MOSEK FAQ

Release 9.2.11

MOSEK ApS

12 June 2020

Contents

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_writetask(task, "dump.task.gz");`
	Java	`task.writetask("dump.task.gz");`
	.NET	`task.writetask("dump.task.gz");`
	Python	`task.writetask("dump.task.gz")`
MATLAB Toolbox		`mosekopt('write(dump.task.gz)',prob)`
Rmosek		`r <- mosek_write(prob, "dump.task.gz")`
Command line		`mosek input_file -out dump.task.gz`
Mosek.jl		`Mosek.writetask(task, "dump.task.gz")`
Third party	CVX	`cvx_solver_settings('write', 'dump.task.gz')`
	CVXPY 1.0+	`prob.solve(solver=MOSEK, save_file="dump.task.gz")`
	YALMIP	`sdpsettings('savedebug',1)` followed by (after optimization): `load('mosekdebug')` `mosekopt('min write(dump.task.gz)', prob, param)`
	YALMIP (devel)	`sdpsettings('mosektaskfile', 'dump.task.gz')`
	JuMP <= 0.18	`JuMP.build(model)` `Mosek.writetask(internalmodel(model).task, "dump.task.gz")`
	JuMP >= 0.19	indirect model `model = Model(with_optimizer(Mosek.Optimizer))` after `optimize!` do: `MOI.write_to_file(backend(model).optimizer.model, "dump.task.gz")` direct model: `model = direct_model(Mosek.Optimizer())` before or after `optimize!` do: `MOI.write_to_file(backend(model), "dump.task.gz")`
	Pyomo 5.6.2+	`solver._solver_model.writetask("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 They may become outdated as the interfaces evolve.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 1.0. 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 (optional)
# save_file    - where to save the data file (optional)
# verbose      - enable full log (optional)
result = prob.solve(solver = cp.MOSEK,
                    mosek_params = {mosek.dparam.optimizer_max_time:  100.0,
                                    mosek.iparam.intpnt_solve_form:   mosek.solveform.dual},
                    save_file = 'dump.opf',
                    verbose = True)
```

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.

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 (only available in very recent YALMIP).

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 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. For example

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

```
with SolverFactory("mosek") as solver:
    # options - MOSEK parameters dictionary, using strings as keys
    # tee - write log output if True
    solver.solve(model, options = {'dparam.optimizer_max_time':  100.0,
                                   'iparam.intpnt_solve_form':   int(mosek.solveform.dual)},
                        tee = True)

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

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.

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

1.2.7 amplpy

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

Listing 1.7: 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.

1.2.8 CVXR

This applies to CVXR before version 1.0. Pass options as a list `mosek_params` using generic parameter names as string keys. For example

```
psolve(problem, solver = "MOSEK", mosek_params = list("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.

1.2.9 PuLP

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

Listing 1.8: 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.

1.3 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()`
MATLAB		`[r, res] = mosekopt('version'); res.version`
R		`ver <- mosek_version()`
Command line		`mosek -v`
Julia		`Mosek.getversion()`

1.4 MOSEK is ignoring the limit on the number of threads.

There are a few possible explanations:

- 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 .

1.5 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.

1.6 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.

1.7 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.

1.8 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.

1.9 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 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`.

1.10 Does MOSEK work with MinGW?

Yes. MOSEK includes libraries that are compatible with MinGW and MinGW-w64. Applications built with MinGW should link with `mosek7_0.lib` and `libmosek64_7_0.a` on the platforms `win32x86` and `win64x86` respectively. The native import library (`.lib`) will not work on 64bit Windows.

1.11 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.

Chapter 2

Installation and configuration

2.1 On Mac OS I get errors about missing libmosek64.8.1.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/9.0/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 Security exceptions in Mac OS 10.15 (Catalina)

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.

2.4 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` or `...\win32x86\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.

2.5 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.

2.6 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
```

Chapter 3

Modeling Issues

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

Please read our white paper. Most likely the tricks described in that paper can help you solve your quadratic optimization problem much faster.

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. MOSEK may not always be able to detect non-convexity, but we *strongly* recommend not to apply MOSEK to non-convex 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 Can I get the solution MOSEK produce in every iteration of the optimizer?

No. Since MOSEK performs a lot of transformations on the problem before starting to solve it, the intermediate solution values may make little sense in the context of the user-specified problem. These solution values are not made available.

3.6 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.

3.7 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.

3.8 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.

3.9 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. For example some numerical routines in `numpy`, such as singular value decomposition, are not fully deterministic.
- different number of threads is used.
- some parameter was changed.
- the task was run on a different machine.

3.10 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.

3.11 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.

3.12 Is the mixed integer optimizer parallelized in MOSEK?

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

3.13 How do I model absolute value?

You can model the constraint

$$t \geq |x|$$

with

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

If you need to model the exact equality $t = |x|$ then integer variables are required - this constraint is not convex.

Chapter 4

License Management

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

Yes. Only the major version number matters.

4.2 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. It caches the license.

4.3 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.

4.4 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.

4.5 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.

4.6 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.

4.7 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 the Standard Base (LSB) package version 3 or newer is not installed on your system. On Ubuntu the latest version of LSB can be installed with the command `apt-get install lsb`.

4.8 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.

4.9 How do I do advanced configuration of the license system?

See the FLEXnet license administration guide.

4.10 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 FLEXnet license administration 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
```

4.11 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 FLEXnet license administration guide.

4.12 How many license token servers should I have?

All MOSEK licenses are floating. That means if you install *one* license on *one* computer the computer can act as a token server for itself. In reality it will often be easier and more economical to use few shared token servers rather than one per MOSEK installation. Each token server would require at least one license, and often this would mean that each license is free most of the time. A best practice is to employ one or two token servers for each site, where 2 token servers are employed if licenses for development and production should be kept separate.

4.13 Is it possible to host a token server on Amazon Cloud?

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

4.14 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.

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 = 2019,
   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, citing the MATLAB toolbox would result in

```
@manual{mosek,
   author = "MOSEK ApS",
   title = "The MOSEK optimization toolbox for MATLAB manual. Version 9.0.",
   year = 2019,
   url = "http://docs.mosek.com/9.0/toolbox/index.html"
 }
```

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