK_CALLBACK_END_DUAL_SIMPLEX`")

The call-back function is called when the dual simplex optimizer is terminated.

callbackcode.end_dual_simplex_bi ("`MSK_CALLBACK_END_DUAL_SIMPLEX_BI`")

The call-back function is called from within the basis identification procedure when the dual clean-up phase is terminated.

callbackcode.end_full_convexity_check ("`MSK_CALLBACK_END_FULL_CONVEXITY_CHECK`")

End full convexity check.

callbackcode.end_infeas_ana ("`MSK_CALLBACK_END_INFEAS_ANA`")

The call-back function is called when the infeasibility analyzer is terminated.

callbackcode.end_intpnt ("`MSK_CALLBACK_END_INTPNT`")

The call-back function is called when the interior-point optimizer is terminated.

callbackcode.end_license_wait ("`MSK_CALLBACK_END_LICENSE_WAIT`")

End waiting for license.

callbackcode.end_mio ("`MSK_CALLBACK_END_MIO`")

The call-back function is called when the mixed-integer optimizer is terminated.

callbackcode.end_network_dual_simplex ("MSK_CALLBACK_END_NETWORK_DUAL_SIMPLEX")

The call-back function is called when the dual network simplex optimizer is terminated.

callbackcode.end_network_primal_simplex ("MSK_CALLBACK_END_NETWORK_PRIMAL_SIMPLEX")

The call-back function is called when the primal network simplex optimizer is terminated.

callbackcode.end_network_simplex ("MSK_CALLBACK_END_NETWORK_SIMPLEX")

The call-back function is called when the simplex network optimizer is terminated.

callbackcode.end_nonconvex ("MSK_CALLBACK_END_NONCONVEX")

The call-back function is called when the nonconvex optimizer is terminated.

callbackcode.end_optimizer ("MSK_CALLBACK_END_OPTIMIZER")

The call-back function is called when the optimizer is terminated.

callbackcode.end_presolve ("MSK_CALLBACK_END_PRESOLVE")

The call-back function is called when the presolve is completed.

callbackcode.end_primal_bi ("MSK_CALLBACK_END_PRIMAL_BI")

The call-back function is called from within the basis identification procedure when the primal phase is terminated.

callbackcode.end_primal_dual_simplex ("MSK_CALLBACK_END_PRIMAL_DUAL_SIMPLEX")

The call-back function is called when the primal-dual simplex optimizer is terminated.

callbackcode.end_primal_dual_simplex_bi ("MSK_CALLBACK_END_PRIMAL_DUAL_SIMPLEX_BI")

The call-back function is called from within the basis identification procedure when the primal-dual clean-up phase is terminated.

callbackcode.end_primal_repair ("MSK_CALLBACK_END_PRIMAL_REPAIR")

End primal feasibility repair.

callbackcode.end_primal_sensitivity ("MSK_CALLBACK_END_PRIMAL_SENSITIVITY")

Primal sensitivity analysis is terminated.

callbackcode.end_primal_setup_bi ("MSK_CALLBACK_END_PRIMAL_SETUP_BI")

The call-back function is called when the primal BI setup is terminated.

callbackcode.end_primal_simplex ("MSK_CALLBACK_END_PRIMAL_SIMPLEX")

The call-back function is called when the primal simplex optimizer is terminated.

callbackcode.end_primal_simplex_bi ("MSK_CALLBACK_END_PRIMAL_SIMPLEX_BI")

The call-back function is called from within the basis identification procedure when the primal clean-up phase is terminated.

callbackcode.end_qcqo_reformulate ("MSK_CALLBACK_END_QCQO_REFORMULATE")

End QCQO reformulation.

callbackcode.end_read ("MSK_CALLBACK_END_READ")

MOSEK has finished reading a problem file.

callbackcode.end_simplex ("MSK_CALLBACK_END_SIMPLEX")

The call-back function is called when the simplex optimizer is terminated.

callbackcode.end_simplex_bi ("MSK_CALLBACK_END_SIMPLEX_BI")

The call-back function is called from within the basis identification procedure when the simplex clean-up phase is terminated.

callbackcode.end_simplex_network_detect ("MSK_CALLBACK_END_SIMPLEX_NETWORK_DETECT")

The call-back function is called when the network detection procedure is terminated.

callbackcode.end_write ("MSK_CALLBACK_END_WRITE")

MOSEK has finished writing a problem file.

callbackcode.im_bi ("MSK_CALLBACK_IM_BI")

The call-back function is called from within the basis identification procedure at an intermediate point.

callbackcode.im_conic ("MSK_CALLBACK_IM_CONIC")

The call-back function is called at an intermediate stage within the conic optimizer where the information database has not been updated.

callbackcode.im_dual_bi ("MSK_CALLBACK_IM_DUAL_BI")

The call-back function is called from within the basis identification procedure at an intermediate point in the dual phase.

callbackcode.im_dual_sensivity ("MSK_CALLBACK_IM_DUAL_SENSIVITY")

The call-back function is called at an intermediate stage of the dual sensitivity analysis.

callbackcode.im_dual_simplex ("MSK_CALLBACK_IM_DUAL_SIMPLEX")

The call-back function is called at an intermediate point in the dual simplex optimizer.

callbackcode.im_full_convexity_check ("MSK_CALLBACK_IM_FULL_CONVEXITY_CHECK")

The call-back function is called at an intermediate stage of the full convexity check.

callbackcode.im_intpnt ("MSK_CALLBACK_IM_INTPNT")

The call-back function is called at an intermediate stage within the interior-point optimizer where the information database has not been updated.

callbackcode.im_license_wait ("MSK_CALLBACK_IM_LICENSE_WAIT")

MOSEK is waiting for a license.

callbackcode.im_lu ("MSK_CALLBACK_IM_LU")

The call-back function is called from within the LU factorization procedure at an intermediate point.

callbackcode.im_mio ("MSK_CALLBACK_IM_MIO")

The call-back function is called at an intermediate point in the mixed-integer optimizer.

callbackcode.im_mio_dual_simplex ("MSK_CALLBACK_IM_MIO_DUAL_SIMPLEX")

The call-back function is called at an intermediate point in the mixed-integer optimizer while running the dual simplex optimizer.

callbackcode.im_mio_intpnt ("MSK_CALLBACK_IM_MIO_INTPNT")

The call-back function is called at an intermediate point in the mixed-integer optimizer while running the interior-point optimizer.

callbackcode.im_mio_presolve ("MSK_CALLBACK_IM_MIO_PRESOLVE")

The call-back function is called at an intermediate point in the mixed-integer optimizer while running the presolve.

callbackcode.im_mio_primal_simplex ("MSK_CALLBACK_IM_MIO_PRIMAL_SIMPLEX")

The call-back function is called at an intermediate point in the mixed-integer optimizer while running the primal simplex optimizer.

callbackcode.im_network_dual_simplex ("MSK_CALLBACK_IM_NETWORK_DUAL_SIMPLEX")

The call-back function is called at an intermediate point in the dual network simplex optimizer.

callbackcode.im_network_primal_simplex ("MSK_CALLBACK_IM_NETWORK_PRIMAL_SIMPLEX")

The call-back function is called at an intermediate point in the primal network simplex optimizer.

callbackcode.im_nonconvex ("MSK_CALLBACK_IM_NONCONVEX")

The call-back function is called at an intermediate stage within the nonconvex optimizer where the information database has not been updated.

callbackcode.im_order ("MSK_CALLBACK_IM_ORDER")

The call-back function is called from within the matrix ordering procedure at an intermediate point.

callbackcode.im_presolve ("MSK_CALLBACK_IM_PRESOLVE")

The call-back function is called from within the presolve procedure at an intermediate stage.

callbackcode.im_primal_bi ("MSK_CALLBACK_IM_PRIMAL_BI")

The call-back function is called from within the basis identification procedure at an intermediate point in the primal phase.

callbackcode.im_primal_dual_simplex ("MSK_CALLBACK_IM_PRIMAL_DUAL_SIMPLEX")

The call-back function is called at an intermediate point in the primal-dual simplex optimizer.

callbackcode.im_primal_sensivity ("MSK_CALLBACK_IM_PRIMAL_SENSIVITY")

The call-back function is called at an intermediate stage of the primal sensitivity analysis.

callbackcode.im_primal_simplex ("`MSK_CALLBACK_IM_PRIMAL_SIMPLEX`")

The call-back function is called at an intermediate point in the primal simplex optimizer.

callbackcode.im_qo_reformulate ("`MSK_CALLBACK_IM_QO_REFORMULATE`")

The call-back function is called at an intermediate stage of the conic quadratic reformulation.

callbackcode.im_read ("`MSK_CALLBACK_IM_READ`")

Intermediate stage in reading.

callbackcode.im_simplex ("`MSK_CALLBACK_IM_SIMPLEX`")

The call-back function is called from within the simplex optimizer at an intermediate point.

callbackcode.im_simplex_bi ("`MSK_CALLBACK_IM_SIMPLEX_BI`")

The call-back function is called from within the basis identification procedure at an intermediate point in the simplex clean-up phase. The frequency of the call-backs is controlled by the `iparam.log_sim_freq` parameter.

callbackcode.intpnt ("`MSK_CALLBACK_INTPNT`")

The call-back function is called from within the interior-point optimizer after the information database has been updated.

callbackcode.new_int_mio ("`MSK_CALLBACK_NEW_INT_MIO`")

The call-back function is called after a new integer solution has been located by the mixed-integer optimizer.

callbackcode.noncovex ("`MSK_CALLBACK_NONCOVEX`")

The call-back function is called from within the nonconvex optimizer after the information database has been updated.

callbackcode.primal_simplex ("`MSK_CALLBACK_PRIMAL_SIMPLEX`")

The call-back function is called from within the primal simplex optimizer.

callbackcode.read_opf ("`MSK_CALLBACK_READ_OPF`")

The call-back function is called from the OPF reader.

callbackcode.read_opf_section ("`MSK_CALLBACK_READ_OPF_SECTION`")

A chunk of Q non-zeos has been read from a problem file.

callbackcode.update_dual_bi ("`MSK_CALLBACK_UPDATE_DUAL_BI`")

The call-back function is called from within the basis identification procedure at an intermediate point in the dual phase.

callbackcode.update_dual_simplex ("`MSK_CALLBACK_UPDATE_DUAL_SIMPLEX`")

The call-back function is called in the dual simplex optimizer.

callbackcode.update_dual_simplex_bi ("`MSK_CALLBACK_UPDATE_DUAL_SIMPLEX_BI`")

The call-back function is called from within the basis identification procedure at an intermediate point in the dual simplex clean-up phase. The frequency of the call-backs is controlled by the `iparam.log_sim_freq` parameter.

callbackcode.update_network_dual_simplex ("`MSK_CALLBACK_UPDATE_NETWORK_DUAL_SIMPLEX`")

The call-back function is called in the dual network simplex optimizer.

callbackcode.update_network_primal_simplex ("`MSK_CALLBACK_UPDATE_NETWORK_PRIMAL_SIMPLEX`")

The call-back function is called in the primal network simplex optimizer.

callbackcode.update_nonconvex ("`MSK_CALLBACK_UPDATE_NONCONVEX`")

The call-back function is called at an intermediate stage within the nonconvex optimizer where the information database has been updated.

callbackcode.update_presolve ("`MSK_CALLBACK_UPDATE_PRESOLVE`")

The call-back function is called from within the presolve procedure.

callbackcode.update_primal_bi ("`MSK_CALLBACK_UPDATE_PRIMAL_BI`")

The call-back function is called from within the basis identification procedure at an intermediate point in the primal phase.

callbackcode.update_primal_dual_simplex ("`MSK_CALLBACK_UPDATE_PRIMAL_DUAL_SIMPLEX`")

The call-back function is called in the primal-dual simplex optimizer.

callbackcode.update_primal_dual_simplex_bi ("`MSK_CALLBACK_UPDATE_PRIMAL_DUAL_SIMPLEX_BI`")

The call-back function is called from within the basis identification procedure at an intermediate point in the primal-dual simplex clean-up phase. The frequency of the call-backs is controlled by the `iparam.log_sim_freq` parameter.

callbackcode.update_primal_simplex ("`MSK_CALLBACK_UPDATE_PRIMAL_SIMPLEX`")

The call-back function is called in the primal simplex optimizer.

callbackcode.update_primal_simplex_bi ("`MSK_CALLBACK_UPDATE_PRIMAL_SIMPLEX_BI`")

The call-back function is called from within the basis identification procedure at an intermediate point in the primal simplex clean-up phase. The frequency of the call-backs is controlled by the `iparam.log_sim_freq` parameter.

callbackcode.write_opf ("`MSK_CALLBACK_WRITE_OPF`")

The call-back function is called from the OPF writer.

3.6 checkconvexitytype

checkconvexitytype.full ("`MSK_CHECK_CONVEXITY_FULL`")

Perform a full convexity check.

checkconvexitytype.none ("`MSK_CHECK_CONVEXITY_NONE`")

No convexity check.

checkconvexitytype.simple ("`MSK_CHECK_CONVEXITY_SIMPLE`")

Perform simple and fast convexity check.

3.7 compresstype

compresstype.free ("`MSK_COMPRESS_FREE`")

The type of compression used is chosen automatically.

compresstype.gzip ("`MSK_COMPRESS_GZIP`")

The type of compression used is gzip compatible.

compresstype.none ("`MSK_COMPRESS_NONE`")

No compression is used.

3.8 conetype

conetype.quad ("`MSK_CT_QUAD`")

The cone is a quadratic cone.

conetype.rquad ("`MSK_CT_RQUAD`")

The cone is a rotated quadratic cone.

3.9 dataformat

dataformat.cb ("`MSK_DATA_FORMAT_CB`")

Conic benchmark format.

dataformat.extension ("`MSK_DATA_FORMAT_EXTENSION`")

The file extension is used to determine the data file format.

dataformat.free_mps ("`MSK_DATA_FORMAT_FREE_MPS`")

The data data a free MPS formatted file.

dataformat.lp ("MSK_DATA_FORMAT_LP")

The data file is LP formatted.

dataformat.mps ("MSK_DATA_FORMAT_MPS")

The data file is MPS formatted.

dataformat.op ("MSK_DATA_FORMAT_OP")

The data file is an optimization problem formatted file.

dataformat.task ("MSK_DATA_FORMAT_TASK")

Generic task dump file.

dataformat.xml ("MSK_DATA_FORMAT_XML")

The data file is an XML formatted file.

3.10 dinfitem

dinfitem.bi_clean_dual_time ("MSK_DINF_BI_CLEAN_DUAL_TIME")

Time spent within the dual clean-up optimizer of the basis identification procedure since its invocation.

dinfitem.bi_clean_primal_dual_time ("MSK_DINF_BI_CLEAN_PRIMAL_DUAL_TIME")

Time spent within the primal-dual clean-up optimizer of the basis identification procedure since its invocation.

dinfitem.bi_clean_primal_time ("MSK_DINF_BI_CLEAN_PRIMAL_TIME")

Time spent within the primal clean-up optimizer of the basis identification procedure since its invocation.

dinfitem.bi_clean_time ("MSK_DINF_BI_CLEAN_TIME")

Time spent within the clean-up phase of the basis identification procedure since its invocation.

dinfitem.bi_dual_time ("MSK_DINF_BI_DUAL_TIME")

Time spent within the dual phase basis identification procedure since its invocation.

dinfitem.bi_primal_time ("MSK_DINF_BI_PRIMAL_TIME")

Time spent within the primal phase of the basis identification procedure since its invocation.

dinfitem.bi_time ("MSK_DINF_BI_TIME")

Time spent within the basis identification procedure since its invocation.

dinfitem.concurrent_time ("MSK_DINF_CONCURRENT_TIME")

Time spent within the concurrent optimizer since its invocation.

dinfitem.intpnt_dual_feas ("MSK_DINF_INTPNT_DUAL_FEAS")

Dual feasibility measure reported by the interior-point optimizer. (For the interior-point optimizer this measure does not directly related to the original problem because a homogeneous model is employed.)

dinfitem.intpnt_dual_obj ("MSK_DINF_INTPNT_DUAL_OBJ")

Dual objective value reported by the interior-point optimizer.

dinfitem.intpnt_factor_num_flops ("MSK_DINF_INTPNT_FACTOR_NUM_FLOPS")

An estimate of the number of flops used in the factorization.

dinfitem.intpnt_opt_status ("MSK_DINF_INTPNT_OPT_STATUS")

This measure should converge to +1 if the problem has a primal-dual optimal solution, and converge to -1 if problem is (strictly) primal or dual infeasible. Furthermore, if the measure converges to 0 the problem is usually ill-posed.

dinfitem.intpnt_order_time ("MSK_DINF_INTPNT_ORDER_TIME")

Order time (in seconds).

dinfitem.intpnt_primal_feas ("MSK_DINF_INTPNT_PRIMAL_FEAS")

Primal feasibility measure reported by the interior-point optimizers. (For the interior-point optimizer this measure does not directly related to the original problem because a homogeneous model is employed).

dinfitem.intpnt_primal_obj ("MSK_DINF_INTPNT_PRIMAL_OBJ")

Primal objective value reported by the interior-point optimizer.

dinfitem.intpnt_time ("MSK_DINF_INTPNT_TIME")

Time spent within the interior-point optimizer since its invocation.

dinfitem.mio_cg_seperation_time ("MSK_DINF_MIO_CG_SEPERATION_TIME")

Seperation time for CG cuts.

dinfitem.mio_cmir_seperation_time ("MSK_DINF_MIO_CMIR_SEPERATION_TIME")

Seperation time for CMIR cuts.

dinfitem.mio_construct_solution_obj ("MSK_DINF_MIO_CONSTRUCT_SOLUTION_OBJ")

If MOSEK has successfully constructed an integer feasible solution, then this item contains the optimal objective value corresponding to the feasible solution.

dinfitem.mio_dual_bound_after_presolve ("MSK_DINF_MIO_DUAL_BOUND_AFTER_PRESOLVE")

Value of the dual bound after presolve but before cut generation.

dinfitem.mio_heuristic_time ("MSK_DINF_MIO_HEURISTIC_TIME")

Time spent in the optimizer while solving the relaxtions.

dinfitem.mio_obj_abs_gap ("MSK_DINF_MIO_OBJ_ABS_GAP")

Given the mixed-integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the absolute gap defined by

$$|(\text{objective value of feasible solution}) - (\text{objective bound})|.$$

Otherwise it has the value -1.0.

dinfitem.mio_obj_bound ("MSK_DINF_MIO_OBJ_BOUND")

The best known bound on the objective function. This value is undefined until at least one relaxation has been solved: To see if this is the case check that `iinfitem.mio_num_relax` is stricly positive.

dinfitem.mio_obj_int ("MSK_DINF_MIO_OBJ_INT")

The primal objective value corresponding to the best integer feasible solution. Please note that at least one integer feasible solution must have located i.e. check `iinfitem.mio_num_int_solutions`.

dinfitem.mio_obj_rel_gap ("MSK_DINF_MIO_OBJ_REL_GAP")

Given that the mixed-integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the relative gap defined by

$$\frac{|(\text{objective value of feasible solution}) - (\text{objective bound})|}{\max(\delta, |(\text{objective value of feasible solution})|)}.$$

where δ is given by the paramater `dparam.mio_rel_gap_const`. Otherwise it has the value -1.0.

dinfitem.mio_optimizer_time ("MSK_DINF_MIO_OPTIMIZER_TIME")

Time spent in the optimizer while solving the relaxtions.

dinfitem.mio_probing_time ("MSK_DINF_MIO_PROBING_TIME")

Total time for probing.

dinfitem.mio_root_cutgen_time ("MSK_DINF_MIO_ROOT_CUTGEN_TIME")

Total time for cut generation.

dinfitem.mio_root_optimizer_time ("MSK_DINF_MIO_ROOT_OPTIMIZER_TIME")

Time spent in the optimizer while solving the root relaxation.

dinfitem.mio_root_presolve_time ("MSK_DINF_MIO_ROOT_PRESOLVE_TIME")

Time spent in while presolveing the root relaxation.

dinfitem.mio_time ("MSK_DINF_MIO_TIME")

Time spent in the mixed-integer optimizer.

dinfitem.mio_user_obj_cut ("MSK_DINF_MIO_USER_OBJ_CUT")

If the objective cut is used, then this information item has the value of the cut.

dinfitem.optimizer_time ("MSK_DINF_OPTIMIZER_TIME")

Total time spent in the optimizer since it was invoked.

dinfitem.presolve_eli_time ("MSK_DINF_PRESOLVE_ELI_TIME")

Total time spent in the eliminator since the presolve was invoked.

dinfitem.presolve_lindep_time ("MSK_DINF_PRESOLVE_LINDEP_TIME")

Total time spent in the linear dependency checker since the presolve was invoked.

dinfitem.presolve_time ("MSK_DINF_PRESOLVE_TIME")

Total time (in seconds) spent in the presolve since it was invoked.

dinfitem.primal_repair_penalty_obj ("MSK_DINF_PRIMAL_REPAIR_PENALTY_OBJ")

The optimal objective value of the penalty function.

dinfitem.qcqo_reformulate_time ("MSK_DINF_QCQO_REFORMULATE_TIME")

Time spent with conic quadratic reformulation.

dinfitem.rd_time ("MSK_DINF_RD_TIME")

Time spent reading the data file.

dinfitem.sim_dual_time ("MSK_DINF_SIM_DUAL_TIME")

Time spent in the dual simplex optimizer since invoking it.

dinfitem.sim_feas ("MSK_DINF_SIM_FEAS")

Feasibility measure reported by the simplex optimizer.

dinfitem.sim_network_dual_time ("MSK_DINF_SIM_NETWORK_DUAL_TIME")

Time spent in the dual network simplex optimizer since invoking it.

dinfitem.sim_network_primal_time ("MSK_DINF_SIM_NETWORK_PRIMAL_TIME")

Time spent in the primal network simplex optimizer since invoking it.

dinfitem.sim_network_time ("MSK_DINF_SIM_NETWORK_TIME")

Time spent in the network simplex optimizer since invoking it.

dinfitem.sim_obj ("MSK_DINF_SIM_OBJ")

Objective value reported by the simplex optimizer.

dinfitem.sim_primal_dual_time ("MSK_DINF_SIM_PRIMAL_DUAL_TIME")

Time spent in the primal-dual simplex optimizer optimizer since invoking it.

dinfitem.sim_primal_time ("MSK_DINF_SIM_PRIMAL_TIME")

Time spent in the primal simplex optimizer since invoking it.

dinfitem.sim_time ("MSK_DINF_SIM_TIME")

Time spent in the simplex optimizer since invoking it.

dinfitem.sol_bas_dual_obj ("MSK_DINF_SOL_BAS_DUAL_OBJ")

Dual objective value of the basic solution.

dinfitem.sol_bas_dviolcon ("MSK_DINF_SOL_BAS_DVIOLCON")

Maximal dual bound violation for x^c in the basic solution.

dinfitem.sol_bas_dviolvar ("MSK_DINF_SOL_BAS_DVIOLVAR")

Maximal dual bound violation for x^x in the basic solution.

dinfitem.sol_bas_primal_obj ("MSK_DINF_SOL_BAS_PRIMAL_OBJ")

Primal objective value of the basic solution.

dinfitem.sol_bas_pviolcon ("MSK_DINF_SOL_BAS_PVIOLCON")

Maximal primal bound violation for x^c in the basic solution.

dinfitem.sol_bas_pviolvar ("MSK_DINF_SOL_BAS_PVIOLVAR")

Maximal primal bound violation for x^x in the basic solution.

dinfitem.sol_itg_primal_obj ("MSK_DINF_SOL_ITG_PRIMAL_OBJ")

Primal objective value of the integer solution.

dinfitem.sol_itg_pviolbarvar ("MSK_DINF_SOL_ITG_PVIOLBARVAR")

Maximal primal bound violation for $\bar{X}$ in the integer solution.

dinfitem.sol_itg_pviolcon ("MSK_DINF_SOL_ITG_PVIOLCON")

Maximal primal bound violation for x^c in the integer solution.

dinfitem.sol_itg_pviolcones ("MSK_DINF_SOL_ITG_PVIOLCONES")

Maximal primal violation for primal conic constraints in the integer solution.

dinfitem.sol_itg_pviolitg ("MSK_DINF_SOL_ITG_PVIOLITG")

Maximal violation for the integer constraints in the integer solution.

dinfitem.sol_itg_pviolvar ("MSK_DINF_SOL_ITG_PVIOLVAR")

Maximal primal bound violation for x^x in the integer solution.

dinfitem.sol_itr_dual_obj ("MSK_DINF_SOL_ITR_DUAL_OBJ")

Dual objective value of the interior-point solution.

dinfitem.sol_itr_dviolbarvar ("MSK_DINF_SOL_ITR_DVIOLBARVAR")

Maximal dual bound violation for $\bar{X}$ in the interior-point solution.

dinfitem.sol_itr_dviolcon ("MSK_DINF_SOL_ITR_DVIOLCON")

Maximal dual bound violation for x^c in the interior-point solution.

dinfitem.sol_itr_dviolcones ("MSK_DINF_SOL_ITR_DVIOLCONES")

Maximal dual violation for dual conic constraints in the interior-point solution.

dinfitem.sol_itr_dviolvar ("MSK_DINF_SOL_ITR_DVIOLVAR")

Maximal dual bound violation for x^x in the interior-point solution.

dinfitem.sol_itr_primal_obj ("MSK_DINF_SOL_ITR_PRIMAL_OBJ")

Primal objective value of the interior-point solution.

dinfitem.sol_itr_pviolbarvar ("MSK_DINF_SOL_ITR_PVIOLBARVAR")

Maximal primal bound violation for $\bar{X}$ in the interior-point solution.

dinfitem.sol_itr_pviolcon ("MSK_DINF_SOL_ITR_PVIOLCON")

Maximal primal bound violation for x^c in the interior-point solution.

dinfitem.sol_itr_pviolcones ("MSK_DINF_SOL_ITR_PVIOLCONES")

Maximal primal violation for primal conic constraints in the interior-point solution.

dinfitem.sol_itr_pviolvar ("MSK_DINF_SOL_ITR_PVIOLVAR")

Maximal primal bound violation for x^x in the interior-point solution.

3.11 feasrepairtype

feasrepairtype.optimize_combined ("MSK_FEASREPAIR_OPTIMIZE_COMBINED")

Minimize with original objective subject to minimal weighted violation of bounds.

feasrepairtype.optimize_none ("MSK_FEASREPAIR_OPTIMIZE_NONE")

Do not optimize the feasibility repair problem.

feasrepairtype.optimize_penalty ("MSK_FEASREPAIR_OPTIMIZE_PENALTY")

Minimize weighted sum of violations.

3.12 feature

feature.ptom ("MSK_FEATURE_PTOM")

Mixed-integer extension.

feature.pton ("MSK_FEATURE_PTON")

Nonlinear extension.

feature.ptox ("MSK_FEATURE_PTOX")

Non-convex extension.

feature.pts ("MSK_FEATURE_PTS")

Base system.

3.13 iinfitem

iinfitem.ana_pro_num_con ("MSK_IINF_ANA_PRO_NUM_CON")

Number of constraints in the problem.

This value is set by Task.analyzeproblem.

iinfitem.ana_pro_num_con_eq ("MSK_IINF_ANA_PRO_NUM_CON_EQ")

Number of equality constraints.

This value is set by Task.analyzeproblem.

iinfitem.ana_pro_num_con_fr ("MSK_IINF_ANA_PRO_NUM_CON_FR")

Number of unbounded constraints.

This value is set by Task.analyzeproblem.

iinfitem.ana_pro_num_con_lo ("MSK_IINF_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 ("MSK_IINF_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 ("MSK_IINF_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 ("MSK_IINF_ANA_PRO_NUM_VAR")

Number of variables in the problem.

This value is set by Task.analyzeproblem.

iinfitem.ana_pro_num_var_bin ("MSK_IINF_ANA_PRO_NUM_VAR_BIN")

Number of binary (0-1) variables.

This value is set by Task.analyzeproblem.

iinfitem.ana_pro_num_var_cont ("MSK_IINF_ANA_PRO_NUM_VAR_CONT")

Number of continuous variables.

This value is set by Task.analyzeproblem.

iinfitem.ana_pro_num_var_eq ("MSK_IINF_ANA_PRO_NUM_VAR_EQ")

Number of fixed variables.

This value is set by Task.analyzeproblem.

iinfitem.ana_pro_num_var_fr ("`MSK_IINF_ANA_PRO_NUM_VAR_FR`")

Number of free variables.

This value is set by `Task.analyzeproblem`.

iinfitem.ana_pro_num_var_int ("`MSK_IINF_ANA_PRO_NUM_VAR_INT`")

Number of general integer variables.

This value is set by `Task.analyzeproblem`.

iinfitem.ana_pro_num_var_lo ("`MSK_IINF_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 ("`MSK_IINF_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 ("`MSK_IINF_ANA_PRO_NUM_VAR_UP`")

Number of variables with an upper bound and an infinite lower bound. This value is set by

This value is set by `Task.analyzeproblem`.

iinfitem.concurrent_fastest_optimizer ("`MSK_IINF_CONCURRENT_FASTEST_OPTIMIZER`")

The type of the optimizer that finished first in a concurrent optimization.

iinfitem.intpnt_factor_dim_dense ("`MSK_IINF_INTPNT_FACTOR_DIM_DENSE`")

Dimension of the dense sub system in factorization.

iinfitem.intpnt_iter ("`MSK_IINF_INTPNT_ITER`")

Number of interior-point iterations since invoking the interior-point optimizer.

iinfitem.intpnt_num_threads ("`MSK_IINF_INTPNT_NUM_THREADS`")

Number of threads that the interior-point optimizer is using.

iinfitem.intpnt_solve_dual ("`MSK_IINF_INTPNT_SOLVE_DUAL`")

Non-zero if the interior-point optimizer is solving the dual problem.

iinfitem.mio_construct_num_roundings ("`MSK_IINF_MIO_CONSTRUCT_NUM_ROUNDINGS`")

Number of values in the integer solution that is rounded to an integer value.

iinfitem.mio_construct_solution ("`MSK_IINF_MIO_CONSTRUCT_SOLUTION`")

If this item has the value 0, then MOSEK did not try to construct an initial integer feasible solution. If the item has a positive value, then MOSEK successfully constructed an initial integer feasible solution.

iinfitem.mio_initial_solution ("`MSK_IINF_MIO_INITIAL_SOLUTION`")

Is non-zero if an initial integer solution is specified.

iinfitem.mio_num_active_nodes ("MSK_IINF_MIO_NUM_ACTIVE_NODES")
Number of active brabch bound nodes.

iinfitem.mio_num_basis_cuts ("MSK_IINF_MIO_NUM_BASIS_CUTS")
Number of basis cuts.

iinfitem.mio_num_branch ("MSK_IINF_MIO_NUM_BRANCH")
Number of branches performed during the optimization.

iinfitem.mio_num_cardgub_cuts ("MSK_IINF_MIO_NUM_CARDGUB_CUTS")
Number of cardgub cuts.

iinfitem.mio_num_clique_cuts ("MSK_IINF_MIO_NUM_CLIQUE_CUTS")
Number of clique cuts.

iinfitem.mio_num_coef_redc_cuts ("MSK_IINF_MIO_NUM_COEF_REDC_CUTS")
Number of coef. redc. cuts.

iinfitem.mio_num_contra_cuts ("MSK_IINF_MIO_NUM_CONTRA_CUTS")
Number of contra cuts.

iinfitem.mio_num_disagg_cuts ("MSK_IINF_MIO_NUM_DISAGG_CUTS")
Number of diasagg cuts.

iinfitem.mio_num_flow_cover_cuts ("MSK_IINF_MIO_NUM_FLOW_COVER_CUTS")
Number of flow cover cuts.

iinfitem.mio_num_gcd_cuts ("MSK_IINF_MIO_NUM_GCD_CUTS")
Number of gcd cuts.

iinfitem.mio_num_gomory_cuts ("MSK_IINF_MIO_NUM_GOMORY_CUTS")
Number of Gomory cuts.

iinfitem.mio_num_gub_cover_cuts ("MSK_IINF_MIO_NUM_GUB_COVER_CUTS")
Number of GUB cover cuts.

iinfitem.mio_num_int_solutions ("MSK_IINF_MIO_NUM_INT_SOLUTIONS")
Number of integer feasible solutions that has been found.

iinfitem.mio_num_knapsur_cover_cuts ("MSK_IINF_MIO_NUM_KNAPSUR_COVER_CUTS")
Number of knapsack cover cuts.

iinfitem.mio_num_lattice_cuts ("MSK_IINF_MIO_NUM_LATTICE_CUTS")
Number of lattice cuts.

iinfitem.mio_num_lift_cuts ("MSK_IINF_MIO_NUM_LIFT_CUTS")
Number of lift cuts.

iinfitem.mio_num_obj_cuts ("MSK_IINF_MIO_NUM_OBJ_CUTS")

Number of obj cuts.

iinfitem.mio_num_plan_loc_cuts ("MSK_IINF_MIO_NUM_PLAN_LOC_CUTS")

Number of loc cuts.

iinfitem.mio_num_relax ("MSK_IINF_MIO_NUM_RELAX")

Number of relaxations solved during the optimization.

iinfitem.mio_numcon ("MSK_IINF_MIO_NUMCON")

Number of constraints in the problem solved be the mixed-integer optimizer.

iinfitem.mio_numint ("MSK_IINF_MIO_NUMINT")

Number of integer variables in the problem solved be the mixed-integer optimizer.

iinfitem.mio_numvar ("MSK_IINF_MIO_NUMVAR")

Number of variables in the problem solved be the mixed-integer optimizer.

iinfitem.mio_obj_bound_defined ("MSK_IINF_MIO_OBJ_BOUND_DEFINED")

Non-zero if a valid objective bound has been found, otherwise zero.

iinfitem.mio_total_num_cuts ("MSK_IINF_MIO_TOTAL_NUM_CUTS")

Total number of cuts generated by the mixed-integer optimizer.

iinfitem.mio_user_obj_cut ("MSK_IINF_MIO_USER_OBJ_CUT")

If it is non-zero, then the objective cut is used.

iinfitem.opt_numcon ("MSK_IINF_OPT_NUMCON")

Number of constraints in the problem solved when the optimizer is called.

iinfitem.opt_numvar ("MSK_IINF_OPT_NUMVAR")

Number of variables in the problem solved when the optimizer is called

iinfitem.optimize_response ("MSK_IINF_OPTIMIZE_RESPONSE")

The reponse code returned by optimize.

iinfitem.rd_numbarvar ("MSK_IINF_RD_NUMBARVAR")

Number of variables read.

iinfitem.rd_numcon ("MSK_IINF_RD_NUMCON")

Number of constraints read.

iinfitem.rd_numcone ("MSK_IINF_RD_NUMCONE")

Number of conic constraints read.

iinfitem.rd_numintvar ("MSK_IINF_RD_NUMINTVAR")

Number of integer-constrained variables read.

iinfitem.rd_numq ("MSK_IINF_RD_NUMQ")

Number of nonempty Q matrixes read.

iinfitem.rd_numvar ("MSK_IINF_RD_NUMVAR")

Number of variables read.

iinfitem.rd_protype ("MSK_IINF_RD_PROTYPE")

Problem type.

iinfitem.sim_dual_deg_iter ("MSK_IINF_SIM_DUAL_DEG_ITER")

The number of dual degenerate iterations.

iinfitem.sim_dual_hotstart ("MSK_IINF_SIM_DUAL_HOTSTART")

If 1 then the dual simplex algorithm is solving from an advanced basis.

iinfitem.sim_dual_hotstart_lu ("MSK_IINF_SIM_DUAL_HOTSTART_LU")

If 1 then a valid basis factorization of full rank was located and used by the dual simplex algorithm.

iinfitem.sim_dual_inf_iter ("MSK_IINF_SIM_DUAL_INF_ITER")

The number of iterations taken with dual infeasibility.

iinfitem.sim_dual_iter ("MSK_IINF_SIM_DUAL_ITER")

Number of dual simplex iterations during the last optimization.

iinfitem.sim_network_dual_deg_iter ("MSK_IINF_SIM_NETWORK_DUAL_DEG_ITER")

The number of dual network degenerate iterations.

iinfitem.sim_network_dual_hotstart ("MSK_IINF_SIM_NETWORK_DUAL_HOTSTART")

If 1 then the dual network simplex algorithm is solving from an advanced basis.

iinfitem.sim_network_dual_hotstart_lu ("MSK_IINF_SIM_NETWORK_DUAL_HOTSTART_LU")

If 1 then a valid basis factorization of full rank was located and used by the dual network simplex algorithm.

iinfitem.sim_network_dual_inf_iter ("MSK_IINF_SIM_NETWORK_DUAL_INF_ITER")

The number of iterations taken with dual infeasibility in the network optimizer.

iinfitem.sim_network_dual_iter ("MSK_IINF_SIM_NETWORK_DUAL_ITER")

Number of dual network simplex iterations during the last optimization.

iinfitem.sim_network_primal_deg_iter ("MSK_IINF_SIM_NETWORK_PRIMAL_DEG_ITER")

The number of primal network degenerate iterations.

iinfitem.sim_network_primal_hotstart ("MSK_IINF_SIM_NETWORK_PRIMAL_HOTSTART")

If 1 then the primal network simplex algorithm is solving from an advanced basis.

iinfitem.sim_network_primal_hotstart_lu ("MSK_IINF_SIM_NETWORK_PRIMAL_HOTSTART_LU")

If 1 then a valid basis factorization of full rank was located and used by the primal network simplex algorithm.

iinfitem.sim_network_primal_inf_iter ("MSK_IINF_SIM_NETWORK_PRIMAL_INF_ITER")

The number of iterations taken with primal infeasibility in the network optimizer.

iinfitem.sim_network_primal_iter ("MSK_IINF_SIM_NETWORK_PRIMAL_ITER")

Number of primal network simplex iterations during the last optimization.

iinfitem.sim_numcon ("MSK_IINF_SIM_NUMCON")

Number of constraints in the problem solved by the simplex optimizer.

iinfitem.sim_numvar ("MSK_IINF_SIM_NUMVAR")

Number of variables in the problem solved by the simplex optimizer.

iinfitem.sim_primal_deg_iter ("MSK_IINF_SIM_PRIMAL_DEG_ITER")

The number of primal degenerate iterations.

iinfitem.sim_primal_dual_deg_iter ("MSK_IINF_SIM_PRIMAL_DUAL_DEG_ITER")

The number of degenerate major iterations taken by the primal dual simplex algorithm.

iinfitem.sim_primal_dual_hotstart ("MSK_IINF_SIM_PRIMAL_DUAL_HOTSTART")

If 1 then the primal dual simplex algorithm is solving from an advanced basis.

iinfitem.sim_primal_dual_hotstart_lu ("MSK_IINF_SIM_PRIMAL_DUAL_HOTSTART_LU")

If 1 then a valid basis factorization of full rank was located and used by the primal dual simplex algorithm.

iinfitem.sim_primal_dual_inf_iter ("MSK_IINF_SIM_PRIMAL_DUAL_INF_ITER")

The number of master iterations with dual infeasibility taken by the primal dual simplex algorithm.

iinfitem.sim_primal_dual_iter ("MSK_IINF_SIM_PRIMAL_DUAL_ITER")

Number of primal dual simplex iterations during the last optimization.

iinfitem.sim_primal_hotstart ("MSK_IINF_SIM_PRIMAL_HOTSTART")

If 1 then the primal simplex algorithm is solving from an advanced basis.

iinfitem.sim_primal_hotstart_lu ("MSK_IINF_SIM_PRIMAL_HOTSTART_LU")

If 1 then a valid basis factorization of full rank was located and used by the primal simplex algorithm.

iinfitem.sim_primal_inf_iter ("MSK_IINF_SIM_PRIMAL_INF_ITER")

The number of iterations taken with primal infeasibility.

iinfitem.sim_primal_iter ("`MSK_IINF_SIM_PRIMAL_ITER`")

Number of primal simplex iterations during the last optimization.

iinfitem.sim_solve_dual ("`MSK_IINF_SIM_SOLVE_DUAL`")

Is non-zero if dual problem is solved.

iinfitem.sol_bas_prosta ("`MSK_IINF_SOL_BAS_PROSTA`")

Problem status of the basic solution. Updated after each optimization.

iinfitem.sol_bas_solsta ("`MSK_IINF_SOL_BAS_SOLSTA`")

Solution status of the basic solution. Updated after each optimization.

iinfitem.sol_int_prosta ("`MSK_IINF_SOL_INT_PROSTA`")

Deprecated.

iinfitem.sol_int_solsta ("`MSK_IINF_SOL_INT_SOLSTA`")

Degrecated.

iinfitem.sol_itg_prosta ("`MSK_IINF_SOL_ITG_PROSTA`")

Problem status of the integer solution. Updated after each optimization.

iinfitem.sol_itg_solsta ("`MSK_IINF_SOL_ITG_SOLSTA`")

Solution status of the integer solution. Updated after each optimization.

iinfitem.sol_itr_prosta ("`MSK_IINF_SOL_ITR_PROSTA`")

Problem status of the interior-point solution. Updated after each optimization.

iinfitem.sol_itr_solsta ("`MSK_IINF_SOL_ITR_SOLSTA`")

Solution status of the interior-point solution. Updated after each optimization.

iinfitem.sto_num_a_cache_flushes ("`MSK_IINF_STO_NUM_A_CACHE_FLUSHES`")

Number of times the cache of A elements is flushed. A large number implies that `maxnumanz` is too small as well as an inefficient usage of MOSEK.

iinfitem.sto_num_a_realloc ("`MSK_IINF_STO_NUM_A_REALLOC`")

Number of times the storage for storing A has been changed. A large value may indicates that memory fragmentation may occur.

iinfitem.sto_num_a_transposes ("`MSK_IINF_STO_NUM_A_TRANSPOSES`")

Number of times the A matrix is transposed. A large number implies that `maxnumanz` is too small or an inefficient usage of MOSEK. This will occur in particular if the code alternate between accessing rows and columns of A.

3.14 inftype

inftype.dou_type ("MSK_INF_DOU_TYPE")

Is a double information type.

inftype.int_type ("MSK_INF_INT_TYPE")

Is an integer.

inftype.lint_type ("MSK_INF_LINT_TYPE")

Is a long integer.

3.15 intpnthotstart

intpnthotstart.dual ("MSK_INTPNT_HOTSTART_DUAL")

The interior-point optimizer exploits the dual solution only.

intpnthotstart.none ("MSK_INTPNT_HOTSTART_NONE")

The interior-point optimizer performs a coldstart.

intpnthotstart.primal ("MSK_INTPNT_HOTSTART_PRIMAL")

The interior-point optimizer exploits the primal solution only.

intpnthotstart.primal_dual ("MSK_INTPNT_HOTSTART_PRIMAL_DUAL")

The interior-point optimizer exploits both the primal and dual solution.

3.16 iomode

iomode.read ("MSK_IOMODE_READ")

The file is read-only.

iomode.readwrite ("MSK_IOMODE_READWRITE")

The file is to read and written.

iomode.write ("MSK_IOMODE_WRITE")

The file is write-only. If the file exists then it is truncated when it is opened. Otherwise it is created when it is opened.

3.17 language

language.dan ("MSK_LANG_DAN")
: Danish language selection

language.eng ("MSK_LANG_ENG")
: English language selection

3.18 liinfitem

liinfitem.bi_clean_dual_deg_iter ("MSK_LIINF_BI_CLEAN_DUAL_DEG_ITER")
: Number of dual degenerate clean iterations performed in the basis identification.

liinfitem.bi_clean_dual_iter ("MSK_LIINF_BI_CLEAN_DUAL_ITER")
: Number of dual clean iterations performed in the basis identification.

liinfitem.bi_clean_primal_deg_iter ("MSK_LIINF_BI_CLEAN_PRIMAL_DEG_ITER")
: Number of primal degenerate clean iterations performed in the basis identification.

liinfitem.bi_clean_primal_dual_deg_iter ("MSK_LIINF_BI_CLEAN_PRIMAL_DUAL_DEG_ITER")
: Number of primal-dual degenerate clean iterations performed in the basis identification.

liinfitem.bi_clean_primal_dual_iter ("MSK_LIINF_BI_CLEAN_PRIMAL_DUAL_ITER")
: Number of primal-dual clean iterations performed in the basis identification.

liinfitem.bi_clean_primal_dual_sub_iter ("MSK_LIINF_BI_CLEAN_PRIMAL_DUAL_SUB_ITER")
: Number of primal-dual subproblem clean iterations performed in the basis identification.

liinfitem.bi_clean_primal_iter ("MSK_LIINF_BI_CLEAN_PRIMAL_ITER")
: Number of primal clean iterations performed in the basis identification.

liinfitem.bi_dual_iter ("MSK_LIINF_BI_DUAL_ITER")
: Number of dual pivots performed in the basis identification.

liinfitem.bi_primal_iter ("MSK_LIINF_BI_PRIMAL_ITER")
: Number of primal pivots performed in the basis identification.

liinfitem.intpnt_factor_num_nz ("MSK_LIINF_INTPNT_FACTOR_NUM_NZ")
: Number of non-zeros in factorization.

liinfitem.mio_intpnt_iter ("MSK_LIINF_MIO_INTPNT_ITER")
: Number of interior-point iterations performed by the mixed-integer optimizer.

liinfitem.mio_simplex_iter ("MSK_LIINF_MIO_SIMPLEX_ITER")
: Number of simplex iterations performed by the mixed-integer optimizer.

liinfitem.rd_numanz ("MSK_LIINF_RD_NUMANZ")

Number of non-zeros in A that is read.

liinfitem.rd_numqnz ("MSK_LIINF_RD_NUMQNZ")

Number of Q non-zeros.

3.19 mark

mark.lo ("MSK_MARK_LO")

The lower bound is selected for sensitivity analysis.

mark.up ("MSK_MARK_UP")

The upper bound is selected for sensitivity analysis.

3.20 miocontsoltype

miocontsoltype.itg ("MSK_MIO_CONT_SOL_ITG")

The reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. A solution is only reported in case the problem has a primal feasible solution.

miocontsoltype.itg_rel ("MSK_MIO_CONT_SOL_ITG_REL")

In case the problem is primal feasible then the reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. If the problem is primal infeasible, then the solution to the root node problem is reported.

miocontsoltype.none ("MSK_MIO_CONT_SOL_NONE")

No interior-point or basic solution are reported when the mixed-integer optimizer is used.

miocontsoltype.root ("MSK_MIO_CONT_SOL_ROOT")

The reported interior-point and basic solutions are a solution to the root node problem when mixed-integer optimizer is used.

3.21 miomode

miomode.ignored ("MSK_MIO_MODE_IGNORED")

The integer constraints are ignored and the problem is solved as a continuous problem.

miomode.lazy ("MSK_MIO_MODE_LAZY")

Integer restrictions should be satisfied if an optimizer is available for the problem.

miomode.satisfied ("MSK_MIO_MODE_SATISFIED")

Integer restrictions should be satisfied.

3.22 mionodeseltype

mionodeseltype.best ("MSK_MIO_NODE_SELECTION_BEST")

The optimizer employs a best bound node selection strategy.

mionodeseltype.first ("MSK_MIO_NODE_SELECTION_FIRST")

The optimizer employs a depth first node selection strategy.

mionodeseltype.free ("MSK_MIO_NODE_SELECTION_FREE")

The optimizer decides the node selection strategy.

mionodeseltype.hybrid ("MSK_MIO_NODE_SELECTION_HYBRID")

The optimizer employs a hybrid strategy.

mionodeseltype.pseudo ("MSK_MIO_NODE_SELECTION_PSEUDO")

The optimizer employs selects the node based on a pseudo cost estimate.

mionodeseltype.worst ("MSK_MIO_NODE_SELECTION_WORST")

The optimizer employs a worst bound node selection strategy.

3.23 mpsformat

mpsformat.free ("MSK_MPS_FORMAT_FREE")

It is assumed that the input file satisfies the free MPS format. This implies that spaces are not allowed in names. Otherwise the format is free.

mpsformat.relaxed ("MSK_MPS_FORMAT_RELAXED")

It is assumed that the input file satisfies a slightly relaxed version of the MPS format.

mpsformat.strict ("MSK_MPS_FORMAT_STRICT")

It is assumed that the input file satisfies the MPS format strictly.

3.24 msgkey

msgkey.mps_selected ("MSK_MSG_MPS_SELECTED")

msgkey.reading_file ("MSK_MSG_READING_FILE")

msgkey.writing_file ("MSK_MSG_WRITING_FILE")

3.25 nametype

nametype.gen ("MSK_NAME_TYPE_GEN")

General names. However, no duplicate and blank names are allowed.

nametype.lp ("MSK_NAME_TYPE_LP")

LP type names.

nametype.mps ("MSK_NAME_TYPE_MPS")

MPS type names.

3.26 objsense

objsense.maximize ("MSK_OBJECTIVE_SENSE_MAXIMIZE")

The problem should be maximized.

objsense.minimize ("MSK_OBJECTIVE_SENSE_MINIMIZE")

The problem should be minimized.

3.27 onoffkey

onoffkey.off ("MSK_OFF")

Switch the option off.

onoffkey.on ("MSK_ON")

Switch the option on.

3.28 optimizertype

optimizertype.concurrent ("MSK_OPTIMIZER_CONCURRENT")

The optimizer for nonconvex nonlinear problems.

optimizertype.conic ("MSK_OPTIMIZER_CONIC")

The optimizer for problems having conic constraints.

optimizertype.dual_simplex ("MSK_OPTIMIZER_DUAL_SIMPLEX")

The dual simplex optimizer is used.

optimizertype.free ("MSK_OPTIMIZER_FREE")

The optimizer is chosen automatically.

optimizertype.free_simplex ("MSK_OPTIMIZER_FREE_SIMPLEX")

One of the simplex optimizers is used.

optimizertype.intpnt ("MSK_OPTIMIZER_INTPNT")

The interior-point optimizer is used.

optimizertype.mixed_int ("MSK_OPTIMIZER_MIXED_INT")

The mixed-integer optimizer.

optimizertype.mixed_int_conic ("MSK_OPTIMIZER_MIXED_INT_CONIC")

The mixed-integer optimizer for conic and linear problems.

optimizertype.network_primal_simplex ("MSK_OPTIMIZER_NETWORK_PRIMAL_SIMPLEX")

The network primal simplex optimizer is used. It is only applicable to pure network problems.

optimizertype.nonconvex ("MSK_OPTIMIZER_NONCONVEX")

The optimizer for nonconvex nonlinear problems.

optimizertype.primal_dual_simplex ("MSK_OPTIMIZER_PRIMAL_DUAL_SIMPLEX")

The primal dual simplex optimizer is used.

optimizertype.primal_simplex ("MSK_OPTIMIZER_PRIMAL_SIMPLEX")

The primal simplex optimizer is used.

3.29 orderingtype

orderingtype.appminloc ("MSK_ORDER_METHOD_APPMINLOC")

Approximate minimum local fill-in ordering is employed.

orderingtype.experimental ("MSK_ORDER_METHOD_EXPERIMENTAL")

This option should not be used.

orderingtype.force_graphpar ("MSK_ORDER_METHOD_FORCE_GRAPHPAR")

Always use the graph partitioning based ordering even if it is worse that the approximate minimum local fill ordering.

orderingtype.free ("MSK_ORDER_METHOD_FREE")

The ordering method is chosen automatically.

orderingtype.none ("MSK_ORDER_METHOD_NONE")

No ordering is used.

orderingtype.try_graphpar ("MSK_ORDER_METHOD_TRY_GRAPHPAR")

Always try the the graph partitioning based ordering.

3.30 parametertype

parametertype.dou_type ("MSK_PAR_DOU_TYPE")

Is a double parameter.

parametertype.int_type ("MSK_PAR_INT_TYPE")

Is an integer parameter.

parametertype.invalid_type ("MSK_PAR_INVALID_TYPE")

Not a valid parameter.

parametertype.str_type ("MSK_PAR_STR_TYPE")

Is a string parameter.

3.31 presolvemode

presolvemode.free ("MSK_PRESOLVE_MODE_FREE")

It is decided automatically whether to presolve before the problem is optimized.

presolvemode.off ("MSK_PRESOLVE_MODE_OFF")

The problem is not presolved before it is optimized.

presolvemode.on ("MSK_PRESOLVE_MODE_ON")

The problem is presolved before it is optimized.

3.32 problemitem

problemitem.con ("MSK_PI_CON")

Item is a constraint.

problemitem.cone ("MSK_PI_CONE")

Item is a cone.

problemitem.var ("MSK_PI_VAR")

Item is a variable.

3.33 problemtype

problemtype.conic ("MSK_PROBTYPE_CONIC")

A conic optimization.

problemtype.geco ("MSK_PROBTYPE_GECO")

General convex optimization.

problemtype.lo ("MSK_PROBTYPE_LO")

The problem is a linear optimization problem.

problemtype.mixed ("MSK_PROBTYPE_MIXED")

General nonlinear constraints and conic constraints. This combination can not be solved by MOSEK.

problemtype.qcqo ("MSK_PROBTYPE_QCQO")

The problem is a quadratically constrained optimization problem.

problemtype.qo ("MSK_PROBTYPE_QO")

The problem is a quadratic optimization problem.

3.34 prosta

prosta.dual_feas ("MSK_PRO_STA_DUAL_FEAS")

The problem is dual feasible.

prosta.dual_infeas ("MSK_PRO_STA_DUAL_INFEAS")

The problem is dual infeasible.

prosta.ill_posed ("MSK_PRO_STA_ILL_POSED")

The problem is ill-posed. For example, it may be primal and dual feasible but have a positive duality gap.

prosta.near_dual_feas ("MSK_PRO_STA_NEAR_DUAL_FEAS")

The problem is at least nearly dual feasible.

prosta.near_prim_and_dual_feas ("MSK_PRO_STA_NEAR_PRIM_AND_DUAL_FEAS")

The problem is at least nearly primal and dual feasible.

prosta.near_prim_feas ("MSK_PRO_STA_NEAR_PRIM_FEAS")

The problem is at least nearly primal feasible.

prosta.prim_and_dual_feas ("MSK_PRO_STA_PRIM_AND_DUAL_FEAS")

The problem is primal and dual feasible.

prosta.prim_and_dual_infeas ("MSK_PRO_STA_PRIM_AND_DUAL_INFEAS")

The problem is primal and dual infeasible.

prosta.prim_feas ("MSK_PRO_STA_PRIM_FEAS")

The problem is primal feasible.

prosta.prim_infeas ("MSK_PRO_STA_PRIM_INFEAS")

The problem is primal infeasible.

prosta.prim_infeas_or_unbounded ("MSK_PRO_STA_PRIM_INFEAS_OR_UNBOUNDED")

The problem is either primal infeasible or unbounded. This may occur for mixed-integer problems.

prosta.unknown ("MSK_PRO_STA_UNKNOWN")

Unknown problem status.

3.35 rescode

rescode.err_ad_invalid_codelist ("MSK_RES_ERR_AD_INVALID_CODELIST")

The code list data was invalid.

rescode.err_ad_invalid_operand ("MSK_RES_ERR_AD_INVALID_OPERAND")

The code list data was invalid. An unknown operand was used.

rescode.err_ad_invalid_operator ("MSK_RES_ERR_AD_INVALID_OPERATOR")

The code list data was invalid. An unknown operator was used.

rescode.err_ad_missing_operand ("MSK_RES_ERR_AD_MISSING_OPERAND")

The code list data was invalid. Missing operand for operator.

rescode.err_ad_missing_return ("MSK_RES_ERR_AD_MISSING_RETURN")

The code list data was invalid. Missing return operation in function.

rescode.err_api_array_too_small ("MSK_RES_ERR_API_ARRAY_TOO_SMALL")

An input array was too short.

rescode.err_api_cb_connect ("MSK_RES_ERR_API_CB_CONNECT")

Failed to connect a callback object.

rescode.err_api_fatal_error ("MSK_RES_ERR_API_FATAL_ERROR")

An internal error occurred in the API. Please report this problem.

rescode.err_api_internal ("MSK_RES_ERR_API_INTERNAL")

An internal fatal error occurred in an interface function.

rescode.err_arg_is_too_large ("MSK_RES_ERR_ARG_IS_TOO_LARGE")

The value of a argument is too small.

rescode.err_arg_is_too_small ("MSK_RES_ERR_ARG_IS_TOO_SMALL")

The value of a argument is too small.

rescode.err_argument_dimension ("MSK_RES_ERR_ARGUMENT_DIMENSION")

A function argument is of incorrect dimension.

rescode.err_argument_is_too_large ("MSK_RES_ERR_ARGUMENT_IS_TOO_LARGE")

The value of a function argument is too large.

rescode.err_argument_lenneq ("MSK_RES_ERR_ARGUMENT_LENNEQ")

Incorrect length of arguments.

rescode.err_argument_perm_array ("MSK_RES_ERR_ARGUMENT_PERM_ARRAY")

An invalid permutation array is specified.

rescode.err_argument_type ("MSK_RES_ERR_ARGUMENT_TYPE")

Incorrect argument type.

rescode.err_bar_var_dim ("MSK_RES_ERR_BAR_VAR_DIM")

The dimension of a symmetric matrix variable has to greater than 0.

rescode.err_basis ("MSK_RES_ERR_BASIS")

An invalid basis is specified. Either too many or too few basis variables are specified.

rescode.err_basis_factor ("MSK_RES_ERR_BASIS_FACTOR")

The factorization of the basis is invalid.

rescode.err_basis_singular ("MSK_RES_ERR_BASIS_SINGULAR")

The basis is singular and hence cannot be factored.

rescode.err_blank_name ("MSK_RES_ERR_BLANK_NAME")

An all blank name has been specified.

rescode.err_cannot_clone_nl ("MSK_RES_ERR_CANNOT_CLONE_NL")

A task with a nonlinear function call-back cannot be cloned.

rescode.err_cannot_handle_nl ("MSK_RES_ERR_CANNOT_HANDLE_NL")

A function cannot handle a task with nonlinear function call-backs.

rescode.err_cbf_duplicate_acoord ("MSK_RES_ERR_CBF_DUPLICATE_ACOORD")

Duplicate index in ACOORD.

rescode.err_cbf_duplicate_bcoord ("MSK_RES_ERR_CBF_DUPLICATE_BCOORD")

Duplicate index in BCOORD.

rescode.err_cbf_duplicate_con ("MSK_RES_ERR_CBF_DUPLICATE_CON")

Duplicate CON keyword.

rescode.err_cbf_duplicate_int ("MSK_RES_ERR_CBF_DUPLICATE_INT")

Duplicate INT keyword.

rescode.err_cbf_duplicate_obj ("MSK_RES_ERR_CBF_DUPLICATE_OBJ")

Duplicate OBJ keyword.

rescode.err_cbf_duplicate_objacoord ("MSK_RES_ERR_CBF_DUPLICATE_OBJACOORD")

Duplicate index in OBJCOORD.

rescode.err_cbf_duplicate_var ("MSK_RES_ERR_CBF_DUPLICATE_VAR")

Duplicate VAR keyword.

rescode.err_cbf_invalid_con_type ("MSK_RES_ERR_CBF_INVALID_CON_TYPE")

Invalid constraint type.

rescode.err_cbf_invalid_domain_dimension ("MSK_RES_ERR_CBF_INVALID_DOMAIN_DIMENSION")

Invalid domain dimension.

rescode.err_cbf_invalid_int_index ("MSK_RES_ERR_CBF_INVALID_INT_INDEX")

Invalid INT index.

rescode.err_cbf_invalid_var_type ("MSK_RES_ERR_CBF_INVALID_VAR_TYPE")

Invalid variable type.

rescode.err_cbf_no_variables ("MSK_RES_ERR_CBF_NO_VARIABLES")

No variables are specified.

rescode.err_cbf_no_version_specified ("MSK_RES_ERR_CBF_NO_VERSION_SPECIFIED")

No version specified.

rescode.err_cbf_obj_sense ("MSK_RES_ERR_CBF_OBJ_SENSE")

An invalid objective sense is specified.

rescode.err_cbf_parse ("MSK_RES_ERR_CBF_PARSE")

An error occurred while parsing an CBF file.

rescode.err_cbf_syntax ("MSK_RES_ERR_CBF_SYNTAX")

Invalid syntax.

rescode.err_cbf_too_few_constraints ("MSK_RES_ERR_CBF_TOO_FEW_CONSTRAINTS")

Too few constraints defined.

rescode.err_cbf_too_few_ints ("MSK_RES_ERR_CBF_TOO_FEW_INTS")

Too few ints are specified.

rescode.err_cbf_too_few_variables ("MSK_RES_ERR_CBF_TOO_FEW_VARIABLES")

Too few variables defined.

rescode.err_cbf_too_many_constraints ("MSK_RES_ERR_CBF_TOO_MANY_CONSTRAINTS")

Too many constraints specified.

rescode.err_cbf_too_many_ints ("MSK_RES_ERR_CBF_TOO_MANY_INTS")

Too many ints are specified.

rescode.err_cbf_too_many_variables ("MSK_RES_ERR_CBF_TOO_MANY_VARIABLES")

Too many variables specified.

rescode.err_cbf_unsupported ("MSK_RES_ERR_CBF_UNSUPPORTED")

Unsupported feature is present.

rescode.err_con_q_not_nsd ("MSK_RES_ERR_CON_Q_NOT_NSD")

The quadratic constraint matrix is not negative semidefinite as expected for a constraint with finite lower bound. This results in a nonconvex problem. The parameter dparam.check_convexity_rel_tol can be used to relax the convexity check.

rescode.err_con_q_not_psd ("MSK_RES_ERR_CON_Q_NOT_PSD")

The quadratic constraint matrix is not positive semidefinite as expected for a constraint with finite upper bound. This results in a nonconvex problem. The parameter dparam.check_convexity_rel_tol can be used to relax the convexity check.

rescode.err_concurrent_optimizer ("MSK_RES_ERR_CONCURRENT_OPTIMIZER")

An unsupported optimizer was chosen for use with the concurrent optimizer.

rescode.err_cone_index ("MSK_RES_ERR_CONE_INDEX")

An index of a non-existing cone has been specified.

rescode.err_cone_overlap ("MSK_RES_ERR_CONE_OVERLAP")

A new cone which variables overlap with an existing cone has been specified.

rescode.err_cone_overlap_append ("MSK_RES_ERR_CONE_OVERLAP_APPEND")

The cone to be appended has one variable which is already memeber of another cone.

rescode.err_cone_rep_var ("MSK_RES_ERR_CONE_REP_VAR")

A variable is included multiple times in the cone.

rescode.err_cone_size ("MSK_RES_ERR_CONE_SIZE")

A cone with too few members is specified.

rescode.err_cone_type ("MSK_RES_ERR_CONE_TYPE")

Invalid cone type specified.

rescode.err_cone_type_str ("MSK_RES_ERR_CONE_TYPE_STR")

Invalid cone type specified.

rescode.err_data_file_ext ("MSK_RES_ERR_DATA_FILE_EXT")

The data file format cannot be determined from the file name.

rescode.err_dup_name ("MSK_RES_ERR_DUP_NAME")

The same name was used multiple times for the same problem item type.

rescode.err_duplicate_barvariable_names ("MSK_RES_ERR_DUPLICATE_BARVARIABLE_NAMES")

Two barvariable names are identical.

rescode.err_duplicate_cone_names ("MSK_RES_ERR_DUPLICATE_CONE_NAMES")

Two cone names are identical.

rescode.err_duplicate_constraint_names ("MSK_RES_ERR_DUPLICATE_CONSTRAINT_NAMES")

Two constraint names are identical.

rescode.err_duplicate_variable_names ("MSK_RES_ERR_DUPLICATE_VARIABLE_NAMES")

Two variable names are identical.

rescode.err_end_of_file ("MSK_RES_ERR_END_OF_FILE")

End of file reached.

rescode.err_factor ("MSK_RES_ERR_FACTOR")

An error occurred while factorizing a matrix.

rescode.err_feasrepair_cannot_relax ("MSK_RES_ERR_FEASREPAIR_CANNOT_RELAX")

An optimization problem cannot be relaxed. This is the case e.g. for general nonlinear optimization problems.

rescode.err_feasrepair_inconsistent_bound ("MSK_RES_ERR_FEASREPAIR_INCONSISTENT_BOUND")

The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.

rescode.err_feasrepair_solving_relaxed ("MSK_RES_ERR_FEASREPAIR_SOLVING_RELAXED")

The relaxed problem could not be solved to optimality. Please consult the log file for further details.

rescode.err_file_license ("MSK_RES_ERR_FILE_LICENSE")

Invalid license file.

rescode.err_file_open ("MSK_RES_ERR_FILE_OPEN")

Error while opening a file.

rescode.err_file_read ("MSK_RES_ERR_FILE_READ")

File read error.

rescode.err_file_write ("MSK_RES_ERR_FILE_WRITE")

File write error.

rescode.err_first ("MSK_RES_ERR_FIRST")

Invalid `first`.

rescode.err_firsti ("MSK_RES_ERR_FIRSTI")

Invalid `firsti`.

rescode.err_firstj ("MSK_RES_ERR_FIRSTJ")

Invalid `firstj`.

rescode.err_fixed_bound_values ("MSK_RES_ERR_FIXED_BOUND_VALUES")

A fixed constraint/variable has been specified using the bound keys but the numerical value of the lower and upper bound is different.

rescode.err_flexlm ("MSK_RES_ERR_FLEXLM")

The FLEXlm license manager reported an error.

rescode.err_global_inv_conic_problem ("MSK_RES_ERR_GLOBAL_INV_CONIC_PROBLEM")

The global optimizer can only be applied to problems without semidefinite variables.

rescode.err_huge_aij ("MSK_RES_ERR_HUGE_AIJ")

A numerically huge value is specified for an $a_{i,j}$ element in A. The parameter `dparam.data_tol_aij_huge` controls when an $a_{i,j}$ is considered huge.

rescode.err_huge_c ("MSK_RES_ERR_HUGE_C")

A huge value in absolute size is specified for one c_j.

rescode.err_identical_tasks ("MSK_RES_ERR_IDENTICAL_TASKS")

Some tasks related to this function call were identical. Unique tasks were expected.

rescode.err_in_argument ("MSK_RES_ERR_IN_ARGUMENT")

A function argument is incorrect.

rescode.err_index ("MSK_RES_ERR_INDEX")

An index is out of range.

rescode.err_index_arr_is_too_large ("MSK_RES_ERR_INDEX_ARR_IS_TOO_LARGE")

An index in an array argument is too large.

rescode.err_index_arr_is_too_small ("MSK_RES_ERR_INDEX_ARR_IS_TOO_SMALL")

An index in an array argument is too small.

rescode.err_index_is_too_large ("MSK_RES_ERR_INDEX_IS_TOO_LARGE")

An index in an argument is too large.

rescode.err_index_is_too_small ("MSK_RES_ERR_INDEX_IS_TOO_SMALL")

An index in an argument is too small.

rescode.err_inf_dou_index ("MSK_RES_ERR_INF_DOU_INDEX")

A double information index is out of range for the specified type.

rescode.err_inf_dou_name ("MSK_RES_ERR_INF_DOU_NAME")

A double information name is invalid.

rescode.err_inf_int_index ("MSK_RES_ERR_INF_INT_INDEX")

An integer information index is out of range for the specified type.

rescode.err_inf_int_name ("MSK_RES_ERR_INF_INT_NAME")

An integer information name is invalid.

rescode.err_inf_lint_index ("MSK_RES_ERR_INF_LINT_INDEX")

A long integer information index is out of range for the specified type.

rescode.err_inf_lint_name ("MSK_RES_ERR_INF_LINT_NAME")

A long integer information name is invalid.

rescode.err_inf_type ("MSK_RES_ERR_INF_TYPE")

The information type is invalid.

rescode.err_infeas_undefined ("MSK_RES_ERR_INFEAS_UNDEFINED")

The requested value is not defined for this solution type.

rescode.err_infinite_bound ("MSK_RES_ERR_INFINITE_BOUND")

A numerically huge bound value is specified.

rescode.err_int64_to_int32_cast ("MSK_RES_ERR_INT64_TO_INT32_CAST")

An 32 bit integer could not cast to a 64 bit integer.

rescode.err_internal ("MSK_RES_ERR_INTERNAL")

An internal error occurred. Please report this problem.

rescode.err_internal_test_failed ("MSK_RES_ERR_INTERNAL_TEST_FAILED")

An internal unit test function failed.

rescode.err_inv_aptre ("MSK_RES_ERR_INV_APTRE")

`aptre[j]` is strictly smaller than `aptrb[j]` for some `j`.

rescode.err_inv_bk ("MSK_RES_ERR_INV_BK")

Invalid bound key.

rescode.err_inv_bkc ("MSK_RES_ERR_INV_BKC")

Invalid bound key is specified for a constraint.

rescode.err_inv_bkx ("MSK_RES_ERR_INV_BKX")

An invalid bound key is specified for a variable.

rescode.err_inv_cone_type ("MSK_RES_ERR_INV_CONE_TYPE")

Invalid cone type code is encountered.

rescode.err_inv_cone_type_str ("MSK_RES_ERR_INV_CONE_TYPE_STR")

Invalid cone type string encountered.

rescode.err_inv_conic_problem ("MSK_RES_ERR_INV_CONIC_PROBLEM")

The conic optimizer can only be applied to problems with linear objective and constraints. Many problems such convex quadratically constrained problems can easily be reformulated to conic problems. See the appropriate MOSEK manual for details.

rescode.err_inv_marki ("MSK_RES_ERR_INV_MARKI")

Invalid value in marki.

rescode.err_inv_markj ("MSK_RES_ERR_INV_MARKJ")

Invalid value in markj.

rescode.err_inv_name_item ("MSK_RES_ERR_INV_NAME_ITEM")

An invalid name item code is used.

rescode.err_inv_numi ("MSK_RES_ERR_INV_NUMI")

Invalid numi.

rescode.err_inv_numj ("MSK_RES_ERR_INV_NUMJ")

Invalid numj.

rescode.err_inv_optimizer ("MSK_RES_ERR_INV_OPTIMIZER")

An invalid optimizer has been chosen for the problem. This means that the simplex or the conic optimizer is chosen to optimize a nonlinear problem.

rescode.err_inv_problem ("MSK_RES_ERR_INV_PROBLEM")

Invalid problem type. Probably a nonconvex problem has been specified.

rescode.err_inv_qcon_subi ("MSK_RES_ERR_INV_QCON_SUBI")

Invalid value in `qcsubi`.

rescode.err_inv_qcon_subj ("MSK_RES_ERR_INV_QCON_SUBJ")

Invalid value in `qcsubj`.

rescode.err_inv_qcon_subk ("MSK_RES_ERR_INV_QCON_SUBK")

Invalid value in `qcsubk`.

rescode.err_inv_qcon_val ("MSK_RES_ERR_INV_QCON_VAL")

Invalid value in `qcval`.

rescode.err_inv_qobj_subi ("MSK_RES_ERR_INV_QOBJ_SUBI")

Invalid value in `qosubi`.

rescode.err_inv_qobj_subj ("MSK_RES_ERR_INV_QOBJ_SUBJ")

Invalid value in `qosubj`.

rescode.err_inv_qobj_val ("MSK_RES_ERR_INV_QOBJ_VAL")

Invalid value in qoval.

rescode.err_inv_sk ("MSK_RES_ERR_INV_SK")

Invalid status key code.

rescode.err_inv_sk_str ("MSK_RES_ERR_INV_SK_STR")

Invalid status key string encountered.

rescode.err_inv_skc ("MSK_RES_ERR_INV_SKC")

Invalid value in skc.

rescode.err_inv_skn ("MSK_RES_ERR_INV_SKN")

Invalid value in skn.

rescode.err_inv_skx ("MSK_RES_ERR_INV_SKX")

Invalid value in skx.

rescode.err_inv_var_type ("MSK_RES_ERR_INV_VAR_TYPE")

An invalid variable type is specified for a variable.

rescode.err_invalid_accmode ("MSK_RES_ERR_INVALID_ACCMODE")

An invalid access mode is specified.

rescode.err_invalid_aij ("MSK_RES_ERR_INVALID_AIJ")

$a_{i,j}$ contains an invalid floating point value, i.e. a NaN or an infinite value.

rescode.err_invalid_ampl_stub ("MSK_RES_ERR_INVALID_AMPL_STUB")

Invalid AMPL stub.

rescode.err_invalid_barvar_name ("MSK_RES_ERR_INVALID_BARVAR_NAME")

An invalid symmetric matrix variable name is used.

rescode.err_invalid_branch_direction ("MSK_RES_ERR_INVALID_BRANCH_DIRECTION")

An invalid branching direction is specified.

rescode.err_invalid_branch_priority ("MSK_RES_ERR_INVALID_BRANCH_PRIORITY")

An invalid branching priority is specified. It should be nonnegative.

rescode.err_invalid_compression ("MSK_RES_ERR_INVALID_COMPRESSION")

Invalid compression type.

rescode.err_invalid_con_name ("MSK_RES_ERR_INVALID_CON_NAME")

An invalid constraint name is used.

rescode.err_invalid_cone_name ("MSK_RES_ERR_INVALID_CONE_NAME")

An invalid cone name is used.

rescode.err_invalid_file_format_for_cones ("`MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_CONES`")

The file format does not support a problem with conic constraints.

rescode.err_invalid_file_format_for_general_nl ("`MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_GENERAL_NL`")

The file format does not support a problem with general nonlinear terms.

rescode.err_invalid_file_format_for_sym_mat ("`MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_SYM_MAT`")

The file format does not support a problem with symmetric matrix variables.

rescode.err_invalid_file_name ("`MSK_RES_ERR_INVALID_FILE_NAME`")

An invalid file name has been specified.

rescode.err_invalid_format_type ("`MSK_RES_ERR_INVALID_FORMAT_TYPE`")

Invalid format type.

rescode.err_invalid_idx ("`MSK_RES_ERR_INVALID_IDX`")

A specified index is invalid.

rescode.err_invalid_iomode ("`MSK_RES_ERR_INVALID_IOMODE`")

Invalid io mode.

rescode.err_invalid_max_num ("`MSK_RES_ERR_INVALID_MAX_NUM`")

A specified index is invalid.

rescode.err_invalid_name_in_sol_file ("`MSK_RES_ERR_INVALID_NAME_IN_SOL_FILE`")

An invalid name occurred in a solution file.

rescode.err_invalid_network_problem ("`MSK_RES_ERR_INVALID_NETWORK_PROBLEM`")

The problem is not a network problem as expected. The error occurs if a network optimizer is applied to a problem that cannot (easily) be converted to a network problem.

rescode.err_invalid_obj_name ("`MSK_RES_ERR_INVALID_OBJ_NAME`")

An invalid objective name is specified.

rescode.err_invalid_objective_sense ("`MSK_RES_ERR_INVALID_OBJECTIVE_SENSE`")

An invalid objective sense is specified.

rescode.err_invalid_problem_type ("`MSK_RES_ERR_INVALID_PROBLEM_TYPE`")

An invalid problem type.

rescode.err_invalid_sol_file_name ("`MSK_RES_ERR_INVALID_SOL_FILE_NAME`")

An invalid file name has been specified.

rescode.err_invalid_stream ("`MSK_RES_ERR_INVALID_STREAM`")

An invalid stream is referenced.

rescode.err_invalid_surplus ("MSK_RES_ERR_INVALID_SURPLUS")

Invalid surplus.

rescode.err_invalid_sym_mat_dim ("MSK_RES_ERR_INVALID_SYM_MAT_DIM")

A sparse symmetric matrix of invalid dimension is specified.

rescode.err_invalid_task ("MSK_RES_ERR_INVALID_TASK")

The `task` is invalid.

rescode.err_invalid_utf8 ("MSK_RES_ERR_INVALID_UTF8")

An invalid UTF8 string is encountered.

rescode.err_invalid_var_name ("MSK_RES_ERR_INVALID_VAR_NAME")

An invalid variable name is used.

rescode.err_invalid_wchar ("MSK_RES_ERR_INVALID_WCHAR")

An invalid `wchar` string is encountered.

rescode.err_invalid_whichsol ("MSK_RES_ERR_INVALID_WHICHSOL")

`whichsol` is invalid.

rescode.err_last ("MSK_RES_ERR_LAST")

Invalid index `last`. A given index was out of expected range.

rescode.err_lasti ("MSK_RES_ERR_LASTI")

Invalid `lasti`.

rescode.err_lastj ("MSK_RES_ERR_LASTJ")

Invalid `lastj`.

rescode.err_lau_arg_k ("MSK_RES_ERR_LAU_ARG_K")

Invalid argument k.

rescode.err_lau_arg_m ("MSK_RES_ERR_LAU_ARG_M")

Invalid argument m.

rescode.err_lau_arg_n ("MSK_RES_ERR_LAU_ARG_N")

Invalid argument n.

rescode.err_lau_arg_trans ("MSK_RES_ERR_LAU_ARG_TRANS")

Invalid argument trans.

rescode.err_lau_arg_transa ("MSK_RES_ERR_LAU_ARG_TRANSA")

Invalid argument transa.

rescode.err_lau_arg_transb ("MSK_RES_ERR_LAU_ARG_TRANSB")

Invalid argument transb.

rescode.err_lau_arg_uplo ("MSK_RES_ERR_LAU_ARG_UPLO")

Invalid argument uplo.

rescode.err_lau_singular_matrix ("MSK_RES_ERR_LAU_SINGULAR_MATRIX")

A matrix is singular.

rescode.err_lau_unknown ("MSK_RES_ERR_LAU_UNKNOWN")

An unknown error.

rescode.err_license ("MSK_RES_ERR_LICENSE")

Invalid license.

rescode.err_license_cannot_allocate ("MSK_RES_ERR_LICENSE_CANNOT_ALLOCATE")

The license system cannot allocate the memory required.

rescode.err_license_cannot_connect ("MSK_RES_ERR_LICENSE_CANNOT_CONNECT")

MOSEK cannot connect to the license server. Most likely the license server is not up and running.

rescode.err_license_expired ("MSK_RES_ERR_LICENSE_EXPIRED")

The license has expired.

rescode.err_license_feature ("MSK_RES_ERR_LICENSE_FEATURE")

A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.

rescode.err_license_invalid_hostid ("MSK_RES_ERR_LICENSE_INVALID_HOSTID")

The host ID specified in the license file does not match the host ID of the computer.

rescode.err_license_max ("MSK_RES_ERR_LICENSE_MAX")

Maximum number of licenses is reached.

rescode.err_license_moseklm_daemon ("MSK_RES_ERR_LICENSE_MOSEKLM_DAEMON")

The MOSEKLM license manager daemon is not up and running.

rescode.err_license_no_server_line ("MSK_RES_ERR_LICENSE_NO_SERVER_LINE")

There is no SERVER line in the license file. All non-zero license count features need at least one SERVER line.

rescode.err_license_no_server_support ("MSK_RES_ERR_LICENSE_NO_SERVER_SUPPORT")

The license server does not support the requested feature. Possible reasons for this error include:

- The feature has expired.
- The feature's start date is later than today's date.
- The version requested is higher than feature's the highest supported version.
- A corrupted license file.

Try restarting the license and inspect the license server debug file, usually called `lmgrd.log`.

rescode.err_license_server ("`MSK_RES_ERR_LICENSE_SERVER`")

The license server is not responding.

rescode.err_license_server_version ("`MSK_RES_ERR_LICENSE_SERVER_VERSION`")

The version specified in the checkout request is greater than the highest version number the daemon supports.

rescode.err_license_version ("`MSK_RES_ERR_LICENSE_VERSION`")

The license is valid for another version of MOSEK.

rescode.err_link_file_dll ("`MSK_RES_ERR_LINK_FILE_DLL`")

A file cannot be linked to a stream in the DLL version.

rescode.err_living_tasks ("`MSK_RES_ERR_LIVING_TASKS`")

All tasks associated with an enviroment must be deleted before the environment is deleted. There are still some undeleted tasks.

rescode.err_lower_bound_is_a_nan ("`MSK_RES_ERR_LOWER_BOUND_IS_A_NAN`")

The lower bound specified is not a number (nan).

rescode.err_lp_dup_slack_name ("`MSK_RES_ERR_LP_DUP_SLACK_NAME`")

The name of the slack variable added to a ranged constraint already exists.

rescode.err_lp_empty ("`MSK_RES_ERR_LP_EMPTY`")

The problem cannot be written to an LP formatted file.

rescode.err_lp_file_format ("`MSK_RES_ERR_LP_FILE_FORMAT`")

Syntax error in an LP file.

rescode.err_lp_format ("`MSK_RES_ERR_LP_FORMAT`")

Syntax error in an LP file.

rescode.err_lp_free_constraint ("`MSK_RES_ERR_LP_FREE_CONSTRAINT`")

Free constraints cannot be written in LP file format.

rescode.err_lp_incompatible ("`MSK_RES_ERR_LP_INCOMPATIBLE`")

The problem cannot be written to an LP formatted file.

rescode.err_lp_invalid_con_name ("`MSK_RES_ERR_LP_INVALID_CON_NAME`")

A constraint name is invalid when used in an LP formatted file.

rescode.err_lp_invalid_var_name ("`MSK_RES_ERR_LP_INVALID_VAR_NAME`")

A variable name is invalid when used in an LP formatted file.

rescode.err_lp_write_conic_problem ("MSK_RES_ERR_LP_WRITE_CONIC_PROBLEM")

The problem contains cones that cannot be written to an LP formatted file.

rescode.err_lp_write_geco_problem ("MSK_RES_ERR_LP_WRITE_GECO_PROBLEM")

The problem contains general convex terms that cannot be written to an LP formatted file.

rescode.err_lu_max_num_tries ("MSK_RES_ERR_LU_MAX_NUM_TRIES")

Could not compute the LU factors of the matrix within the maximum number of allowed tries.

rescode.err_max_len_is_too_small ("MSK_RES_ERR_MAX_LEN_IS_TOO_SMALL")

An maximum length that is too small has been specfified.

rescode.err_maxnumbarvar ("MSK_RES_ERR_MAXNUMBARVAR")

The maximum number of semidefinite variables specified is smaller than the number of semidefinite variables in the task.

rescode.err_maxnumcon ("MSK_RES_ERR_MAXNUMCON")

The maximum number of constraints specified is smaller than the number of constraints in the task.

rescode.err_maxnumcone ("MSK_RES_ERR_MAXNUMCONE")

The value specified for `maxnumcone` is too small.

rescode.err_maxnumqnz ("MSK_RES_ERR_MAXNUMQNZ")

The maximum number of non-zeros specified for the Q matrixes is smaller than the number of non-zeros in the current Q matrixes.

rescode.err_maxnumvar ("MSK_RES_ERR_MAXNUMVAR")

The maximum number of variables specified is smaller than the number of variables in the task.

rescode.err_mbt_incompatible ("MSK_RES_ERR_MBT_INCOMPATIBLE")

The MBT file is incompatible with this platform. This results from reading a file on a 32 bit platform generated on a 64 bit platform.

rescode.err_mbt_invalid ("MSK_RES_ERR_MBT_INVALID")

The MBT file is invalid.

rescode.err_mio_internal ("MSK_RES_ERR_MIO_INTERNAL")

A fatal error occurred in the mixed integer optimizer. Please contact MOSEK support.

rescode.err_mio_invalid_node_optimizer ("MSK_RES_ERR_MIO_INVALID_NODE_OPTIMIZER")

An invalid node optimizer was selected for the problem type.

rescode.err_mio_invalid_root_optimizer ("MSK_RES_ERR_MIO_INVALID_ROOT_OPTIMIZER")

An invalid root optimizer was selected for the problem type.

rescode.err_mio_no_optimizer ("MSK_RES_ERR_MIO_NO_OPTIMIZER")

No optimizer is available for the current class of integer optimization problems.

rescode.err_mio_not_loaded ("MSK_RES_ERR_MIO_NOT_LOADED")

The mixed-integer optimizer is not loaded.

rescode.err_missing_license_file ("MSK_RES_ERR_MISSING_LICENSE_FILE")

MOSEK cannot license file or a token server. See the MOSEK installation manual for details.

rescode.err_mixed_problem ("MSK_RES_ERR_MIXED_PROBLEM")

The problem contains both conic and nonlinear constraints.

rescode.err_mps_cone_overlap ("MSK_RES_ERR_MPS_CONE_OVERLAP")

A variable is specified to be a member of several cones.

rescode.err_mps_cone_repeat ("MSK_RES_ERR_MPS_CONE_REPEAT")

A variable is repeated within the CSECTION.

rescode.err_mps_cone_type ("MSK_RES_ERR_MPS_CONE_TYPE")

Invalid cone type specified in a CSECTION.

rescode.err_mps_duplicate_q_element ("MSK_RES_ERR_MPS_DUPLICATE_Q_ELEMENT")

Duplicate elements is specfied in a Q matrix.

rescode.err_mps_file ("MSK_RES_ERR_MPS_FILE")

An error occurred while reading an MPS file.

rescode.err_mps_inv_bound_key ("MSK_RES_ERR_MPS_INV_BOUND_KEY")

An invalid bound key occurred in an MPS file.

rescode.err_mps_inv_con_key ("MSK_RES_ERR_MPS_INV_CON_KEY")

An invalid constraint key occurred in an MPS file.

rescode.err_mps_inv_field ("MSK_RES_ERR_MPS_INV_FIELD")

A field in the MPS file is invalid. Probably it is too wide.

rescode.err_mps_inv_marker ("MSK_RES_ERR_MPS_INV_MARKER")

An invalid marker has been specified in the MPS file.

rescode.err_mps_inv_sec_name ("MSK_RES_ERR_MPS_INV_SEC_NAME")

An invalid section name occurred in an MPS file.

rescode.err_mps_inv_sec_order ("MSK_RES_ERR_MPS_INV_SEC_ORDER")

The sections in the MPS data file are not in the correct order.

rescode.err_mps_invalid_obj_name ("MSK_RES_ERR_MPS_INVALID_OBJ_NAME")

An invalid objective name is specified.

rescode.err_mps_invalid_objsense ("MSK_RES_ERR_MPS_INVALID_OBJSENSE")

An invalid objective sense is specified.

rescode.err_mps_mul_con_name ("MSK_RES_ERR_MPS_MUL_CON_NAME")

A constraint name was specified multiple times in the ROWS section.

rescode.err_mps_mul_csec ("MSK_RES_ERR_MPS_MUL_CSEC")

Multiple CSECTIONs are given the same name.

rescode.err_mps_mul_qobj ("MSK_RES_ERR_MPS_MUL_QOBJ")

The Q term in the objective is specified multiple times in the MPS data file.

rescode.err_mps_mul_qsec ("MSK_RES_ERR_MPS_MUL_QSEC")

Multiple QSECTIONs are specified for a constraint in the MPS data file.

rescode.err_mps_no_objective ("MSK_RES_ERR_MPS_NO_OBJECTIVE")

No objective is defined in an MPS file.

rescode.err_mps_non_symmetric_q ("MSK_RES_ERR_MPS_NON_SYMMETRIC_Q")

A non symmetric matrice has been speciefied.

rescode.err_mps_null_con_name ("MSK_RES_ERR_MPS_NULL_CON_NAME")

An empty constraint name is used in an MPS file.

rescode.err_mps_null_var_name ("MSK_RES_ERR_MPS_NULL_VAR_NAME")

An empty variable name is used in an MPS file.

rescode.err_mps_splitted_var ("MSK_RES_ERR_MPS_SPLITTED_VAR")

All elements in a column of the A matrix must be specified consecutively. Hence, it is illegal to specify non-zero elements in A for variable 1, then for variable 2 and then variable 1 again.

rescode.err_mps_tab_in_field2 ("MSK_RES_ERR_MPS_TAB_IN_FIELD2")

A tab char occurred in field 2.

rescode.err_mps_tab_in_field3 ("MSK_RES_ERR_MPS_TAB_IN_FIELD3")

A tab char occurred in field 3.

rescode.err_mps_tab_in_field5 ("MSK_RES_ERR_MPS_TAB_IN_FIELD5")

A tab char occurred in field 5.

rescode.err_mps_undef_con_name ("MSK_RES_ERR_MPS_UNDEF_CON_NAME")

An undefined constraint name occurred in an MPS file.

rescode.err_mps_undef_var_name ("MSK_RES_ERR_MPS_UNDEF_VAR_NAME")

An undefined variable name occurred in an MPS file.

rescode.err_mul_a_element ("`MSK_RES_ERR_MUL_A_ELEMENT`")

An element in A is defined multiple times.

rescode.err_name_is_null ("`MSK_RES_ERR_NAME_IS_NULL`")

The name buffer is a NULL pointer.

rescode.err_name_max_len ("`MSK_RES_ERR_NAME_MAX_LEN`")

A name is longer than the buffer that is supposed to hold it.

rescode.err_nan_in_blc ("`MSK_RES_ERR_NAN_IN_BLC`")

l^c contains an invalid floating point value, i.e. a `NaN`.

rescode.err_nan_in_blx ("`MSK_RES_ERR_NAN_IN_BLX`")

l^x contains an invalid floating point value, i.e. a `NaN`.

rescode.err_nan_in_buc ("`MSK_RES_ERR_NAN_IN_BUC`")

u^c contains an invalid floating point value, i.e. a `NaN`.

rescode.err_nan_in_bux ("`MSK_RES_ERR_NAN_IN_BUX`")

u^x contains an invalid floating point value, i.e. a `NaN`.

rescode.err_nan_in_c ("`MSK_RES_ERR_NAN_IN_C`")

c contains an invalid floating point value, i.e. a `NaN`.

rescode.err_nan_in_double_data ("`MSK_RES_ERR_NAN_IN_DOUBLE_DATA`")

An invalid floating point value was used in some double data.

rescode.err_negative_append ("`MSK_RES_ERR_NEGATIVE_APPEND`")

Cannot append a negative number.

rescode.err_negative_surplus ("`MSK_RES_ERR_NEGATIVE_SURPLUS`")

Negative surplus.

rescode.err_newer_dll ("`MSK_RES_ERR_NEWER_DLL`")

The dynamic link library is newer than the specified version.

rescode.err_no_bars_for_solution ("`MSK_RES_ERR_NO_BARS_FOR_SOLUTION`")

There is no $\bar{s}$ available for the solution specified. In particular note there are no $\bar{s}$ defined for the basic and integer solutions.

rescode.err_no_barx_for_solution ("`MSK_RES_ERR_NO_BARX_FOR_SOLUTION`")

There is no $\bar{X}$ available for the solution specified. In particular note there are no $\bar{X}$ defined for the basic and integer solutions.

rescode.err_no_basis_sol ("`MSK_RES_ERR_NO_BASIS_SOL`")

No basic solution is defined.

rescode.err_no_dual_for_itg_sol ("MSK_RES_ERR_NO_DUAL_FOR_ITG_SOL")

No dual information is available for the integer solution.

rescode.err_no_dual_infeas_cer ("MSK_RES_ERR_NO_DUAL_INFEAS_CER")

A certificate of infeasibility is not available.

rescode.err_no_dual_info_for_itg_sol ("MSK_RES_ERR_NO_DUAL_INFO_FOR_ITG_SOL")

Dual information is not available for the integer solution.

rescode.err_no_init_env ("MSK_RES_ERR_NO_INIT_ENV")

env is not initialized.

rescode.err_no_optimizer_var_type ("MSK_RES_ERR_NO_OPTIMIZER_VAR_TYPE")

No optimizer is available for this class of optimization problems.

rescode.err_no_primal_infeas_cer ("MSK_RES_ERR_NO_PRIMAL_INFEAS_CER")

A certificate of primal infeasibility is not available.

rescode.err_no_snx_for_bas_sol ("MSK_RES_ERR_NO_SNX_FOR_BAS_SOL")

s_n^x is not available for the basis solution.

rescode.err_no_solution_in_callback ("MSK_RES_ERR_NO_SOLUTION_IN_CALLBACK")

The required solution is not available.

rescode.err_non_unique_array ("MSK_RES_ERR_NON_UNIQUE_ARRAY")

An array does not contain unique elements.

rescode.err_nonconvex ("MSK_RES_ERR_NONCONVEX")

The optimization problem is nonconvex.

rescode.err_nonlinear_equality ("MSK_RES_ERR_NONLINEAR_EQUALITY")

The model contains a nonlinear equality which defines a nonconvex set.

rescode.err_nonlinear_functions_not_allowed ("MSK_RES_ERR_NONLINEAR_FUNCTIONS_NOT_ALLOWED")

An operation that is invalid for problems with nonlinear functions defined has been attempted.

rescode.err_nonlinear_ranged ("MSK_RES_ERR_NONLINEAR_RANGED")

The model contains a nonlinear ranged constraint which by definition defines a nonconvex set.

rescode.err_nr_arguments ("MSK_RES_ERR_NR_ARGUMENTS")

Incorrect number of function arguments.

rescode.err_null_env ("MSK_RES_ERR_NULL_ENV")

env is a NULL pointer.

rescode.err_null_pointer ("MSK_RES_ERR_NULL_POINTER")

An argument to a function is unexpectedly a NULL pointer.

rescode.err_null_task ("MSK_RES_ERR_NULL_TASK")

task is a NULL pointer.

rescode.err_numconlim ("MSK_RES_ERR_NUMCONLIM")

Maximum number of constraints limit is exceeded.

rescode.err_numvarlim ("MSK_RES_ERR_NUMVARLIM")

Maximum number of variables limit is exceeded.

rescode.err_obj_q_not_nsd ("MSK_RES_ERR_OBJ_Q_NOT_NSD")

The quadratic coefficient matrix in the objective is not negative semidefinite as expected for a maximization problem. The parameter dparam.check_convexity_rel_tol can be used to relax the convexity check.

rescode.err_obj_q_not_psd ("MSK_RES_ERR_OBJ_Q_NOT_PSD")

The quadratic coefficient matrix in the objective is not positive semidefinite as expected for a minimization problem. The parameter dparam.check_convexity_rel_tol can be used to relax the convexity check.

rescode.err_objective_range ("MSK_RES_ERR_OBJECTIVE_RANGE")

Empty objective range.

rescode.err_older_dll ("MSK_RES_ERR_OLDER_DLL")

The dynamic link library is older than the specified version.

rescode.err_open_dl ("MSK_RES_ERR_OPEN_DL")

A dynamic link library could not be opened.

rescode.err_opf_format ("MSK_RES_ERR_OPF_FORMAT")

Syntax error in an OPF file

rescode.err_opf_new_variable ("MSK_RES_ERR_OPF_NEW_VARIABLE")

Introducing new variables is now allowed. When a [variables] section is present, it is not allowed to introduce new variables later in the problem.

rescode.err_opf_premature_eof ("MSK_RES_ERR_OPF_PREMATURE_EOF")

Premature end of file in an OPF file.

rescode.err_optimizer_license ("MSK_RES_ERR_OPTIMIZER_LICENSE")

The optimizer required is not licensed.

rescode.err_ord_invalid ("MSK_RES_ERR_ORD_INVALID")

Invalid content in branch ordering file.

rescode.err_ord_invalid_branch_dir ("MSK_RES_ERR_ORD_INVALID_BRANCH_DIR")

An invalid branch direction key is specified.

rescode.err_overflow ("`MSK_RES_ERR_OVERFLOW`")

A computation produced an overflow i.e. a very large number.

rescode.err_param_index ("`MSK_RES_ERR_PARAM_INDEX`")

Parameter index is out of range.

rescode.err_param_is_too_large ("`MSK_RES_ERR_PARAM_IS_TOO_LARGE`")

The parameter value is too large.

rescode.err_param_is_too_small ("`MSK_RES_ERR_PARAM_IS_TOO_SMALL`")

The parameter value is too small.

rescode.err_param_name ("`MSK_RES_ERR_PARAM_NAME`")

The parameter name is not correct.

rescode.err_param_name_dou ("`MSK_RES_ERR_PARAM_NAME_DOU`")

The parameter name is not correct for a double parameter.

rescode.err_param_name_int ("`MSK_RES_ERR_PARAM_NAME_INT`")

The parameter name is not correct for an integer parameter.

rescode.err_param_name_str ("`MSK_RES_ERR_PARAM_NAME_STR`")

The parameter name is not correct for a string parameter.

rescode.err_param_type ("`MSK_RES_ERR_PARAM_TYPE`")

The parameter type is invalid.

rescode.err_param_value_str ("`MSK_RES_ERR_PARAM_VALUE_STR`")

The parameter value string is incorrect.

rescode.err_platform_not_licensed ("`MSK_RES_ERR_PLATFORM_NOT_LICENSED`")

A requested license feature is not available for the required platform.

rescode.err_postsolve ("`MSK_RES_ERR_POSTSOLVE`")

An error occurred during the postsolve. Please contact MOSEK support.

rescode.err_pro_item ("`MSK_RES_ERR_PRO_ITEM`")

An invalid problem is used.

rescode.err_prob_license ("`MSK_RES_ERR_PROB_LICENSE`")

The software is not licensed to solve the problem.

rescode.err_qcon_subi_too_large ("`MSK_RES_ERR_QCON_SUBI_TOO_LARGE`")

Invalid value in `qcsubi`.

rescode.err_qcon_subi_too_small ("`MSK_RES_ERR_QCON_SUBI_TOO_SMALL`")

Invalid value in `qcsubi`.

rescode.err_qcon_upper_triangle ("MSK_RES_ERR_QCON_UPPER_TRIANGLE")

An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.

rescode.err_qobj_upper_triangle ("MSK_RES_ERR_QOBJ_UPPER_TRIANGLE")

An element in the upper triangle of Q^o is specified. Only elements in the lower triangle should be specified.

rescode.err_read_format ("MSK_RES_ERR_READ_FORMAT")

The specified format cannot be read.

rescode.err_read_lp_missing_end_tag ("MSK_RES_ERR_READ_LP_MISSING_END_TAG")

Syntax error in LP file. Possibly missing End tag.

rescode.err_read_lp_nonexisting_name ("MSK_RES_ERR_READ_LP_NONEXISTING_NAME")

A variable never occurred in objective or constraints.

rescode.err_remove_cone_variable ("MSK_RES_ERR_REMOVE_CONE_VARIABLE")

A variable cannot be removed because it will make a cone invalid.

rescode.err_repair_invalid_problem ("MSK_RES_ERR_REPAIR_INVALID_PROBLEM")

The feasibility repair does not support the specified problem type.

rescode.err_repair_optimization_failed ("MSK_RES_ERR_REPAIR_OPTIMIZATION_FAILED")

Computation the optimal relaxation failed. The cause may have been numerical problems.

rescode.err_sen_bound_invalid_lo ("MSK_RES_ERR_SEN_BOUND_INVALID_LO")

Analysis of lower bound requested for an index, where no lower bound exists.

rescode.err_sen_bound_invalid_up ("MSK_RES_ERR_SEN_BOUND_INVALID_UP")

Analysis of upper bound requested for an index, where no upper bound exists.

rescode.err_sen_format ("MSK_RES_ERR_SEN_FORMAT")

Syntax error in sensitivity analysis file.

rescode.err_sen_index_invalid ("MSK_RES_ERR_SEN_INDEX_INVALID")

Invalid range given in the sensitivity file.

rescode.err_sen_index_range ("MSK_RES_ERR_SEN_INDEX_RANGE")

Index out of range in the sensitivity analysis file.

rescode.err_sen_invalid_regexp ("MSK_RES_ERR_SEN_INVALID_REGEXP")

Syntax error in regexp or regexp longer than 1024.

rescode.err_sen_numerical ("MSK_RES_ERR_SEN_NUMERICAL")

Numerical difficulties encountered performing the sensitivity analysis.

rescode.err_sen_solution_status ("`MSK_RES_ERR_SEN_SOLUTION_STATUS`")

No optimal solution found to the original problem given for sensitivity analysis.

rescode.err_sen_undef_name ("`MSK_RES_ERR_SEN_UNDEF_NAME`")

An undefined name was encountered in the sensitivity analysis file.

rescode.err_sen_unhandled_problem_type ("`MSK_RES_ERR_SEN_UNHANDLED_PROBLEM_TYPE`")

Sensitivity analysis cannot be performed for the spcified problem. Sensitivity analysis is only possible for linear problems.

rescode.err_size_license ("`MSK_RES_ERR_SIZE_LICENSE`")

The problem is bigger than the license.

rescode.err_size_license_con ("`MSK_RES_ERR_SIZE_LICENSE_CON`")

The problem has too many constraints to be solved with the available license.

rescode.err_size_license_intvar ("`MSK_RES_ERR_SIZE_LICENSE_INTVAR`")

The problem contains too many integer variables to be solved with the available license.

rescode.err_size_license_numcores ("`MSK_RES_ERR_SIZE_LICENSE_NUMCORES`")

The computer contains more cpu cores than the license allows for.

rescode.err_size_license_var ("`MSK_RES_ERR_SIZE_LICENSE_VAR`")

The problem has too many variables to be solved with the available license.

rescode.err_sol_file_invalid_number ("`MSK_RES_ERR_SOL_FILE_INVALID_NUMBER`")

An invalid number is specified in a solution file.

rescode.err_solitem ("`MSK_RES_ERR_SOLITEM`")

The solution item number `solitem` is invalid. Please note that `solitem.snx` is invalid for the basic solution.

rescode.err_solver_probtype ("`MSK_RES_ERR_SOLVER_PROBTYPE`")

Problem type does not match the chosen optimizer.

rescode.err_space ("`MSK_RES_ERR_SPACE`")

Out of space.

rescode.err_space_leaking ("`MSK_RES_ERR_SPACE_LEAKING`")

MOSEK is leaking memory. This can be due to either an incorrect use of MOSEK or a bug.

rescode.err_space_no_info ("`MSK_RES_ERR_SPACE_NO_INFO`")

No available information about the space usage.

rescode.err_sym_mat_duplicate ("`MSK_RES_ERR_SYM_MAT_DUPLICATE`")

A value in a symmetric matric as been specified more than once.

rescode.err_sym_mat_invalid_col_index ("MSK_RES_ERR_SYM_MAT_INVALID_COL_INDEX")

A column index specified for sparse symmetric maxtrix is invalid.

rescode.err_sym_mat_invalid_row_index ("MSK_RES_ERR_SYM_MAT_INVALID_ROW_INDEX")

A row index specified for sparse symmetric maxtrix is invalid.

rescode.err_sym_mat_invalid_value ("MSK_RES_ERR_SYM_MAT_INVALID_VALUE")

The numerical value specified in a sparse symmetric matrix is not a value floating value.

rescode.err_sym_mat_not_lower_tringular ("MSK_RES_ERR_SYM_MAT_NOT_LOWER_TRINGULAR")

Only the lower triangular part of sparse symmetric matrix should be specified.

rescode.err_task_incompatible ("MSK_RES_ERR_TASK_INCOMPATIBLE")

The Task file is incompatible with this platform. This results from reading a file on a 32 bit platform generated on a 64 bit platform.

rescode.err_task_invalid ("MSK_RES_ERR_TASK_INVALID")

The Task file is invalid.

rescode.err_thread_cond_init ("MSK_RES_ERR_THREAD_COND_INIT")

Could not initialize a condition.

rescode.err_thread_create ("MSK_RES_ERR_THREAD_CREATE")

Could not create a thread. This error may occur if a large number of environments are created and not deleted again. In any case it is a good practice to minimize the number of environments created.

rescode.err_thread_mutex_init ("MSK_RES_ERR_THREAD_MUTEX_INIT")

Could not initialize a mutex.

rescode.err_thread_mutex_lock ("MSK_RES_ERR_THREAD_MUTEX_LOCK")

Could not lock a mutex.

rescode.err_thread_mutex_unlock ("MSK_RES_ERR_THREAD_MUTEX_UNLOCK")

Could not unlock a mutex.

rescode.err_toconic_conversion_fail ("MSK_RES_ERR_TOCONIC_CONVERSION_FAIL")

A constraint could not be converted in conic form.

rescode.err_too_many_concurrent_tasks ("MSK_RES_ERR_TOO_MANY_CONCURRENT_TASKS")

Too many concurrent tasks specified.

rescode.err_too_small_max_num_nz ("MSK_RES_ERR_TOO_SMALL_MAX_NUM_NZ")

The maximum number of non-zeros specified is too small.

rescode.err_too_small_maxnumanz ("MSK_RES_ERR_TOO_SMALL_MAXNUMANZ")

The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A.

rescode.err_unb_step_size ("MSK_RES_ERR_UNB_STEP_SIZE")

A step size in an optimizer was unexpectedly unbounded. For instance, if the step-size becomes unbounded in phase 1 of the simplex algorithm then an error occurs. Normally this will happen only if the problem is badly formulated. Please contact MOSEK support if this error occurs.

rescode.err_undef_solution ("MSK_RES_ERR_UNDEF_SOLUTION")

MOSEK has the following solution types:

- an interior-point solution,
- an basic solution,
- and an integer solution.

Each optimizer may set one or more of these solutions; e.g by default a successful optimization with the interior-point optimizer defines the interior-point solution, and, for linear problems, also the basic solution. This error occurs when asking for a solution or for information about a solution that is not defined.

rescode.err_undefined_objective_sense ("MSK_RES_ERR_UNDEFINED_OBJECTIVE_SENSE")

The objective sense has not been specified before the optimization.

rescode.err_unhandled_solution_status ("MSK_RES_ERR_UNHANDLED_SOLUTION_STATUS")

Unhandled solution status.

rescode.err_unknown ("MSK_RES_ERR_UNKNOWN")

Unknown error.

rescode.err_upper_bound_is_a_nan ("MSK_RES_ERR_UPPER_BOUND_IS_A_NAN")

The upper bound specified is not a number (nan).

rescode.err_upper_triangle ("MSK_RES_ERR_UPPER_TRIANGLE")

An element in the upper triangle of a lower triangular matrix is specified.

rescode.err_user_func_ret ("MSK_RES_ERR_USER_FUNC_RET")

An user function reported an error.

rescode.err_user_func_ret_data ("MSK_RES_ERR_USER_FUNC_RET_DATA")

An user function returned invalid data.

rescode.err_user_nlo_eval ("MSK_RES_ERR_USER_NLO_EVAL")

The user-defined nonlinear function reported an error.

rescode.err_user_nlo_eval_hessubi ("MSK_RES_ERR_USER_NLO_EVAL_HESSUBI")

The user-defined nonlinear function reported an invalid subscript in the Hessian.

rescode.err_user_nlo_eval_hessubj ("MSK_RES_ERR_USER_NLO_EVAL_HESSUBJ")

The user-defined nonlinear function reported an invalid subscript in the Hessian.

rescode.err_user_nlo_func ("MSK_RES_ERR_USER_NLO_FUNC")

The user-defined nonlinear function reported an error.

rescode.err_whichitem_not_allowed ("MSK_RES_ERR_WHICHITEM_NOT_ALLOWED")

`whichitem` is unacceptable.

rescode.err_whichsol ("MSK_RES_ERR_WHICHSOL")

The solution defined by compwhichsol does not exists.

rescode.err_write_lp_format ("MSK_RES_ERR_WRITE_LP_FORMAT")

Problem cannot be written as an LP file.

rescode.err_write_lp_non_unique_name ("MSK_RES_ERR_WRITE_LP_NON_UNIQUE_NAME")

An auto-generated name is not unique.

rescode.err_write_mps_invalid_name ("MSK_RES_ERR_WRITE_MPS_INVALID_NAME")

An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.

rescode.err_write_opf_invalid_var_name ("MSK_RES_ERR_WRITE_OPF_INVALID_VAR_NAME")

Empty variable names cannot be written to OPF files.

rescode.err_writing_file ("MSK_RES_ERR_WRITING_FILE")

An error occurred while writing file

rescode.err_xml_invalid_problem_type ("MSK_RES_ERR_XML_INVALID_PROBLEM_TYPE")

The problem type is not supported by the XML format.

rescode.err_y_is_undefined ("MSK_RES_ERR_Y_IS_UNDEFINED")

The solution item y is undefined.

rescode.ok ("MSK_RES_OK")

No error occurred.

rescode.trm_internal ("MSK_RES_TRM_INTERNAL")

The optimizer terminated due to some internal reason. Please contact MOSEK support.

rescode.trm_internal_stop ("MSK_RES_TRM_INTERNAL_STOP")

The optimizer terminated for internal reasons. Please contact MOSEK support.

rescode.trm_max_iterations ("MSK_RES_TRM_MAX_ITERATIONS")

The optimizer terminated at the maximum number of iterations.

rescode.trm_max_num_setbacks ("MSK_RES_TRM_MAX_NUM_SETBACKS")

The optimizer terminated as the maximum number of set-backs was reached. This indicates numerical problems and a possibly badly formulated problem.

rescode.trm_max_time ("MSK_RES_TRM_MAX_TIME")

The optimizer terminated at the maximum amount of time.

rescode.trm_mio_near_abs_gap ("MSK_RES_TRM_MIO_NEAR_ABS_GAP")

The mixed-integer optimizer terminated because the near optimal absolute gap tolerance was satisfied.

rescode.trm_mio_near_rel_gap ("MSK_RES_TRM_MIO_NEAR_REL_GAP")

The mixed-integer optimizer terminated because the near optimal relative gap tolerance was satisfied.

rescode.trm_mio_num_branches ("MSK_RES_TRM_MIO_NUM_BRANCHES")

The mixed-integer optimizer terminated as to the maximum number of branches was reached.

rescode.trm_mio_num_relaxs ("MSK_RES_TRM_MIO_NUM_RELAXS")

The mixed-integer optimizer terminated as the maximum number of relaxations was reached.

rescode.trm_num_max_num_int_solutions ("MSK_RES_TRM_NUM_MAX_NUM_INT_SOLUTIONS")

The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.

rescode.trm_numerical_problem ("MSK_RES_TRM_NUMERICAL_PROBLEM")

The optimizer terminated due to numerical problems.

rescode.trm_objective_range ("MSK_RES_TRM_OBJECTIVE_RANGE")

The optimizer terminated on the bound of the objective range.

rescode.trm_stall ("MSK_RES_TRM_STALL")

The optimizer is terminated due to slow progress.

Stalling means that numerical problems prevent the optimizer from making reasonable progress and that it make no sense to continue. In many cases this happens if the problem is badly scaled or otherwise ill-conditioned. There is no guarantee that the solution will be (near) feasible or near optimal. However, often stalling happens near the optimum, and the returned solution may be of good quality. Therefore, it is recommended to check the status of then solution. If the solution near optimal the solution is most likely good enough for most practical purposes.

Please note that if a linear optimization problem is solved using the interior-point optimizer with basis identification turned on, the returned basic solution likely to have high accuracy, even though the optimizer stalled.

Some common causes of stalling are a) badly scaled models, b) near feasible or near infeasible problems and c) a non-convex problems. Case c) is only relevant for general non-linear problems. It is not possible in general for MOSEK to check if a specific problems is convex since such a check would be NP hard in itself. This implies that care should be taken when solving problems involving general user defined functions.

rescode.trm_user_callback ("MSK_RES_TRM_USER_CALLBACK")

The optimizer terminated due to the return of the user-defined call-back function.

rescode.wrn_ana_almost_int_bounds ("MSK_RES_WRN_ANA_ALMOST_INT_BOUNDS")

This warning is issued by the problem analyzer if a constraint is bound nearly integral.

rescode.wrn_ana_c_zero ("MSK_RES_WRN_ANA_C_ZERO")

This warning is issued by the problem analyzer, if the coefficients in the linear part of the objective are all zero.

rescode.wrn_ana_close_bounds ("MSK_RES_WRN_ANA_CLOSE_BOUNDS")

This warning is issued by problem analyzer, if ranged constraints or variables with very close upper and lower bounds are detected. One should consider treating such constraints as equalities and such variables as constants.

rescode.wrn_ana_empty_cols ("MSK_RES_WRN_ANA_EMPTY_COLS")

This warning is issued by the problem analyzer, if columns, in which all coefficients are zero, are found.

rescode.wrn_ana_large_bounds ("MSK_RES_WRN_ANA_LARGE_BOUNDS")

This warning is issued by the problem analyzer, if one or more constraint or variable bounds are very large. One should consider omitting these bounds entirely by setting them to +inf or -inf.

rescode.wrn_construct_invalid_sol_itg ("MSK_RES_WRN_CONSTRUCT_INVALID_SOL_ITG")

The intial value for one or more of the integer variables is not feasible.

rescode.wrn_construct_no_sol_itg ("MSK_RES_WRN_CONSTRUCT_NO_SOL_ITG")

The construct solution requires an integer solution.

rescode.wrn_construct_solution_infeas ("MSK_RES_WRN_CONSTRUCT_SOLUTION_INFEAS")

After fixing the integer variables at the suggested values then the problem is infeasible.

rescode.wrn_dropped_nz_qobj ("MSK_RES_WRN_DROPPED_NZ_QOBJ")

One or more non-zero elements were dropped in the Q matrix in the objective.

rescode.wrn_duplicate_barvariable_names ("MSK_RES_WRN_DUPLICATE_BARVARIABLE_NAMES")

Two barvariable names are identical.

rescode.wrn_duplicate_cone_names ("MSK_RES_WRN_DUPLICATE_CONE_NAMES")

Two cone names are identical.

rescode.wrn_duplicate_constraint_names ("MSK_RES_WRN_DUPLICATE_CONSTRAINT_NAMES")

Two constraint names are identical.

rescode.wrn_duplicate_variable_names ("MSK_RES_WRN_DUPLICATE_VARIABLE_NAMES")

Two variable names are identical.

rescode.wrn_eliminator_space ("`MSK_RES_WRN_ELIMINATOR_SPACE`")

The eliminator is skipped at least once due to lack of space.

rescode.wrn_empty_name ("`MSK_RES_WRN_EMPTY_NAME`")

A variable or constraint name is empty. The output file may be invalid.

rescode.wrn_ignore_integer ("`MSK_RES_WRN_IGNORE_INTEGER`")

Ignored integer constraints.

rescode.wrn_incomplete_linear_dependency_check ("`MSK_RES_WRN_INCOMPLETE_LINEAR_DEPENDENCY_CHECK`")

The linear dependency check(s) is not completed. Normally this is not an important warning unless the optimization problem has been formulated with linear dependencies which is bad practice.

rescode.wrn_large_aij ("`MSK_RES_WRN_LARGE_AIJ`")

A numerically large value is specified for an $a_{i,j}$ element in A. The parameter `dparam.data_tol_aij_large` controls when an $a_{i,j}$ is considered large.

rescode.wrn_large_bound ("`MSK_RES_WRN_LARGE_BOUND`")

A numerically large bound value is specified.

rescode.wrn_large_cj ("`MSK_RES_WRN_LARGE_CJ`")

A numerically large value is specified for one c_j.

rescode.wrn_large_con_fx ("`MSK_RES_WRN_LARGE_CON_FX`")

An equality constraint is fixed to a numerically large value. This can cause numerical problems.

rescode.wrn_large_lo_bound ("`MSK_RES_WRN_LARGE_LO_BOUND`")

A numerically large lower bound value is specified.

rescode.wrn_large_up_bound ("`MSK_RES_WRN_LARGE_UP_BOUND`")

A numerically large upper bound value is specified.

rescode.wrn_license_expire ("`MSK_RES_WRN_LICENSE_EXPIRE`")

The license expires.

rescode.wrn_license_feature_expire ("`MSK_RES_WRN_LICENSE_FEATURE_EXPIRE`")

The license expires.

rescode.wrn_license_server ("`MSK_RES_WRN_LICENSE_SERVER`")

The license server is not responding.

rescode.wrn_lp_drop_variable ("`MSK_RES_WRN_LP_DROP_VARIABLE`")

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

rescode.wrn_lp_old_quad_format ("MSK_RES_WRN_LP_OLD_QUAD_FORMAT")

Missing '/2' after quadratic expressions in bound or objective.

rescode.wrn_mio_infeasible_final ("MSK_RES_WRN_MIO_INFEASIBLE_FINAL")

The final mixed-integer problem with all the integer variables fixed at their optimal values is infeasible.

rescode.wrn_mps_split_bou_vector ("MSK_RES_WRN_MPS_SPLIT_BOU_VECTOR")

A BOUNDS vector is split into several nonadjacent parts in an MPS file.

rescode.wrn_mps_split_ran_vector ("MSK_RES_WRN_MPS_SPLIT_RAN_VECTOR")

A RANGE vector is split into several nonadjacent parts in an MPS file.

rescode.wrn_mps_split_rhs_vector ("MSK_RES_WRN_MPS_SPLIT_RHS_VECTOR")

An RHS vector is split into several nonadjacent parts in an MPS file.

rescode.wrn_name_max_len ("MSK_RES_WRN_NAME_MAX_LEN")

A name is longer than the buffer that is supposed to hold it.

rescode.wrn_no_dualizer ("MSK_RES_WRN_NO_DUALIZER")

No automatic dualizer is available for the specified problem. The primal problem is solved.

rescode.wrn_no_global_optimizer ("MSK_RES_WRN_NO_GLOBAL_OPTIMIZER")

No global optimizer is available.

rescode.wrn_no_nonlinear_function_write ("MSK_RES_WRN_NO_NONLINEAR_FUNCTION_WRITE")

The problem contains a general nonlinear function in either the objective or the constraints. Such a nonlinear function cannot be written to a disk file. Note that quadratic terms when inputted explicitly can be written to disk.

rescode.wrn_nz_in_upr_tri ("MSK_RES_WRN_NZ_IN_UPR_TRI")

Non-zero elements specified in the upper triangle of a matrix were ignored.

rescode.wrn_open_param_file ("MSK_RES_WRN_OPEN_PARAM_FILE")

The parameter file could not be opened.

rescode.wrn_param_ignored_cmio ("MSK_RES_WRN_PARAM_IGNORED_CMIO")

A parameter was ignored by the conic mixed integer optimizer.

rescode.wrn_param_name_dou ("MSK_RES_WRN_PARAM_NAME_DOU")

The parameter name is not recognized as a double parameter.

rescode.wrn_param_name_int ("MSK_RES_WRN_PARAM_NAME_INT")

The parameter name is not recognized as a integer parameter.

rescode.wrn_param_name_str ("MSK_RES_WRN_PARAM_NAME_STR")

The parameter name is not recognized as a string parameter.

rescode.wrn_param_str_value ("MSK_RES_WRN_PARAM_STR_VALUE")

The string is not recognized as a symbolic value for the parameter.

rescode.wrn_presolve_outofspace ("MSK_RES_WRN_PRESOLVE_OUTOFSPACE")

The presolve is incomplete due to lack of space.

rescode.wrn_quad_cones_with_root_fixed_at_zero ("MSK_RES_WRN_QUAD_CONES_WITH_ROOT_FIXED_AT_ZERO")

For at least one quadratic cone the root is fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problems, or to fix all the variables in the cone to 0.

rescode.wrn_rquad_cones_with_root_fixed_at_zero ("MSK_RES_WRN_RQUAD_CONES_WITH_ROOT_FIXED_AT_ZERO")

For at least one rotated quadratic cone at least one of the root variables are fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problems, or to fix all the variables in the cone to 0.

rescode.wrn_sol_file_ignored_con ("MSK_RES_WRN_SOL_FILE_IGNORED_CON")

One or more lines in the constraint section were ignored when reading a solution file.

rescode.wrn_sol_file_ignored_var ("MSK_RES_WRN_SOL_FILE_IGNORED_VAR")

One or more lines in the variable section were ignored when reading a solution file.

rescode.wrn_sol_filter ("MSK_RES_WRN_SOL_FILTER")

Invalid solution filter is specified.

rescode.wrn_spar_max_len ("MSK_RES_WRN_SPAR_MAX_LEN")

A value for a string parameter is longer than the buffer that is supposed to hold it.

rescode.wrn_too_few_basis_vars ("MSK_RES_WRN_TOO_FEW_BASIS_VARS")

An incomplete basis has been specified. Too few basis variables are specified.

rescode.wrn_too_many_basis_vars ("MSK_RES_WRN_TOO_MANY_BASIS_VARS")

A basis with too many variables has been specified.

rescode.wrn_too_many_threads_concurrent ("MSK_RES_WRN_TOO_MANY_THREADS_CONCURRENT")

The concurrent optimizer employs more threads than available. This will lead to poor performance.

rescode.wrn_undef_sol_file_name ("MSK_RES_WRN_UNDEF_SOL_FILE_NAME")

Undefined name occurred in a solution.

rescode.wrn_using_generic_names ("MSK_RES_WRN_USING_GENERIC_NAMES")

Generic names are used because a name is not valid. For instance when writing an LP file the names must not contain blanks or start with a digit.

rescode.wrn_write_changed_names ("MSK_RES_WRN_WRITE_CHANGED_NAMES")

Some names were changed because they were invalid for the output file format.

rescode.wrn_write_discarded_cfix ("MSK_RES_WRN_WRITE_DISCARDED_CFIX")

The fixed objective term could not be converted to a variable and was discarded in the output file.

rescode.wrn_zero_aij ("MSK_RES_WRN_ZERO_AIJ")

One or more zero elements are specified in A.

rescode.wrn_zeros_in_sparse_col ("MSK_RES_WRN_ZEROS_IN_SPARSE_COL")

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

rescode.wrn_zeros_in_sparse_row ("MSK_RES_WRN_ZEROS_IN_SPARSE_ROW")

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

3.36 rescodetype

rescodetype.err ("MSK_RESPONSE_ERR")

The response code is an error.

rescodetype.ok ("MSK_RESPONSE_OK")

The response code is OK.

rescodetype.trm ("MSK_RESPONSE_TRM")

The response code is an optimizer termination status.

rescodetype.unk ("MSK_RESPONSE_UNK")

The response code does not belong to any class.

rescodetype.wrn ("MSK_RESPONSE_WRN")

The response code is a warning.

3.37 scalingmethod

scalingmethod.free ("MSK_SCALING_METHOD_FREE")

The optimizer chooses the scaling heuristic.

scalingmethod.pow2 ("MSK_SCALING_METHOD_POW2")

Scales only with power of 2 leaving the mantissa untouched.

3.38 scalingtype

scalingtype.aggressive ("`MSK_SCALING_AGGRESSIVE`")

A very aggressive scaling is performed.

scalingtype.free ("`MSK_SCALING_FREE`")

The optimizer chooses the scaling heuristic.

scalingtype.moderate ("`MSK_SCALING_MODERATE`")

A conservative scaling is performed.

scalingtype.none ("`MSK_SCALING_NONE`")

No scaling is performed.

3.39 sensitivitytype

sensitivitytype.basis ("`MSK_SENSITIVITY_TYPE_BASIS`")

Basis sensitivity analysis is performed.

sensitivitytype.optimal_partition ("`MSK_SENSITIVITY_TYPE_OPTIMAL_PARTITION`")

Optimal partition sensitivity analysis is performed.

3.40 simdegen

simdegen.aggressive ("`MSK_SIM_DEGEN_AGGRESSIVE`")

The simplex optimizer should use an aggressive degeneration strategy.

simdegen.free ("`MSK_SIM_DEGEN_FREE`")

The simplex optimizer chooses the degeneration strategy.

simdegen.minimum ("`MSK_SIM_DEGEN_MINIMUM`")

The simplex optimizer should use a minimum degeneration strategy.

simdegen.moderate ("`MSK_SIM_DEGEN_MODERATE`")

The simplex optimizer should use a moderate degeneration strategy.

simdegen.none ("`MSK_SIM_DEGEN_NONE`")

The simplex optimizer should use no degeneration strategy.

3.41 simdupvec

simdupvec.free ("MSK_SIM_EXPLOIT_DUPVEC_FREE")

The simplex optimizer can choose freely.

simdupvec.off ("MSK_SIM_EXPLOIT_DUPVEC_OFF")

Disallow the simplex optimizer to exploit duplicated columns.

simdupvec.on ("MSK_SIM_EXPLOIT_DUPVEC_ON")

Allow the simplex optimizer to exploit duplicated columns.

3.42 simhotstart

simhotstart.free ("MSK_SIM_HOTSTART_FREE")

The simplex optimize chooses the hot-start type.

simhotstart.none ("MSK_SIM_HOTSTART_NONE")

The simplex optimizer performs a coldstart.

simhotstart.status_keys ("MSK_SIM_HOTSTART_STATUS_KEYS")

Only the status keys of the constraints and variables are used to choose the type of hot-start.

3.43 simreform

simreform.aggressive ("MSK_SIM_REFORMULATION_AGGRESSIVE")

The simplex optimizer should use an aggressive reformulation strategy.

simreform.free ("MSK_SIM_REFORMULATION_FREE")

The simplex optimizer can choose freely.

simreform.off ("MSK_SIM_REFORMULATION_OFF")

Disallow the simplex optimizer to reformulate the problem.

simreform.on ("MSK_SIM_REFORMULATION_ON")

Allow the simplex optimizer to reformulate the problem.

3.44 simseltype

simseltype.ase ("MSK_SIM_SELECTION_ASE")

The optimizer uses approximate steepest-edge pricing.

simseltype.devex ("MSK_SIM_SELECTION_DEVEX")

The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).

simseltype.free ("MSK_SIM_SELECTION_FREE")

The optimizer chooses the pricing strategy.

simseltype.full ("MSK_SIM_SELECTION_FULL")

The optimizer uses full pricing.

simseltype.partial ("MSK_SIM_SELECTION_PARTIAL")

The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.

simseltype.se ("MSK_SIM_SELECTION_SE")

The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

3.45 solitem

solitem.slc ("MSK_SOL_ITEM_SLC")

Lagrange multipliers for lower bounds on the constraints.

solitem.slx ("MSK_SOL_ITEM_SLX")

Lagrange multipliers for lower bounds on the variables.

solitem.snx ("MSK_SOL_ITEM_SNX")

Lagrange multipliers corresponding to the conic constraints on the variables.

solitem.suc ("MSK_SOL_ITEM_SUC")

Lagrange multipliers for upper bounds on the constraints.

solitem.sux ("MSK_SOL_ITEM_SUX")

Lagrange multipliers for upper bounds on the variables.

solitem.xc ("MSK_SOL_ITEM_XC")

Solution for the constraints.

solitem.xx ("MSK_SOL_ITEM_XX")

Variable solution.

solitem.y ("MSK_SOL_ITEM_Y")

Lagrange multipliers for equations.

3.46 solsta

solsta.dual_feas ("MSK_SOL_STA_DUAL_FEAS")

The solution is dual feasible.

solsta.dual_infeas_cer ("MSK_SOL_STA_DUAL_INFEAS_CER")

The solution is a certificate of dual infeasibility.

solsta.integer_optimal ("MSK_SOL_STA_INTEGER_OPTIMAL")

The primal solution is integer optimal.

solsta.near_dual_feas ("MSK_SOL_STA_NEAR_DUAL_FEAS")

The solution is nearly dual feasible.

solsta.near_dual_infeas_cer ("MSK_SOL_STA_NEAR_DUAL_INFEAS_CER")

The solution is almost a certificate of dual infeasibility.

solsta.near_integer_optimal ("MSK_SOL_STA_NEAR_INTEGER_OPTIMAL")

The primal solution is near integer optimal.

solsta.near_optimal ("MSK_SOL_STA_NEAR_OPTIMAL")

The solution is nearly optimal.

solsta.near_prim_and_dual_feas ("MSK_SOL_STA_NEAR_PRIM_AND_DUAL_FEAS")

The solution is nearly both primal and dual feasible.

solsta.near_prim_feas ("MSK_SOL_STA_NEAR_PRIM_FEAS")

The solution is nearly primal feasible.

solsta.near_prim_infeas_cer ("MSK_SOL_STA_NEAR_PRIM_INFEAS_CER")

The solution is almost a certificate of primal infeasibility.

solsta.optimal ("MSK_SOL_STA_OPTIMAL")

The solution is optimal.

solsta.prim_and_dual_feas ("MSK_SOL_STA_PRIM_AND_DUAL_FEAS")

The solution is both primal and dual feasible.

solsta.prim_feas ("MSK_SOL_STA_PRIM_FEAS")

The solution is primal feasible.

solsta.prim_infeas_cer ("MSK_SOL_STA_PRIM_INFEAS_CER")

The solution is a certificate of primal infeasibility.

solsta.unknown ("MSK_SOL_STA_UNKNOWN")

Status of the solution is unknown.

3.47 soltype

soltype.bas ("MSK_SOL_BAS")

The basic solution.

soltype.itg ("MSK_SOL_ITG")

The integer solution.

soltype.itr ("MSK_SOL_ITR")

The interior solution.

3.48 solveform

solveform.dual ("MSK_SOLVE_DUAL")

The optimizer should solve the dual problem.

solveform.free ("MSK_SOLVE_FREE")

The optimizer is free to solve either the primal or the dual problem.

solveform.primal ("MSK_SOLVE_PRIMAL")

The optimizer should solve the primal problem.

3.49 stakey

stakey.bas ("MSK_SK_BAS")

The constraint or variable is in the basis.

stakey.fix ("MSK_SK_FIX")

The constraint or variable is fixed.

stakey.inf ("MSK_SK_INF")

The constraint or variable is infeasible in the bounds.

stakey.low ("MSK_SK_LOW")

The constraint or variable is at its lower bound.

stakey.supbas ("MSK_SK_SUPBAS")

The constraint or variable is super basic.

stakey.unk ("MSK_SK_UNK")

The status for the constraint or variable is unknown.

stakey.upr ("MSK_SK_UPR")

The constraint or variable is at its upper bound.

3.50 startpointtype

startpointtype.constant ("MSK_STARTING_POINT_CONSTANT")

The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.

startpointtype.free ("MSK_STARTING_POINT_FREE")

The starting point is chosen automatically.

startpointtype.guess ("MSK_STARTING_POINT_GUESS")

The optimizer guesses a starting point.

startpointtype.satisfy_bounds ("MSK_STARTING_POINT_SATISFY_BOUNDS")

The starting point is choosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

3.51 streamtype

streamtype.err ("MSK_STREAM_ERR")

Error stream. Error messages are written to this stream.

streamtype.log ("MSK_STREAM_LOG")

Log stream. Contains the aggregated contents of all other streams. This means that a message written to any other stream will also be written to this stream.

streamtype.msg ("MSK_STREAM_MSG")

Message stream. Log information relating to performance and progress of the optimization is written to this stream.

streamtype.wrn ("MSK_STREAM_WRN")

Warning stream. Warning messages are written to this stream.

3.52 symmattype

symmattype.sparse ("MSK_SYMMAT_TYPE_SPARSE")

Sparse symmetric matrix.

3.53 transpose

transpose.no ("MSK_TRANSPOSE_NO")

No transpose is applied.

transpose.yes ("MSK_TRANSPOSE_YES")

A transpose is applied.

3.54 uplo

uplo.lo ("MSK_UPLO_LO")

Lower part.

uplo.up ("MSK_UPLO_UP")

Upper part

3.55 value

value.license_buffer_length ("MSK_LICENSE_BUFFER_LENGTH")

The length of a license key buffer.

value.max_str_len ("MSK_MAX_STR_LEN")

Maximum string length allowed in MOSEK.

3.56 variabletype

variabletype.type_cont ("MSK_VAR_TYPE_CONT")

Is a continuous variable.

variabletype.type_int ("MSK_VAR_TYPE_INT")

Is an integer variable.

3.57 xmlwriteroutputtype

xmlwriteroutputtype.col ("MSK_WRITE_XML_MODE_COL")

Write in column order.

xmlwriteroutputtype.row ("MSK_WRITE_XML_MODE_ROW")

Write in row order.

3.58 Parameter dparam

dparam.ana_sol_infeas_tol (`MSK_DPAR_ANA_SOL_INFEAS_TOL`)

If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer.

Default parameter value:

1e-6

Possible Values:

Any number between 0.0 and +inf.

dparam.basis_rel_tol_s (`MSK_DPAR_BASIS_REL_TOL_S`)

Maximum relative dual bound violation allowed in an optimal basic solution.

Default parameter value:

1.0e-12

Possible Values:

Any number between 0.0 and +inf.

dparam.basis_tol_s (`MSK_DPAR_BASIS_TOL_S`)

Maximum absolute dual bound violation in an optimal basic solution.

Default parameter value:

1.0e-6

Possible Values:

Any number between 1.0e-9 and +inf.

dparam.basis_tol_x (`MSK_DPAR_BASIS_TOL_X`)

Maximum absolute primal bound violation allowed in an optimal basic solution.

Default parameter value:

1.0e-6

Possible Values:

Any number between 1.0e-9 and +inf.

dparam.check_convexity_rel_tol (`MSK_DPAR_CHECK_CONVEXITY_REL_TOL`)

This parameter controls when the full convexity check declares a problem to be non-convex. Increasing this tolerance relaxes the criteria for declaring the problem non-convex.

A problem is declared non-convex if negative (positive) pivot elements are detected in the cholesky factor of a matrix which is required to be PSD (NSD). This parameter controles how much this non-negativity requirement may be violated.

If d_i is the pivot element for column i, then the matrix Q is considered to not be PSD if:

$$d_i \leq -|Q_{ii}| * \texttt{check\_convexity\_rel\_tol}$$

Default parameter value:

1e-10

Possible Values:

Any number between 0 and +inf.

dparam.data_tol_aij (`MSK_DPAR_DATA_TOL_AIJ`)

Absolute zero tolerance for elements in A. If any value A_{ij} is smaller than this parameter in absolute terms MOSEK will treat the values as zero and generate a warning.

Default parameter value:

1.0e-12

Possible Values:

Any number between 1.0e-16 and 1.0e-6.

dparam.data_tol_aij_huge (`MSK_DPAR_DATA_TOL_AIJ_HUGE`)

An element in A which is larger than this value in absolute size causes an error.

Default parameter value:

1.0e20

Possible Values:

Any number between 0.0 and +inf.

dparam.data_tol_aij_large (`MSK_DPAR_DATA_TOL_AIJ_LARGE`)

An element in A which is larger than this value in absolute size causes a warning message to be printed.

Default parameter value:

1.0e10

Possible Values:

Any number between 0.0 and +inf.

dparam.data_tol_bound_inf (`MSK_DPAR_DATA_TOL_BOUND_INF`)

Any bound which in absolute value is greater than this parameter is considered infinite.

Default parameter value:

1.0e16

Possible Values:

Any number between 0.0 and +inf.

dparam.data_tol_bound_wrn (`MSK_DPAR_DATA_TOL_BOUND_WRN`)

If a bound value is larger than this value in absolute size, then a warning message is issued.

Default parameter value:

1.0e8

Possible Values:

Any number between 0.0 and +inf.

dparam.data_tol_c_huge (`MSK_DPAR_DATA_TOL_C_HUGE`)

An element in c which is larger than the value of this parameter in absolute terms is considered to be huge and generates an error.

Default parameter value:

1.0e16

Possible Values:

Any number between 0.0 and +inf.

dparam.data_tol_cj_large (`MSK_DPAR_DATA_TOL_CJ_LARGE`)

An element in c which is larger than this value in absolute terms causes a warning message to be printed.

Default parameter value:

1.0e8

Possible Values:

Any number between 0.0 and +inf.

dparam.data_tol_qij (`MSK_DPAR_DATA_TOL_QIJ`)

Absolute zero tolerance for elements in Q matrixes.

Default parameter value:

1.0e-16

Possible Values:

Any number between 0.0 and +inf.

dparam.data_tol_x (`MSK_DPAR_DATA_TOL_X`)

Zero tolerance for constraints and variables i.e. if the distance between the lower and upper bound is less than this value, then the lower and lower bound is considered identical.

Default parameter value:

1.0e-8

Possible Values:

Any number between 0.0 and +inf.

dparam.feasrepair_tol (`MSK_DPAR_FEASREPAIR_TOL`)

Tolerance for constraint enforcing upper bound on sum of weighted violations in feasibility repair.

Default parameter value:

1.0e-10

Possible Values:

Any number between 1.0e-16 and 1.0e+16.

dparam.intpnt_co_tol_dfeas (`MSK_DPAR_INTPNT_CO_TOL_DFEAS`)

Dual feasibility tolerance used by the conic interior-point optimizer.

Default parameter value:
: 1.0e-8

Possible Values:
: Any number between 0.0 and 1.0.

dparam.intpnt_co_tol_infeas (`MSK_DPAR_INTPNT_CO_TOL_INFEAS`)

Controls when the conic interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default parameter value:
: 1.0e-10

Possible Values:
: Any number between 0.0 and 1.0.

dparam.intpnt_co_tol_mu_red (`MSK_DPAR_INTPNT_CO_TOL_MU_RED`)

Relative complementarity gap tolerance feasibility tolerance used by the conic interior-point optimizer.

Default parameter value:
: 1.0e-8

Possible Values:
: Any number between 0.0 and 1.0.

dparam.intpnt_co_tol_near_rel (`MSK_DPAR_INTPNT_CO_TOL_NEAR_REL`)

If MOSEK cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Default parameter value:
: 1000

Possible Values:
: Any number between 1.0 and +inf.

dparam.intpnt_co_tol_pfeas (`MSK_DPAR_INTPNT_CO_TOL_PFEAS`)

Primal feasibility tolerance used by the conic interior-point optimizer.

Default parameter value:
: 1.0e-8

Possible Values:
: Any number between 0.0 and 1.0.

dparam.intpnt_co_tol_rel_gap (MSK_DPAR_INTPNT_CO_TOL_REL_GAP)

Relative gap termination tolerance used by the conic interior-point optimizer.

Default parameter value:
1.0e-7

Possible Values:
Any number between 0.0 and 1.0.

dparam.intpnt_nl_merit_bal (MSK_DPAR_INTPNT_NL_MERIT_BAL)

Controls if the complementarity and infeasibility is converging to zero at about equal rates.

Default parameter value:
1.0e-4

Possible Values:
Any number between 0.0 and 0.99.

dparam.intpnt_nl_tol_dfeas (MSK_DPAR_INTPNT_NL_TOL_DFEAS)

Dual feasibility tolerance used when a nonlinear model is solved.

Default parameter value:
1.0e-8

Possible Values:
Any number between 0.0 and 1.0.

dparam.intpnt_nl_tol_mu_red (MSK_DPAR_INTPNT_NL_TOL_MU_RED)

Relative complementarity gap tolerance.

Default parameter value:
1.0e-12

Possible Values:
Any number between 0.0 and 1.0.

dparam.intpnt_nl_tol_near_rel (MSK_DPAR_INTPNT_NL_TOL_NEAR_REL)

If the MOSEK nonlinear interior-point optimizer cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Default parameter value:
1000.0

Possible Values:
Any number between 1.0 and +inf.

dparam.intpnt_nl_tol_pfeas (MSK_DPAR_INTPNT_NL_TOL_PFEAS)

Primal feasibility tolerance used when a nonlinear model is solved.

Default parameter value:

1.0e-8

Possible Values:

Any number between 0.0 and 1.0.

dparam.intpnt_nl_tol_rel_gap (`MSK_DPAR_INTPNT_NL_TOL_REL_GAP`)

Relative gap termination tolerance for nonlinear problems.

Default parameter value:

1.0e-6

Possible Values:

Any number between 1.0e-14 and +inf.

dparam.intpnt_nl_tol_rel_step (`MSK_DPAR_INTPNT_NL_TOL_REL_STEP`)

Relative step size to the boundary for general nonlinear optimization problems.

Default parameter value:

0.995

Possible Values:

Any number between 1.0e-4 and 0.9999999.

dparam.intpnt_tol_dfeas (`MSK_DPAR_INTPNT_TOL_DFEAS`)

Dual feasibility tolerance used for linear and quadratic optimization problems.

Default parameter value:

1.0e-8

Possible Values:

Any number between 0.0 and 1.0.

dparam.intpnt_tol_dsafe (`MSK_DPAR_INTPNT_TOL_DSAFE`)

Controls the initial dual starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly.

Default parameter value:

1.0

Possible Values:

Any number between 1.0e-4 and +inf.

dparam.intpnt_tol_infeas (`MSK_DPAR_INTPNT_TOL_INFEAS`)

Controls when the optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default parameter value:

1.0e-10

Possible Values:

Any number between 0.0 and 1.0.

dparam.intpnt_tol_mu_red (`MSK_DPAR_INTPNT_TOL_MU_RED`)

Relative complementarity gap tolerance.

Default parameter value:

1.0e-16

Possible Values:

Any number between 0.0 and 1.0.

dparam.intpnt_tol_path (`MSK_DPAR_INTPNT_TOL_PATH`)

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

Default parameter value:

1.0e-8

Possible Values:

Any number between 0.0 and 0.9999.

dparam.intpnt_tol_pfeas (`MSK_DPAR_INTPNT_TOL_PFEAS`)

Primal feasibility tolerance used for linear and quadratic optimization problems.

Default parameter value:

1.0e-8

Possible Values:

Any number between 0.0 and 1.0.

dparam.intpnt_tol_psafe (`MSK_DPAR_INTPNT_TOL_PSAFE`)

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

Default parameter value:

1.0

Possible Values:

Any number between 1.0e-4 and +inf.

dparam.intpnt_tol_rel_gap (`MSK_DPAR_INTPNT_TOL_REL_GAP`)

Relative gap termination tolerance.

Default parameter value:

1.0e-8

Possible Values:

Any number between 1.0e-14 and +inf.

dparam.intpnt_tol_rel_step (`MSK_DPAR_INTPNT_TOL_REL_STEP`)

Relative step size to the boundary for linear and quadratic optimization problems.

Default parameter value:

0.9999

Possible Values:

Any number between 1.0e-4 and 0.999999.

dparam.intpnt_tol_step_size (`MSK_DPAR_INTPNT_TOL_STEP_SIZE`)

If the step size falls below the value of this parameter, then the interior-point optimizer assumes that it is stalled. In other words the interior-point optimizer does not make any progress and therefore it is better stop.

Default parameter value:

1.0e-6

Possible Values:

Any number between 0.0 and 1.0.

dparam.lower_obj_cut (`MSK_DPAR_LOWER_OBJ_CUT`)

If either a primal or dual feasible solution is found proving that the optimal objective value is outside, the interval [`dparam.lower_obj_cut`, `dparam.upper_obj_cut`], then MOSEK is terminated.

Default parameter value:

-1.0e30

Possible Values:

Any number between -inf and +inf.

dparam.lower_obj_cut_finite_trh (`MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH`)

If the lower objective cut is less than the value of this parameter value, then the lower objective cut i.e. `dparam.lower_obj_cut` is treated as $-\infty$.

Default parameter value:

-0.5e30

Possible Values:

Any number between -inf and +inf.

dparam.mio_disable_term_time (`MSK_DPAR_MIO_DISABLE_TERM_TIME`)

The termination criteria governed by

- `iparam.mio_max_num_relaxs`
- `iparam.mio_max_num_branches`

- dparam.mio_near_tol_abs_gap
- dparam.mio_near_tol_rel_gap

is disabled the first n seconds. This parameter specifies the number n. A negative value is identical to infinity i.e. the termination criteria are never checked.

Default parameter value:

-1.0

Possible Values:

Any number between -inf and +inf.

dparam.mio_heuristic_time (`MSK_DPAR_MIO_HEURISTIC_TIME`)

Minimum amount of time to be used in the heuristic search for a good feasible integer solution. A negative values implies that the optimizer decides the amount of time to be spent in the heuristic.

Default parameter value:

-1.0

Possible Values:

Any number between -inf and +inf.

dparam.mio_max_time (`MSK_DPAR_MIO_MAX_TIME`)

This parameter limits the maximum time spent by the mixed-integer optimizer. A negative number means infinity.

Default parameter value:

-1.0

Possible Values:

Any number between -inf and +inf.

dparam.mio_max_time_aprx_opt (`MSK_DPAR_MIO_MAX_TIME_APRX_OPT`)

Number of seconds spent by the mixed-integer optimizer before the dparam.mio_tol_rel_relax_int is applied.

Default parameter value:

60

Possible Values:

Any number between 0.0 and +inf.

dparam.mio_near_tol_abs_gap (`MSK_DPAR_MIO_NEAR_TOL_ABS_GAP`)

Relaxed absolute optimality tolerance employed by the mixed-integer optimizer. This termination criteria is delayed. See dparam.mio_disable_term_time for details.

Default parameter value:

0.0

Possible Values:

Any number between 0.0 and +inf.

dparam.mio_near_tol_rel_gap (`MSK_DPAR_MIO_NEAR_TOL_REL_GAP`)

The mixed-integer optimizer is terminated when this tolerance is satisfied. This termination criteria is delayed. See `dparam.mio_disable_term_time` for details.

Default parameter value:

1.0e-3

Possible Values:

Any number between 0.0 and +inf.

dparam.mio_rel_add_cut_limited (`MSK_DPAR_MIO_REL_ADD_CUT_LIMITED`)

Controls how many cuts the mixed-integer optimizer is allowed to add to the problem. Let α be the value of this parameter and m the number constraints, then mixed-integer optimizer is allowed to αm cuts.

Default parameter value:

0.75

Possible Values:

Any number between 0.0 and 2.0.

dparam.mio_rel_gap_const (`MSK_DPAR_MIO_REL_GAP_CONST`)

This value is used to compute the relative gap for the solution to an integer optimization problem.

Default parameter value:

1.0e-10

Possible Values:

Any number between 1.0e-15 and +inf.

dparam.mio_tol_abs_gap (`MSK_DPAR_MIO_TOL_ABS_GAP`)

Absolute optimality tolerance employed by the mixed-integer optimizer.

Default parameter value:

0.0

Possible Values:

Any number between 0.0 and +inf.

dparam.mio_tol_abs_relax_int (`MSK_DPAR_MIO_TOL_ABS_RELAX_INT`)

Absolute relaxation tolerance of the integer constraints. I.e. $\min(|x| - \lfloor x \rfloor, \lceil x \rceil - |x|)$ is less than the tolerance then the integer restrictions assumed to be satisfied.

Default parameter value:

1.0e-5

Possible Values:

Any number between 1e-9 and +inf.

dparam.mio_tol_feas (`MSK_DPAR_MIO_TOL_FEAS`)

Feasibility tolerance for mixed integer solver. Any solution with maximum infeasibility below this value will be considered feasible.

Default parameter value:

1.0e-7

Possible Values:

Any number between 0.0 and +inf.

dparam.mio_tol_max_cut_frac_rhs (`MSK_DPAR_MIO_TOL_MAX_CUT_FRAC_RHS`)

Maximum value of fractional part of right hand side to generate CMIR and CG cuts for. A value of 0.0 means that the value is selected automatically.

Default parameter value:

0.0

Possible Values:

Any number between 0.0 and 1.0.

dparam.mio_tol_min_cut_frac_rhs (`MSK_DPAR_MIO_TOL_MIN_CUT_FRAC_RHS`)

Minimum value of fractional part of right hand side to generate CMIR and CG cuts for. A value of 0.0 means that the value is selected automatically.

Default parameter value:

0.0

Possible Values:

Any number between 0.0 and 1.0.

dparam.mio_tol_rel_dual_bound_improvement (`MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT`)

If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation. A value of 0.0 means that the value is selected automatically.

Default parameter value:

0.0

Possible Values:

Any number between 0.0 and 1.0.

dparam.mio_tol_rel_gap (`MSK_DPAR_MIO_TOL_REL_GAP`)

Relative optimality tolerance employed by the mixed-integer optimizer.

Default parameter value:

1.0e-4

Possible Values:

Any number between 0.0 and +inf.

dparam.mio_tol_rel_relax_int (`MSK_DPAR_MIO_TOL_REL_RELAX_INT`)

Relative relaxation tolerance of the integer constraints. I.e $(\min(|x| - \lfloor x \rfloor, \lceil x \rceil - |x|))$ is less than the tolerance times $|x|$ then the integer restrictions assumed to be satisfied.

Default parameter value:

1.0e-6

Possible Values:

Any number between 0.0 and +inf.

dparam.mio_tol_x (`MSK_DPAR_MIO_TOL_X`)

Absolute solution tolerance used in mixed-integer optimizer.

Default parameter value:

1.0e-6

Possible Values:

Any number between 0.0 and +inf.

dparam.nonconvex_tol_feas (`MSK_DPAR_NONCONVEX_TOL_FEAS`)

Feasibility tolerance used by the nonconvex optimizer.

Default parameter value:

1.0e-6

Possible Values:

Any number between 0.0 and +inf.

dparam.nonconvex_tol_opt (`MSK_DPAR_NONCONVEX_TOL_OPT`)

Optimality tolerance used by the nonconvex optimizer.

Default parameter value:

1.0e-7

Possible Values:

Any number between 0.0 and +inf.

dparam.optimizer_max_time (`MSK_DPAR_OPTIMIZER_MAX_TIME`)

Maximum amount of time the optimizer is allowed to spent on the optimization. A negative number means infinity.

Default parameter value:

-1.0

Possible Values:

Any number between -inf and +inf.

dparam.presolve_tol_abs_lindep (`MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP`)

Absolute tolerance employed by the linear dependency checker.

Default parameter value:
1.0e-6

Possible Values:
Any number between 0.0 and +inf.

dparam.presolve_tol_aij (`MSK_DPAR_PRESOLVE_TOL_AIJ`)

Absolute zero tolerance employed for a_{ij} in the presolve.

Default parameter value:
1.0e-12

Possible Values:
Any number between 1.0e-15 and +inf.

dparam.presolve_tol_rel_lindep (`MSK_DPAR_PRESOLVE_TOL_REL_LINDEP`)

Relative tolerance employed by the linear dependency checker.

Default parameter value:
1.0e-10

Possible Values:
Any number between 0.0 and +inf.

dparam.presolve_tol_s (`MSK_DPAR_PRESOLVE_TOL_S`)

Absolute zero tolerance employed for s_i in the presolve.

Default parameter value:
1.0e-8

Possible Values:
Any number between 0.0 and +inf.

dparam.presolve_tol_x (`MSK_DPAR_PRESOLVE_TOL_X`)

Absolute zero tolerance employed for x_j in the presolve.

Default parameter value:
1.0e-8

Possible Values:
Any number between 0.0 and +inf.

dparam.qcqo_reformulate_rel_drop_tol (`MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL`)

This parameter determines when columns are dropped in incomplete cholesky factorization doing reformulation of quadratic problems.

Default parameter value:

1e-15

Possible Values:

Any number between 0 and +inf.

dparam.sim_lu_tol_rel_piv (`MSK_DPAR_SIM_LU_TOL_REL_PIV`)

Relative pivot tolerance employed when computing the LU factorization of the basis in the simplex optimizers and in the basis identification procedure.

A value closer to 1.0 generally improves numerical stability but typically also implies an increase in the computational work.

Default parameter value:

0.01

Possible Values:

Any number between 1.0e-6 and 0.999999.

dparam.simplex_abs_tol_piv (`MSK_DPAR_SIMPLEX_ABS_TOL_PIV`)

Absolute pivot tolerance employed by the simplex optimizers.

Default parameter value:

1.0e-7

Possible Values:

Any number between 1.0e-12 and +inf.

dparam.upper_obj_cut (`MSK_DPAR_UPPER_OBJ_CUT`)

If either a primal or dual feasible solution is found proving that the optimal objective value is outside, [`dparam.lower_obj_cut`, `dparam.upper_obj_cut`], then MOSEK is terminated.

Default parameter value:

1.0e30

Possible Values:

Any number between -inf and +inf.

dparam.upper_obj_cut_finite_trh (`MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH`)

If the upper objective cut is greater than the value of this value parameter, then the the upper objective cut `dparam.upper_obj_cut` is treated as ∞.

Default parameter value:

0.5e30

Possible Values:

Any number between -inf and +inf.

3.59 Parameter sparam

sparam.bas_sol_file_name (`MSK_SPAR_BAS_SOL_FILE_NAME`)

Name of the `bas` solution file.

Default parameter value:

Possible Values:

Any valid file name.

sparam.data_file_name (`MSK_SPAR_DATA_FILE_NAME`)

Data are read and written to this file.

Default parameter value:

Possible Values:

Any valid file name.

sparam.debug_file_name (`MSK_SPAR_DEBUG_FILE_NAME`)

MOSEK debug file.

Default parameter value:

Possible Values:

Any valid file name.

sparam.feasrepair_name_prefix (`MSK_SPAR_FEASREPAIR_NAME_PREFIX`)

Not applicable.

Default parameter value:

MSK-

Possible Values:

Any valid string.

sparam.feasrepair_name_separator (`MSK_SPAR_FEASREPAIR_NAME_SEPARATOR`)

Not applicable.

Default parameter value:

-

Possible Values:

Any valid string.

sparam.feasrepair_name_wsumviol (`MSK_SPAR_FEASREPAIR_NAME_WSUMVIOL`)

The constraint and variable associated with the total weighted sum of violations are each given the name of this parameter postfixed with `CON` and `VAR` respectively.

Default parameter value:

WSUMVIOL

Possible Values:

Any valid string.

sparam.int_sol_file_name (`MSK_SPAR_INT_SOL_FILE_NAME`)

Name of the `int` solution file.

Default parameter value:

Possible Values:

Any valid file name.

sparam.itr_sol_file_name (`MSK_SPAR_ITR_SOL_FILE_NAME`)

Name of the `itr` solution file.

Default parameter value:

Possible Values:

Any valid file name.

sparam.mio_debug_string (`MSK_SPAR_MIO_DEBUG_STRING`)

For internal use only.

Default parameter value:

Possible Values:

Any valid string.

sparam.param_comment_sign (`MSK_SPAR_PARAM_COMMENT_SIGN`)

Only the first character in this string is used. It is considered as a start of comment sign in the MOSEK parameter file. Spaces are ignored in the string.

Default parameter value:

%%

Possible Values:

Any valid string.

sparam.param_read_file_name (`MSK_SPAR_PARAM_READ_FILE_NAME`)

Modifications to the parameter database is read from this file.

Default parameter value:

Possible Values:

Any valid file name.

sparam.param_write_file_name (`MSK_SPAR_PARAM_WRITE_FILE_NAME`)

The parameter database is written to this file.

Default parameter value:

Possible Values:

Any valid file name.

sparam.read_mps_bou_name (`MSK_SPAR_READ_MPS_BOU_NAME`)

Name of the BOUNDS vector used. An empty name means that the first BOUNDS vector is used.

Default parameter value:

Possible Values:

Any valid MPS name.

sparam.read_mps_obj_name (`MSK_SPAR_READ_MPS_OBJ_NAME`)

Name of the free constraint used as objective function. An empty name means that the first constraint is used as objective function.

Default parameter value:

Possible Values:

Any valid MPS name.

sparam.read_mps_ran_name (`MSK_SPAR_READ_MPS_RAN_NAME`)

Name of the RANGE vector used. An empty name means that the first RANGE vector is used.

Default parameter value:

Possible Values:

Any valid MPS name.

sparam.read_mps_rhs_name (`MSK_SPAR_READ_MPS_RHS_NAME`)

Name of the RHS used. An empty name means that the first RHS vector is used.

Default parameter value:

Possible Values:

Any valid MPS name.

sparam.sensitivity_file_name (`MSK_SPAR_SENSITIVITY_FILE_NAME`)

If defined `Task.sensitivityreport` reads this file as a sensitivity analysis data file specifying the type of analysis to be done.

Default parameter value:

Possible Values:

Any valid string.

sparam.sensitivity_res_file_name (`MSK_SPAR_SENSITIVITY_RES_FILE_NAME`)

If this is a nonempty string, then `Task.sensitivityreport` writes results to this file.

Default parameter value:

Possible Values:

Any valid string.

sparam.sol_filter_xc_low (`MSK_SPAR_SOL_FILTER_XC_LOW`)

A filter used to determine which constraints should be listed in the solution file. A value of "0.5" means that all constraints having `xc[i]>0.5` should be listed, whereas "+0.5" means that all constraints having `xc[i]>=blc[i]+0.5` should be listed. An empty filter means that no filter is applied.

Default parameter value:

Possible Values:

Any valid filter.

sparam.sol_filter_xc_upr (`MSK_SPAR_SOL_FILTER_XC_UPR`)

A filter used to determine which constraints should be listed in the solution file. A value of "0.5" means that all constraints having `xc[i]<0.5` should be listed, whereas "-0.5" means all constraints having `xc[i]<=buc[i]-0.5` should be listed. An empty filter means that no filter is applied.

Default parameter value:

Possible Values:

Any valid filter.

sparam.sol_filter_xx_low (`MSK_SPAR_SOL_FILTER_XX_LOW`)

A filter used to determine which variables should be listed in the solution file. A value of "0.5" means that all constraints having `xx[j]>=0.5` should be listed, whereas "+0.5" means that all constraints having `xx[j]>=blx[j]+0.5` should be listed. An empty filter means no filter is applied.

Default parameter value:

Possible Values:

Any valid filter.

sparam.sol_filter_xx_upr (`MSK_SPAR_SOL_FILTER_XX_UPR`)

A filter used to determine which variables should be listed in the solution file. A value of "0.5" means that all constraints having `xx[j]<0.5` should be printed, whereas "-0.5" means all constraints having `xx[j]<=bux[j]-0.5` should be listed. An empty filter means no filter is applied.

Default parameter value:

Possible Values:

Any valid file name.

sparam.stat_file_name (`MSK_SPAR_STAT_FILE_NAME`)

Statistics file name.

Default parameter value:

Possible Values:

Any valid file name.

sparam.stat_key (`MSK_SPAR_STAT_KEY`)

Key used when writing the summary file.

Default parameter value:

Possible Values:

Any valid XML string.

sparam.stat_name (`MSK_SPAR_STAT_NAME`)

Name used when writing the statistics file.

Default parameter value:

Possible Values:

Any valid XML string.

sparam.write_lp_gen_var_name (`MSK_SPAR_WRITE_LP_GEN_VAR_NAME`)

Sometimes when an LP file is written additional variables must be inserted. They will have the prefix denoted by this parameter.

Default parameter value:

xmskgen

Possible Values:

Any valid string.

3.60 Parameter iparam

iparam.alloc_add_qnz (`MSK_IPAR_ALLOC_ADD_QNZ`)

Additional number of Q non-zeros that are allocated space for when `numanz` exceeds `maxnumqnz` during addition of new Q entries.

Default parameter value:

5000

Possible Values:

Any number between 0 and +inf.

iparam.ana_sol_basis (`MSK_IPAR_ANA_SOL_BASIS`)

Controls whether the basis matrix is analyzed in solaution analyzer.

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.ana_sol_print_violated (MSK_IPAR_ANA_SOL_PRINT_VIOLATED)

Controls whether a list of violated constraints is printed when calling Task.analyzesolution. All constraints violated by more than the value set by the parameter dparam.ana_sol_infeas_tol will be printed.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.auto_sort_a_before_opt (MSK_IPAR_AUTO_SORT_A_BEFORE_OPT)

Controls whether the elements in each column of A are sorted before an optimization is performed. This is not required but makes the optimization more deterministic.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.auto_update_sol_info (MSK_IPAR_AUTO_UPDATE_SOL_INFO)

Controls whether the solution information items are automatically updated after an optimization is performed.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.basis_solve_use_plus_one (MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE)

Unused.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.bi_clean_optimizer (MSK_IPAR_BI_CLEAN_OPTIMIZER)

Controls which simplex optimizer is used in the clean-up phase.

Default parameter value:

optimizertype.free

Possible values:

- optimizertype.concurrent The optimizer for nonconvex nonlinear problems.
- optimizertype.conic The optimizer for problems having conic constraints.
- optimizertype.dual_simplex The dual simplex optimizer is used.
- optimizertype.free The optimizer is chosen automatically.
- optimizertype.free_simplex One of the simplex optimizers is used.
- optimizertype.intpnt The interior-point optimizer is used.
- optimizertype.mixed_int The mixed-integer optimizer.
- optimizertype.mixed_int_conic The mixed-integer optimizer for conic and linear problems.
- optimizertype.network_primal_simplex The network primal simplex optimizer is used. It is only applicable to pure network problems.
- optimizertype.nonconvex The optimizer for nonconvex nonlinear problems.
- optimizertype.primal_dual_simplex The primal dual simplex optimizer is used.
- optimizertype.primal_simplex The primal simplex optimizer is used.

iparam.bi_ignore_max_iter (MSK_IPAR_BI_IGNORE_MAX_ITER)

If the parameter iparam.intpnt_basis has the value basindtype.no_error and the interior-point optimizer has terminated due to maximum number of iterations, then basis identification is performed if this parameter has the value onoffkey.on.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.bi_ignore_num_error (MSK_IPAR_BI_IGNORE_NUM_ERROR)

If the parameter iparam.intpnt_basis has the value basindtype.no_error and the interior-point optimizer has terminated due to a numerical problem, then basis identification is performed if this parameter has the value onoffkey.on.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.bi_max_iterations (MSK_IPAR_BI_MAX_ITERATIONS)

Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification.

Default parameter value:

1000000

Possible Values:

Any number between 0 and +inf.

iparam.cache_license (MSK_IPAR_CACHE_LICENSE)

Specifies if the license is kept checked out for the lifetime of the mosek environment (on) or returned to the server immediately after the optimization (off).

Check-in and check-out of licenses have an overhead. Frequent communication with the license server should be avoided.

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.check_convexity (MSK_IPAR_CHECK_CONVEXITY)

Specify the level of convexity check on quadratic problems

Default parameter value:

`checkconvexitytype.full`

Possible values:

- `checkconvexitytype.full` Perform a full convexity check.
- `checkconvexitytype.none` No convexity check.
- `checkconvexitytype.simple` Perform simple and fast convexity check.

iparam.compress_statfile (MSK_IPAR_COMPRESS_STATFILE)

Control compression of stat files.

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.concurrent_num_optimizers (MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS)

The maximum number of simultaneous optimizations that will be started by the concurrent optimizer.

Default parameter value:

2

Possible Values:

Any number between 0 and +inf.

iparam.concurrent_priority_dual_simplex (`MSK_IPAR_CONCURRENT_PRIORITY_DUAL_SIMPLEX`)

Priority of the dual simplex algorithm when selecting solvers for concurrent optimization.

Default parameter value:

2

Possible Values:

Any number between 0 and +inf.

iparam.concurrent_priority_free_simplex (`MSK_IPAR_CONCURRENT_PRIORITY_FREE_SIMPLEX`)

Priority of the free simplex optimizer when selecting solvers for concurrent optimization.

Default parameter value:

3

Possible Values:

Any number between 0 and +inf.

iparam.concurrent_priority_intpnt (`MSK_IPAR_CONCURRENT_PRIORITY_INTPNT`)

Priority of the interior-point algorithm when selecting solvers for concurrent optimization.

Default parameter value:

4

Possible Values:

Any number between 0 and +inf.

iparam.concurrent_priority_primal_simplex (`MSK_IPAR_CONCURRENT_PRIORITY_PRIMAL_SIMPLEX`)

Priority of the primal simplex algorithm when selecting solvers for concurrent optimization.

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.feasrepair_optimize (`MSK_IPAR_FEASREPAIR_OPTIMIZE`)

Controls which type of feasibility analysis is to be performed.

Default parameter value:

`feasrepairtype.optimize_none`

Possible values:

- `feasrepairtype.optimize_combined` Minimize with original objective subject to minimal weighted violation of bounds.
- `feasrepairtype.optimize_none` Do not optimize the feasibility repair problem.

- `feasrepairtype.optimize_penalty` Minimize weighted sum of violations.

iparam.infeas_generic_names (`MSK_IPAR_INFEAS_GENERIC_NAMES`)

Controls whether generic names are used when an infeasible subproblem is created.

Default parameter value:

`onoffkey.off`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.infeas_prefer_primal (`MSK_IPAR_INFEAS_PREFER_PRIMAL`)

If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on.

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.infeas_report_auto (`MSK_IPAR_INFEAS_REPORT_AUTO`)

Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible.

Default parameter value:

`onoffkey.off`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.infeas_report_level (`MSK_IPAR_INFEAS_REPORT_LEVEL`)

Controls the amount of information presented in an infeasibility report. Higher values imply more information.

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.intpnt_basis (`MSK_IPAR_INTPNT_BASIS`)

Controls whether the interior-point optimizer also computes an optimal basis.

Default parameter value:

`basindtype.always`

Possible values:

- basindtype.always Basis identification is always performed even if the interior-point optimizer terminates abnormally.
- basindtype.if_feasible Basis identification is not performed if the interior-point optimizer terminates with a problem status saying that the problem is primal or dual infeasible.
- basindtype.never Never do basis identification.
- basindtype.no_error Basis identification is performed if the interior-point optimizer terminates without an error.
- basindtype.reservered Not currently in use.

iparam.intpnt_diff_step (MSK_IPAR_INTPNT_DIFF_STEP)

Controls whether different step sizes are allowed in the primal and dual space.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.intpnt_factor_debug_lvl (MSK_IPAR_INTPNT_FACTOR_DEBUG_LVL)

Controls factorization debug level.

Default parameter value:

0

Possible Values:

Any number between 0 and +inf.

iparam.intpnt_factor_method (MSK_IPAR_INTPNT_FACTOR_METHOD)

Controls the method used to factor the Newton equation system.

Default parameter value:

0

Possible Values:

Any number between 0 and +inf.

iparam.intpnt_hotstart (MSK_IPAR_INTPNT_HOTSTART)

Currently not in use.

Default parameter value:

intpnthotstart.none

Possible values:

- intpnthotstart.dual The interior-point optimizer exploits the dual solution only.

- intpnthotstart.none The interior-point optimizer performs a coldstart.
- intpnthotstart.primal The interior-point optimizer exploits the primal solution only.
- intpnthotstart.primal_dual The interior-point optimizer exploits both the primal and dual solution.

iparam.intpnt_max_iterations (`MSK_IPAR_INTPNT_MAX_ITERATIONS`)

Controls the maximum number of iterations allowed in the interior-point optimizer.

Default parameter value:

400

Possible Values:

Any number between 0 and +inf.

iparam.intpnt_max_num_cor (`MSK_IPAR_INTPNT_MAX_NUM_COR`)

Controls the maximum number of correctors allowed by the multiple corrector procedure. A negative value means that MOSEK is making the choice.

Default parameter value:

-1

Possible Values:

Any number between -1 and +inf.

iparam.intpnt_max_num_refinement_steps (`MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS`)

Maximum number of steps to be used by the iterative refinement of the search direction. A negative value implies that the optimizer Chooses the maximum number of iterative refinement steps.

Default parameter value:

-1

Possible Values:

Any number between -inf and +inf.

iparam.intpnt_off_col_trh (`MSK_IPAR_INTPNT_OFF_COL_TRH`)

Controls how many offending columns are detected in the Jacobian of the constraint matrix.

1 means aggressive detection, higher values mean less aggressive detection.

0 means no detection.

Default parameter value:

40

Possible Values:

Any number between 0 and +inf.

iparam.intpnt_order_method (`MSK_IPAR_INTPNT_ORDER_METHOD`)

Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.

Default parameter value:

orderingtype.free

Possible values:

- orderingtype.appminloc Approximate minimum local fill-in ordering is employed.
- orderingtype.experimental This option should not be used.
- orderingtype.force_graphpar Always use the graph partitioning based ordering even if it is worse that the approximate minimum local fill ordering.
- orderingtype.free The ordering method is chosen automatically.
- orderingtype.none No ordering is used.
- orderingtype.try_graphpar Always try the the graph partitioning based ordering.

iparam.intpnt_regularization_use (MSK_IPAR_INTPNT_REGULARIZATION_USE)

Controls whether regularization is allowed.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.intpnt_scaling (MSK_IPAR_INTPNT_SCALING)

Controls how the problem is scaled before the interior-point optimizer is used.

Default parameter value:

scalingtype.free

Possible values:

- scalingtype.aggressive A very aggressive scaling is performed.
- scalingtype.free The optimizer chooses the scaling heuristic.
- scalingtype.moderate A conservative scaling is performed.
- scalingtype.none No scaling is performed.

iparam.intpnt_solve_form (MSK_IPAR_INTPNT_SOLVE_FORM)

Controls whether the primal or the dual problem is solved.

Default parameter value:

solveform.free

Possible values:

- solveform.dual The optimizer should solve the dual problem.
- solveform.free The optimizer is free to solve either the primal or the dual problem.
- solveform.primal The optimizer should solve the primal problem.

iparam.intpnt_starting_point (MSK_IPAR_INTPNT_STARTING_POINT)

Starting point used by the interior-point optimizer.

Default parameter value:

`startpointtype.free`

Possible values:

- `startpointtype.constant` The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.
- `startpointtype.free` The starting point is chosen automatically.
- `startpointtype.guess` The optimizer guesses a starting point.
- `startpointtype.satisfy_bounds` The starting point is choosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

iparam.lic_trh_expiry_wrn (`MSK_IPAR_LIC_TRH_EXPIRY_WRN`)

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

Default parameter value:

7

Possible Values:

Any number between 0 and +inf.

iparam.license_debug (`MSK_IPAR_LICENSE_DEBUG`)

This option is used to turn on debugging of the incense manager.

Default parameter value:

`onoffkey.off`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.license_pause_time (`MSK_IPAR_LICENSE_PAUSE_TIME`)

If `iparam.license_wait`=`onoffkey.on` and no license is available, then MOSEK sleeps a number of milliseconds between each check of whether a license has become free.

Default parameter value:

100

Possible Values:

Any number between 0 and 1000000.

iparam.license_suppress_expire_wrns (`MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS`)

Controls whether license features expire warnings are suppressed.

Default parameter value:

`onoffkey.off`

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.license_wait (MSK_IPAR_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 parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.log (MSK_IPAR_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 parameter value:

10

Possible Values:

Any number between 0 and +inf.

iparam.log_bi (MSK_IPAR_LOG_BI)

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

Default parameter value:

4

Possible Values:

Any number between 0 and +inf.

iparam.log_bi_freq (MSK_IPAR_LOG_BI_FREQ)

Controls how frequent the optimizer outputs information about the basis identification and how frequent the user-defined call-back function is called.

Default parameter value:

2500

Possible Values:

Any number between 0 and +inf.

iparam.log_check_convexity (`MSK_IPAR_LOG_CHECK_CONVEXITY`)

Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on.

If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

Default parameter value:

0

Possible Values:

Any number between 0 and +inf.

iparam.log_concurrent (`MSK_IPAR_LOG_CONCURRENT`)

Controls amount of output printed by the concurrent optimizer.

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.log_cut_second_opt (`MSK_IPAR_LOG_CUT_SECOND_OPT`)

If a task is employed to solve a sequence of optimization problems, then the value of the log levels is reduced by the value of this parameter. E.g `iparam.log` and `iparam.log_sim` are reduced by the value of this parameter for the second and any subsequent optimizations.

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.log_expand (`MSK_IPAR_LOG_EXPAND`)

Controls the amount of logging when a data item such as the maximum number constrains is expanded.

Default parameter value:

0

Possible Values:

Any number between 0 and +inf.

iparam.log_factor (`MSK_IPAR_LOG_FACTOR`)

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

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.log_feas_repair (`MSK_IPAR_LOG_FEAS_REPAIR`)

Controls the amount of output printed when performing feasibility repair. A value higher than one means extensive logging.

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.log_file (`MSK_IPAR_LOG_FILE`)

If turned on, then some log info is printed when a file is written or read.

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.log_head (`MSK_IPAR_LOG_HEAD`)

If turned on, then a header line is added to the log.

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.log_infeas_ana (`MSK_IPAR_LOG_INFEAS_ANA`)

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

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.log_intpnt (`MSK_IPAR_LOG_INTPNT`)

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

Default parameter value:

4

Possible Values:

Any number between 0 and +inf.

iparam.log_mio (`MSK_IPAR_LOG_MIO`)

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

Default parameter value:

4

Possible Values:

Any number between 0 and +inf.

iparam.log_mio_freq (`MSK_IPAR_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 parameter value:

1000

Possible Values:

A integer value.

iparam.log_nonconvex (`MSK_IPAR_LOG_NONCONVEX`)

Controls amount of output printed by the nonconvex optimizer.

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.log_optimizer (`MSK_IPAR_LOG_OPTIMIZER`)

Controls the amount of general optimizer information that is logged.

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.log_order (`MSK_IPAR_LOG_ORDER`)

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

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.log_param (`MSK_IPAR_LOG_PARAM`)

Controls the amount of information printed out about parameter changes.

Default parameter value:

0

Possible Values:

Any number between 0 and +inf.

iparam.log_presolve (MSK_IPAR_LOG_PRESOLVE)

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

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.log_response (MSK_IPAR_LOG_RESPONSE)

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

Default parameter value:

0

Possible Values:

Any number between 0 and +inf.

iparam.log_sensitivity (MSK_IPAR_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 parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.log_sensitivity_opt (MSK_IPAR_LOG_SENSITIVITY_OPT)

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

Default parameter value:

0

Possible Values:

Any number between 0 and +inf.

iparam.log_sim (MSK_IPAR_LOG_SIM)

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

Default parameter value:

4

Possible Values:

Any number between 0 and +inf.

iparam.log_sim_freq (MSK_IPAR_LOG_SIM_FREQ)

Controls how frequent the simplex optimizer outputs information about the optimization and how frequent the user-defined call-back function is called.

Default parameter value:

1000

Possible Values:

Any number between 0 and +inf.

iparam.log_sim_minor (MSK_IPAR_LOG_SIM_MINOR)

Currently not in use.

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.log_sim_network_freq (MSK_IPAR_LOG_SIM_NETWORK_FREQ)

Controls how frequent the network simplex optimizer outputs information about the optimization and how frequent the user-defined call-back function is called. The network optimizer will use a logging frequency equal to iparam.log_sim_freq times iparam.log_sim_network_freq.

Default parameter value:

1000

Possible Values:

Any number between 0 and +inf.

iparam.log_storage (MSK_IPAR_LOG_STORAGE)

When turned on, MOSEK prints messages regarding the storage usage and allocation.

Default parameter value:

0

Possible Values:

Any number between 0 and +inf.

iparam.max_num_warnings (MSK_IPAR_MAX_NUM_WARNINGS)

A negtive number means all warnings are logged. Otherwise the parameter specifies the maximum number times each warning is logged.

Default parameter value:

6

Possible Values:

Any number between -inf and +inf.

iparam.mio_branch_dir (`MSK_IPAR_MIO_BRANCH_DIR`)

Controls whether the mixed-integer optimizer is branching up or down by default.

Default parameter value:

`branchdir.free`

Possible values:

- `branchdir.down` The mixed-integer optimizer always chooses the down branch first.
- `branchdir.free` The mixed-integer optimizer decides which branch to choose.
- `branchdir.up` The mixed-integer optimizer always chooses the up branch first.

iparam.mio_branch_priorities_use (`MSK_IPAR_MIO_BRANCH_PRIORITIES_USE`)

Controls whether branching priorities are used by the mixed-integer optimizer.

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.mio_construct_sol (`MSK_IPAR_MIO_CONSTRUCT_SOL`)

If set to `onoffkey.on` and all integer variables have been given a value for which a feasible mixed integer solution exists, then MOSEK generates an initial solution to the mixed integer problem by fixing all integer values and solving the remaining problem.

Default parameter value:

`onoffkey.off`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.mio_cont_sol (`MSK_IPAR_MIO_CONT_SOL`)

Controls the meaning of the interior-point and basic solutions in mixed integer problems.

Default parameter value:

`miocontsoltype.none`

Possible values:

- `miocontsoltype.itg` The reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. A solution is only reported in case the problem has a primal feasible solution.

- miocontsoltype.itg_rel In case the problem is primal feasible then the reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. If the problem is primal infeasible, then the solution to the root node problem is reported.
- miocontsoltype.none No interior-point or basic solution are reported when the mixed-integer optimizer is used.
- miocontsoltype.root The reported interior-point and basic solutions are a solution to the root node problem when mixed-integer optimizer is used.

iparam.mio_cut_cg (MSK_IPAR_MIO_CUT_CG)

Controls whether CG cuts should be generated.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.mio_cut_cmir (MSK_IPAR_MIO_CUT_CMIR)

Controls whether mixed integer rounding cuts should be generated.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.mio_cut_level_root (MSK_IPAR_MIO_CUT_LEVEL_ROOT)

Controls the cut level employed by the mixed-integer optimizer at the root node. A negative value means a default value determined by the mixed-integer optimizer is used. By adding the appropriate values from the following table the employed cut types can be controlled.

GUB cover	+2
Flow cover	+4
Lifting	+8
Plant location	+16
Disaggregation	+32
Knapsack cover	+64
Lattice	+128
Gomory	+256
Coefficient reduction	+512
GCD	+1024
Obj. integrality	+2048

Default parameter value:

-1

Possible Values:

Any value.

iparam.mio_cut_level_tree (`MSK_IPAR_MIO_CUT_LEVEL_TREE`)

Controls the cut level employed by the mixed-integer optimizer at the tree. See `iparam.mio_cut_level_root` for an explanation of the parameter values.

Default parameter value:

-1

Possible Values:

Any value.

iparam.mio_feaspump_level (`MSK_IPAR_MIO_FEASPUMP_LEVEL`)

Feasibility pump is a heuristic designed to compute an initial feasible solution. A value of 0 implies that the feasibility pump heuristic is not used. A value of -1 implies that the mixed-integer optimizer decides how the feasibility pump heuristic is used. A larger value than 1 implies that the feasibility pump is employed more aggressively. Normally a value beyond 3 is not worthwhile.

Default parameter value:

-1

Possible Values:

Any number between -inf and 3.

iparam.mio_heuristic_level (`MSK_IPAR_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 parameter value:

-1

Possible Values:

Any value.

iparam.mio_hotstart (`MSK_IPAR_MIO_HOTSTART`)

Controls whether the integer optimizer is hot-started.

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.mio_keep_basis (`MSK_IPAR_MIO_KEEP_BASIS`)

Controls whether the integer presolve keeps bases in memory. This speeds on the solution process at cost of bigger memory consumption.

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.mio_local_branch_number (`MSK_IPAR_MIO_LOCAL_BRANCH_NUMBER`)

Controls the size of the local search space when doing local branching.

Default parameter value:

-1

Possible Values:

Any number between -inf and +inf.

iparam.mio_max_num_branches (`MSK_IPAR_MIO_MAX_NUM_BRANCHES`)

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

Default parameter value:

-1

Possible Values:

Any number between -inf and +inf.

iparam.mio_max_num_relaxs (`MSK_IPAR_MIO_MAX_NUM_RELAXS`)

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

Default parameter value:

-1

Possible Values:

Any number between -inf and +inf.

iparam.mio_max_num_solutions (`MSK_IPAR_MIO_MAX_NUM_SOLUTIONS`)

The mixed-integer optimizer can be terminated after a certain number of different feasible solutions has been located. If this parameter has the value n and n is strictly positive, then the mixed-integer optimizer will be terminated when n feasible solutions have been located.

Default parameter value:

-1

Possible Values:

Any number between -inf and +inf.

iparam.mio_mode (MSK_IPAR_MIO_MODE)

Controls whether the optimizer includes the integer restrictions when solving a (mixed) integer optimization problem.

Default parameter value:

miomode.satisfied

Possible values:

- miomode.ignored The integer constraints are ignored and the problem is solved as a continuous problem.
- miomode.lazy Integer restrictions should be satisfied if an optimizer is available for the problem.
- miomode.satisfied Integer restrictions should be satisfied.

iparam.mio_mt_user_cb (MSK_IPAR_MIO_MT_USER_CB)

It true user callbacks are called from each thread used by this optimizer. If false the user callback is only called from a single thread.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.mio_node_optimizer (MSK_IPAR_MIO_NODE_OPTIMIZER)

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

Default parameter value:

optimizertype.free

Possible values:

- optimizertype.concurrent The optimizer for nonconvex nonlinear problems.
- optimizertype.conic The optimizer for problems having conic constraints.
- optimizertype.dual_simplex The dual simplex optimizer is used.
- optimizertype.free The optimizer is chosen automatically.
- optimizertype.free_simplex One of the simplex optimizers is used.
- optimizertype.intpnt The interior-point optimizer is used.
- optimizertype.mixed_int The mixed-integer optimizer.
- optimizertype.mixed_int_conic The mixed-integer optimizer for conic and linear problems.
- optimizertype.network_primal_simplex The network primal simplex optimizer is used. It is only applicable to pure network problems.
- optimizertype.nonconvex The optimizer for nonconvex nonlinear problems.
- optimizertype.primal_dual_simplex The primal dual simplex optimizer is used.

- optimizertype.primal_simplex The primal simplex optimizer is used.

iparam.mio_node_selection (MSK_IPAR_MIO_NODE_SELECTION)

Controls the node selection strategy employed by the mixed-integer optimizer.

Default parameter value:

mionodeseltype.free

Possible values:

- mionodeseltype.best The optimizer employs a best bound node selection strategy.
- mionodeseltype.first The optimizer employs a depth first node selection strategy.
- mionodeseltype.free The optimizer decides the node selection strategy.
- mionodeseltype.hybrid The optimizer employs a hybrid strategy.
- mionodeseltype.pseudo The optimizer employs selects the node based on a pseudo cost estimate.
- mionodeseltype.worst The optimizer employs a worst bound node selection strategy.

iparam.mio_optimizer_mode (MSK_IPAR_MIO_OPTIMIZER_MODE)

An exprimental feature.

Default parameter value:

0

Possible Values:

Any number between 0 and 1.

iparam.mio_presolve_aggregate (MSK_IPAR_MIO_PRESOLVE_AGGREGATE)

Controls whether the presolve used by the mixed-integer optimizer tries to aggregate the constraints.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.mio_presolve_probing (MSK_IPAR_MIO_PRESOLVE_PROBING)

Controls whether the mixed-integer presolve performs probing. Probing can be very time consuming.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.mio_presolve_use (`MSK_IPAR_MIO_PRESOLVE_USE`)

Controls whether presolve is performed by the mixed-integer optimizer.

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.mio_probing_level (`MSK_IPAR_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 parameter value:

-1

Possible Values:

An integer value in the range of -1 to 3.

iparam.mio_rins_max_nodes (`MSK_IPAR_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 parameter value:

-1

Possible Values:

Any number between -1 and +inf.

iparam.mio_root_optimizer (`MSK_IPAR_MIO_ROOT_OPTIMIZER`)

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

Default parameter value:

`optimizertype.free`

Possible values:

- `optimizertype.concurrent` The optimizer for nonconvex nonlinear problems.
- `optimizertype.conic` The optimizer for problems having conic constraints.
- `optimizertype.dual_simplex` The dual simplex optimizer is used.
- `optimizertype.free` The optimizer is chosen automatically.

- optimizertype.free_simplex One of the simplex optimizers is used.
- optimizertype.intpnt The interior-point optimizer is used.
- optimizertype.mixed_int The mixed-integer optimizer.
- optimizertype.mixed_int_conic The mixed-integer optimizer for conic and linear problems.
- optimizertype.network_primal_simplex The network primal simplex optimizer is used. It is only applicable to pure network problems.
- optimizertype.nonconvex The optimizer for nonconvex nonlinear problems.
- optimizertype.primal_dual_simplex The primal dual simplex optimizer is used.
- optimizertype.primal_simplex The primal simplex optimizer is used.

iparam.mio_strong_branch (MSK_IPAR_MIO_STRONG_BRANCH)

The value specifies the depth from the root in which strong branching is used. A negative value means that the optimizer chooses a default value automatically.

Default parameter value:

-1

Possible Values:

Any number between -inf and +inf.

iparam.mio_use_multithreaded_optimizer (MSK_IPAR_MIO_USE_MULTITHREADED_OPTIMIZER)

Controls wheter the new multithreaded optimizer should be used for Mixed integer problems.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.mt_spincount (MSK_IPAR_MT_SPINCOUNT)

Set the number of iterations to spin before sleeping.

Default parameter value:

0

Possible Values:

Any integer greater or equal to 0.

iparam.nonconvex_max_iterations (MSK_IPAR_NONCONVEX_MAX_ITERATIONS)

Maximum number of iterations that can be used by the nonconvex optimizer.

Default parameter value:

100000

Possible Values:

Any number between 0 and +inf.

iparam.num_threads (MSK_IPAR_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.

Default parameter value:

0

Possible Values:

Any integer greater or equal to 0.

iparam.opf_max_terms_per_line (MSK_IPAR_OPF_MAX_TERMS_PER_LINE)

The maximum number of terms (linear and quadratic) per line when an OPF file is written.

Default parameter value:

5

Possible Values:

Any number between 0 and +inf.

iparam.opf_write_header (MSK_IPAR_OPF_WRITE_HEADER)

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

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.opf_write_hints (MSK_IPAR_OPF_WRITE_HINTS)

Write a hint section with problem dimensions in the beginning of an OPF file.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.opf_write_parameters (MSK_IPAR_OPF_WRITE_PARAMETERS)

Write a parameter section in an OPF file.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.opf_write_problem (MSK_IPAR_OPF_WRITE_PROBLEM)

Write objective, constraints, bounds etc. to an OPF file.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.opf_write_sol_bas (MSK_IPAR_OPF_WRITE_SOL_BAS)

If iparam.opf_write_solutions is onoffkey.on and a basic solution is defined, include the basic solution in OPF files.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.opf_write_sol_itg (MSK_IPAR_OPF_WRITE_SOL_ITG)

If iparam.opf_write_solutions is onoffkey.on and an integer solution is defined, write the integer solution in OPF files.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.opf_write_sol_itr (MSK_IPAR_OPF_WRITE_SOL_ITR)

If iparam.opf_write_solutions is onoffkey.on and an interior solution is defined, write the interior solution in OPF files.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.opf_write_solutions (MSK_IPAR_OPF_WRITE_SOLUTIONS)

Enable inclusion of solutions in the OPF files.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.optimizer (MSK_IPAR_OPTIMIZER)

The paramter controls which optimizer is used to optimize the task.

Default parameter value:

optimizertype.free

Possible values:

- optimizertype.concurrent The optimizer for nonconvex nonlinear problems.
- optimizertype.conic The optimizer for problems having conic constraints.
- optimizertype.dual_simplex The dual simplex optimizer is used.
- optimizertype.free The optimizer is chosen automatically.
- optimizertype.free_simplex One of the simplex optimizers is used.
- optimizertype.intpnt The interior-point optimizer is used.
- optimizertype.mixed_int The mixed-integer optimizer.
- optimizertype.mixed_int_conic The mixed-integer optimizer for conic and linear problems.
- optimizertype.network_primal_simplex The network primal simplex optimizer is used. It is only applicable to pure network problems.
- optimizertype.nonconvex The optimizer for nonconvex nonlinear problems.
- optimizertype.primal_dual_simplex The primal dual simplex optimizer is used.
- optimizertype.primal_simplex The primal simplex optimizer is used.

iparam.param_read_case_name (MSK_IPAR_PARAM_READ_CASE_NAME)

If turned on, then names in the parameter file are case sensitive.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.param_read_ign_error (MSK_IPAR_PARAM_READ_IGN_ERROR)

If turned on, then errors in paramter settings is ignored.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.presolve_elim_fill (`MSK_IPAR_PRESOLVE_ELIM_FILL`)

Controls the maximum amount of fill-in that can be created during the elimination phase of the presolve. This parameter times (`numcon`+`numvar`) denotes the amount of fill-in.

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.presolve_eliminator_max_num_tries (`MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES`)

Control the maximum number of times the eliminator is tried.

Default parameter value:

-1

Possible Values:

A negative value implies MOSEK decides maximum number of times.

iparam.presolve_eliminator_use (`MSK_IPAR_PRESOLVE_ELIMINATOR_USE`)

Controls whether free or implied free variables are eliminated from the problem.

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.presolve_level (`MSK_IPAR_PRESOLVE_LEVEL`)

Currently not used.

Default parameter value:

-1

Possible Values:

Any number between -inf and +inf.

iparam.presolve_lindep_abs_work_trh (`MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH`)

The linear dependency check is potentially computationally expensive.

Default parameter value:

100

Possible Values:

Any number between 0 and +inf.

iparam.presolve_lindep_rel_work_trh (`MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH`)

The linear dependency check is potentially computationally expensive.

Default parameter value:

100

Possible Values:

Any number between 0 and +inf.

iparam.presolve_lindep_use (`MSK_IPAR_PRESOLVE_LINDEP_USE`)

Controls whether the linear constraints are checked for linear dependencies.

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.presolve_max_num_reductions (`MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS`)

Controls the maximum number reductions performed by the presolve. The value of the parameter is normally only changed in connection with debugging. A negative value implies that an infinite number of reductions are allowed.

Default parameter value:

-1

Possible Values:

Any number between -inf and +inf.

iparam.presolve_use (`MSK_IPAR_PRESOLVE_USE`)

Controls whether the presolve is applied to a problem before it is optimized.

Default parameter value:

`presolvemode.free`

Possible values:

- `presolvemode.free` It is decided automatically whether to presolve before the problem is optimized.
- `presolvemode.off` The problem is not presolved before it is optimized.
- `presolvemode.on` The problem is presolved before it is optimized.

iparam.primal_repair_optimizer (`MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER`)

Controls which optimizer that is used to find the optimal repair.

Default parameter value:

`optimizertype.free`

Possible values:

- `optimizertype.concurrent` The optimizer for nonconvex nonlinear problems.
- `optimizertype.conic` The optimizer for problems having conic constraints.

- `optimizertype.dual_simplex` The dual simplex optimizer is used.
- `optimizertype.free` The optimizer is chosen automatically.
- `optimizertype.free_simplex` One of the simplex optimizers is used.
- `optimizertype.intpnt` The interior-point optimizer is used.
- `optimizertype.mixed_int` The mixed-integer optimizer.
- `optimizertype.mixed_int_conic` The mixed-integer optimizer for conic and linear problems.
- `optimizertype.network_primal_simplex` The network primal simplex optimizer is used. It is only applicable to pure network problems.
- `optimizertype.nonconvex` The optimizer for nonconvex nonlinear problems.
- `optimizertype.primal_dual_simplex` The primal dual simplex optimizer is used.
- `optimizertype.primal_simplex` The primal simplex optimizer is used.

iparam.qo_separable_reformulation (`MSK_IPAR_QO_SEPARABLE_REFORMULATION`)

Determine if Quadratic programing problems should be reformulated to separable form.

Default parameter value:

`onoffkey.off`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.read_anz (`MSK_IPAR_READ_ANZ`)

Expected maximum number of A non-zeros to be read. The option is used only by fast MPS and LP file readers.

Default parameter value:

100000

Possible Values:

Any number between 0 and +inf.

iparam.read_con (`MSK_IPAR_READ_CON`)

Expected maximum number of constraints to be read. The option is only used by fast MPS and LP file readers.

Default parameter value:

10000

Possible Values:

Any number between 0 and +inf.

iparam.read_cone (`MSK_IPAR_READ_CONE`)

Expected maximum number of conic constraints to be read. The option is used only by fast MPS and LP file readers.

Default parameter value:

2500

Possible Values:

Any number between 0 and +inf.

iparam.read_data_compressed (`MSK_IPAR_READ_DATA_COMPRESSED`)

If this option is turned on,it is assumed that the data file is compressed.

Default parameter value:

`compresstype.free`

Possible values:

- `compresstype.free` The type of compression used is chosen automatically.
- `compresstype.gzip` The type of compression used is gzip compatible.
- `compresstype.none` No compression is used.

iparam.read_data_format (`MSK_IPAR_READ_DATA_FORMAT`)

Format of the data file to be read.

Default parameter value:

`dataformat.extension`

Possible values:

- `dataformat.cb` Conic benchmark format.
- `dataformat.extension` The file extension is used to determine the data file format.
- `dataformat.free_mps` The data data a free MPS formatted file.
- `dataformat.lp` The data file is LP formatted.
- `dataformat.mps` The data file is MPS formatted.
- `dataformat.op` The data file is an optimization problem formatted file.
- `dataformat.task` Generic task dump file.
- `dataformat.xml` The data file is an XML formatted file.

iparam.read_debug (`MSK_IPAR_READ_DEBUG`)

Turns on additional debugging information when reading files.

Default parameter value:

`onoffkey.off`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.read_keep_free_con (`MSK_IPAR_READ_KEEP_FREE_CON`)

Controls whether the free constraints are included in the problem.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.read_lp_drop_new_vars_in_bou (MSK_IPAR_READ_LP_DROP_NEW_VARS_IN_BOU)

If this option is turned on, MOSEK will drop variables that are defined for the first time in the bounds section.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.read_lp_quoted_names (MSK_IPAR_READ_LP_QUOTED_NAMES)

If a name is in quotes when reading an LP file, the quotes will be removed.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.read_mps_format (MSK_IPAR_READ_MPS_FORMAT)

Controls how strictly the MPS file reader interprets the MPS format.

Default parameter value:

mpsformat.relaxed

Possible values:

- mpsformat.free It is assumed that the input file satisfies the free MPS format. This implies that spaces are not allowed in names. Otherwise the format is free.
- mpsformat.relaxed It is assumed that the input file satisfies a slightly relaxed version of the MPS format.
- mpsformat.strict It is assumed that the input file satisfies the MPS format strictly.

iparam.read_mps_keep_int (MSK_IPAR_READ_MPS_KEEP_INT)

Controls whether MOSEK should keep the integer restrictions on the variables while reading the MPS file.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.read_mps_obj_sense (MSK_IPAR_READ_MPS_OBJ_SENSE)

If turned on, the MPS reader uses the objective sense section. Otherwise the MPS reader ignores it.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.read_mps_relax (MSK_IPAR_READ_MPS_RELAX)

If this option is turned on, then mixed integer constraints are ignored when a problem is read.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.read_mps_width (MSK_IPAR_READ_MPS_WIDTH)

Controls the maximal number of characters allowed in one line of the MPS file.

Default parameter value:

1024

Possible Values:

Any positive number greater than 80.

iparam.read_qnz (MSK_IPAR_READ_QNZ)

Expected maximum number of Q non-zeros to be read. The option is used only by MPS and LP file readers.

Default parameter value:

20000

Possible Values:

Any number between 0 and +inf.

iparam.read_task_ignore_param (MSK_IPAR_READ_TASK_IGNORE_PARAM)

Controls whether MOSEK should ignore the parameter setting defined in the task file and use the default parameter setting instead.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.read_var (MSK_IPAR_READ_VAR)

Expected maximum number of variable to be read. The option is used only by MPS and LP file readers.

Default parameter value:

10000

Possible Values:

Any number between 0 and +inf.

iparam.sensitivity_all (MSK_IPAR_SENSITIVITY_ALL)

If set to onoffkey.on, then Task.sensitivityreport analyzes all bounds and variables instead of reading a specification from the file.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.sensitivity_optimizer (MSK_IPAR_SENSITIVITY_OPTIMIZER)

Controls which optimizer is used for optimal partition sensitivity analysis.

Default parameter value:

optimizertype.free_simplex

Possible values:

- optimizertype.concurrent The optimizer for nonconvex nonlinear problems.
- optimizertype.conic The optimizer for problems having conic constraints.
- optimizertype.dual_simplex The dual simplex optimizer is used.
- optimizertype.free The optimizer is chosen automatically.
- optimizertype.free_simplex One of the simplex optimizers is used.
- optimizertype.intpnt The interior-point optimizer is used.
- optimizertype.mixed_int The mixed-integer optimizer.
- optimizertype.mixed_int_conic The mixed-integer optimizer for conic and linear problems.
- optimizertype.network_primal_simplex The network primal simplex optimizer is used. It is only applicable to pure network problems.

- optimizertype.nonconvex The optimizer for nonconvex nonlinear problems.
- optimizertype.primal_dual_simplex The primal dual simplex optimizer is used.
- optimizertype.primal_simplex The primal simplex optimizer is used.

iparam.sensitivity_type (MSK_IPAR_SENSITIVITY_TYPE)

Controls which type of sensitivity analysis is to be performed.

Default parameter value:

sensitivitytype.basis

Possible values:

- sensitivitytype.basis Basis sensitivity analysis is performed.
- sensitivitytype.optimal_partition Optimal partition sensitivity analysis is performed.

iparam.sim_basis_factor_use (MSK_IPAR_SIM_BASIS_FACTOR_USE)

Controls whether a (LU) factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penanlty.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.sim_degen (MSK_IPAR_SIM_DEGEN)

Controls how aggressively degeneration is handled.

Default parameter value:

simdegen.free

Possible values:

- simdegen.aggressive The simplex optimizer should use an aggressive degeneration strategy.
- simdegen.free The simplex optimizer chooses the degeneration strategy.
- simdegen.minimum The simplex optimizer should use a minimum degeneration strategy.
- simdegen.moderate The simplex optimizer should use a moderate degeneration strategy.
- simdegen.none The simplex optimizer should use no degeneration strategy.

iparam.sim_dual_crash (MSK_IPAR_SIM_DUAL_CRASH)

Controls whether crashing is performed in the dual simplex optimizer.

In general if a basis consists of more than (100-this parameter value)% fixed variables, then a crash will be performed.

Default parameter value:

90

Possible Values:

Any number between 0 and +inf.

iparam.sim_dual_phaseone_method (`MSK_IPAR_SIM_DUAL_PHASEONE_METHOD`)

An exprimental feature.

Default parameter value:

0

Possible Values:

Any number between 0 and 10.

iparam.sim_dual_restrict_selection (`MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION`)

The dual simplex optimizer can use a so-called restricted selection/pricing strategy to chooses the outgoing variable. Hence, if restricted selection is applied, then the dual simplex optimizer first choose a subset of all the potential outgoing variables. Next, for some time it will choose the outgoing variable only among the subset. From time to time the subset is redefined.

A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Default parameter value:

50

Possible Values:

Any number between 0 and 100.

iparam.sim_dual_selection (`MSK_IPAR_SIM_DUAL_SELECTION`)

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

Default parameter value:

`simseltype.free`

Possible values:

- `simseltype.ase` The optimizer uses approximate steepest-edge pricing.
- `simseltype.devex` The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).
- `simseltype.free` The optimizer chooses the pricing strategy.
- `simseltype.full` The optimizer uses full pricing.
- `simseltype.partial` The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.
- `simseltype.se` The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

iparam.sim_exploit_dupvec (`MSK_IPAR_SIM_EXPLOIT_DUPVEC`)

Controls if the simplex optimizers are allowed to exploit duplicated columns.

Default parameter value:

`simdupvec.off`

Possible values:

- `simdupvec.free` The simplex optimizer can choose freely.
- `simdupvec.off` Disallow the simplex optimizer to exploit duplicated columns.
- `simdupvec.on` Allow the simplex optimizer to exploit duplicated columns.

iparam.sim_hotstart (`MSK_IPAR_SIM_HOTSTART`)

Controls the type of hot-start that the simplex optimizer perform.

Default parameter value:

`simhotstart.free`

Possible values:

- `simhotstart.free` The simplex optimize chooses the hot-start type.
- `simhotstart.none` The simplex optimizer performs a coldstart.
- `simhotstart.status_keys` Only the status keys of the constraints and variables are used to choose the type of hot-start.

iparam.sim_hotstart_lu (`MSK_IPAR_SIM_HOTSTART_LU`)

Determines if the simplex optimizer should exploit the initial factorization.

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.sim_integer (`MSK_IPAR_SIM_INTEGER`)

An exprimental feature.

Default parameter value:

0

Possible Values:

Any number between 0 and 10.

iparam.sim_max_iterations (`MSK_IPAR_SIM_MAX_ITERATIONS`)

Maximum number of iterations that can be used by a simplex optimizer.

Default parameter value:

10000000

Possible Values:

Any number between 0 and +inf.

iparam.sim_max_num_setbacks (`MSK_IPAR_SIM_MAX_NUM_SETBACKS`)

Controls how many set-backs are allowed within a simplex optimizer. A set-back is an event where the optimizer moves in the wrong direction. This is impossible in theory but may happen due to numerical problems.

Default parameter value:

250

Possible Values:

Any number between 0 and +inf.

iparam.sim_non_singular (`MSK_IPAR_SIM_NON_SINGULAR`)

Controls if the simplex optimizer ensures a non-singular basis, if possible.

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.sim_primal_crash (`MSK_IPAR_SIM_PRIMAL_CRASH`)

Controls whether crashing is performed in the primal simplex optimizer.

In general, if a basis consists of more than (100-this parameter value)% fixed variables, then a crash will be performed.

Default parameter value:

90

Possible Values:

Any nonnegative integer value.

iparam.sim_primal_phaseone_method (`MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD`)

An exprimental feature.

Default parameter value:

0

Possible Values:

Any number between 0 and 10.

iparam.sim_primal_restrict_selection (`MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION`)

The primal simplex optimizer can use a so-called restricted selection/pricing strategy to chooses the outgoing variable. Hence, if restricted selection is applied, then the primal simplex optimizer first choose a subset of all the potential incoming variables. Next, for some time it will choose the incoming variable only among the subset. From time to time the subset is redefined.

A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Default parameter value:

50

Possible Values:

Any number between 0 and 100.

iparam.sim_primal_selection (MSK_IPAR_SIM_PRIMAL_SELECTION)

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

Default parameter value:

simseltype.free

Possible values:

- simseltype.ase The optimizer uses approximate steepest-edge pricing.
- simseltype.devex The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).
- simseltype.free The optimizer chooses the pricing strategy.
- simseltype.full The optimizer uses full pricing.
- simseltype.partial The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.
- simseltype.se The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

iparam.sim_refactor_freq (MSK_IPAR_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 parameter value:

0

Possible Values:

Any number between 0 and +inf.

iparam.sim_reformulation (MSK_IPAR_SIM_REFORMULATION)

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

Default parameter value:

simreform.off

Possible values:

- simreform.aggressive The simplex optimizer should use an aggressive reformulation strategy.

- simreform.free The simplex optimizer can choose freely.
- simreform.off Disallow the simplex optimizer to reformulate the problem.
- simreform.on Allow the simplex optimizer to reformulate the problem.

iparam.sim_save_lu (MSK_IPAR_SIM_SAVE_LU)

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

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.sim_scaling (MSK_IPAR_SIM_SCALING)

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

Default parameter value:

scalingtype.free

Possible values:

- scalingtype.aggressive A very aggressive scaling is performed.
- scalingtype.free The optimizer chooses the scaling heuristic.
- scalingtype.moderate A conservative scaling is performed.
- scalingtype.none No scaling is performed.

iparam.sim_scaling_method (MSK_IPAR_SIM_SCALING_METHOD)

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

Default parameter value:

scalingmethod.pow2

Possible values:

- scalingmethod.free The optimizer chooses the scaling heuristic.
- scalingmethod.pow2 Scales only with power of 2 leaving the mantissa untouched.

iparam.sim_solve_form (MSK_IPAR_SIM_SOLVE_FORM)

Controls whether the primal or the dual problem is solved by the primal-/dual- simplex optimizer.

Default parameter value:

solveform.free

Possible values:

- solveform.dual The optimizer should solve the dual problem.
- solveform.free The optimizer is free to solve either the primal or the dual problem.
- solveform.primal The optimizer should solve the primal problem.

iparam.sim_stability_priority (MSK_IPAR_SIM_STABILITY_PRIORITY)

Controls how high priority the numerical stability should be given.

Default parameter value:

50

Possible Values:

Any number between 0 and 100.

iparam.sim_switch_optimizer (MSK_IPAR_SIM_SWITCH_OPTIMIZER)

The simplex optimizer sometimes chooses to solve the dual problem instead of the primal problem. This implies that if you have chosen to use the dual simplex optimizer and the problem is dualized, then it actually makes sense to use the primal simplex optimizer instead. If this parameter is on and the problem is dualized and furthermore the simplex optimizer is chosen to be the primal (dual) one, then it is switched to the dual (primal).

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.sol_filter_keep_basic (MSK_IPAR_SOL_FILTER_KEEP_BASIC)

If turned on, then basic and super basic constraints and variables are written to the solution file independent of the filter setting.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.sol_filter_keep_ranged (MSK_IPAR_SOL_FILTER_KEEP_RANGED)

If turned on, then ranged constraints and variables are written to the solution file independent of the filter setting.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.sol_read_name_width (MSK_IPAR_SOL_READ_NAME_WIDTH)

When a solution is read by MOSEK and some constraint, variable or cone names contain blanks, then a maximum name width much be specified. A negative value implies that no name contain blanks.

Default parameter value:

-1

Possible Values:

Any number between -inf and +inf.

iparam.sol_read_width (`MSK_IPAR_SOL_READ_WIDTH`)

Controls the maximal acceptable width of line in the solutions when read by MOSEK.

Default parameter value:

1024

Possible Values:

Any positive number greater than 80.

iparam.solution_callback (`MSK_IPAR_SOLUTION_CALLBACK`)

Indicates whether solution call-backs will be performed during the optimization.

Default parameter value:

`onoffkey.off`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.timing_level (`MSK_IPAR_TIMING_LEVEL`)

Controls the a amount of timing performed inside MOSEK.

Default parameter value:

1

Possible Values:

Any integer greater or equal to 0.

iparam.warning_level (`MSK_IPAR_WARNING_LEVEL`)

Deprecated and not in use

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.write_bas_constraints (`MSK_IPAR_WRITE_BAS_CONSTRAINTS`)

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

Default parameter value:

`onoffkey.on`

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_bas_head (MSK_IPAR_WRITE_BAS_HEAD)

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

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_bas_variables (MSK_IPAR_WRITE_BAS_VARIABLES)

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

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_data_compressed (MSK_IPAR_WRITE_DATA_COMPRESSED)

Controls whether the data file is compressed while it is written. 0 means no compression while higher values mean more compression.

Default parameter value:

0

Possible Values:

Any number between 0 and +inf.

iparam.write_data_format (MSK_IPAR_WRITE_DATA_FORMAT)

Controls the data format when a task is written using Task.writedata.

Default parameter value:

dataformat.extension

Possible values:

- dataformat.cb Conic benchmark format.
- dataformat.extension The file extension is used to determine the data file format.
- dataformat.free_mps The data data a free MPS formatted file.
- dataformat.lp The data file is LP formatted.
- dataformat.mps The data file is MPS formatted.
- dataformat.op The data file is an optimization problem formatted file.
- dataformat.task Generic task dump file.

- `dataformat.xml` The data file is an XML formatted file.

iparam.write_data_param (`MSK_IPAR_WRITE_DATA_PARAM`)

If this option is turned on the parameter settings are written to the data file as parameters.

Default parameter value:

`onoffkey.off`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.write_free_con (`MSK_IPAR_WRITE_FREE_CON`)

Controls whether the free constraints are written to the data file.

Default parameter value:

`onoffkey.off`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.write_generic_names (`MSK_IPAR_WRITE_GENERIC_NAMES`)

Controls whether the generic names or user-defined names are used in the data file.

Default parameter value:

`onoffkey.off`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.write_generic_names_io (`MSK_IPAR_WRITE_GENERIC_NAMES_IO`)

Index origin used in generic names.

Default parameter value:

1

Possible Values:

Any number between 0 and +inf.

iparam.write_ignore_incompatible_conic_items (`MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_CONIC_ITEMS`)

If the output format is not compatible with conic quadratic problems this parameter controls if the writer ignores the conic parts or produces an error.

Default parameter value:

`onoffkey.off`

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_ignore_incompatible_items (MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS)

Controls if the writer ignores incompatible problem items when writing files.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_ignore_incompatible_nl_items (MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_NL_ITEMS)

Controls if the writer ignores general non-linear terms or produces an error.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_ignore_incompatible_psd_items (MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_PSD_ITEMS)

If the output format is not compatible with semidefinite problems this parameter controls if the writer ignores the conic parts or produces an error.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_int_constraints (MSK_IPAR_WRITE_INT_CONSTRAINTS)

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

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_int_head (MSK_IPAR_WRITE_INT_HEAD)

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

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_int_variables (MSK_IPAR_WRITE_INT_VARIABLES)

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

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_lp_line_width (MSK_IPAR_WRITE_LP_LINE_WIDTH)

Maximum width of line in an LP file written by MOSEK.

Default parameter value:

80

Possible Values:

Any positive number.

iparam.write_lp_quoted_names (MSK_IPAR_WRITE_LP_QUOTED_NAMES)

If this option is turned on, then MOSEK will quote invalid LP names when writing an LP file.

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_lp_strict_format (MSK_IPAR_WRITE_LP_STRICT_FORMAT)

Controls whether LP output files satisfy the LP format strictly.

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_lp_terms_per_line (MSK_IPAR_WRITE_LP_TERMS_PER_LINE)

Maximum number of terms on a single line in an LP file written by MOSEK. 0 means unlimited.

Default parameter value:

10

Possible Values:

Any number between 0 and +inf.

iparam.write_mps_int (`MSK_IPAR_WRITE_MPS_INT`)

Controls if marker records are written to the MPS file to indicate whether variables are integer restricted.

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.write_precision (`MSK_IPAR_WRITE_PRECISION`)

Controls the precision with which `double` numbers are printed in the MPS data file. In general it is not worthwhile to use a value higher than 15.

Default parameter value:

8

Possible Values:

Any number between 0 and +inf.

iparam.write_sol_barvariables (`MSK_IPAR_WRITE_SOL_BARVARIABLES`)

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

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.write_sol_constraints (`MSK_IPAR_WRITE_SOL_CONSTRAINTS`)

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

Default parameter value:

`onoffkey.on`

Possible values:

- `onoffkey.off` Switch the option off.
- `onoffkey.on` Switch the option on.

iparam.write_sol_head (`MSK_IPAR_WRITE_SOL_HEAD`)

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

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_sol_ignore_invalid_names (MSK_IPAR_WRITE_SOL_IGNORE_INVALID_NAMES)

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

Default parameter value:

onoffkey.off

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_sol_variables (MSK_IPAR_WRITE_SOL_VARIABLES)

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

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_task_inc_sol (MSK_IPAR_WRITE_TASK_INC_SOL)

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

Default parameter value:

onoffkey.on

Possible values:

- onoffkey.off Switch the option off.
- onoffkey.on Switch the option on.

iparam.write_xml_mode (MSK_IPAR_WRITE_XML_MODE)

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

Default parameter value:

xmlwriteroutputtype.row

Possible values:

- xmlwriteroutputtype.col Write in column order.
- xmlwriteroutputtype.row Write in row order.

Chapter 4

Mosek file formats

MOSEK supports a range of problem and solution formats. The Task formats is MOSEK's native binary format and it supports all features that MOSEK supports. OPF is the corresponding ASCII format and this supports nearly all features (everything except semidefinite problems). In general, the text formats are significantly slower to read, but they can be examined and edited directly in any text editor.

MOSEK supports GZIP compression of files. Problem files with an additional ".gz" extension are assumed to be compressed when read, and is automatically compressed when written. For example, a file called

```
problem.mps.gz
```

will be read as a GZIP compressed MPS file.

4.1 The MPS file format

MOSEK supports the standard MPS format with some extensions. For a detailed description of the MPS format see the book by Nazareth [2].

4.1.1 MPS file structure

The version of the MPS format supported by MOSEK allows specification of an optimization problem on the form

$$\begin{array}{ccccc} l^c & \leq & Ax + q(x) & \leq & u^c, \\ l^x & \leq & x & \leq & u^x, \\ & & x \in \mathcal{C}, & & \\ & & x_{\mathcal{J}} \text{ integer}, & & \end{array} \tag{4.1}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \to \mathbb{R}$ is a vector of quadratic functions. Hence,

  $$q_i(x) = 1/2 x^T Q^i x$$

  where it is assumed that

  $$Q^i = (Q^i)^T.$$

  Please note the explicit 1/2 in the quadratic term and that Q^i is required to be symmetric.
- $\mathcal{C}$ is a convex cone.
- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer-constrained variables.

An MPS file with one row and one column can be illustrated like this:

```
*        1         2         3         4         5         6
*23456789012345678901234567890123456789012345678901234567890
NAME          [name]
OBJSENSE
    [objsense]
OBJNAME
    [objname]
ROWS
 ?  [cname1]
COLUMNS
    [vname1]  [cname1]    [value1]     [vname3]  [value2]
RHS
    [name]    [cname1]    [value1]     [cname2]  [value2]
RANGES
    [name]    [cname1]    [value1]     [cname2]  [value2]
QSECTION      [cname1]
    [vname1]  [vname2]    [value1]     [vname3]  [value2]
BOUNDS
 ?? [name]    [vname1]    [value1]
CSECTION      [kname1]    [value1]     [ktype]
    [vname1]
ENDATA
```

Here the names in capitals are keywords of the MPS format and names in brackets are custom defined names or values. A couple of notes on the structure:

Fields:

All items surrounded by brackets appear in *fields*. The fields named "valueN" are numerical values. Hence, they must have the format

```
[+|-]XXXXXXX.XXXXXX[[e|E][+|-]XXX]
```

where

```
X = [0|1|2|3|4|5|6|7|8|9].
```

Sections:

The MPS file consists of several sections where the names in capitals indicate the beginning of a new section. For example, COLUMNS denotes the beginning of the columns section.

Comments:

Lines starting with an "*" are comment lines and are ignored by MOSEK.

Keys:

The question marks represent keys to be specified later.

Extensions:

The sections QSECTION and CSECTION are MOSEK specific extensions of the MPS format.

The standard MPS format is a fixed format, i.e. everything in the MPS file must be within certain fixed positions. MOSEK also supports a *free format*. See Section 4.1.5 for details.

4.1.1.1 Linear example lo1.mps

A concrete example of a MPS file is presented below:

```
* File: lo1.mps
NAME          lo1
OBJSENSE
    MAX
ROWS
 N  obj
 E  c1
 G  c2
 L  c3
COLUMNS
    x1        obj       3
    x1        c1        3
    x1        c2        2
    x2        obj       1
    x2        c1        1
    x2        c2        1
    x2        c3        2
    x3        obj       5
    x3        c1        2
    x3        c2        3
    x4        obj       1
    x4        c2        1
    x4        c3        3
```

```
RHS
    rhs       c1        30
    rhs       c2        15
    rhs       c3        25
RANGES
BOUNDS
 UP bound     x2        10
ENDATA
```

Subsequently each individual section in the MPS format is discussed.

4.1.1.2 NAME

In this section a name (`[name]`) is assigned to the problem.

4.1.1.3 OBJSENSE (optional)

This is an optional section that can be used to specify the sense of the objective function. The `OBJSENSE` section contains one line at most which can be one of the following

```
MIN
MINIMIZE
MAX
MAXIMIZE
```

It should be obvious what the implication is of each of these four lines.

4.1.1.4 OBJNAME (optional)

This is an optional section that can be used to specify the name of the row that is used as objective function. The `OBJNAME` section contains one line at most which has the form

```
objname
```

`objname` should be a valid row name.

4.1.1.5 ROWS

A record in the `ROWS` section has the form

```
 ?  [cname1]
```

where the requirements for the fields are as follows:

Field	Starting position	Maximum width	Required	Description
?	2	1	Yes	Constraint key
`[cname1]`	5	8	Yes	Constraint name

Hence, in this section each constraint is assigned an unique name denoted by `[cname1]`. Please note that `[cname1]` starts in position 5 and the field can be at most 8 characters wide. An initial key (?)

must be present to specify the type of the constraint. The key can have the values `E`, `G`, `L`, or `N` with the following interpretation:

Constraint type	l_i^c	u_i^c
`E`	finite	l_i^c
`G`	finite	∞
`L`	$-\infty$	finite
`N`	$-\infty$	∞

In the MPS format an objective vector is not specified explicitly, but one of the constraints having the key `N` will be used as the objective vector c . In general, if multiple `N` type constraints are specified, then the first will be used as the objective vector c .

4.1.1.6 `COLUMNS`

In this section the elements of A are specified using one or more records having the form

```
 [vname1]  [cname1]    [value1]     [cname2]  [value2]
```

where the requirements for each field are as follows:

Field	Starting position	Maximum width	Re-quired	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.

4.1.1.7 `RHS` (optional)

A record in this section has the format

```
 [name]    [cname1]    [value1]     [cname2]  [value2]
```

where the requirements for each field are as follows:

Field	Starting position	Maximum width	Required	Description
[name]	5	8	Yes	Name of the RHS vector
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

The interpretation of a record is that [name] is the name of the RHS vector to be specified. In general, several vectors can be specified. [cname1] denotes a constraint name previously specified in the ROWS section. Now, assume that this name has been assigned to the i th constraint and v_1 denotes the value specified by [value1], then the interpretation of v_1 is:

Constraint type	l_i^c	u_i^c
E	v_1	v_1
G	v_1	
L		v_1
N		

An optional second element is specified by [cname2] and [value2] and is interpreted in the same way. Please note that it is not necessary to specify zero elements, because elements are assumed to be zero.

4.1.1.8 RANGES (optional)

A record in this section has the form

```
[name]    [cname1]    [value1]     [cname2]  [value2]
```

where the requirements for each fields are as follows:

Field	Starting position	Maximum width	Required	Description
[name]	5	8	Yes	Name of the RANGE vector
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

The records in this section are used to modify the bound vectors for the constraints, i.e. the values in l^c and u^c . A record has the following interpretation: [name] is the name of the RANGE vector and [cname1] is a valid constraint name. Assume that [cname1] is assigned to the i th constraint and let v_1 be the value specified by [value1], then a record has the interpretation:

Constraint type	Sign of v_1	l_i^c	u_i^c
`E`	-	$u_i^c + v_1$	
`E`	+		$l_i^c + v_1$
`G`	- or +		$l_i^c + \|v_1\|$
`L`	- or +	$u_i^c - \|v_1\|$	
`N`			

4.1.1.9 `QSECTION` (optional)

Within the `QSECTION` the label `[cname1]` must be a constraint name previously specified in the `ROWS` section. The label `[cname1]` denotes the constraint to which the quadratic term belongs. A record in the `QSECTION` has the form

```
 [vname1]  [vname2]    [value1]     [vname3]  [value2]
```

where the requirements for each field are:

Field	Starting position	Maximum width	Required	Description
`[vname1]`	5	8	Yes	Variable name
`[vname2]`	15	8	Yes	Variable name
`[value1]`	25	12	Yes	Numerical value
`[vname3]`	40	8	No	Variable name
`[value2]`	50	12	No	Numerical value

A record specifies one or two elements in the lower triangular part of the Q^i matrix where `[cname1]` specifies the i . Hence, if the names `[vname1]` and `[vname2]` have been assigned to the k th and j th variable, then Q^i_{kj} is assigned the value given by `[value1]` An optional second element is specified in the same way by the fields `[vname1]`, `[vname3]`, and `[value2]`.

The example

$$
\begin{array}{lccc}
\text{minimize} & -x_2 + 0.5(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) & & \\
\text{subject to} & x_1 + x_2 + x_3 & \geq & 1, \\
& x \geq 0 & &
\end{array}
$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
 N  obj
 G  c1
COLUMNS
    x1        c1        1.0
    x2        obj       -1.0
    x2        c1        1.0
    x3        c1        1.0
RHS
    rhs       c1        1.0
```

```
QSECTION      obj
    x1        x1        2.0
    x1        x3        -1.0
    x2        x2        0.2
    x3        x3        2.0
ENDATA
```

Regarding the QSECTIONs please note that:

- Only one QSECTION is allowed for each constraint.
- The QSECTIONs can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- All entries specified in a QSECTION are assumed to belong to the lower triangular part of the quadratic term of Q .

4.1.1.10 BOUNDS (optional)

In the BOUNDS section changes to the default bounds vectors l^x and u^x are specified. The default bounds vectors are $l^x = 0$ and $u^x = \infty$. Moreover, it is possible to specify several sets of bound vectors. A record in this section has the form

```
?? [name]    [vname1]    [value1]
```

where the requirements for each field are:

Field	Starting position	Maximum width	Required	Description
??	2	2	Yes	Bound key
[name]	5	8	Yes	Name of the BOUNDS vector
[vname1]	15	8	Yes	Variable name
[value1]	25	12	No	Numerical value

Hence, a record in the BOUNDS section has the following interpretation: [name] is the name of the bound vector and [vname1] is the name of the variable which bounds are modified by the record. ?? and [value1] are used to modify the bound vectors according to the following table:

??	l_j^x	u_j^x	Made integer (added to $\mathcal{J}$)
FR	$-\infty$	∞	No
FX	v_1	v_1	No
LO	v_1	unchanged	No
MI	$-\infty$	unchanged	No
PL	unchanged	∞	No
UP	unchanged	v_1	No
BV	0	1	Yes
LI	$\lceil v_1 \rceil$	*unchanged*	Yes
UI	unchanged	$\lfloor v_1 \rfloor$	Yes

v_1 is the value specified by [value1].

4.1.1.11 CSECTION (optional)

The purpose of the CSECTION is to specify the constraint

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

in (4.1).

It is assumed that $\mathcal{C}$ satisfies the following requirements. Let

$$x^t \in \mathbb{R}^{n^t},\ t = 1, \ldots, k$$

be vectors comprised of parts of the decision variables x so that each decision variable is a member of exactly **one** vector x^t , for example

$$x^1 = \begin{bmatrix} x_1 \\ x_4 \\ x_7 \end{bmatrix} \text{ and } x^2 = \begin{bmatrix} x_6 \\ x_5 \\ x_3 \\ x_2 \end{bmatrix}.$$

Next define

$$\mathcal{C} := \left\{ x \in \mathbb{R}^n :\ x^t \in \mathcal{C}_t,\ t = 1, \ldots, k \right\}$$

where $\mathcal{C}_t$ must have one of the following forms

- $\mathbb{R}$ set:

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

- Quadratic cone:

$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \tag{4.2}$$

- Rotated quadratic cone:

$$\mathcal{C}_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2,\ x_1, x_2 \geq 0 \right\}. \tag{4.3}$$

In general, only quadratic and rotated quadratic cones are specified in the MPS file whereas membership of the $\mathbb{R}$ set is not. If a variable is not a member of any other cone then it is assumed to be a member of an $\mathbb{R}$ cone.

Next, let us study an example. Assume that the quadratic cone

$$x_4 \geq \sqrt{x_5^2 + x_8^2} \tag{4.4}$$

and the rotated quadratic cone

$$2x_3x_7 \geq x_1^2 + x_0^2, \; x_3, x_7 \geq 0, \tag{4.5}$$

should be specified in the MPS file. One CSECTION is required for each cone and they are specified as follows:

```
*        1         2         3         4         5         6
*23456789012345678901234567890123456789012345678901234567890
CSECTION      konea       0.0          QUAD
    x4
    x5
    x8
CSECTION      koneb       0.0          RQUAD
    x7
    x3
    x1
    x0
```

This first CSECTION specifies the cone (4.4) which is given the name konea. This is a quadratic cone which is specified by the keyword QUAD in the CSECTION header. The 0.0 value in the CSECTION header is not used by the QUAD cone.

The second CSECTION specifies the rotated quadratic cone (4.5). Please note the keyword RQUAD in the CSECTION which is used to specify that the cone is a rotated quadratic cone instead of a quadratic cone. The 0.0 value in the CSECTION header is not used by the RQUAD cone.

In general, a CSECTION header has the format

```
CSECTION      [kname1]    [value1]     [ktype]
```

where the requirement for each field are as follows:

Field	Starting position	Maximum width	Required	Description
[kname1]	5	8	Yes	Name of the cone
[value1]	15	12	No	Cone parameter
[ktype]	25		Yes	Type of the cone.

The possible cone type keys are:

Cone type key	Members	Interpretation.
QUAD	≥ 1	Quadratic cone i.e. (4.2).
RQUAD	≥ 2	Rotated quadratic cone i.e. (4.3).

Please note that a quadratic cone must have at least one member whereas a rotated quadratic cone must have at least two members. A record in the CSECTION has the format

```
[vname1]
```

where the requirements for each field are

Field	Starting position	Maximum width	Re-quired	Description
[vname1]	2	8	Yes	A valid variable name

The most important restriction with respect to the CSECTION is that a variable must occur in only one CSECTION.

4.1.1.12 ENDATA

This keyword denotes the end of the MPS file.

4.1.2 Integer variables

Using special bound keys in the BOUNDS section it is possible to specify that some or all of the variables should be integer-constrained i.e. be members of $\mathcal{J}$. However, an alternative method is available.

This method is available only for backward compatibility and we recommend that it is not used. This method requires that markers are placed in the COLUMNS section as in the example:

```
COLUMNS
    x1        obj       -10.0         c1        0.7
    x1        c2        0.5           c3        1.0
    x1        c4        0.1
* Start of integer-constrained variables.
    MARK000   'MARKER'                'INTORG'
    x2        obj       -9.0          c1        1.0
    x2        c2        0.8333333333  c3        0.66666667
    x2        c4        0.25
    x3        obj       1.0           c6        2.0
    MARK001   'MARKER'                'INTEND'
* End of integer-constrained variables.
```

Please note that special marker lines are used to indicate the start and the end of the integer variables. Furthermore be aware of the following

- IMPORTANT: All variables between the markers are assigned a default lower bound of 0 and a default upper bound of 1. **This may not be what is intended.** If it is not intended, the correct bounds should be defined in the BOUNDS section of the MPS formatted file.
- MOSEK ignores field 1, i.e. MARK0001 and MARK001, however, other optimization systems require them.
- Field 2, i.e. 'MARKER', must be specified including the single quotes. This implies that no row can be assigned the name 'MARKER'.

- Field 3 is ignored and should be left blank.
- Field 4, i.e. `'INTORG'` and `'INTEND'`, must be specified.
- It is possible to specify several such integer marker sections within the `COLUMNS` section.

4.1.3 General limitations

- An MPS file should be an ASCII file.

4.1.4 Interpretation of the MPS format

Several issues related to the MPS format are not well-defined by the industry standard. However, MOSEK uses the following interpretation:

- If a matrix element in the `COLUMNS` section is specified multiple times, then the multiple entries are added together.
- If a matrix element in a `QSECTION` section is specified multiple times, then the multiple entries are added together.

4.1.5 The free MPS format

MOSEK supports a free format variation of the MPS format. The free format is similar to the MPS file format but less restrictive, e.g. it allows longer names. However, it also presents two main limitations:

- By default a line in the MPS file must not contain more than 1024 characters. However, by modifying the parameter `iparam.read_mps_width` an arbitrary large line width will be accepted.
- A name must not contain any blanks.

To use the free MPS format instead of the default MPS format the MOSEK parameter `iparam.read_mps_format` should be changed.

4.2 The LP file format

MOSEK supports the LP file format with some extensions i.e. MOSEK can read and write LP formatted files.

Please note that the LP format is not a completely well-defined standard and hence different optimization packages may interpret the same LP file in slightly different ways. MOSEK tries to emulate as closely as possible CPLEX's behavior, but tries to stay backward compatible.

The LP file format can specify problems on the form

$$\begin{array}{lcccccc} \text{minimize/maximize} & & & c^T x + \frac{1}{2} q^o(x) & & \\ \text{subject to} & l^c & \leq & Ax + \frac{1}{2} q(x) & \leq & u^c, \\ & l^x & \leq & x & \leq & u^x, \\ & & & x_{\mathcal{J}} \text{integer}, & & \end{array}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear term in the objective.
- $q^o :\in \mathbb{R}^n \rightarrow \mathbb{R}$ is the quadratic term in the objective where

  $$q^o(x) = x^T Q^o x$$

  and it is assumed that

  $$Q^o = (Q^o)^T.$$

- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \rightarrow \mathbb{R}$ is a vector of quadratic functions. Hence,

  $$q_i(x) = x^T Q^i x$$

  where it is assumed that

  $$Q^i = (Q^i)^T.$$

- $\mathcal{J} \subseteq \{1, 2, \ldots, n\}$ is an index set of the integer constrained variables.

4.2.1 The sections

An LP formatted file contains a number of sections specifying the objective, constraints, variable bounds, and variable types. The section keywords may be any mix of upper and lower case letters.

4.2.1.1 The objective

The first section beginning with one of the keywords

```
max
maximum
maximize
min
minimum
minimize
```

defines the objective sense and the objective function, i.e.

$$c^T x + \frac{1}{2} x^T Q^o x.$$

The objective may be given a name by writing

```
myname:
```

before the expressions. If no name is given, then the objective is named `obj`.

The objective function contains linear and quadratic terms. The linear terms are written as

```
4 x1 + x2 - 0.1 x3
```

and so forth. The quadratic terms are written in square brackets (`[]`) and are either squared or multiplied as in the examples

```
x1^2
```

and

```
x1 * x2
```

There may be zero or more pairs of brackets containing quadratic expressions.

An example of an objective section is:

```
minimize
myobj: 4 x1 + x2 - 0.1 x3 + [ x1^2 + 2.1 x1 * x2 ]/2
```

Please note that the quadratic expressions are multiplied with $\frac{1}{2}$, so that the above expression means

$$\text{minimize} \quad 4x_1 + x_2 - 0.1 \cdot x_3 + \frac{1}{2}(x_1^2 + 2.1 \cdot x_1 \cdot x_2)$$

If the same variable occurs more than once in the linear part, the coefficients are added, so that `4 x1 + 2 x1` is equivalent to `6 x1`. In the quadratic expressions `x1 * x2` is equivalent to `x2 * x1` and as in the linear part , if the same variables multiplied or squared occur several times their coefficients are added.

4.2.1.2 The constraints

The second section beginning with one of the keywords

```
subj to
subject to
s.t.
```

```
st
```

defines the linear constraint matrix (A) and the quadratic matrices (Q^i).

A constraint contains a name (optional), expressions adhering to the same rules as in the objective and a bound:

```
subject to
con1: x1 + x2 + [ x3^2 ]/2 <= 5.1
```

The bound type (here `<=`) may be any of `<`, `<=`, `=`, `>`, `>=` (`<` and `<=` mean the same), and the bound may be any number.

In the standard LP format it is not possible to define more than one bound, but MOSEK supports defining ranged constraints by using double-colon (`'’::'’`) instead of a single-colon (”`:`”) after the constraint name, i.e.

$$-5 \leq x_1 + x_2 \leq 5 \tag{4.6}$$

may be written as

```
con:: -5 < x_1 + x_2 < 5
```

By default MOSEK writes ranged constraints this way.

If the files must adhere to the LP standard, ranged constraints must either be split into upper bounded and lower bounded constraints or be written as en equality with a slack variable. For example the expression (4.6) may be written as

$$x_1 + x_2 - sl_1 = 0, -5 \leq sl_1 \leq 5.$$

4.2.1.3 Bounds

Bounds on the variables can be specified in the bound section beginning with one of the keywords

```
bound
bounds
```

The bounds section is optional but should, if present, follow the `subject to` section. All variables listed in the bounds section must occur in either the objective or a constraint.

The default lower and upper bounds are 0 and $+\infty$. A variable may be declared free with the keyword `free`, which means that the lower bound is $-\infty$ and the upper bound is $+\infty$. Furthermore it may be assigned a finite lower and upper bound. The bound definitions for a given variable may be written in one or two lines, and bounds can be any number or $\pm\infty$ (written as `+inf`/`-inf`/`+infinity`/`-infinity`) as in the example

```
bounds
x1 free
x2 <= 5
0.1 <= x2
x3 = 42
2 <= x4 < +inf
```

4.2.1.4 Variable types

The final two sections are optional and must begin with one of the keywords

```
bin
binaries
binary
```

and

```
gen
general
```

Under `general` all integer variables are listed, and under `binary` all binary (integer variables with bounds 0 and 1) are listed:

```
general
x1 x2
binary
x3 x4
```

Again, all variables listed in the binary or general sections must occur in either the objective or a constraint.

4.2.1.5 Terminating section

Finally, an LP formatted file must be terminated with the keyword

```
end
```

4.2.1.6 Linear example `lo1.lp`

A simple example of an LP file is:

```
\ File: lo1.lp
maximize
obj: 3 x1 + x2 + 5 x3 + x4
subject to
c1:  3 x1 + x2 + 2 x3 = 30
c2:  2 x1 + x2 + 3 x3 + x4 >= 15
c3:  2 x2 + 3 x4 <= 25
bounds
 0 <= x1 <= +infinity
 0 <= x2 <= 10
 0 <= x3 <= +infinity
 0 <= x4 <= +infinity
end
```

4.2.1.7 Mixed integer example `milo1.lp`

```
maximize
obj: x1 + 6.4e-01 x2
subject to
c1:  5e+01 x1 + 3.1e+01 x2 <= 2.5e+02
c2:  3e+00 x1 - 2e+00 x2 >= -4e+00
```

```
bounds
 0 <= x1 <= +infinity
 0 <= x2 <= +infinity
general
 x1 x2
end
```

4.2.2 LP format peculiarities

4.2.2.1 Comments

Anything on a line after a "\" is ignored and is treated as a comment.

4.2.2.2 Names

A name for an objective, a constraint or a variable may contain the letters a-z, A-Z, the digits 0-9 and the characters

```
!"#$%&()/,.;?@_'`|~
```

The first character in a name must not be a number, a period or the letter 'e' or 'E'. Keywords must not be used as names.

MOSEK accepts any character as valid for names, except `'\0'`. When writing a name that is not allowed in LP files, it is changed and a warning is issued.

The algorithm for making names LP valid works as follows: The name is interpreted as an `utf-8` string. For a unicode character `c`:

- If `c=='_'` (underscore), the output is `'__'` (two underscores).
- If `c` is a valid LP name character, the output is just `c`.
- If `c` is another character in the ASCII range, the output is `_XX`, where `XX` is the hexadecimal code for the character.
- If `c` is a character in the range 127—65535, the output is `_uXXXX`, where `XXXX` is the hexadecimal code for the character.
- If `c` is a character above 65535, the output is `_UXXXXXXXX`, where `XXXXXXXX` is the hexadecimal code for the character.

Invalid `utf-8` substrings are escaped as `'_XX'`, and if a name starts with a period, 'e' or 'E', that character is escaped as `'_XX'`.

4.2.2.3 Variable bounds

Specifying several upper or lower bounds on one variable is possible but MOSEK uses only the tightest bounds. If a variable is fixed (with `=`), then it is considered the tightest bound.

4.2.2.4 MOSEK specific extensions to the LP format

Some optimization software packages employ a more strict definition of the LP format that the one used by MOSEK. The limitations imposed by the strict LP format are the following:

- Quadratic terms in the constraints are not allowed.
- Names can be only 16 characters long.
- Lines must not exceed 255 characters in length.

If an LP formatted file created by MOSEK should satisfies the strict definition, then the parameter

`iparam.write_lp_strict_format`

should be set; note, however, that some problems cannot be written correctly as a strict LP formatted file. For instance, all names are truncated to 16 characters and hence they may loose their uniqueness and change the problem.

To get around some of the inconveniences converting from other problem formats, MOSEK allows lines to contain 1024 characters and names may have any length (shorter than the 1024 characters).

Internally in MOSEK names may contain any (printable) character, many of which cannot be used in LP names. Setting the parameters

`iparam.read_lp_quoted_names`

and

`iparam.write_lp_quoted_names`

allows MOSEK to use quoted names. The first parameter tells MOSEK to remove quotes from quoted names e.g, `"x1"`, when reading LP formatted files. The second parameter tells MOSEK to put quotes around any semi-illegal name (names beginning with a number or a period) and fully illegal name (containing illegal characters). As double quote is a legal character in the LP format, quoting semi-illegal names makes them legal in the pure LP format as long as they are still shorter than 16 characters. Fully illegal names are still illegal in a pure LP file.

4.2.3 The strict LP format

The LP format is not a formal standard and different vendors have slightly different interpretations of the LP format. To make MOSEK's definition of the LP format more compatible with the definitions of other vendors, use the parameter setting

`iparam.write_lp_strict_format` = `onoffkey.on`

This setting may lead to truncation of some names and hence to an invalid LP file. The simple solution to this problem is to use the parameter setting iparam.write_generic_names = onoffkey.on

which will cause all names to be renamed systematically in the output file.

4.2.4 Formatting of an LP file

A few parameters control the visual formatting of LP files written by MOSEK in order to make it easier to read the files. These parameters are

iparam.write_lp_line_width

iparam.write_lp_terms_per_line

The first parameter sets the maximum number of characters on a single line. The default value is 80 corresponding roughly to the width of a standard text document.

The second parameter sets the maximum number of terms per line; a term means a sign, a coefficient, and a name (for example "`+ 42 elephants`"). The default value is 0, meaning that there is no maximum.

4.2.4.1 Speeding up file reading

If the input file should be read as fast as possible using the least amount of memory, then it is important to tell MOSEK how many non-zeros, variables and constraints the problem contains. These values can be set using the parameters

iparam.read_con

iparam.read_var

iparam.read_anz

iparam.read_qnz

4.2.4.2 Unnamed constraints

Reading and writing an LP file with MOSEK may change it superficially. If an LP file contains unnamed constraints or objective these are given their generic names when the file is read (however unnamed constraints in MOSEK are written without names).

4.3 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.

4.3.1 Intended use

The OPF file format is meant to replace several other files:

- The LP file format. Any problem that can be written as an LP file can be written as an OPF file to; furthermore it naturally accommodates ranged constraints and variables as well as arbitrary characters in names, fixed expressions in the objective, empty constraints, and conic constraints.
- Parameter files. It is possible to specify integer, double and string parameters along with the problem (or in a separate OPF file).
- Solution files. It is possible to store a full or a partial solution in an OPF file and later reload it.

4.3.2 The file format

The format uses tags to structure data. A simple example with the basic sections may look like this:

```
[comment]
  This is a comment. You may write almost anything here...
[/comment]

# This is a single-line comment.

[objective min 'myobj']
  x + 3 y + x^2 + 3 y^2 + z + 1
[/objective]

[constraints]
  [con 'con01'] 4 <= x + y  [/con]
[/constraints]

[bounds]
  [b] -10 <= x,y <= 10  [/b]

  [cone quad] x,y,z [/cone]
[/bounds]
```

A scope is opened by a tag of the form `[tag]` and closed by a tag of the form `[/tag]`. An opening tag may accept a list of unnamed and named arguments, for examples

```
[tag value] tag with one unnamed argument [/tag]
[tag arg=value] tag with one named argument in quotes [/tag]
```

Unnamed arguments are identified by their order, while named arguments may appear in any order, but never before an unnamed argument. The `value` can be a quoted, single-quoted or double-quoted text string, i.e.

```
[tag 'value']     single-quoted value [/tag]
[tag arg='value'] single-quoted value [/tag]
```

```
[tag "value"]     double-quoted value [/tag]
[tag arg="value"] double-quoted value [/tag]
```

4.3.2.1 Sections

The recognized tags are

- `[comment]` A comment section. This can contain *almost* any text: Between single quotes (`'`) or double quotes (`"`) any text may appear. Outside quotes the markup characters (`[` and `]`) must be prefixed by backslashes. Both single and double quotes may appear alone or inside a pair of quotes if it is prefixed by a backslash.

- `[objective]` The objective function: This accepts one or two parameters, where the first one (in the above example '`min`') is either `min` or `max` (regardless of case) and defines the objective sense, and the second one (above '`myobj`'), if present, is the objective name. The section may contain linear and quadratic expressions.

  If several objectives are specified, all but the last are ignored.

- `[constraints]` This does not directly contain any data, but may contain the subsection '`con`' defining a linear constraint.

  `[con]` defines a single constraint; if an argument is present (`[con NAME]`) this is used as the name of the constraint, otherwise it is given a null-name. The section contains a constraint definition written as linear and quadratic expressions with a lower bound, an upper bound, with both or with an equality. Examples:

  ```
  [constraints]
    [con 'con1'] 0 <= x + y       [/con]
    [con 'con2'] 0 >= x + y       [/con]
    [con 'con3'] 0 <= x + y <= 10 [/con]
    [con 'con4']      x + y  = 10 [/con]
  [/constraints]
  ```

  Constraint names are unique. If a constraint is specified which has the same name as a previously defined constraint, the new constraint replaces the existing one.

- `[bounds]` This does not directly contain any data, but may contain the subsections '`b`' (linear bounds on variables) and `cone`' (quadratic cone).

  - `[b]`. Bound definition on one or several variables separated by comma ('`,`'). An upper or lower bound on a variable replaces any earlier defined bound on that variable. If only one bound (upper or lower) is given only this bound is replaced. This means that upper and lower bounds can be specified separately. So the OPF bound definition:

    ```
    [b]  x,y >= -10  [/b]
    [b]  x,y <= 10   [/b]
    ```

    results in the bound

    $$-10 \leq x, y \leq 10.$$

– [cone]. Currently, the supported cones are the *quadratic cone* and the *rotated quadratic cone* A conic constraint is defined as a set of variables which belongs to a single unique cone.

A quadratic cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1^2 > \sum_{i=2}^{n} x_i^2.$$

A rotated quadratic cone of n variables $x_1, \ldots, x_n$ defines a constraint of the form

$$x_1 x_2 > \sum_{i=3}^{n} x_i^2.$$

A [bounds]-section example:

```
[bounds]
  [b]  0 <= x,y <= 10  [/b] # ranged bound
  [b] 10 >= x,y >=  0  [/b] # ranged bound
  [b]  0 <= x,y <= inf [/b] # using inf
  [b]       x,y free   [/b] # free variables
  # Let (x,y,z,w) belong to the cone K
  [cone quad]  x,y,z,w  [/cone] # quadratic cone
  [cone rquad] x,y,z,w  [/cone] # rotated quadratic cone
[/bounds]
```

By default all variables are free.

- [variables] This defines an ordering of variables as they should appear in the problem. This is simply a space-separated list of variable names.

- [integer] This contains a space-separated list of variables and defines the constraint that the listed variables must be integer values.

- [hints] This may contain only non-essential data; for example estimates of the number of variables, constraints and non-zeros. Placed before all other sections containing data this may reduce the time spent reading the file.

  In the hints section, any subsection which is not recognized by MOSEK is simply ignored. In this section a hint in a subsection is defined as follows:

  ```
  [hint ITEM] value [/hint]
  ```

  where ITEM may be replaced by numvar (number of variables), numcon (number of linear/quadratic constraints), numanz (number of linear non-zeros in constraints) and numqnz (number of quadratic non-zeros in constraints).

- [solutions] This section can contain a set of full or partial solutions to a problem. Each solution must be specified using a [solution]-section, i.e.

  ```
  [solutions]
         [solution]...[/solution] #solution 1
         [solution]...[/solution] #solution 2
                           #other solutions....
         [solution]...[/solution] #solution n
  [/solutions]
  ```

Note that a [solution]-section must be always specified inside a [solutions]-section. The syntax of a [solution]-section is the following:

```
[solution SOLTYPE status=STATUS]...[/solution]
```

where SOLTYPE is one of the strings

- 'interior', a non-basic solution,
- 'basic', a basic solution,
- 'integer', an integer solution,

and STATUS is one of the strings

- 'UNKNOWN',
- 'OPTIMAL',
- 'INTEGER_OPTIMAL',
- 'PRIM_FEAS',
- 'DUAL_FEAS',
- 'PRIM_AND_DUAL_FEAS',
- 'NEAR_OPTIMAL',
- 'NEAR_PRIM_FEAS',
- 'NEAR_DUAL_FEAS',
- 'NEAR_PRIM_AND_DUAL_FEAS',
- 'PRIM_INFEAS_CER',
- 'DUAL_INFEAS_CER',
- 'NEAR_PRIM_INFEAS_CER',
- 'NEAR_DUAL_INFEAS_CER',
- 'NEAR_INTEGER_OPTIMAL'.

Most of these values are irrelevant for input solutions; when constructing a solution for simplex hot-start or an initial solution for a mixed integer problem the safe setting is UNKNOWN.

A [solution]-section contains [con] and [var] sections. Each [con] and [var] section defines solution information for a single variable or constraint, specified as list of KEYWORD/value pairs, in any order, written as

```
KEYWORD=value
```

Allowed keywords are as follows:

- sk. The status of the item, where the value is one of the following strings:
  * LOW, the item is on its lower bound.
  * UPR, the item is on its upper bound.
  * FIX, it is a fixed item.
  * BAS, the item is in the basis.
  * SUPBAS, the item is super basic.

    * `UNK`, the status is unknown.
    * `INF`, the item is outside its bounds (infeasible).
  - `lvl` Defines the level of the item.
  - `sl` Defines the level of the dual variable associated with its lower bound.
  - `su` Defines the level of the dual variable associated with its upper bound.
  - `sn` Defines the level of the variable associated with its cone.
  - `y` Defines the level of the corresponding dual variable (for constraints only).

  A `[var]` section should always contain the items `sk`, `lvl`, `sl` and `su`. Items `sl` and `su` are not required for `integer` solutions.

  A `[con]` section should always contain `sk`, `lvl`, `sl`, `su` and `y`.

  An example of a solution section

```
[solution basic status=UNKNOWN]
    [var x0] sk=LOW    lvl=5.0      [/var]
    [var x1] sk=UPR    lvl=10.0     [/var]
    [var x2] sk=SUPBAS lvl=2.0  sl=1.5 su=0.0 [/var]

    [con c0] sk=LOW    lvl=3.0 y=0.0 [/con]
    [con c0] sk=UPR    lvl=0.0 y=5.0 [/con]
[/solution]
```

- `[vendor]` This contains solver/vendor specific data. It accepts one argument, which is a vendor ID – for MOSEK the ID is simply `mosek` – and the section contains the subsection `parameters` defining solver parameters. When reading a vendor section, any unknown vendor can be safely ignored. This is described later.

Comments using the '`#`' may appear anywhere in the file. Between the '`#`' and the following line-break any text may be written, including markup characters.

4.3.2.2 Numbers

Numbers, when used for parameter values or coefficients, are written in the usual way by the `printf` function. That is, they may be prefixed by a sign (`+` or `-`) and may contain an integer part, decimal part and an exponent. The decimal point is always '`.`' (a dot). Some examples are

```
1
1.0
 .0
1.
1e10
1e+10
1e-10
```

Some *invalid* examples are

```
e10   # invalid, must contain either integer or decimal part
.     # invalid
.e10  # invalid
```

More formally, the following standard regular expression describes numbers as used:

```
[+|-]?([0-9]+[.][0-9]*|[.][0-9]+)([eE][+|-]?[0-9]+)?
```

4.3.2.3 Names

Variable names, constraint names and objective name may contain arbitrary characters, which in some cases must be enclosed by quotes (single or double) that in turn must be preceded by a backslash. Unquoted names must begin with a letter (a-z or A-Z) and contain only the following characters: the letters a-z and A-Z, the digits 0-9, braces ({ and }) and underscore (_).

Some examples of legal names:

```
an_unquoted_name
another_name{123}
'single quoted name'
"double quoted name"
"name with \\"quote\\" in it"
"name with []s in it"
```

4.3.3 Parameters section

In the vendor section solver parameters are defined inside the parameters subsection. Each parameter is written as

```
[p PARAMETER_NAME] value [/p]
```

where PARAMETER_NAME is replaced by a MOSEK parameter name, usually of the form MSK_IPAR_..., MSK_DPAR_... or MSK_SPAR_..., and the value is replaced by the value of that parameter; both integer values and named values may be used. Some simple examples are:

```
[vendor mosek]
  [parameters]
    [p MSK_IPAR_OPF_MAX_TERMS_PER_LINE] 10     [/p]
    [p MSK_IPAR_OPF_WRITE_PARAMETERS]   MSK_ON [/p]
    [p MSK_DPAR_DATA_TOL_BOUND_INF]     1.0e18 [/p]
  [/parameters]
[/vendor]
```

4.3.4 Writing OPF files from MOSEK

The function Task.writedata can be used to produce an OPF file from a task.

To write an OPF file set the parameter iparam.write_data_format to dataformat.op as this ensures that OPF format is used. Then modify the following parameters to define what the file should contain:

- iparam.opf_write_header, include a small header with comments.
- iparam.opf_write_hints, include hints about the size of the problem.
- iparam.opf_write_problem, include the problem itself — objective, constraints and bounds.
- iparam.opf_write_solutions, include solutions if they are defined. If this is off, no solutions are included.
- iparam.opf_write_sol_bas, include basic solution, if defined.

- iparam.opf_write_sol_itg, include integer solution, if defined.
- iparam.opf_write_sol_itr, include interior solution, if defined.
- iparam.opf_write_parameters, include all parameter settings.

4.3.5 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

4.3.5.1 Linear example lo1.opf

Consider the example:

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

having the bounds

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

In the OPF format the example is displayed as shown below:

```
[comment]
  The lo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 4 [/hint]
  [hint NUMCON] 3 [/hint]
  [hint NUMANZ] 9 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4
[/variables]

[objective maximize 'obj']
   3 x1 + x2 + 5 x3 + x4
[/objective]

[constraints]
  [con 'c1'] 3 x1 +   x2 + 2 x3         = 30 [/con]
  [con 'c2'] 2 x1 +   x2 + 3 x3 +   x4 >= 15 [/con]
  [con 'c3']        2 x2        + 3 x4 <= 25 [/con]
[/constraints]
```

```
[bounds]
  [b] 0 <= * [/b]
  [b] 0 <= x2 <= 10 [/b]
[/bounds]
```

4.3.5.2 Quadratic example `qo1.opf`

An example of a quadratic optimization problem is

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

This can be formulated in `opf` as shown below.

```
[comment]
  The qo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 3 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
  [hint NUMQNZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3
[/variables]

[objective minimize 'obj']
  # The quadratic terms are often written with a factor of 1/2 as here,
  # but this is not required.

   - x2 + 0.5 ( 2.0 x1 ^ 2 - 2.0 x3 * x1 + 0.2 x2 ^ 2 + 2.0 x3 ^ 2 )
[/objective]

[constraints]
  [con 'c1'] 1.0 <= x1 + x2 + x3 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]
```

4.3.5.3 Conic quadratic example `cqo1.opf`

Consider the example:

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

Please note that the type of the cones is defined by the parameter to `[cone ...]`; the content of the `cone`-section is the names of variables that belong to the cone.

```
[comment]
  The cqo1 example in OPF format.
[/comment]

[hints]
  [hint NUMVAR] 6 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4 x5 x6
[/variables]

[objective minimize 'obj']
   x4 + x5 + x6
[/objective]

[constraints]
  [con 'c1']  x1 + x2 + 2e+00 x3 = 1e+00 [/con]
[/constraints]

[bounds]
  # We let all variables default to the positive orthant
  [b] 0 <= * [/b]

  # ...and change those that differ from the default
  [b] x4,x5,x6 free [/b]

  # Define quadratic cone: x4 >= sqrt( x1^2 + x2^2 )
  [cone quad 'k1'] x4, x1, x2 [/cone]

  # Define rotated quadratic cone: 2 x5 x6 >= x3^2
  [cone rquad 'k2'] x5, x6, x3 [/cone]
[/bounds]
```

4.3.5.4 Mixed integer example `milo1.opf`

Consider the mixed integer problem:

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

This can be implemented in OPF with:

```
[comment]
  The milo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 2 [/hint]
  [hint NUMCON] 2 [/hint]
  [hint NUMANZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2
[/variables]

[objective maximize 'obj']
   x1 + 6.4e-1 x2
[/objective]

[constraints]
  [con 'c1'] 5e+1 x1 + 3.1e+1 x2 <= 2.5e+2 [/con]
  [con 'c2'] -4 <= 3 x1 - 2 x2 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]

[integer]
  x1 x2
[/integer]
```

4.4 The Task format

The Task format is MOSEK's native binary format. It contains a complete image of a MOSEK task, i.e.

- Problem data: Linear, conic quadratic, semidefinite and quadratic data
- Problem item names: Variable names, constraints names, cone names etc.
- Parameter settings
- Solutions

There are a few things to be aware of:

- The task format *does not* support General Convex problems since these are defined by arbitrary user-defined functions.
- Status of a solution read from a file will *always* be unknown.

```
*        1         2         3         4         5         6
*23456789012345678901234567890123456789012345678901234567890
NAME          [name]
 ?? [vname1]              [value1]
ENDATA
```

Figure 4.1: The standard ORD format.

The format is based on the TAR (USTar) file format. This means that the individual pieces of data in a `.task` file can be examined by unpacking it as a TAR file. Please note that the inverse may not work: Creating a file using TAR will most probably not create a valid MOSEK Task file since the order of the entries is important.

4.5 The XML (OSiL) format

MOSEK can write data in the standard `OSiL` xml format. For a definition of the `OSiL` format please see http://www.optimizationservices.org/. Only linear constraints (possibly with integer variables) are supported. By default output files with the extension `.xml` are written in the `OSiL` format.

The parameter `iparam.write_xml_mode` controls if the linear coefficients in the A matrix are written in row or column order.

4.6 The ORD file format

An ORD formatted file specifies in which order the mixed integer optimizer branches on variables. The format of an ORD file is shown in Figure 4.1. In the figure names in capitals are keywords of the ORD format, whereas names in brackets are custom names or values. The `??` is an optional key specifying the preferred branching direction. The possible keys are `DN` and `UP` which indicate that down or up is the preferred branching direction respectively. The branching direction key is optional and is left blank the mixed integer optimizer will decide whether to branch up or down.

4.6.1 An example

A concrete example of a ORD file is presented below:

```
NAME          EXAMPLE
 DN x1                2
 UP x2                1
    x3                10
ENDATA
```

This implies that the priorities 2, 1, and 10 are assigned to variable `x1`, `x2`, and `x3` respectively. The higher the priority value assigned to a variable the earlier the mixed integer optimizer will branch on that variable. The key `DN` implies that the mixed integer optimizer first will branch down on variable whereas the key `UP` implies that the mixed integer optimizer will first branch up on a variable.

If no branch direction is specified for a variable then the mixed integer optimizer will automatically choose the branching direction for that variable. Similarly, if no priority is assigned to a variable then it is automatically assigned the priority of 0.

4.7 The solution file format

MOSEK provides one or two solution files depending on the problem type and the optimizer used. If a problem is optimized using the interior-point optimizer and no basis identification is required, then a file named `probname.sol` is provided. `probname` is the name of the problem and `.sol` is the file extension. If the problem is optimized using the simplex optimizer or basis identification is performed, then a file named `probname.bas` is created presenting the optimal basis solution. Finally, if the problem contains integer constrained variables then a file named `probname.int` is created. It contains the integer solution.

4.7.1 The basic and interior solution files

In general both the interior-point and the basis solution files have the format:

```
NAME                : <problem name>
PROBLEM STATUS      : <status of the problem>
SOLUTION STATUS     : <status of the solution>
OBJECTIVE NAME      : <name of the objective function>
PRIMAL OBJECTIVE    : <primal objective value corresponding to the solution>
DUAL OBJECTIVE      : <dual objective value corresponding to the solution>
CONSTRAINTS
INDEX  NAME     AT ACTIVITY    LOWER LIMIT   UPPER LIMIT   DUAL LOWER   DUAL UPPER
?      <name>   ?? <a value>   <a value>     <a value>     <a value>    <a value>
VARIABLES
INDEX  NAME     AT ACTIVITY    LOWER LIMIT   UPPER LIMIT   DUAL LOWER   DUAL UPPER    CONIC DUAL
?      <name>   ?? <a value>   <a value>     <a value>     <a value>    <a value>     <a value>
```

In the example the fields `?` and `<>` will be filled with problem and solution specific information. As can be observed a solution report consists of three sections, i.e.

`HEADER`

In this section, first the name of the problem is listed and afterwards the problem and solution statuses are shown. In this case the information shows that the problem is primal and dual feasible and the solution is optimal. Next the primal and dual objective values are displayed.

`CONSTRAINTS`

Subsequently in the constraint section the following information is listed for each constraint:

`INDEX`

A sequential index assigned to the constraint by MOSEK

`NAME`

The name of the constraint assigned by the user.

Status key	Interpretation
UN	Unknown status
BS	Is basic
SB	Is superbasic
LL	Is at the lower limit (bound)
UL	Is at the upper limit (bound)
EQ	Lower limit is identical to upper limit
**	Is infeasible i.e. the lower limit is greater than the upper limit.

Table 4.1: Status keys.

`AT`

The status of the constraint. In Table 4.1 the possible values of the status keys and their interpretation are shown.

`ACTIVITY`

Given the i th constraint on the form

$$l_i^c \leq \sum_{j=1}^{n} a_{ij} x_j \leq u_i^c, \tag{4.7}$$

then activity denote the quantity $\sum_{j=1}^{n} a_{ij} x_j^*$, where x^* is the value for the x solution.

`LOWER LIMIT`

Is the quantity l_i^c (see (4.7)).

`UPPER LIMIT`

Is the quantity u_i^c (see (4.7)).

`DUAL LOWER`

Is the dual multiplier corresponding to the lower limit on the constraint.

`DUAL UPPER`

Is the dual multiplier corresponding to the upper limit on the constraint.

`VARIABLES`

The last section of the solution report lists information for the variables. This information has a similar interpretation as for the constraints. However, the column with the header `[CONIC DUAL]` is only included for problems having one or more conic constraints. This column shows the dual variables corresponding to the conic constraints.

4.7.2 The integer solution file

The integer solution is equivalent to the basic and interior solution files except that no dual information is included.

Bibliography

[1] Chvátal, V.. Linear programming, 1983. W.H. Freeman and Company

[2] Nazareth, J. L.. Computer Solution of Linear Programs, 1987. Oxford University Press, New York

[3] Bazaraa, M. S., Sherali, H. D. and Shetty, C. M.. Nonlinear programming: Theory and algorithms, 2 edition, 1993. John Wiley and Sons, New York

[4] Williams, H. P.. Model building in mathematical programming, 3 edition, 1993. John Wiley and Sons

[5] G. W. Stewart. Matrix Algorithms. Volume 1: Basic decompositions, 1998. P_SIAM

Index