

# MOSEK FAQ

*Release 11.2.8*

MOSEK ApS

05 November 2026

# Contents


**1 Technical Issues** **1**

1.1 How do I dump the problem to a file to attach with my support question? . . . 1

1.2 I am using a third-party interface to MOSEK. How do I set solver parameters and other options? . . . 3

1.2.1 CVXPY . . . 3

1.2.2 YALMIP . . . 3

1.2.3 CVX . . . 4

1.2.4 JuMP . . . 4

1.2.5 CVXOPT . . . 4

1.2.6 PyOptInterface . . . 4

1.2.7 Pyomo . . . 5

1.2.8 amplpy . . . 5

1.2.9 CVXR . . . 6

1.2.10 PuLP . . . 6

1.2.11 linopy . . . 7

1.3 How do I find my exact **MOSEK** version? . . . 7

1.4 What to do about Fatal error / stoptask / Assertion failed? . . . 7

1.5 **MOSEK** is ignoring the limit on the number of threads. . . . 8

1.6 When using the Python interface I get a RuntimeWarning: Item size computed from the PEP 3118. . . . 8

1.7 The Python API or Python Fusion API will not accept a numpy array, only a list. . . . 8

1.8 Python: why do I see `` UserWarning:`` Incorrect array format causing data to be copied? 8

1.9 Python AttributeError: Task object has no attribute Taskobj . . . 8

1.10 Can the C API be used from Fortran? . . . 9

1.11 Can MOSEK run on virtualized server such as VmWare server? . . . 9

1.12 How do I write an MPS file using GAMS and MOSEK? . . . 9

1.13 Does MOSEK work with MinGW? . . . 9

1.14 Is it possible to link MOSEK as a static library? . . . 9

1.15 Can you provide MOSEK linked with another `libc` implementation? . . . 9

1.16 Does MOSEK support Cygwin? . . . 9

1.17 Can MOSEK exploit GPU? . . . 10

**2 Installation and configuration** **11**

2.1 On Mac OS I get errors about missing libmosek64.10.1.dylib or similar files. . . . 11

2.2 Errors running the install script on Mac OS: missing otool. . . . 11

2.3 Installing **MOSEK** with CVX. . . . 11

2.4 Security exceptions in Mac OS 10.15+ . . . 12

2.5 In MATLAB on Windows I get Invalid MEX-file . . . mosekopt.mexw64: The specified module could not be found. . . . 12

2.6 I cannot link a C++ *Fusion* application on Windows: error LNK2038. . . . 12

2.7 Missing VCRUNTIME140_1.dll . . . 12

2.8 error: could not create 'build': Access is denied . . . 12

2.9 How to use `toolbox` on Apple M1/M2/M3 up to **MOSEK** 10 ? . . . 13

2.10 How to use `toolbox` on Apple M1/M2/M3 from **MOSEK** 11 ? . . . 13

2.11 How to use **MOSEK** in a Google Colab notebook? . . . 13

2.12 How to use a **MOSEK** license in a GitHub CI/CD pipeline? . . . 13

2.13 How to optimize remotely from a third-party interface? . . . 14
2.14 Rmosek: "Error in mosek(prob, opts) : Unknown exported object to be built" . . . 14
2.15 Rmosek: can I install Rmosek offline? . . . 14

**3 Modeling Issues** **15**
3.1 MOSEK is not solving my quadratic optimization problem fast enough. What should I do? 15
3.2 Can MOSEK solve nonconvex problems? . . . 15
3.3 Can MOSEK handle SOS (or SOS2) sets in mixed integer optimization? . . . 15
3.4 Why does MOSEK not report a dual solution when I have solved a mixed integer problem? 15
3.5 How big problems can MOSEK solve? . . . 16
3.6 Can I get the solution MOSEK produces in every iteration of the optimizer? . . . 16
3.7 Can the MOSEK interior-point optimizers be hot-started? . . . 16
3.8 What is the best hardware for **MOSEK**? . . . 16
3.9 Why do not I get a good speedup when using multiple threads? . . . 16
3.10 Why does MOSEK give slightly different answers on different machines/OS/... ? . . . 17
3.11 Why does the solution slightly violate constraints? . . . 18
3.12 Is the simplex optimizer in MOSEK parallelized? . . . 18
3.13 Is the mixed integer optimizer parallelized in MOSEK? . . . 18
3.14 How do I model absolute value? . . . 18
3.15 Why is my SDP/LMI slow or running out of memory? . . . 18

**4 License Management** **19**
4.1 I have a problem setting up a floating license server. . . . 19
4.2 My license file says VER=8.0. Can I use version 8.1? . . . 19
4.3 The Optimization Toolbox for Matlab says license has expired although I downloaded a new one. . . . 19
4.4 Do I want a floating license or a server (node-locked) license? . . . 19
4.5 I have one floating license. How many processes/threads can I use? . . . 19
4.6 Does the licensing software make some sort of a network connection to **MOSEK** to validate the license? . . . 20
4.7 When I change hostid, can I first test the new license while still using the old one? . . . 20
4.8 The lmutil and lmgrd will not start on Linux: No such file or directory. . . . 20
4.9 Can a MOSEK license be used under Citrix? . . . 20
4.10 How do I do advanced configuration of the license system? . . . 20
4.11 Is it possible to reserve a number licenses to particular group of users or hosts? . . . 20
4.12 Is it possible to use MOSEK with a floating license on a machine detached from the LAN(WAN)? . . . 21
4.13 How many license token servers should I have? . . . 21
4.14 Is it possible to host a token server in the cloud? . . . 21
4.15 Can I use the same personal academic license on more than machine? . . . 21
4.16 Why is `os.environ["MOSEKLM_LICENSE_FILE"]` ignored by **MOSEK**? . . . 21

**5 Miscellanea** **22**
5.1 Where is the user manual, API reference, PDF versions, examples? . . . 22
5.2 How do I cite MOSEK in an academic publication? . . . 22

# Chapter 1

# Technical Issues

## 1.1 How do I dump the problem to a file to attach with my support question?

Just before or after optimization do:

| | | |
|---|---|---|
| Fusion API | C++ | `M->writeTask("dump.task.gz");` |
| | Java | `M.writeTask("dump.task.gz");` |
| | .NET | `M.WriteTask("dump.task.gz");` |
| | Python | `M.writeTask("dump.task.gz")` |
| Optimizer API | C | `MSK_writedata(task, "dump.task.gz");` |
| | Java | `task.writedata("dump.task.gz");` |
| | .NET | `task.writedata("dump.task.gz");` |
| | Python | `task.writedata("dump.task.gz")` |
| | Julia | `writedata(task, "dump.task.gz")` |
| | Rust | `task.write_data("dump.task.gz")?;` |
| MOSEK API for MATLAB | | `model.write("dump.task.gz")` |
| MATLAB Toolbox (old) | | `mosekopt('write(dump.task.gz)',prob)` |
| Rmosek | | `r <- mosek_write(prob, "dump.task.gz")` |
| Command line | | `mosek input_file -out dump.task.gz` |
| Third party | CVX | `cvx_solver_settings('write', 'dump.task.gz')` |
| | AMPL | `option mosek_auxfiles rc;`<br>`option mosek_options 'writeprob=dump.task.gz';`<br>`solve;` |
| | CVXPY | `prob.solve(solver=MOSEK, save_file="dump.task.gz")` |
| | YALMIP | `sdpsettings('mosektaskfile', 'dump.task.gz')` |
| | JuMP <= 0.18 | `JuMP.build(model)`<br>`Mosek.writedata(internalmodel(model).task, "dump.task.gz")` |
| | JuMP >= 0.19 | indirect model `model = Model(with_optimizer(Mosek.Optimizer))`<br>after `optimize!` do:<br>`MOI.write_to_file(backend(model).optimizer.model, "dump.task.gz")`<br><br>direct model: `model = direct_model(Mosek.Optimizer())`<br>before or after `optimize!` do:<br>`MOI.write_to_file(backend(model), "dump.task.gz")` |
| | PyOptInterface | `model.write("dump.task.gz")` |
| | Pyomo 5.6.2+ | `solver._solver_model.writedata("dump.task.gz")` |
| | PuLP (master) | `prob.solve(solver=MOSEK(task_file_name = 'dump.task.gz'))` |

## 1.2 I am using a third-party interface to MOSEK. How do I set solver parameters and other options?

Here are instructions for some interfaces we are aware of. They may become outdated as the interfaces evolve.

### 1.2.1 CVXPY

Applies to CVXPY at least version 1.3.1. When invoking the solver, pass a dictionary `mosek_params` with pairs `parameter: value`. For example

Listing 1.1: Setting **MOSEK** options from CVXPY.

```
# mosek_params - dictionary with MOSEK parameters set as in the C API manual␣
↪(optional)
# save_file    - where to save the data file (optional)
# verbose      - enable full log (optional)
result = prob.solve(solver = cp.MOSEK,
                    mosek_params = {'MSK_DPAR_OPTIMIZER_MAX_TIME':  100.0,
                                    'MSK_IPAR_INTPNT_SOLVE_FORM':   'MSK_SOLVE_DUAL' }
↪,
                    save_file = 'dump.ptf',
                    verbose = True)
```

will set the time limit to 100s and force the interior-point solver to solve the dual. See the C Optimizer API manual for parameter names and values.

(In older versions of CVXPY it may be impossible to use a string as the value of a parameter, in this case use `mosek.solveform.dual`, i.e. constants from the Python Optimizer API).

To enable log use `verbose=True`. To retrieve basic solution instead of interior-point solution (for linear problems) use `bfs=True`.

### 1.2.2 YALMIP

Pass options using `sdpsettings`. For example

Listing 1.2: Setting **MOSEK** options from YALMIP.

```
% mosek.*      - MOSEK parameters identified by generic names
% verbose      - enable full log
% mosektaskfile- where to save the data file
ops = sdpsettings('solver', 'mosek', ...
                  'mosek.MSK_DPAR_OPTIMIZER_MAX_TIME', 100.0, ...
                  'mosek.MSK_IPAR_INTPNT_SOLVE_FORM', 'MSK_SOLVE_DUAL', ...
                  'verbose', 1, ...
                  'mosektaskfile', 'dump.task.gz');
optimize(constraints, objective, ops);
```

will set the time limit to 100s and force the interior-point solver to solve the dual. See the **MOSEK** Optimization Toolbox for MATLAB manual for parameters and values.

To enable log set `verbose`.

To save the **MOSEK** input (`prob` structure) to the file called `mosekdebug.m` set `savedebug`.

To save an internal **MOSEK** data file to one of the supported file formats set `mosektaskfile` to the file name.

### 1.2.3 CVX

Set parameters using `cvx_solver_settings`. For example

Listing 1.3: Setting **MOSEK** options from CVX.

```
cvx_solver mosek;
% Set MOSEK parameters using generic names
cvx_solver_settings('MSK_DPAR_OPTIMIZER_MAX_TIME', 100.0, ...
                    'MSK_IPAR_INTPNT_SOLVE_FORM', 'MSK_SOLVE_DUAL');
% Shows how to save the MOSEK task file
cvx_solver_settings('write', 'dump.task.gz')
```

will set the time limit to 100s and force the interior-point solver to solve the dual. See the **MOSEK** Optimization Toolbox for MATLAB manual for parameters and values.

Use `write` option to specify where to save the problem data.

### 1.2.4 JuMP

This example applies to JuMP version at least 0.19. Set parameters when creating the solver using full generic names. For example

Listing 1.4: Setting **MOSEK** options from JuMP.

```
model = Model(with_optimizer(Mosek.Optimizer,
                             MSK_DPAR_OPTIMIZER_MAX_TIME = 100.0,
                             MSK_IPAR_INTPNT_SOLVE_FORM  = MSK_SOLVE_DUAL));
```

will set the time limit to 100s and force the interior-point solver to solve the dual. See the C API manual for parameters and values.

Use `LOG=0` to disable log output.

### 1.2.5 CVXOPT

Pass options through the dictionary `solvers.options['mosek']`. For example

Listing 1.5: Setting **MOSEK** options from CVXOPT.

```
cvxopt.solvers.options['mosek'] = {mosek.dparam.optimizer_max_time:  100.0,
                                   mosek.iparam.intpnt_solve_form:   mosek.solveform.
↪dual}

cvxopt.solvers.options['verbose'] = False

sol=cvxopt.solvers.qp(Q, p, G, h, A, b, solver='mosek')
```

will set the time limit to 100s and force the interior-point solver to solve the dual. See the Python Optimizer API manual for parameter names and values.

### 1.2.6 PyOptInterface

Based on version 0.4.1. Pass options by calling `model.set_raw_parameter`. For example

Listing 1.6: Setting **MOSEK** options from PyOptInterface.

```
# Set some MOSEK parameters
model.set_raw_parameter("MSK_DPAR_OPTIMIZER_MAX_TIME", 10.0)
model.set_raw_parameter("MSK_IPAR_INTPNT_SOLVE_FORM",  int(mosek.solveform.dual))    #␣
↪Must explicitly convert to int or know the value
model.set_raw_parameter("MSK_IPAR_NUM_THREADS",  "2")
```

(continues on next page)

(continued from previous page)

```
# How to write a task file
model.write("dump.task.gz")    # Binary
model.write("dump.ptf")        # Human-readable
```

will set the time limit to 10s, force the interior-point solver to solve the dual, and use 2 threads. See the C Optimizer API manual for parameter names and values.

### 1.2.7 Pyomo

When invoking the solver, pass a dictionary `options` with pairs `parameter: value`. Parameter names should be passed as strings starting with `iparam`, `dparam` or `sparam` and the value should be of the corresponding type int, float or string. Alternatively, if one uses the generic parameter name then the value must be a string, which will automatically be converted to the correct type as needed. For example

Listing 1.7: Setting **MOSEK** options from Pyomo.

```
with SolverFactory("mosek") as solver:
    # options - MOSEK parameters dictionary, using strings as keys (optional)
    # tee - write log output if True (optional)
    # soltype - accepts three values : bas, itr and itg for basic,
    # interior point and integer solution, respectively. (optional)
    solver.solve(model, options = {'MSK_DPAR_INTPNT_CO_TOL_REL_GAP': '1.0e-7',
                                   'dparam.optimizer_max_time':  100.0,
                                   'iparam.intpnt_solve_form':   int(mosek.solveform.
↪dual)},
                        tee = True, soltype='itr')

    # Save data to file (after solve())
    solver._solver_model.writedata("dump.task.gz")
```

will change relative gap tolerance (note the use of generic parameter name and the string value), set the time limit to 100s and force the interior-point solver to solve the dual. See the Python Optimizer API manual for valid parameters, their generic names and valid values.

Use `tee=True` to enable **MOSEK** log output.

### 1.2.8 amplpy

Set the option `mosek_options` to a space-separated string `param1=value1 param2=value2` etc. For example

Listing 1.8: Setting **MOSEK** options from amplpy.

```
ampl.setOption('solver', 'mosek')
ampl.setOption('mosek_options', 'outlev=2 msk_dpar_optimizer_max_time=100.0 msk_ipar_
↪intpnt_solve_form=msk_solve_dual')
```

will set verbosity level to 2, time limit to 100s and force the interior-point solver to solve the dual. See the **MOSEK** Command-line Tool manual for parameter names and values.

### 1.2.9 CVXR

The following illustration applies to CVXR version 1.0. In addition to the default parameters for the CVXR solve function, MOSEK specific parameters can be set by passing integer/double/string parameters in their own separate lists, as shown in the code below.

Listing 1.9: Setting **MOSEK** options from CVXR.

```
# iparam: integer parameters (similar names for double and string parameters)
iparam <- list()
iparam$MSK_IPAR_INTPNT_SOLVE_FORM="MSK_SOLVE_DUAL"
result <- solve(prob, solver="MOSEK", verbose=TRUE, iparam=iparam)
```

This code example illustrates how to set the relevant iparam to force the interior-point solver to solve the dual. Also check the CVXR documentation for MOSEK parameters.

### 1.2.10 PuLP

This applies to current master branch of PuLP. When invoking the solver, pass a dictionary `options` with pairs `parameter: value`. For example:

Listing 1.10: Setting **MOSEK** options from PuLP.

```
# msg - activate MOSEK log output (optional)
# options - a dictionary with MOSEK parameters (optional)
# task_file_name - name of the file where to save the data (optional)
prob.solve(solver=MOSEK(mip = False,
                        msg = True,
                        options = {mosek.dparam.optimizer_max_time:  100.0,
                                   mosek.iparam.optimizer:           mosek.
↪optimizertype.dual_simplex},
                        task_file_name = 'dump.task.gz'))
```

will set the time limit to 100s and force the dual simplex optimizer. Instead of passing the enumeration type objects as keys to the dictionary as shown above, one could use the generic names of **MOSEK** parameters as strings to play the role of keys in the dictionary. For example, use `options = {"MSK_DPAR_OPTIMIZER_MAX_TIME": 100}` to achieve the same effect as before. See the Python Optimizer API manual for parameter names (including the generic **MOSEK** names for the each parameter) and values.

Use `msg=True` to enable **MOSEK** log output.

### 1.2.11 linopy

Applies to linopy with full **MOSEK** direct support, version at least `0.3.7`. When invoking the solver, pass **MOSEK** parameters using their generic names. For example:

Listing 1.11: Setting **MOSEK** options from linopy.

```
M.solve(solver_name="mosek",
        io_api="direct",
        log_fn="lo1.log",                    # Mosek log file
        basis_fn="lo1.bas",
        MSK_DPAR_OPTIMIZER_MAX_TIME=100.0,   # Setting Mosek parameters using generic␣
↪names
        MSK_IPAR_INTPNT_SOLVE_FORM='MSK_SOLVE_DUAL')
```

will set the time limit to 100s and force the optimizer to solve the dual formulation. See the Python Optimizer API manual for generic parameter names and values.

Use `log_fn` to save the solver's log output to a file and the standard `linopy.Model.to_file` to write the model to a file.

Download `LO example`, `QO example`.

## 1.3 How do I find my exact MOSEK version?

| | | |
|---|---|---|
| Fusion API | C++ | `mosek::fusion::Model::getVersion();` |
| | Java | `mosek.fusion.Model.getVersion();` |
| | .NET | `mosek.fusion.Model.GetVersion();` |
| | Python | `mosek.fusion.Model.getVersion()` |
| Optimizer API | C | `MSK_getversion(&major, &minor, &rev);` |
| | Java | `mosek.Env.getversion(major, minor, rev);` |
| | .NET | `mosek.Env.getversion(out major, out minor, out rev);` |
| | Python | `mosek.Env.getversion()` |
| | Julia | `Mosek.getversion()` |
| | Rust | `mosek::get_version(& mut major, & mut minor, & mut rev)?;` |
| MATLAB | | `[major, minor, rev] = mosekenv("version");` |
| MATLAB (old toolbox) | | `[r, res] = mosekopt('version'); res.version` |
| R | | `ver <- mosek_version()` |
| Command line | | `mosek -v` |

## 1.4 What to do about Fatal error / stoptask / Assertion failed?

Maybe **MOSEK** stops and causes your whole process to terminate abruptly with a message such as:

```
MOSEK fatal error stoptask
Version : 11.0.26
File    : src\prslv\prlog.c
Line    : 18363
Msg     : Assertion failed at src\prslv\prlog.c(18363).
```

Congratulations! You just found a bug in **MOSEK**. There is no prize but we promise to fix it ASAP if you just send us the task file which caused it.

- Upgrade to the latest **MOSEK** version with the same major version as yours and see if that helps. Maybe somebody found the same bug before you did.
- If the issue persists in the latest version then save the task file of the offending problem (see Sec. 1.1) and send to **MOSEK** support, ideally together with your log output, platform and OS information and any other relevant info.

- Alternatively send a fully reproducible code sample demonstrating the fatal error.

## 1.5 MOSEK is ignoring the limit on the number of threads.

There are a few possible explanations:

- (Only relevant up to MOSEK 9.3). If you are using the conic optimizer, the number of threads should be set before the first optimization. After that the thread pool is reserved for the process and its size will not be affected by subsequent changes of the parameter. It will only be possible to switch between using all those threads and a single-threaded run.
- If you are using Python, it is possible that `numpy`, and not the **MOSEK** optimizer, is using many threads. Set the environment variable `MKL_NUM_THREADS`. For more information on limiting the thread usage in `numpy` see for example this thread or this one .

## 1.6 When using the Python interface I get a RuntimeWarning: Item size computed from the PEP 3118.

This is caused by a bug in the CTypes module in Python 2.7 and 3.x, related to Issue 10744 and Issue 10746. These issues are only registered as being related to Python 3.1+, but the ctypes code from 3.x is routinely back-ported to 2.7. The problem is that CTypes reports the size and shape of arrays correctly, but provides a wrong element formatting string: Numpy and other libraries may check this and issue a warning. The bug appears to be harmless; we know of no cases where the invalid information is used.

## 1.7 The Python API or Python Fusion API will not accept a numpy array, only a list.

The Python interfaces accept numpy arrays. The problem here is most likely that the numpy array is of wrong type, usually `int` instead of `float` where **MOSEK** expects floating-point data. Use `dtype=float` when constructing the array, if necessary. Moreover, the argument must be an actual numpy array and not, for instance, a slice or another indirect view of a part of the array.

## 1.8 Python: why do I see `` UserWarning:`` Incorrect array format causing data to be copied?

You will see this warning if you a pass a `numpy` array of type `int32` where the API expects an `int64` array or vice versa.

This is not an error; the Python module will perform the necessary type conversion. However, if the input array had the exact expect type from the start, then it could be passed directly to the **MOSEK** library, avoiding an additional copy. For huge arrays this can have a noticeable impact on efficiency, hence the warning.

## 1.9 Python AttributeError: Task object has no attribute Taskobj

If accessing a task object in Python generates an error such as

```
AttributeError: 'Task' object has no attribute '_Task__obj'
```

then most likely the task object had already been (manually or automatically) garbage collected at the time of the call. Typically you are trying to use the task object outside of the `with` scope where it was created. You should track the structure of your code for such issues.

## 1.10 Can the C API be used from Fortran?

MOSEK has no official support for FORTRAN, but if the FORTRAN compiler supports C calling convention it should be possible use to the C API. The appropriate FORTRAN compiler documentation will have to tell you how to do it.

## 1.11 Can MOSEK run on virtualized server such as VmWare server?

Yes, MOSEK runs fine on a virtualized server. However, particularly when solving large problems, MOSEK can exploit all the resources of computer and hence using a virtualized server may cost performance. If performance is important it is recommended to test whether MOSEK runs as fast on the virtualized server as on a native server for the relevant applications.

## 1.12 How do I write an MPS file using GAMS and MOSEK?

GAMS can write native MPS files but they do not include any nonlinear information so they are not useful for nonlinear problems. However, it is possible to write MPS files using MOSEK as follows. Create a file name `mosek.opt` that has the content

```
MSK_SPAR_DATA_FILE_NAME somename.mps
```

and then execute the command `gams mygamsprogram`. MOSEK should now write the file `somename.mps`.

## 1.13 Does MOSEK work with MinGW?

Yes. MOSEK includes libraries that are compatible with MinGW-w64. Applications built with MinGW should link with `libmosek64_*.a`. The native import library (`.lib`) will not work on 64bit Windows.

## 1.14 Is it possible to link MOSEK as a static library?

No. The design of MOSEK requires it to be a shared/dynamic library.

## 1.15 Can you provide MOSEK linked with another `libc` implementation?

No. MOSEK requires the functionality of standard `libc` and we cannot provide a version linked with other implementations (e.g. `musl libc`).

## 1.16 Does MOSEK support Cygwin?

No. Cygwin is not supported. An alternative to using Cygwin is the use of MinGW toolchain that is supported by MOSEK.

## 1.17 Can MOSEK exploit GPU?

No. GPUs are not particularly well-suited for implementing sparse linear algebra like the one in a general-purpose interior-point solver. We are watching it closely but there is no indication GPUs can become useful for MOSEK or similar optimizers in near future. See for example this stackexchange answer.

# Chapter 2

# Installation and configuration

If you cannot find the answer here check also the *Installation* section of your APIs manual for possibly more troubleshooting tips.

## 2.1 On Mac OS I get errors about missing libmosek64.10.1.dylib or similar files.

Most likely you forgot to run the Mac OS installation script as described in the installation manual. Go to the `bin` directory of your MOSEK installation and run

```
python install.py
```

## 2.2 Errors running the install script on Mac OS: missing otool.

If running the `install.py` script produces errors such as:

```
xcrun: error: invalid active developer path (/Library/Developer/CommandLineTools),␣
↪missing xcrun at: /Library/Developer/CommandLineTools/usr/bin/xcrun
...
CalledProcessError: Command '['otool', '-L', '/users/username/mosek/10.0/tools/
↪platform/osx64x86/bin/MOSEKLM']' returned non-zero exit status 1
```

then you need to install the command line tools, in particular `otool`. Depending on the OS version, this should be possible with one of the commands:

```
xcode-select --install
xcode-select --switch /Library/Developer/CommandLineTools
```

## 2.3 Installing MOSEK with CVX.

We have written a comprehensive step-by-step guide to installing **MOSEK** with CVX here https://themosekblog.blogspot.com/2025/03/using-mosek-with-cvx.html

## 2.4 Security exceptions in Mac OS 10.15+

If an attempt to run **MOSEK** on Mac OS 10.15 (Catalina) and later produces security exceptions (`developer cannot be verified` and similar) then use `xattr` to remove the quarantine attribute from all **MOSEK** executables and binaries. This can be done in one go with

```
xattr -dr com.apple.quarantine mosek
```

where `mosek` is the folder which contains the full **MOSEK** installation or **MOSEK** binaries. See https://themosekblog.blogspot.com/2019/12/macos-1015-catalina-mosek-installation.html for more information. If that does not help, use the system settings to allow running arbitrary unverified applications.

## 2.5 In MATLAB on Windows I get Invalid MEX-file ...mosekopt.mexw64: The specified module could not be found.

Most likely the folder with **MOSEK** DLLs, `...\win64x86\bin`, is not in the environment variable `PATH` and the shared libraries cannot be found. Add this folder to the environment variable `PATH`. You can also do it inside MATLAB using the setenv command.

## 2.6 I cannot link a C++ *Fusion* application on Windows: error LNK2038.

The *Fusion* library we pre-built and distribute for convenience is built in `StaticRelease` mode. Attempt to link it with an application built in `DynamicDebug` mode will result in errors such as:

```
error LNK2038: mismatch detected for 'RuntimeLibrary': value 'MT_StaticRelease' doesn
↪'t match value 'MDd_DynamicDebug' in lo1.obj
```

The most robust solution is to import the *Fusion* source code from directly into your project and compile within the project with whatever compiler and linker settings are required.

## 2.7 Missing VCRUNTIME140_1.dll

The missing redistributable DLLs on Windows can be installed directly from Microsoft, see Windows support forum. For a direct link to the installer click here.

## 2.8 error: could not create 'build': Access is denied

If an attempt to install the Python interface results in an error such as

```
error: could not create 'build': Access is denied
```

then you have no write permissions to the folder where **MOSEK** is installed. This can happen for example if the package was installed by an administrator, and a user is trying to set up the Python interface. One solution is to install **MOSEK** in another location. Another solution is to specify the location of the build folder in a place the user can write to, for example:

```
python setup.py build --build-base=SOME_FOLDER install --user
```

## 2.9 How to use `toolbox` on Apple M1/M2/M3 up to MOSEK 10 ?

The choice of toolbox is determined by the architecture of your MATLAB.

- If you use a MATLAB installation native for Apple Silicon (`MACA64`, officially available from R2023b) then use the `osxaarch64` installation of **MOSEK** and the toolbox from there.
- If you use a MATLAB installation for Intel (`MACI64`, runs via Rosetta) then install and use the `toolbox` form the `osx64x86` distribution of **MOSEK**. (You can still install and use the `osxaarch64` **MOSEK** for other applications outside of MATLAB).

## 2.10 How to use `toolbox` on Apple M1/M2/M3 from MOSEK 11 ?

- You must use a MATLAB installation native for Apple Silicon (`MACA64`, officially available from R2023b) and the `osxaarch64` installation of **MOSEK** and the toolbox from there.
- The **MOSEK** package for `osx64x86` was discontinued so you cannot use **MOSEK** 11+ in a MATLAB installation for Intel CPUs (`MACI64`) via Rosetta. If you are using that MATLAB release then you are restricted to **MOSEK** 10.

## 2.11 How to use MOSEK in a Google Colab notebook?

Install **MOSEK** using `pip`:

```
%pip install mosek
```

The default location for the license file is

```
/root/mosek/mosek.lic
```

inside the internal filesystem of the notebook.

If you have the license file in a different location you can point **MOSEK** to it via an environment variable

```
%env MOSEKLM_LICENSE_FILE /your/license/file.lic
```

provided that the environment variable is set before the first time **MOSEK** is imported.

## 2.12 How to use a MOSEK license in a GitHub CI/CD pipeline?

If **MOSEK** is used in a project being tested in a GitHub CI/CD then it may be necessary to provide a **MOSEK** license to the pipeline. This should be done without releasing the license content publicly. The recommended way is to use secret variables. Here is an outline.

Go to *Project* > *Settings* > *Secrets* > *Actions* > *Actions secret* > *Add secret* and create a secret called `MSK_LICENSE` with content:

```
START_LICENSE\n
FEATURE PTS ....
...
... here copy the text of the "FEATURE" sections in the license ...
...
...                                                    ... 5FE1 5DBC"
END_LICENSE\n
```

that is the text of the license features from the license file, surrounded by `START_LICENSE\n` and `END_LICENSE\n` tags.

This secret variable can then be used to initialize an environment variable `MOSEKLM_LICENSE_FILE`. For example, a section of a `python-app.yml` file could look as follows:

```
- name: Setup MOSEK & Run Tests
  env:
    MOSEKLM_LICENSE_FILE: ${{ secrets.MSK_LICENSE }}
  run: |
    pytest -sv
```

On runtime **MOSEK** checks the environment variable `MOSEKLM_LICENSE_FILE` to look for a license (see Licensing Guide), and in this case it will just consume the license embedded in the variable's text.

## 2.13 How to optimize remotely from a third-party interface?

When optimizing, set the parameter `MSK_SPAR_REMOTE_OPTSERVER_HOST` to the URL of the OptServer or OptServerLight you want to optimize remotly with, in the format `http://host:port` or `https://host:port`. You can see in Sec. 1.2 or in the documentation of your tool/API how to set **MOSEK** parameters.

## 2.14 Rmosek: "Error in mosek(prob, opts) : Unknown exported object to be built"

Simply calling `install.packages("Rmosek")` will install a **meta**-package from the CRAN repository. Loading this meta-package and calling `mosek(prob, opts)` will result in the said error. To resolve the error, you must complete the installation procedure as shown below (followed by a re-start of the R-session):

```
mosek_attachbuilder("<MSKHOME>/mosek/<VERSION>/tools/platform/<PLATFORM>/bin/")
install.rmosek()
```

Note that the CRAN package is due to be removed from CRAN. Follow the installation instructions in Rmosek install guide.

## 2.15 Rmosek: can I install Rmosek offline?

The default Rmosek installation process will attempt to automatically download the source package from an online repository. An offline installation can be performed as follows.

- Download the package description:

  https://download.mosek.com/R/11.0/src/contrib/PACKAGES

  Replace `11.0` with `X.Y` for your version `X.Y` of **MOSEK**.

- In the downloaded file find the exact latest version of Rmosek, for example:

  ```
  Package: Rmosek
  Version: 11.0.26
  ```

- Download the tarball:

  https://download.mosek.com/R/11.0/src/contrib/Rmosek_11.0.26.tar.gz

  Replace `11.0` with `X.Y` and `11.0.26` with the located latest version `X.Y.Z`.

- Perform the default installation, but in the last step instruct the installer to use the locally downloaded tarball instead of accessing the remote repository:

  ```
  install.rmosek("path\to\file\Rmosek_X.Y.Z.tar.gz", repos=NULL)
  ```

# Chapter 3

# Modeling Issues

For a general introduction to conic modeling and practical hints for formulating optimization problems, including numerical issues, see the Modeling Cookbook.

## 3.1 MOSEK is not solving my quadratic optimization problem fast enough. What should I do?

Please read our white paperor a section in the Modeling Cookbook. Most likely the tricks described in that paper can help you solve your quadratic optimization problem much faster. We recommend formulating quadratic problems in conic form and, in the financial optimization setting, exploiting the factor model structure.

## 3.2 Can MOSEK solve nonconvex problems?

MOSEK cannot solve continuous non-convex optimization problems, and if non-convexity is detected, the solver will terminate. All continuous conic problems one can formulate in MOSEK are convex by construction; the remark about detecting non-convexity applies only to QO and QCQO problems.

Integer variables are the only allowed type of non-convexity and the mixed-integer solver will be invoked in this case.

## 3.3 Can MOSEK handle SOS (or SOS2) sets in mixed integer optimization?

MOSEK has no special support for SOS1 or SOS2 (special ordered sets), but they can easily be implemented with linear constraints of the form $x_1 + x_2 + \ldots \leq 1$ and similar.

## 3.4 Why does MOSEK not report a dual solution when I have solved a mixed integer problem?

In general, the dual problem corresponding to a mixed-integer problem is not well-defined. In some cases the desired *dual solution* is the one obtained by fixing all integer variables at their optimal solution values and re-solving the problem with the continuous optimizer.

## 3.5 How big problems can MOSEK solve?

The **MOSEK** API will accept, as a rule of thumb, up to $2^{31} \approx 2 \cdot 10^9$ variables and constraints and up to $2^{63} \approx 10^{18}$ nonzeros (details can vary and there can be additional restrictions depending on the API). That sets the theoretical limit on the input size. **MOSEK** has been successfully used to solve practical problems with tens of millions of variables/constraints as well as with billions of nonzeros.

In practice the solver's performance will depend on the sparsity (number of nonzeros), structure (positions of nonzeros), how well the problem can be presolved, and other factors, so it is not just a simple function of the number of variables or constraints.

For mixed-integer problems the relation between size and solvability is even less direct, since one can easily construct MIPs with millions of variables that solve efficiently as well as MIPs with hundreds of variables that most likely will not be solved to optimality within the predicted lifetime of Earth by any solver.

## 3.6 Can I get the solution MOSEK produces in every iteration of the optimizer?

Not for continuous problems, due to the design of the interior-point optimizer. Since MOSEK transforms the problem before starting to solve it, the intermediate solution values may make little sense in the context of the user-specified problem, and moreover they are typically not even feasible. These solution values are not made available.

For mixed-integer problems new integer feasible points can be obtained through a callback function.

## 3.7 Can the MOSEK interior-point optimizers be hot-started?

No, since there are no known generally reliable ways to hot-start interior-point methods. This is an open research topic.

## 3.8 What is the best hardware for MOSEK?

**MOSEK** runs best on physical hardware with fast RAM and a fast CPU. Low latency RAM and a good memory controller seem to have the greatest impact on runtime, followed closely by a fast CPU.

MOSEK will take advantage of threading in a variety of places, but for one smallish optimization problem a fast CPU with a high clock speed using one thread will perform best. For large optimization problems using multiple threads is often beneficial. However, the optimal number of threads is problem specific.

Finally, the amount of RAM is problem dependent.

## 3.9 Why do not I get a good speedup when using multiple threads?

There are many reasons why a good speedup cannot be achieved when using more threads. Some of them are:

- Not all operations can be parallelized, as for instance the presolve phase.
- There is an overhead associated with using multiple threads because coordinating the multiple threads requires some management operations which cost some of the computational performance. This is particular visible if more threads than the number of cores is employed.
- Assume you are parallelizing the inner product of two long vectors using two threads. Now each thread will have to move data from main memory to CPU. Since the amount data that can be moved from main memory and to the CPUs is limited then data movement easily becomes the bottleneck. This will limit the potential speedup.

- For many operations such as matrix multiplication then the sequential version runs relatively faster the larger the matrices are. Now if matrix multiplication is parallelized then one big matrix multiplication is replaced by several small ones leading to a loss of efficiency compared to the sequential one. Hence, a linear speedup is not archived.

- Some recent CPUs may boost the clock frequency if the CPU is not loaded too heavily. This is likely to be the case if only one thread is employed. See Intel's explanation for details. This will of course offset some of the benefits of using multiple threads.

To summarize obtaining a linear speedup when using multiple threads is nearly impossible. In fact in many cases it is not possible to obtain any speedup if the job to be parallelized is small.

## 3.10 Why does MOSEK give slightly different answers on different machines/OS/... ?

This is because using different number of threads, different operating system, different compiler etc. lead to arithmetic operations being executed in different order, leading to different rounding errors and in consequence a different solution. For instance, the associative law $(a+b)+c = a+(b+c)$ does not always hold in floating-point arithmetic, so the order in which the computations are performed affects the result. This is known as *floating point indeterminism*. These differences should be miniscule, otherwise it may be a sign that the problem is ill-posed. However, differences of order $1e-8$ in the solution or objective are not a surprise.

MOSEK *is* run-to-run deterministic only under the following conditions:

- exactly the same problem is being solved,
- it is run on the same machine,
- exactly the same parameter settings are used,
- no time limits are specified.

Typical causes of apparent non-determinism include:

- the tasks are not 100% identical, for example constraints or variables appear in different order,
- data entered into the task is not always the same,
- different number of threads is used.
- some parameter was changed.
- the task was run on a different machine.

You will most likely find the cause of non-determinism by saving the task files from a few runs (see Sec. 1.1) and comparing them. Use a text format such as PTF, LP, OPF, depending on your problem type (save as `dump.ptf`, `dump.lp`, `dump.opf`) and compare with `diff`.

Typical reasons for differences are:

- Constraints and variables are differently ordered, that is the tasks are permutations of one another. Common if input data comes from a Python dictionary, set, or dataframe, which can have non-deterministic ordering.
- There are small variations in some numerical values. Can happen if the numerical routines used for data preparation, for instance multi-threaded singular value decomposition in `numpy`, were not deterministic.

## 3.11 Why does the solution slightly violate constraints?

It is completely expected that the solution will violate the constraints by a small amount such as $10^{-8}$. This is the result of performing computations in floating-point arithmetic. It is almost impossible, for instance, that a non-trivial linear equality constraint will be satisfied *exactly in floating point numbers* when a non-trivial solution is plugged in. The same holds for other types of constraints, for example a variable with bound $x \geq 0$ can well be returned as $-10^{-8}$ in the solution. If required, it is the user's responsibility to clean up the solution from such values. See also Modeling Cookbook for an example.

The solution summary contains the amount of violations.

Of course very large violations may indicate an issue with the solution or numerical issues in the problem formulation.

## 3.12 Is the simplex optimizer in MOSEK parallelized?

The simplex optimizers in MOSEK are not parallelized and hence cannot exploit multiple threads. The iterations in the simplex algorithm very serial in nature, and each iteration performs so little work that they cannot efficiently be parallelized. To our knowledge, high-performance large scale parallel simplex is currently unavailable.

## 3.13 Is the mixed integer optimizer parallelized in MOSEK?

Yes, the branch-and-bound algorithm is parallelized.

## 3.14 How do I model absolute value?

You can model the constraint

$$t \geq |x|$$

with

$$t \geq x, \; t \geq -x.$$

If you really need to model the exact equality $t = |x|$ or inequality $t \leq |x|$ then integer variables are required - this constraint is not convex. In this case see the Modeling Cookbook.

## 3.15 Why is my SDP/LMI slow or running out of memory?

One of the key factors in the time and memory complexity of solving an SDP with MOSEK is the *number of constraint rows which contain semidefinite terms* (in fact complexity depends on it quadratically). Therefore one should as much as possible try to reduce the number of constraints involving symmetric matrix variables. In particular consider whether the primal or dual variant is better in this respect.

The most prominent situation is a sparse $n$-dimensional LMI: $A_0 + \sum_{i=1}^{k} x_i A_i \succeq 0$. The direct representation in MOSEK will be

$$A_0 + \sum_{i=1}^{k} x_i A_i - \overline{X} = 0, \quad \overline{X} \succeq 0$$

with $n(n+1)/2$ equality constraints, one per each entry coordinate $(k, \ell)$, $0 \leq \ell \leq k < n$. (In some APIs one can write the problem indirectly and hide the semidefinite variable $\overline{X}$, but it will still appear internally; for a sparse LMI most of these constraints will just say $\overline{X}_{k,\ell} = 0$). On the other hand the dual of this LMI will have only $k$ constraints of the form

$$\langle A_i, \overline{S} \rangle = c_i, \quad i = 1, \ldots, k, \quad \overline{S} \succeq 0,$$

which is typically orders of magnitude better. Therefore *one should always input a sparse LMI in its dual form.* Already for moderate sizes, like $n \approx 200$ this can mean the difference between not being able to solve the problem at all, and solving it in a few seconds.

You can read more in the Modeling Cookbook and this google groups thread.

# Chapter 4

# License Management

See also the Licensing FAQ where you will find more technical questions about license setup.

## 4.1 I have a problem setting up a floating license server.

See Licensing FAQ for a list of typical problems with solutions.

## 4.2 My license file says VER=8.0. Can I use version 8.1?

Yes. Only the major version number matters.

## 4.3 The Optimization Toolbox for Matlab says license has expired although I downloaded a new one.

After you put the license file in the right place restart Matlab.

## 4.4 Do I want a floating license or a server (node-locked) license?

If you want to perform a moderate amount of optimizations, but they are spread over many machines, then you are more likely to use a floating license. If you are only going to run **MOSEK** on one machine, but you want unlimited number of simultaneous optimizations, then you are more likely to use a server license.

## 4.5 I have one floating license. How many processes/threads can I use?

With one floating license token, one process can use **MOSEK** at a time because at any moment in time there can be at most one process that holds the license token checked out. Inside that process the organization of **MOSEK** optimizations and level of multithreading is not limited by the license.

## 4.6 Does the licensing software make some sort of a network connection to MOSEK to validate the license?

No. Never. All the license system needs is in the license file.

## 4.7 When I change hostid, can I first test the new license while still using the old one?

Of course. We understand that a switch requires some amount of testing, while the old license is still being used. When you are happy that the new setup works, you must disable the old license server.

## 4.8 The lmutil and lmgrd will not start on Linux: No such file or directory.

If you run a command from the licensing system and get this error:

```
user@hostname:~/path_to_mosek$ ./lmutil
./lmutil: No such file or directory
```

then most likely you are using an older version of the token server which required the Linux Standard Base (LSB) package. Either install the package or use the token server from the latest **MOSEK** distribution (at least 10.2).

## 4.9 Can a MOSEK license be used under Citrix?

This is not a case that we test for, but MOSEK employs a floating license scheme by default and that works under Citrix.

## 4.10 How do I do advanced configuration of the license system?

See the FLEXnet license administration guide.

## 4.11 Is it possible to reserve a number licenses to particular group of users or hosts?

Yes it is. To reserve a certain number license features for a particular user or IP address, you must create an additional options file and use the `RESERVE` option. For details see the licensing guide. For example, to reserve one `PTS` token for user `username` make an options file `res.opt` with content

```
RESERVE 1 PTS USER username
```

and add the following to your license file:

```
VENDOR MOSEKLM OPTIONS=res.opt
```

## 4.12 Is it possible to use MOSEK with a floating license on a machine detached from the LAN(WAN)?

Yes. It is possible to use the `lmborrow` functionality. See the licensing guide.

## 4.13 How many license token servers should I have?

You have to consider your usage.

If you are a single user and only want to use **MOSEK** yourself on your machine, then very likely you only want one floating license, with one token server installed on your computer and everything running locally.

In bigger environments it will often be easier and more economical to use a few shared tokens, rather than one per **MOSEK** client installation. You can have them all on one token server, or split between any number of token servers, for example for development and production, or servers dedicated to specific sites or units.

## 4.14 Is it possible to host a token server in the cloud?

It is definitely possible, although some specific steps must be performed. Please see the licensing guide for details.

## 4.15 Can I use the same personal academic license on more than machine?

Yes, as long as it does not violate other license terms, i.e. it is your personal academic/research/educational use.

## 4.16 Why is `os.environ["MOSEKLM_LICENSE_FILE"]` ignored by MOSEK?

When setting the license path via the environment variable `MOSEKLM_LICENSE_FILE` it is generally recommended that the environment variable is set in the environment *before* starting the process.

If that is not possible, and the path is set by modifying the environment from inside the process, then such changes will only be meaningful *before* the first time the **MOSEK** library is loaded. For example, in Python, it must be before the first time `mosek` is imported:

```
import os
os.environ["MOSEKLM_LICENSE_FILE"] = "your/license/path.lic"

import mosek
```

Note that other modules may import `mosek`. For example, the same logic applies to `import cvxpy`.

Another solution is to put the license path in any **MOSEK** environment; that will also put the license path in the global environment. This must be done *before* the first optimization:

```
import mosek
mosek.Env().putlicensepath("your/license/path.lic")
```

# Chapter 5

# Miscellanea

## 5.1 Where is the user manual, API reference, PDF versions, examples?

- The user manuals for all interfaces are available online at http://www.mosek.com/documentation .
- All manuals have a PDF version which can be accessed from the corresponding HTML page (look for *Download PDF* in the left menu).
- PDF manuals are also contained in the distribution (directory `docs`).
- All example files are contained in the distribution package.
- Each manual contains a detailed API reference as the last few sections.

## 5.2 How do I cite MOSEK in an academic publication?

The preferred way to cite MOSEK is to use a BibTex entry as

```
@manual{mosek,
   author = "MOSEK ApS",
   title = "XXX",
   year = 2025,
   url = "YYY"
 }
```

where `YYY` is the link to the relevant manual of the specific API used, and `XXX` is the title of the manual main page. For instance, if you were using the Python Fusion API then you could cite it as:

```
@manual{mosek,
   author = "MOSEK ApS",
   title = "The MOSEK Python Fusion API manual. Version 11.0.",
   year = 2025,
   url = "https://docs.mosek.com/latest/pythonfusion/index.html"
 }
```

The same applies for all other material we provide on the web site.