Modeling with MOSEK Fusion

Ulf Worsøe

INFORMS Minneapolis
October 5 2013

What is Fusion?

What is Fusion?

- Fusion is a modern object oriented API for conic optimization in MOSEK available for
  - Matlab
  - Java 1.6+
  - .NET 2.2+
  - Python 2.6+
- Fusion is designed to be as efficient as possible while making it easy to develop models.
- Fusion includes a library of generic functionality to assist model building.

Why Fusion?

- Developing complex models in the optimizer API is time consuming and error prone — especially so for semi-definite programming introduced in MOSEK 7.0.
- Several customers have built their own Fusion-like functionality to be able to implement complex models.
- Fusion allows only conic models which can be solved very efficiently in MOSEK.
- Fusion *allows and encourages* vectorized formulations making the model building more efficient than many third party interfaces and modeling languages.

Finally, Fusion is implemented to be as efficient as possible: Conic optimization can be solved very efficiently, and the model building phase should not dominate in terms of time.

Even if the last seconds mean everything, using Fusion for prototyping decreases the model development time.

mosek
http://www.mosek.com

Let’s get to the code part!

Portfolio model

This is a variant of the Markowitz portfolio model that we often see:

$$\begin{array}{rl} \text{minimize} & x^T(G^TG)x + \sum_{i=1}^{n} m_i x^{3/2} \\ \text{such that} & r^T x = t \\ & x \in \mathbb{R}^n,\ x \geq 0 \end{array}$$

This model assumes that we have no initial investment and that we require a certain return.
Here:

- $x^T G^T G x$ is the variance (or *risk*) of the portfolio x,
- $\sum_{i=1}^{n} m_i x^{3/2}$ is the market impact term, and
- $r^T x$ is the expected return of the portfolio x

Portfolio model transformed

The conic form of this:

$$
\begin{array}{rl}
\text{minimize} & z + m^T y \\
\text{such that} & r^T x = t \\
& 2 \cdot (1/2) \cdot z \geq ||Gx||_2^2 \\
& 2 y_i w_i \geq x_i^2, \text{ for } i = 1 \ldots n \\
& 2 \cdot (1/8) \cdot x_i \geq w_i^2, \text{ for } i = 1 \ldots n \\
& x \in \mathbb{R}^n, \; x \geq 0G
\end{array}
$$

The three non-linear constraints can be implemented using the rotated quadratic cone of dimension 3:

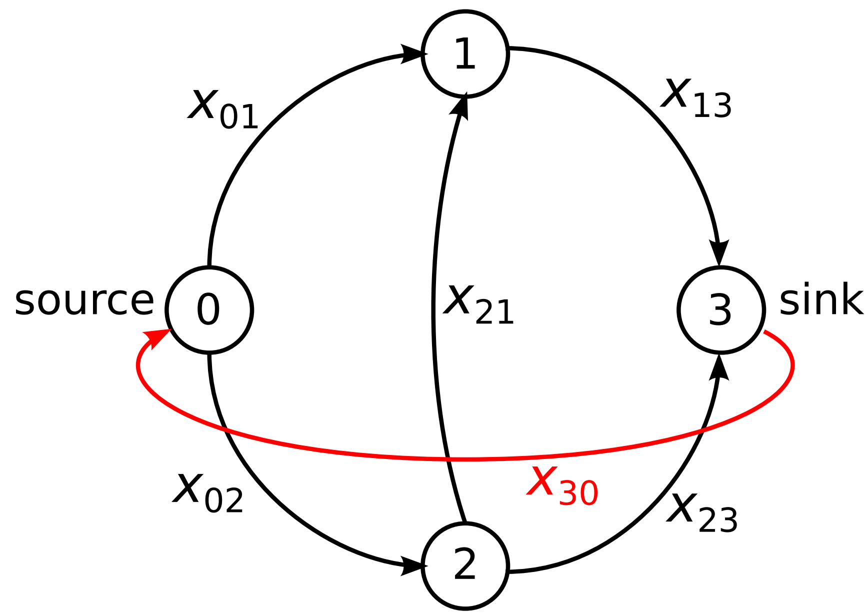
$$Q_r^3 = \{x \in \mathbb{R}^3 | 2 x_1 x_2 \geq x_3^2\}$$

Portfolio in Fusion/Python

```
from mosek.fusion import *

def portfolio(G,m,r,t):
  n = len(m)

  with Model("Markowitz") as M:
    x = M.variable(n,Domain.greaterThan(0.0))
    y = M.variable(n,Domain.unbounded())
    z = M.variable(1,Domain.unbounded())
    w = M.variable(n,Domain.unbounded())

    M.constraint(Expr.mul(r,x), Domain.equalsTo(t))
    M.constraint(Expr.vstack(0.5,z,Expr.mul(G,x)),
                 Domain.inRotatedQCone())
    M.constraint(Expr.hstack(y,w,x), Domain.inRotatedQCone())
    M.constraint(Expr.hstack(Expr.constTerm(n,.125),x,w), Domain.inRotatedQCone())
    M.objective(ObjectiveSense.Minimize, Expr.add(z,Expr.dot(m,y)))

    M.solve()
    return x.level()

if __name__ == '__main__':
  G = DenseMatrix(3,3, [ 0.16667,0.02322, 0.00126,
                         0,      0.10286,-0.00223,
                         0,      0,       0.03381 ])
  r = [ 0.1073, 0.0737, 0.0627 ]
  m = [ 0.01,   0.01,   0.01 ]

  print "x␣=",portfolio(G,m,r,0.08)
```

Traffic flow network

Traffic network model based on a presentation by Robert Fourer (Convexity Detection in Large-Scale Optimization., 2011. OR 53 Nottingham 6-8 September 2011).



The red arc is added to simplify the formulation of the model, but it has infinite capacity and is not included in the objective.

Traffic flow network: original form

$$
\begin{array}{ll}
\text{minimize} & \displaystyle\sum_{(i,j)\in A} t_{ij} x_{ij}/T \\
\text{such that} & t_{ij} = b_{ij} \dfrac{s_{ij} x_{ij}}{1 - x_{ij} c_{ij}},\ (i,j) \in A \\
& \displaystyle\sum_{j:(i,j)\in A_+} x_{ij} = \sum_{j:(j,i)\in A_+} x_{ji},\ i \in N \\
& x_{es} = T \\
& 0 \le x_{ij} \le c_{ij}, (i,j) \in A
\end{array}
$$

where N is the set of nodes, A is the set of arcs and A_+ is the set of arcs plus an arc from sink to source. c_{ij} is the capacity and s_{ij} is the sensitivity of arc (i,j).

Traffic flow network: conic form

$$
\begin{aligned}
\text{minimize} \quad & \sum_{(i,j)\in A} \frac{1}{T}\left(b_{ij}x_{ij} + y_{ij}\right) && (1)\\
\text{such that} \quad & 2\frac{1 - x_{ij}c_{ij}}{2s_{ij}} y_{ij} \geq x_{ij}^2,\ (i,j) \in A && (2)\\
& \sum_{j:(i,j)\in A_+} x_{ij} = \sum_{j:(j,i)\in A_+} x_{ji},\ i \in N && (3)\\
& x_{es} = T && (4)\\
& 0 \leq x_{ij} \leq c_{ij}, (i,j) \in A && (5)
\end{aligned}
$$

- Objective (1) is now linear.
- The term t is completely gone (in fact we substituted t into the original objective).
- Constraint (2) is a rotated quadratic cone.

Traffic flow in Fusion/Python

```
def main(N,E, source,sink, arc_sensitivity, arc_capacity, arc_baseTravelTime, T):
  with Model("Traffic␣Network") as M:
    arc_i = [ i for i,j in E ]
    arc_j = [ j for i,j in E ]
    e     = Matrix.sparse(N,N, arc_i,arc_j, 1.0)

    c = Matrix.sparse(N,N,arc_i,arc_j, arc_capacity)
    cplus = Matrix.sparse(N,N,arc_i + [sink],arc_j + [source], arc_capacity + [T])
    # Set up (5)
    x = M.sparseVariable('x', NDSet(N,N), Domain.inRange(0.0, cplus))
    y = M.sparseVariable('y', NDSet(N,N), Domain.unbounded())
    # Set up (1)
    b = Matrix.sparse(N,N, arc_i,arc_j,arc_baseTravelTime)
    M.objective(ObjectiveSense.Minimize,
                Expr.mul(1.0/T, Expr.add((Expr.dot(x,b), Expr.dot(y,e)))))
    # Set up (2)
    y_sel = y.pick_flat([ i*N+j for (i,j) in E])
    x_sel = x.pick_flat([ i*N+j for (i,j) in E])
    one_div_2s = [0.5/s for s in arc_sensitivity]

    M.constraint('(2)',
      Expr.hstack(Expr.mulElm(Expr.sub(1.0,Expr.mulElm(x_sel,[ 1.0/c for c in arc_capacity ])),
                              one_div_2s),
                  y_sel,
                  x_sel),
      Domain.inRotatedQCone())
    # Set up (3)
    eplus_T = Matrix.sparse(N,N, arc_j+[source],arc_i+[sink], 1.0)
    M.constraint('(3)', Expr.sub(Expr.mulDiag(x,eplus_T), Expr.mulDiag(eplus_T,x)),
                 Domain.equalsTo(0.0))
    # Set up (4)
    M.constraint('(4)', x.index(sink,source), Domain.equalsTo(T))

    M.solve()
    return x_sel.level()
```

Modeling a complex cone: Geometric mean cone

It is possible to model several complex sets using quadratic cones. One example: The geometric mean (GM) cone:

$$t \leq \left(\prod_{i=1}^{n} x_i \right)^{1/n}, \; x_i \geq 0$$

We notice first that the GM cone of size 3 is in fact almost the rotated quadratic cone of size 3:

$$t \leq \sqrt{2 x_1 x_2} \;\Leftrightarrow\; (x_1, x_2, t) \in \mathcal{Q}_r^3$$

so, e.g., the GM cone of size 5 can then be implemented as:

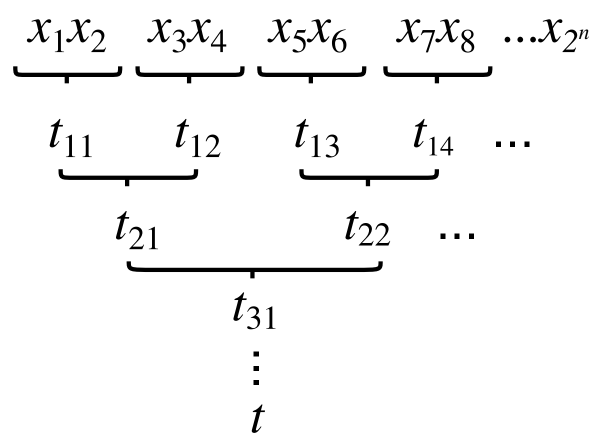
$$\sqrt{2}t \leq, \; \sqrt{2 t_1 t_2}, \; \sqrt{2} t_1 \leq \sqrt{2 x_1 x_2}, \; \sqrt{2} t_2 \leq \sqrt{2 x_3 x_4}$$

$$\Leftrightarrow$$

$$(t_1, t_2 \sqrt{2} t), \; (x_1, x_2, \sqrt{2} t_1), \; (x_3, x_4, \sqrt{2} t_2) \in \mathcal{Q}_r^3$$

GM cone of power 2

We notice that this approach allows us to build GM cones of size $1 + 2^n$.



$$
\begin{gathered}
(x_1, x_2, \sqrt{2}t_{11}), (x_3, x_4, \sqrt{2}t_{12}), (x_5, x_6, \sqrt{2}t_{13}), \ldots \in \mathcal{Q}_r^3 \\
(t_{11}, t_{12}, \sqrt{2}t_{21}), (t_{13}, t_{14}, \sqrt{2}t_{22}), \ldots \in \mathcal{Q}_r^3 \\
(t_{21}, t_{22}, \sqrt{2}t_{31}), \ldots \in \mathcal{Q}_r^3 \\
\vdots \\
(t_{\log_2 n-2,1}, t_{\log_2 n-2,2}, \sqrt{2}t) \in \mathcal{Q}_r^3
\end{gathered}
$$

GM cone

We can now formulate

$$(t, x_1, x_2, \ldots, x_{2^p}) \in \mathcal{G}^{2^p+1} : \; t^n \leq x_1 x_2 x_3 \ldots x_{2^p}$$

It now only remains to observe that for $n < 2^p$ we can write the GM cone of $n+1$ elements as the GM cone of 2^p+1 elements:

$$t^{2^p} \leq x_1 x_2 \cdots x_n \cdot t^{2^p-n}$$

or

$$(t, x_1, x_2, \ldots, x_n, t, \ldots, t) \in \mathcal{G}^{2^p+1}$$

A 2^p+1 GM cone requires 2^p-2 extra variables and 2^p-1 rotated quadratic cones of size 3.

Geometric mean cone in Fusion/Python

```
def geometric_mean(M,x,t):
  '''
  Models the convex set

    S = { (x, t) \in R^n x R | x >= 0, t <= (x1 * x2 * ... * xn)^(1/n) }

  as the intersection of rotated quadratic cones and affine hyperplanes.
  '''
  def rec(x):
    n = x.shape.dim(0)
    if n > 1:
      y = M.variable(n/2, Domain.unbounded())
      M.constraint(Variable.hstack(Variable.reshape(x, NDSet(n/2,2)), y), Domain.inRotatedQCone())
      return rec(y)
    else:
      return x

  n = x.shape.dim(0)
  l = int(ceil(log(n, 2)))
  m = int(2**l) - n

  # if size of x is not a power of 2 we pad it:
  if m > 0:
    x_padding = M.variable(m,Domain.unbounded())
    M.constraint(Expr.sub(x_padding, Variable.repeat(t,m)), Domain.equalsTo(0.0))
    # set the last m elements equal to t
    x = Variable.stack(x,x_padding)

  M.constraint(Expr.sub(Expr.mul(2.0**(l/2.0), t),rec(x)), Domain.equalsTo(0.0))
```

Performance

Modeling and solving

MOSEK solves purely continuous problems very efficiently. This means that:

- Setting up a model in a modeling language is often slower than solving it,
- ...it may even have worse run-time complexity!
- Setting up a model in e.g. the Python API or a similar API is sometimes slower than solving it.

When creating mixed-integer models this is rarely an issue. Fusion is designed to make model development simpler while

- minimizing the overhead of loops and function calls by encouraging vectorized operations, and
- minimizing the run-time complexity when handling sparse structures.

A sparse conic problem

Conic formulation of the CHAINSING model:

$$\begin{array}{ll} \text{minimize} & e^T s + e^T t + e^T p + e^T q \\ \text{such that} & (1/2, s_j, x_i + 10x_{i+1}) \in \mathcal{Q}_r^3 \\ & (1/2, t_j, 5^{1/2}(x_{i+2} - x_{i+3})) \in \mathcal{Q}_r^3 \\ & (1/2, r_j, x_{i+1} - 2x_{i+2}) \in \mathcal{Q}_r^3 \\ & (1/2, u_j, 10^{1/4}(x_i - 10x_{i+3})) \in \mathcal{Q}_r^3 \\ & (1/2, p_j, r_j) \in \mathcal{Q}_r^3 \\ < & (1/2, q_j, u_j) \in \mathcal{Q}_r^3, \quad j = 0, \ldots, (n-2)/2,\ i = 2j \\ & 0.1 \leq x_i \leq 1.1, \quad i = 0, 2, \ldots, n-2. \end{array}$$

A sparse conic problem we can scale easily.

"Sparse second order cone programming formulations for convex optimization problems." K. Kobayashi, S.-Y. Kim, M. Kojima, Journal of the Operations Research Society of Japan, Vol. 51, No. 3 (2008), pp. 241-264.

Performance test: chainsing.java

```
public static void chainsing4(Model M, int n)
{
  int m = (n-2) / 2;
  Variable x = M.variable(n, Domain.unbounded());
  Variable p = M.variable(m, Domain.unbounded());
  Variable q = M.variable(m, Domain.unbounded());
  Variable r = M.variable(m, Domain.unbounded());
  Variable s = M.variable(m, Domain.unbounded());
  Variable t = M.variable(m, Domain.unbounded());
  Variable u = M.variable(m, Domain.unbounded());

  Variable x_i      = Variable.reshape(x,n/2,2).slice(new int[]{0,0},new int[]{n/2-1,1});
  Variable x_iplus1 = Variable.reshape(x,n/2,2).slice(new int[]{0,0},new int[]{n/2-1,1});
  Variable x_iplus2 = Variable.reshape(x,n/2,2).slice(new int[]{1,0},new int[]{n/2,1});
  Variable x_iplus3 = Variable.reshape(x,n/2,2).slice(new int[]{1,0},new int[]{n/2,1});
  Expression c = Expr.constTerm(m,0.5);
  // s[j] >= (x[i] + 10*x[i+1])^2
  M.constraint(Expr.hstack(c, s, Expr.add(x_i, Expr.mul(10.0,x_iplus1))), Domain.inRotatedQCone());
  // t[j] >= 5*(x[i+2] - x[i+3])^2
  M.constraint(Expr.hstack(c, t, Expr.mul(Math.sqrt(5), Expr.sub(x_iplus2,x_iplus3))),
               Domain.inRotatedQCone());
  // r[j] >= (x[i+1] - 2*x[i+2])^2
  M.constraint(Expr.hstack(c, r, Expr.sub(x_iplus1, Expr.mul(2.0,x_iplus2))),
               Domain.inRotatedQCone());
  // u[j] >= sqrt(10)*(x[i] - 10*x[i+3])^2
  M.constraint(Expr.hstack(Expr.constTerm(m,0.5/Math.sqrt(10)),
                           u,
                           Expr.sub(x_i, Expr.mul(10,x_iplus3))), Domain.inRotatedQCone());
  // p[j] >= r[j]^2
  M.constraint(Expr.hstack(c,p,r), Domain.inRotatedQCone());
  // q[j] >= u[j]^2
  M.constraint(Expr.hstack(c,q,u), Domain.inRotatedQCone());
  // 0.1 <= x[j] <= 1.1
  M.constraint(x,Domain.inRange(0.1, 1.1));
  M.objective(ObjectiveSense.Minimize, Expr.sum(Variable.vstack(new Variable[]{s, t, p, q})));
}
```

Performance: Fusion vs. solver API

	Solver API				Fusion Java			
n	C		Java		scalar		vectorized	
2000	0.01	0.25	0.08	0.25	0.52	0.20	0.17	0.28
4000	0.01	0.58	0.15	0.49	0.86	0.39	0.19	0.54
8000	0.03	1.16	0.27	1.01	1.49	0.81	0.33	1.10
16000	0.06	2.43	0.50	2.09	3.07	1.72	0.50	2.25
32000	0.12	5.14	0.97	4.47	8.74	3.60	0.89	4.85
64000	0.25	10.81	1.91	9.40	33.38	7.92	1.65	10.83
128000	0.50	23.82	3.64	21.30	115.46	18.73	3.15	25.66

Model setup time and solver time in seconds for each implementation of CHAINSING. The numbers do not include JVM startup time.

Conclusions

Conclusions

- Fusion handles non-linearities in conic form, but many complex sets can be constructed from these.
- Fusion makes it significantly faster to build complex models.
- Fusion overhead is small giving a good compromise between efficiency and ease of use.

Fusion is included in MOSEK 7.0 and requires no extra license.

These slides and source code for examples are available at
`http://mosek.com/resources/presentations/`