MOSEK Optimization Suite

Release 10.2.5

MOSEK ApS

17 September 2024

Contents

Chapter 1

Overview

An optimization problem such as

$$\begin{array}{lcl} \text{minimize} & 1x_1 + 2x_2 & \\ \text{subject to} & x_1 + x_2 & = \quad 1, \\ & x_1, x_2 \geq 0 & \end{array}$$

is an example of a linear optimization problem. In general such problems may be very large and are often compactly written in one of the standard forms such as:

$$\begin{array}{lcl} \text{minimize} & c^T x & \\ \text{subject to} & Ax + b & = \quad 0, \\ & x & \geq \quad 0. \end{array} \tag{1.1}$$

A natural generalization of linear constraints $Ax + b = 0$ are constraints of the form

$$Ax + b \in \mathcal{K}, \tag{1.2}$$

where $\mathcal{K}$ is a more general nonlinear convex set.

Here enters the **MOSEK** Optimization Suite, which is a software package for solving large optimization problems with many constraints and variables, in particular:

- linear problems as in (1.1),
- nonlinear conic problems with constraints as in (1.2), where $\mathcal{K}$ can be the second-order cone, power cone, exponential cone, semidefinite cone and some cones derived from them,
- as well as, independently, problems with quadratic objective and constraints,
- mixed-integer variants of those problems and problems with disjunctive constraints.

In addition to optimization algorithms the **MOSEK** Optimization Suite provides interfaces to mainstream programming languages such as C, Java, MATLAB, .NET, Python, R, Julia or Rust.

Below are the typical constraints and expressions which can be modeled using conic form:

- Conic Quadratic Optimization: $t \geq x^2$, sums of squares, 2-norm $t \geq \|x\|_2$, variance, $\|Ax - b\|_2$.
- Power Cone Optimization: powers x^p, geometric mean, products $x^\alpha y^\beta$, p-norm.
- Exponential Cone: e^x, $\ln x$, log-sum-exp, entropy $x \ln x$, relative entropy, geometric programming.
- Semidefinite Optimization: $X \succeq 0$ is positive semidefinite.

MOSEK provides three solvers:

- A simplex solver for linear problems.
- An interior-point conic solver for linear and nonlinear continuous problems, conic problems and quadratic problems.
- A mixed-integer solver when the problem contains integer variables or disjunctive constraints.

The **MOSEK** Optimization Suite includes

- the low-level optimizer API for C, Java, .NET, Python, Julia and Rust.
- the object-oriented *Fusion* API for C++, Java, .NET and Python.
- an optimization toolbox for MATLAB.
- an Rmosek package for R.
- a command line tool.
- optimizer server for remote optimization.

In addition **MOSEK** Optimization Suite provides

- sensitivity analysis for linear problems.
- infeasibility diagnostic tools.
- problem analyses for diagnosing numerical issues in input data.
- reading and writing optimization problems and solutions from/to files.

Most large optimization problems are very sparse i.e. most of the input data consists of zeros. Therefore, the APIs and optimization algorithms in **MOSEK** Optimization Suite is designed to exploit sparsity to reduce both storage and time.

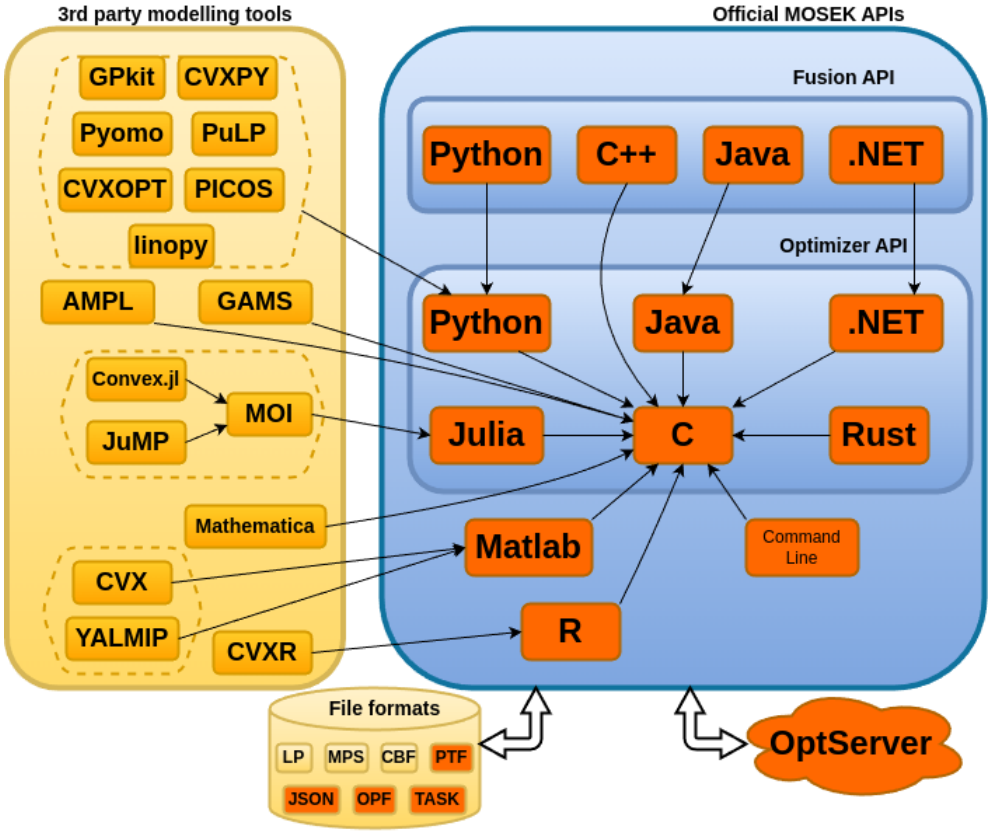
Chapter 2

Interfaces

In this section we give a quick tour of the kinds of interfaces available for **MOSEK**:

- Sec. 2.1 - an object-oriented Fusion API for conic optimization,
- Sec. 2.2 - lower-level matrix-oriented APIs,
- Sec. 2.3 - external modeling languages where **MOSEK** can be plugged in as a solver.

The image below presents a schematic view of the **MOSEK** ecosystem including the official **MOSEK** interfaces and major third-party interfaces.



2.1 Object oriented

An object oriented interface deals directly with variable, expression and constraint objects which can be put together using familiar linear algebra operations. The implemented model can be made similar to the mathematical model one would write directly on paper and the translation from the mathematical model to the implementation is very efficient.

2.1.1 The *Fusion* API

The **MOSEK** *Fusion* API is an object orientated API for expressing conic optimization problems which can include

- linear objective,
- linear constraints of the form $Ax = b$, $Ax \geq b$, $Ax \leq b$, $b_1 \leq Ax \leq b_2$ and similar,
- conic constraints of the form $Ax + b \in K$ where K is a convex cone,
- integer variables,
- disjunctive constraints (a constraint which is an alternative of a number of conditions).

Perhaps surprisingly most convex optimization problems occurring in practice can be expressed in conic form. The advantages of the conic form are

- the problem is convex by construction.
- the problem description is explicit.
- it is not necessary for the user to provide derivative information e.g. gradients.
- almost all the concepts known from linear optimization, like duality, generalize to conic problems.
- a powerful solution algorithm exists.

A typical model in *Fusion* is a direct reflection of the mathematical formulation of the problem. For example, here is a code sample in Python:

```
def BasicMarkowitz(n,mu,GT,x0,w,gamma):

    with  Model("Basic Markowitz") as M:

        # Redirect log output from the solver to stdout for debugging.
        # if uncommented.
        # M.setLogHandler(sys.stdout)

        # Defines the variables (holdings). Shortselling is not allowed.
        x = M.variable("x", n, Domain.greaterThan(0.0))

        #  Maximize expected return
        M.objective('obj', ObjectiveSense.Maximize, x.T @ mu)

        # The amount invested  must be identical to initial wealth
        M.constraint('budget', Expr.sum(x) == w+sum(x0))

        # Imposes a bound on the risk
        M.constraint('risk', Expr.vstack(gamma, GT @ x) == Domain.inQCone())
        #M.constraint('risk', Expr.vstack(gamma, 0.5, Expr.mul(GT, x)), Domain.
↪inRotatedQCone())

        # Solves the model.
        M.solve()
```

(continues on next page)

(continued from previous page)

```
        # Check if the solution is an optimal point
        solsta = M.getPrimalSolutionStatus()
        if (solsta != SolutionStatus.Optimal):
            # See https://docs.mosek.com/latest/pythonfusion/accessing-solution.html␣
↪about handling solution statuses.
            raise Exception(f"Unexpected solution status: {solsta}")

        return np.dot(mu, x.level()), x.level()
```

It is a very compact and straightforward mapping of the mathematical model to code, which looks similar in all other languages supporting *Fusion*.

Fusion is a thin layer on top of the *Optimizer API* and it uses objects to represent

- multi-dimensional variables,
- linear operators and
- domains (typical bounds or cones).

Fusion has been designed with the following principles in mind:

- Expressive: *Fusion* yields readable code.
- Seamlessly multi-language: A *Fusion* model can easily be ported from one supported language and to another supported language.
- Predictability: *Fusion* does very little transformations to the problem before sending the problem to **MOSEK**. The advantage is that the problem solved is predictable for *Fusion* user.
- Efficiency: *Fusion* should only add moderate computational overhead compared to using the optimizer APIs.

Currently, *Fusion* is available for

- Python,
- Java,
- .NET,
- C++ (except Windows 32bit).

Fusion is ideal for fast prototyping of models and in most cases fast enough for production use.

2.2 Matrix oriented

An interface is said to be matrix oriented if it allows inputting optimization problems of the form

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \begin{bmatrix} \leq \\ = \\ \geq \end{bmatrix} b \end{array}$$

where the vectors c and b and matrix A are parts of input. This form is close to the form employed by the optimization algorithm and hence this type of interface has low computational overhead. The disadvantage of a matrix oriented interface is that the problem description is quite different from the formulation the modeler thinks about. Therefore, a lot of work goes into casting the problem into the matrix form. This recasting is typically time consuming and error prone.

For instance consider the problem

$$\begin{array}{lrcl} \text{minimize} & c^T y + t & & \\ \text{subject to} & Fy & = & b, \\ & Gy - z & = & 0, \\ & t \geq \|z\|, & & \\ & y \geq 0. & & \end{array}$$

Observe the problem has three variables i.e. y, z and t. In order to solve the problem with a matrix oriented interface these variables must be mapped to a single variable x, the matrix A has to be formed out of F, G, and so on. This can be cumbersome.

The different matrix oriented interfaces available in the **MOSEK** Optimization Suite are discussed subsequently.

2.2.1 The Optimizer API

The *C Optimizer Application Programming Interface (API)* is the core of the **MOSEK** Optimization Suite because it contains optimization algorithms and a matrix orientated interface that can be used from any C compatible programming language. The C Optimizer API is the most comprehensive API and all other APIs are built on top of that. Hence, it is also the interface with lowest computational overhead. Almost all of its functionality is available from

- Java,
- Python,
- .NET,
- Julia,
- Rust.

The Optimizer API accepts problems with:

- linear objective,
- linear constraints of the form $Ax = b$, $Ax \geq b$, $Ax \leq b$, $b_1 \leq Ax \leq b_2$ and similar,
- integer variables,

and nonlinearities can be inputted either in conic form as:

- conic constraints of the form $Ax + b \in K$ where K is a convex cone,
- disjunctive constraints (a constraint which is an alternative of a number of conditions),

or for the pure QP users as:

- quadratic objective and quadratic constraints.

The code sample

```
for (j = 0; j < numvar && r == MSK_RES_OK; ++j)
{
  /* Set the linear term c_j in the objective.*/
  if (r == MSK_RES_OK)
    r = MSK_putcj(task, j, c[j]);


  /* Set the bounds on variable j.
   blx[j] <= x_j <= bux[j] */
  if (r == MSK_RES_OK)
    r = MSK_putvarbound(task,
                        j,           /* Index of variable.*/
```

(continues on next page)

(continued from previous page)

```
                            bkx[j],      /* Bound key.*/
                            blx[j],      /* Numerical value of lower bound.*/
                            bux[j]);     /* Numerical value of upper bound.*/

      /* Input column j of A */
      if (r == MSK_RES_OK)
        r = MSK_putacol(task,
                        j,                 /* Variable (column) index.*/
                        aptre[j] - aptrb[j], /* Number of non-zeros in column j.*/
                        asub + aptrb[j],   /* Pointer to row indexes of column j.*/
                        aval + aptrb[j]);  /* Pointer to Values of column j.*/
    }
```

illustrates how to input the vector c and matrix A. This should provide the flavor of the interface. A common feature of all the optimizer APIs is a low performance overhead.

2.2.2 The Optimization Toolbox for MATLAB

MOSEK provides its own Optimization Toolbox for MATLAB that gives access to most of the **Optimizer API** functionalities, plus some specialized drivers. It solves problems with:

- linear objective,
- linear constraints of the form $Ax = b$, $Ax \geq b$, $Ax \leq b$, $b_1 \leq Ax \leq b_2$ and similar,
- integer variables,

and nonlinearities can be inputted either:

- in conic form as conic constraints $Ax + b \in K$ where K is a convex cone,
- or for the pure QP users as quadratic objective and quadratic constraints.

The following code

```
c     = [3 1 5 1]';
a     = [[3 1 2 0];[2 1 3 1];[0 2 0 3]];
blc   = [30 15  -inf]';
buc   = [30 inf 25 ]';
blx   = zeros(4,1);
bux   = [inf 10 inf inf]';

[res] = msklpopt(c,a,blc,buc,blx,bux,[],'maximize');
sol   = res.sol;
```

illustrates how to solve a linear optimization problem using the toolbox. Some of the main advantages of using MATLAB compared to say C are: no memory management required and direct operations with sparse vectors and matrices.

There is a MATLAB Optimization Toolbox available from the company MathWorks. For convenience the **MOSEK** Optimization Toolbox for MATLAB provides functions compatible with those in the MATLAB Optimization Toolbox, e.g.

- `linprog`: Solves linear optimization problems.
- `intlinprog`: Solves a linear optimization problem with integer variables.
- `quadprog`: Solves quadratic optimization problems.
- `lsqlin`: Minimizes a least-squares objective with linear constraints.
- `lsqnonneg`: Minimizes a least-squares objective with nonnegativity constraints.

In general the **MOSEK** Optimization Toolbox for MATLAB is not capable of handling all the problem types that the MATLAB Optimization Toolbox can deal with and vice versa. For instance only **MOSEK** can deal with conic optimization problems.

2.2.3 Rmosek

Rmosek is a simple R interface to the **MOSEK** solver. It has features that mimic those of the *Optimizer API*. It solves problems with:

- linear objective,
- linear constraints of the form $Ax = b$, $Ax \geq b$, $Ax \leq b$, $b_1 \leq Ax \leq b_2$ and similar,
- integer variables,

and nonlinearities can be inputted either:

- in conic form as conic constraints $Ax + b \in K$ where K is a convex cone,
- or for the pure QP users as quadratic objective and quadratic constraints.

```
# Objective coefficients
prob$c <- c(3, 1, 5, 1)

# Specify matrix 'A' in sparse format.
asubi  <- c(1, 1, 1, 2, 2, 2, 2, 3, 3)
asubj  <- c(1, 2, 3, 1, 2, 3, 4, 2, 4)
aval   <- c(3, 1, 2, 2, 1, 3, 1, 2, 3)

prob$A <- sparseMatrix(asubi,asubj,x=aval)

# Bound values for constraints
prob$bc <- rbind(blc=c(30,  15, -Inf),
                 buc=c(30, Inf,   25))

# Bound values for variables
prob$bx <- rbind(blx=rep(0,4),
                 bux=c(Inf, 10, Inf, Inf))

# Solve the problem
r <- mosek(prob)
```

2.2.4 The Command Line Tool

The **MOSEK** Optimization Suite includes a command line tool that allows to use **MOSEK** directly. This is quite convenient in many situations:

- testing the license setup,
- performing tests bypassing any API,
- benchmarking the solver without involving API calling.
- solving an optimization problem stored in a file,
- performing infeasibility analysis on a problem dumped to disk from an API.

2.3 Modeling languages

MOSEK is available from several external modeling languages such as

- AMPL
- JuMP
- GAMS
- CVXPY
- Pyomo
- YALMIP
- CVX

(see full list on https://www.mosek.com/resources/third-party/) that make it easy to build optimization models that look almost like the one the modeler has in mind. Hence, the big advantage of modeling languages is convenience and prototyping optimization models is typically extremely fast. However, it is important to remember that such tools:

- add an extra layer between the user and the optimizer,
- can make debugging much more cumbersome,
- do not provide access to full capabilities of **MOSEK**.

2.3.1 AMPL

MOSEK can be used with the modeling language AMPL via the AMPL/MP Library available at https://github.com/ampl/mp

2.3.2 GAMS

MOSEK can be used with the modeling language GAMS. However, a special link must be purchased from GAMS in order to do that. GAMS also provides documentation for how to use **MOSEK** from GAMS.

Chapter 3

Remote optimization

MOSEK is able to off-load optimization problems remotely to a listening remote server. The user's client code is executed locally, but the part that actually invokes the optimizer will be off-loaded to the remote server.

3.1 The OptServer

The OptServer is a server that accepts and executes optimization problems from a **MOSEK** client or other software through a REST API.

The main functionalities are

- receive optimization problems using HTTP/HTTPS protocol,
- accept incoming problem in any file format supported by **MOSEK**,
- store jobs and results for further analysis,
- configure load level,
- run jobs in both synchronous and asynchronous mode,
- manage users, permissions and access tokens.

This full version of the OptServer is only available for Linux 64bit platform but can be used from any client platform.

3.2 The OptServerLight

The OptServerLight is a minimal version of the full OptServer which only supports receiving and solving optimization problems through an API, without any administrative capabilities (job storage, user management, authentication etc. are all stripped away). It runs with almost no configuration and is suitable for use as a stateless **MOSEK** solver microservice. It is optimized to work best with the **MOSEK** clients of the same version.

The OptServerLight is available on all 64-bit platforms supported by **MOSEK**.

Chapter 4

Contact Information

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

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

Blogger	https://blog.mosek.com/
Google Group	https://groups.google.com/forum/#!forum/mosek
Twitter	https://twitter.com/mosektw
Linkedin	https://www.linkedin.com/company/mosek-aps
Youtube	https://www.youtube.com/channel/UCvIyectEVLP31NXeD5mIbEw

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