MOSEK Optimization Suite

Release 8.1.0.37

MOSEK ApS

2018

CONTENTS

CHAPTER

ONE

OVERVIEW

The problem

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

is an example of a linear optimization problem. Now finding a solution to this particular problem is easy. However, in general such optimization problems may be very large. Here enters the **MOSEK** Optimization Suite which is a software package for solving large optimization problems with many constraints and variables. In addition to optimization algorithms the **MOSEK** Optimization Suite provides interfaces to mainstream programming languages such as C, Java, MATLAB, .NET, Python and R.

The purpose of this manual is to provide an overview of the capabilities of the **MOSEK** Optimization Suite.

1.1 Problem types

Table 1.1 summarizes the types of optimization problem that are solvable by the **MOSEK** Optimization Suite.

Table 1.1: Summary of optimization problem types that can be solved with the **MOSEK** Optimization Suite.

Problem type	Available algorithms	Mixed-Integer	Mul
Linear Optimization (LO)	Primal and Dual Simplex	Yes	
	Interior-point	Yes	
Convex Quadratically Constrained (QCQO)	Interior-point	Yes	
Conic Quadratic (Second-Order Cone) Optimization (CQO, SOCO)	Interior-point	Yes	
Semidefinite Optimization (SDO)	Interior-point	No	
General and Separable Convex Optimization (SCO)	Interior-point	No	

1.2 Capabilities

The **MOSEK** Optimization Suite includes

- the low-level optimizer API for C, Java, .NET and Python.
- the object-oriented *Fusion* API for C++, Java, MATLAB, .NET and Python.
- an optimization toolbox for MATLAB.
- an Rmosek package for R.
- an interface to AMPL.

- a command line tool.
- optimizer server for remote optimization.

Fig. 1.1 illustrates the relationship between the parts.

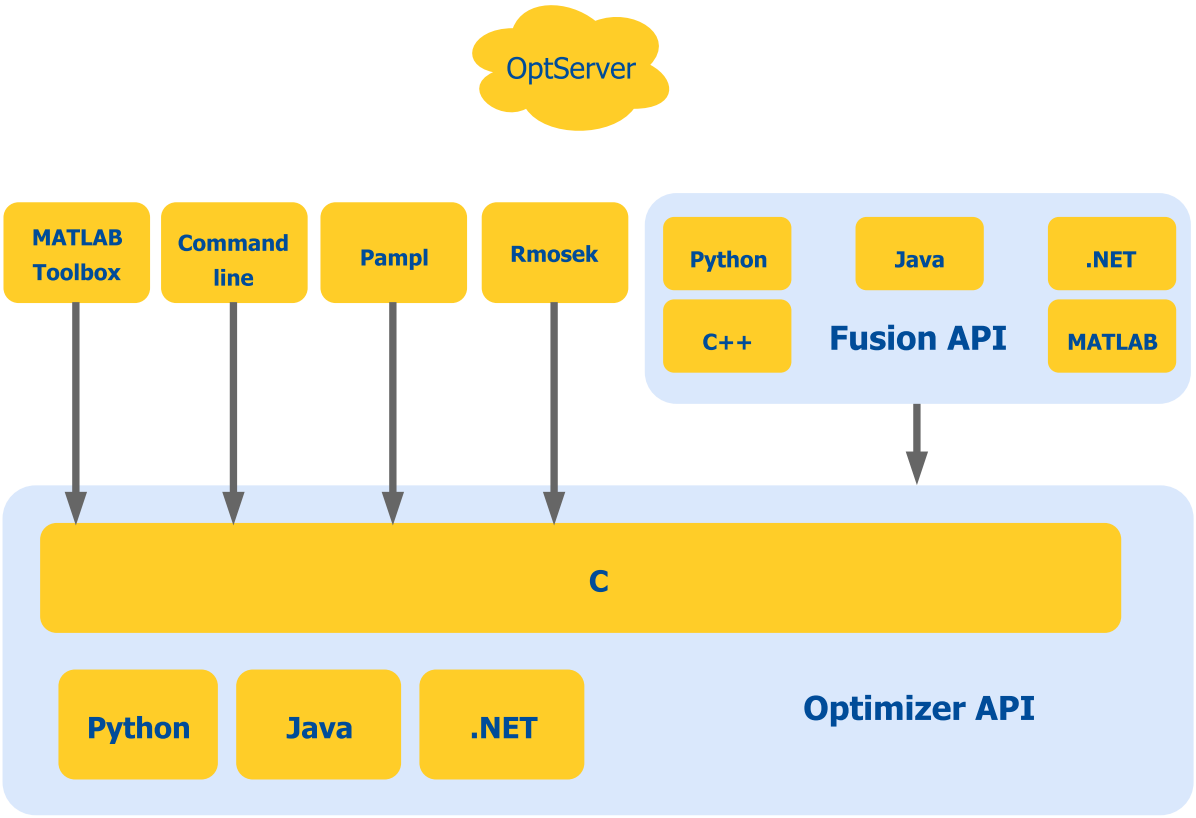

Fig. 1.1: An overview of the API and interfaces available in the **MOSEK** Optimization Suite.

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.

Table 1.2 illustrates the problem types which can be solved via different interfaces.

Table 1.2: Summary of optimization problem types that can be expressed with various **MOSEK** interfaces.

Problem type	Command line	C API	Python, Java, .NET API	Fusion	Matlab Toolbox	RMo
Linear (LO)	Yes	Yes	Yes	Yes	Yes	Ye
Quadratic (QCQO)	Yes	Yes	Yes	No (1)	Yes	Ye
Conic Quadratic (CQO, SOCO)	Yes	Yes	Yes	Yes	Yes	Ye
Semidefinite (SDO)	Yes	Yes	Yes	Yes	Yes	Ye
Separable Convex (SCOPT)	No	Yes	Yes	No	Yes	Ye
General Convex	No	Yes	No	No	No	No

Remarks:

- (1) Quadratic problems can (and should) be recast into Conic Quadratic form and passed to *Fusion*.

CHAPTER

TWO

INTERFACES

2.1 The matrix orientated interfaces

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

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

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 orientated interface is that the problem description is quite different from the formulation the modeller 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 orientated 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 orientated interfaces available in the **MOSEK** Optimization Suite are discussed subsequently.

2.1.1 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.

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]);

  /* 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 flavour of the interface.

Almost all the functionality of the C optimizer API is also available for

- Java,
- Python and
- .NET.

A common feature of all the optimizer APIs is a low performance overhead. However, only the C API makes it possible to solve General Convex Optimization problems.

One could say the Optimizer API for C is the fastest whereas Python is the easiest to use. On the other hand the Java and .NET optimizer APIs add very low extra overhead compared to using the C optimizer API.

2.1.2 The Optimization Toolbox for MATLAB

MATLAB is a popular platform for numerical computing which is also used to solve optimization problems such as constrained least squares problems. **MOSEK** provides a MATLAB toolbox that gives access to most of the **Optimizer API** functionalities, plus some specialized drivers.

The following code

```
c     = [1 2 0]';
a     = sparse([[1 0 1];[1 1 0]]);
blc   = [4 1]';
buc   = [6 inf]';
blx   = sparse(3,1);
bux   = [];
[res] = msklpopt(c,a,blc,buc,blx,bux);
```

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 **MOSEK** optimization toolbox 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 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.

2.1.3 Rmosek

Rmosek is a simple R interface to the **MOSEK** solver. For more information see http://rmosek.r-forge.r-project.org/ .

2.1.4 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.

2.2 An object orientated interface

An object orientated interface deals directly with variable and constraint objects and the implemented model can be made similar to the model the modeller/user have in mind. This typically reduces the time to build a correct optimization dramatically.

2.2.1 The *Fusion* API

The **MOSEK** *Fusion* API is an object orientated API for expressing conic optimization problems on the form

$$\begin{array}{lcl} \text{minimize} & c^T x & \\ \text{subject to} & A^i x + b \in K^i & \forall i. \end{array}$$

where K^i is a convex cone i.e. a quadratic, rotated quadratic or semidefinite cone. Although not shown it is possible to have multiple variables and each variable can be multi-dimensional.

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.

Next let us use the model

$$\begin{array}{lcl} \text{maximize} & \mu^T x & \\ \text{subject to} & \sum_j x_j = 1, & \\ & \begin{bmatrix} \gamma \\ G^T x \end{bmatrix} \in \mathcal{Q} & \end{array}$$

where $\mathcal{Q}$ denotes the quadratic cone, i.e. the last constraint is equivalent to $\|G^T x\| \leq \gamma$. The implementation in Python *Fusion* is the following:

```
with  Model("Basic Markowitz") as M:

    x = M.variable("x", n, Domain.greaterThan(0.0))

    M.objective('obj', ObjectiveSense.Maximize, Expr.dot(mu,x))

    M.constraint('budget', Expr.sum(x), Domain.equalsTo(1.0))

    M.constraint('risk', Expr.vstack(gamma, Expr.mul(GT,x)), Domain.inQCone())

    M.solve()

    print(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,
- MATLAB,
- C++ (except Windows 32bit).

Fusion is ideal for fast prototyping of models and in many cases fast enough for production use. However, it should be mentioned that the Python *Fusion* is noticeably slower than its counterparts in other languages. However, the Python *Fusion* is extremely convenient to use and ideal for fast prototyping.

2.3 Modelling languages

There exist several modelling languages such as

- AMPL and
- GAMS

that make it easy to build optimization models that look almost like the one the modeller has in mind. Hence, the big advantage of modelling languages is convenience and prototyping optimization models is typically extremely fast. In a **MOSEK** context modelling languages have a big advantage for general nonlinear models because they compute all derivative information such as gradients and Hessians

needed by **MOSEK**. Therefore, it is strongly recommended to use a modelling language for prototyping nonlinear convex models because the possibilities for errors are reduced dramatically.

The drawbacks of modelling languages are

- they do not integrate so well with common programming languages.
- they do not support conic optimization very well if at all.
- they can add nontrivial computational overhead.

2.3.1 AMPL

The **MOSEK** command line tool provides a link to AMPL. Please consult the **MOSEK** command line tool documentation for how to use it.

2.3.2 GAMS

MOSEK can be used with the modelling 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

THREE

REMOTE OPTIMIZATION

3.1 The OptServer

Since version 8 **MOSEK** is able to off-load optimization problems remotely to a listening server both in a synchronous and asynchronous way. The OptServer is a simple server that accepts and executes optimization problems from a **MOSEK** client or using HTTP commands.

The main functionalities are

- receive optimization problems using HTTP/HTTPS protocol,
- accept incoming problem in any file format supported by **MOSEK**,
- OptServer also acts as a tiny web server to provide a minimal GUI for management.

Observe the OptServer is only available for Linux 64bit platform but can be used from any client platform. The OptServer is distributed as a binary along with a few Python scripts that can be easily modified by the user.

CHAPTER

FOUR

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

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