`Fusion` : an object-oriented API for conic optimization by MOSEK

OptALI Industry Day
June 1-2, 2015, Copenhagen (DK)

Andrea Cassioli, PhD

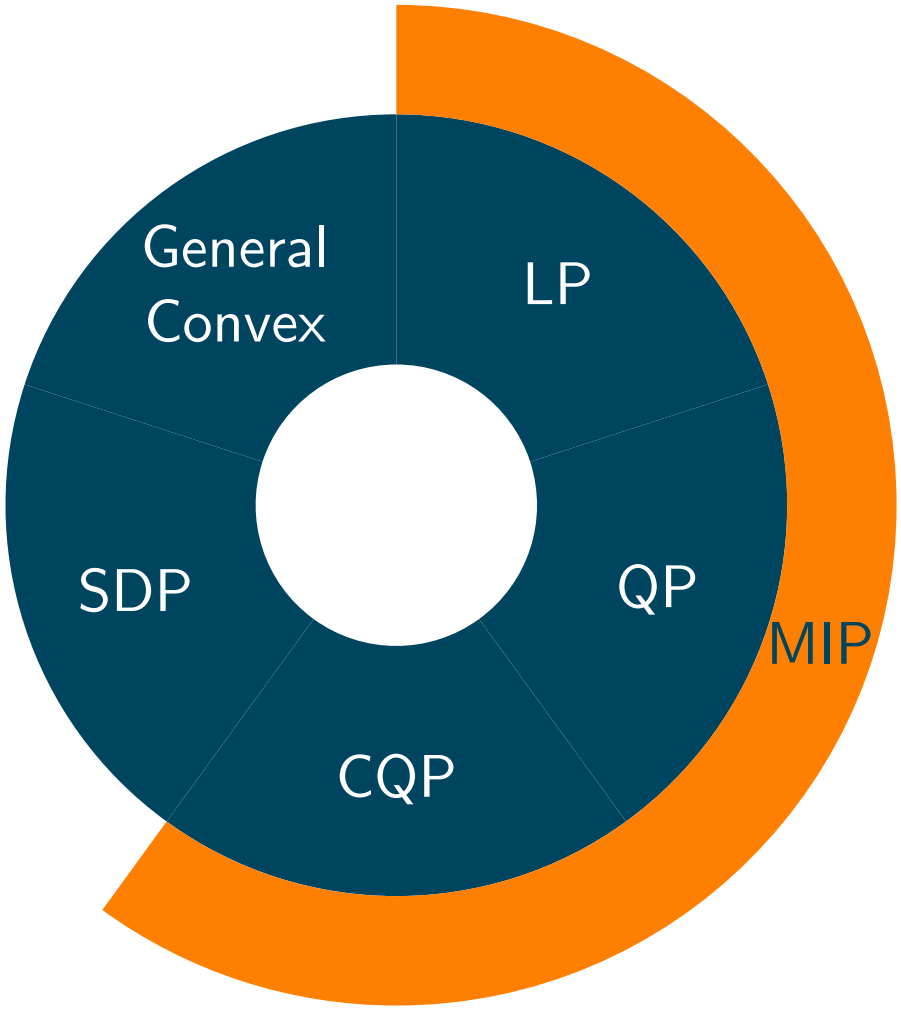
andrea.cassioli@mosek.com

www.mosek.com

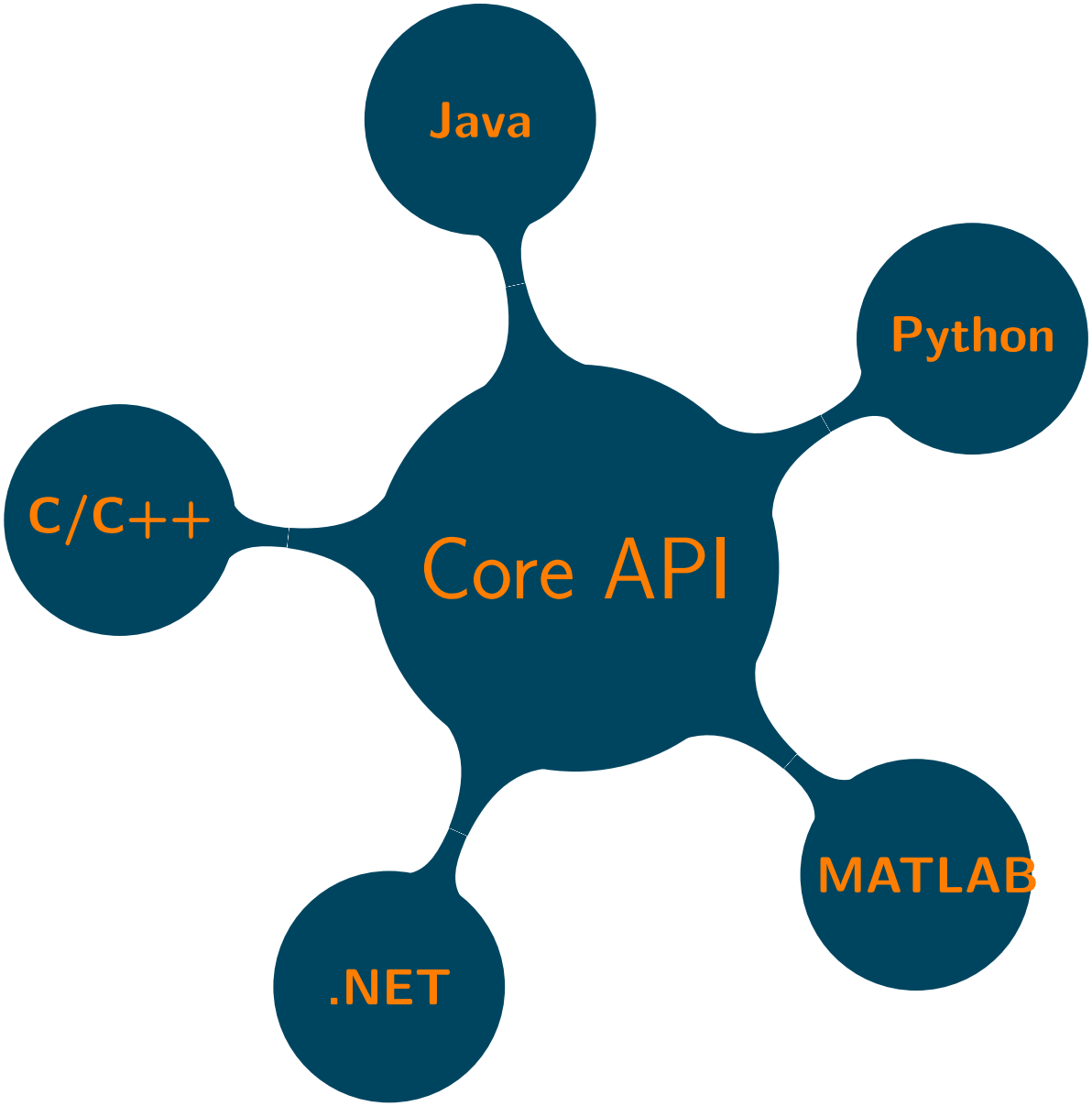
Overview

- About **MOSEK**
- Conic Optimization in a Nutshell
- Conic Optimization with `Fusion`
- Conclusion

MOSEK is one of the leading provider of high-quality optimization software world-wide.



MOSEK APIs



Conic Optimization - Definition

$$\min \quad \sum_j \langle C_j, X_j \rangle$$

$$s.t.$$

$$\sum_j \langle A_{ij}, X_j \rangle + B_i \in \mathcal{K}_i^{n_i} \quad i = 1, \ldots, k$$

where $\mathcal{K}_i^{n_i}$ is one of the following

$$\mathbb{R}_+^{n_i} = \{x \in \mathbb{R}^{n_i} | x_j \geq 0 \quad \forall j = 1, \ldots, n_i\}$$

$$\mathcal{Q}^{n_i} = \left\{x \in \mathbb{R}^{n_i} | x_1 \geq \sqrt{\sum_{j=2}^{n_i} x_j^2}\right\}$$

$$\mathbb{S}_+^{n_i} = \left\{X \in \mathbb{M}^{n_i \times n_i} | X = X^T \text{ and positive semidefinite}\right\}$$

Conic Optimization Modeling

A huge number of functions are *conic representable*, eg.

- geometric and harmonic means
- convex (de)increasing rational powers
- sum of eigenvalues
- spectral radius
- roots of the determinant
- nonnegativity and convexity of polynomials

...other than plain LPs, QCQPs, CQPs, SPDs!

Conic Optimization - Main benefits

- conicity implies convexity
- rich duality theory
- polinomial complexity upper bound

Users should use conic formulations directly and with full understanding of the benefits.

Sadly conic optimization not yet well know:

- not yet a common teaching
- sometimes hidden under the hood
- lack of material on modeling (see our Modeling Cook Book)
- cumbersome interfaces!

Modern frameworks (Yalmip, CVX, PICOS,...) help a lot!

But....

- Lack of transparency and control
- Performance degradation
- Additional dependencies, support, etc....

Modern frameworks (Yalmip, CVX, PICOS,...) help a lot!

But....

- Lack of transparency and control
- Performance degradation
- Additional dependencies, support, etc....

Fusion

Object-Oriented API for Conic Optimization

(almost) Seamlessly multi-language

(C#, Java, Python, Matlab)

Minimalistic and Safe

Extremely Disciplined Conic Opt.

Limited overhead

MOSEK gets what you write

Fusion

Object-Oriented API for Conic Optimization

(almost) Seamlessly multi-language

(C#, Java, Python, Matlab)

Minimalistic and Safe

Extremely Disciplined Conic Opt.

Limited overhead

MOSEK gets what you write

Fusion

Object-Oriented API for Conic Optimization

(almost) Seamlessly multi-language

(C#, Java, Python, Matlab)

Minimalistic and Safe

Extremely Disciplined Conic Opt.

Limited overhead

MOSEK gets what you write

Fusion

Object-Oriented API for Conic Optimization

(almost) Seamlessly multi-language

(C#, Java, Python, Matlab)

Minimalistic and Safe

Extremely Disciplined Conic Opt.

Limited overhead

MOSEK gets what you write

Fusion - Basic Idea

$$\min \quad \sum_j \langle C_j, X_j \rangle$$

$$s.t.$$

$$\sum_j \langle A_{ij}, X_j \rangle + B_i \in \mathcal{K}_i^{n_i} \quad i = 1, \ldots, k$$

Basic constructs:

```
obj. fun. := ( linear expression , sense)
variable  := ( dimension , domain)
constraint:= ( linear expression , domain)
```

Linear expressions need:

```
linear operators
matrices
variables
```

Fusion - Basic Idea

$$\min \quad \sum_j \langle C_j, X_j \rangle$$

$$s.t.$$

$$\sum_j \langle A_{ij}, X_j \rangle + B_i \in \mathcal{K}_i^{n_i} \quad i = 1, \ldots, k$$

Basic constructs:

```
obj. fun. := ( linear expression , sense)
variable  := ( dimension , domain)
constraint:= ( linear expression , domain)
```

Linear expressions need:

```
linear operators
matrices
variables
```

Fusion - Basic Idea

$$\min \quad \sum_j \langle C_j, X_j \rangle$$

$$s.t.$$

$$\sum_j \langle A_{ij}, X_j \rangle + B_i \in \mathcal{K}_i^{n_i} \quad i = 1, \ldots, k$$

Basic constructs:

```
obj. fun. := ( linear expression , sense)
variable  := ( dimension , domain)
constraint:= ( linear expression , domain)
```

Linear expressions need:

```
linear operators
matrices
variables
```

Fusion

A single class to rule them all: `Model`

Model building:

```
var  = M.variable( [name], dimension, domain)
cons = M.constraint( [name], linear expression , domain)
M.objective( [name], sense, linear expression)
```

Solver access:

```
M.solve()
...
```

Note: each model owns the objects it builds!

`Fusion` only performs the following transformation:

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

becomes

$$\begin{aligned} Ax + b &= y \\ y &\in \mathcal{K}, \end{aligned}$$

All other manipulations are left to the solver.

`Fusion` Example - Portfolio Optimization

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & 1, \\ & x^T F x & \leq & \gamma^2, \\ & x & \geq & 0, \end{array}$$

Since $F = GG^T$, then $x^T F x = \|G^T x\|^2 \leq \gamma^2$,

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & 1, \\ & (\gamma, G^T x) & \in & \mathcal{Q}^{n+1}, \\ & x & \geq & 0, \end{array}$$

Fusion Example - Portfolio Optimization

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & 1, \\ & x^T F x & \leq & \gamma^2, \\ & x & \geq & 0, \end{array}$$

Since $F = GG^T$, then $x^T F x = \|G^T x\|^2 \leq \gamma^2$,

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & 1, \\ & (\gamma, G^T x) & \in & \mathcal{Q}^{n+1}, \\ & x & \geq & 0, \end{array}$$

Fusion Example - Portfolio Optimization

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & 1, \\ & x^T F x & \leq & \gamma^2, \\ & x & \geq & 0, \end{array}$$

Since $F = GG^T$, then $x^T F x = \|G^T x\|^2 \leq \gamma^2$,

$$\begin{array}{lrcl} \text{maximize} & \mu^T x & & \\ \text{subject to} & e^T x & = & 1, \\ & (\gamma, G^T x) & \in & \mathcal{Q}^{n+1}, \\ & x & \geq & 0, \end{array}$$

```
M = Model('Basic Markowitz')

# Defines the variables (holdings).
x = M.variable(n, Domain.greaterThan(0.0))

# The amount invested  must be 1
M.constraint(Expr.sum(x), Domain.equalsTo(1.))

# Imposes a bound on the risk
conevars = Expr.vstack( gamma,Expr.mul(GT,x))
M.constraint(conevars, Domain.inQCone())

#  Maximize expected return
M.objective(ObjectiveSense.Maximize, Expr.dot(mu,x))

M.solve()

print x.level()
```

Example - Nearest Correlation

$$
\begin{array}{ll}
\min & \|A - X\|_F \\
s.t. & \\
 & X_{ii} = 1 \qquad i = 1, \ldots, n \\
 & X \in \mathbb{S}^n_+
\end{array}
$$

but being $\|A - X\|_F = \sqrt{\sum_{ij}(A_{ij} - X_{ij})^2}$ we get

$$
\begin{array}{ll}
\min & t \\
s.t. & \\
 & X_{ii} = 1 \qquad\qquad\qquad i = 1, \ldots, n \\
 & (t, A_{11} - X_{11}, \ldots) \in \mathcal{Q} \\
 & X \in \mathbb{S}^n_+
\end{array}
$$

Example - Nearest Correlation

$$\begin{array}{ll} \min & \|A - X\|_F \\ s.t. & \\ & X_{ii} = 1 \qquad i = 1, \ldots, n \\ & X \in \mathbb{S}^n_+ \end{array}$$

but being $\|A - X\|_F = \sqrt{\sum_{ij}(A_{ij} - X_{ij})^2}$ we get

$$\begin{array}{ll} \min & t \\ s.t. & \\ & X_{ii} = 1 \qquad i = 1, \ldots, n \\ & (t, A_{11} - X_{11}, \ldots) \in \mathcal{Q} \\ & X \in \mathbb{S}^n_+ \end{array}$$

Example - Nearest Correlation

$$
\begin{array}{ll}
\min & \|A - X\|_F \\
s.t. & \\
 & X_{ii} = 1 \qquad i = 1, \ldots, n \\
 & X \in \mathbb{S}^n_+
\end{array}
$$

but being $\|A - X\|_F = \sqrt{\sum_{ij}(A_{ij} - X_{ij})^2}$ we get

$$
\begin{array}{ll}
\min & t \\
s.t. & \\
 & X_{ii} = 1 \qquad\qquad\qquad i = 1, \ldots, n \\
 & (t, A_{11} - X_{11}, \ldots) \in \mathcal{Q} \\
 & X \in \mathbb{S}^n_+
\end{array}
$$

```
M = Model('NearestCorrelation')

X = M.variable('X', Domain.inPSDCone(N))
t = M.variable('t', 1, Domain.unbounded())

# (t, vec(A-X)) \in Q
delta= Expr.sub(A,X).flatten()
M.constraint(Expr.vstack(t, delta), Domain.inQCone())

# diag(X) = e
M.constraint(X.diag(), Domain.equalsTo(1.0))

M.objective(ObjectiveSense.Minimize, t)

M.solve()

print X.level(), t.level()
```

A primer

MOSEK and SolverStudio!

Thanks to the Python interface, **MOSEK** can be called from SolverStudio with no additional setup!

- all functionalities of the Optimizer and `Fusion` Python APIs are available
- `Fusion` fits particularly well!

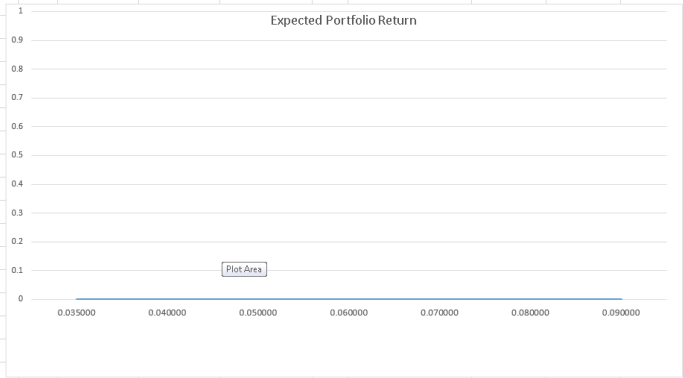
diet - Excel

Assets	Returns	x0
stock1	0.1073	0
stock2	0.0737	0
stock3	0.0627	0

Factorized Covariance		
0.5271	0.0734	0.004
0	0.3253	-0.007
0	0	0.1069

Total Assets	Covariance multiplier
1	0.316228

	Solutions						
Assets\Gamma	0.035000	0.040000	0.050000	0.060000	0.070000	0.080000	0.090000
expected returns	0	0	0	0	0	0	0



Sheet1 | Sheet2 | Sheet3 | Sheet4 | Sheet4 (2)

READY

SolverStudio ©Andrew Mason

File Edit Language Python

```
import mosek
from mosek.fusion import *
from SolverStudio import *
import sys


mu = [ Returns[i] for i in Assets]
x0= [ InitAssets[i] for i in Assets]
GT= [ [ CovarMult*FactCovariance[j,i] for i in Assets] for j in Assets]


for g in Gamma:
    with  Model("Basic Markowitz") as M:


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

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

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

        # Imposes a bound on the risk
        M.constraint('risk', Expr.vstack( g, Expr.mul(GT,x)), Domain.inQCone())

        # Solves the model.
        M.solve()

        for i,xj in zip( Assets, x.level()):
            Solutions[(i,g)] = xj
```

Model Output

```
## Executing C:\Users\andreacassioli\programs\SolverStudio\SolverStudio\CPython\RunCPython.py
## Building CPython input file, module 'SolverStudio.py', for 8 data items
## Writing data items
## Running CPython (C:\Python27\python.exe)
## with file: C:\Users\andreacassioli\AppData\Local\Temp\4\SolverStudio hdgwdrfs\model.py


## CPython run completed successfully.
## New values loaded for data items: Solutions
## Done
```

Row 24

diet - Excel

Assets	Returns	x0
stock1	0.1073	0
stock2	0.0737	0
stock3	0.0627	0

Factorized Covariance		
0.5271	0.0734	0.004
0	0.3253	-0.007
0	0	0.1069

Total Assets
1

Covariance multiplier
0.316228

	Solutions						
Assets\Gamma	0.035000	0.040000	0.050000	0.060000	0.070000	0.080000	0.090000
	0.101014	0.155231	0.236472	0.306385	0.371880	0.434836	0.496367
	0.115437	0.124898	0.139337	0.152042	0.163330	0.174648	0.185514
	0.783549	0.719872	0.624190	0.541573	0.464790	0.390517	0.318120
expected returns	0.06848	0.071	0.07478	0.07804	0.08108	0.08401	0.08688

SolverStudio ©Andrew Mason

File Edit Language Python

```
import mosek
from mosek.fusion import *
from SolverStudio import *
import sys


mu = [ Returns[i] for i in Assets]
x0= [ InitAssets[i] for i in Assets]
GT= [ [ CovarMult*FactCovariance[j,i] for i in Assets] for j in Assets]


for g in Gamma:
    with  Model("Basic Markowitz") as M:


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

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

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

        # Imposes a bound on the risk
        M.constraint('risk', Expr.vstack( g, Expr.mul(GT,x)), Domain.inQCone())

        # Solves the model.
        M.solve()

        for i,xj in zip( Assets, x.level()):
            Solutions[(i,g)] = xj
```

Model Output

```
## Executing C:\Users\andreacassioli\programs\SolverStudio\SolverStudio\CPython\RunCPython.py
## Building CPython input file, module 'SolverStudio.py', for 8 data items
## Writing data items
## Running CPython (C:\Python27\python.exe)
## with file: C:\Users\andreacassioli\AppData\Local\Temp\4\SolverStudio hdgwdrfs\model.py


## CPython run completed successfully.
## New values loaded for data items: Solutions
## Done
```

Sheet1 Sheet2 Sheet3 Sheet4 Sheet4 (2)

READY

Row 24

`Fusion` - What's next

Upcoming **MOSEK** version 8 will include an extensive `Fusion` refactoring, including

- improved perfomance
- support for C++11
- new expressions and syntactic sugar
- pretty printing

`Fusion` - What's next

Upcoming **MOSEK** version 8 will include an extensive `Fusion` refactoring, including

- improved perfomance
- support for C++11
- new expressions and syntactic sugar
- pretty printing

`Fusion` - What's next

Upcoming **MOSEK** version 8 will include an extensive `Fusion` refactoring, including

- improved perfomance
- support for C++11
- new expressions and syntactic sugar
- pretty printing

`Fusion` - What's next

Upcoming **MOSEK** version 8 will include an extensive `Fusion` refactoring, including

- improved perfomance
- support for C++11
- new expressions and syntactic sugar
- pretty printing

Thank you!

Andrea Cassioli, PhD

andrea.cassioli@mosek.com

www.mosek.com