Semidefinite Optimization using MOSEK

Joachim Dahl

ISMP Berlin, August 23, 2012

Introduction

- MOSEK is a state-of-the-art solver for large-scale linear and conic quadratic problems.
- Based on the homogeneous model using Nesterov-Todd scaling.
- Next version includes **semidefinite optimization**.
- Goal for first version is to beat SeDuMi.

Why SDP in MOSEK?

- It is very powerful and flexible for modeling.
- We want the SDP work you developed to be available to our customers!
- Customers have been asking about it for awhile.

Semidefinite optimization

Semidefinite optimization

- Primal and dual problems:

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{p} C_j \bullet X_j \\ \text{subject to} & \sum_{j=1}^{p} A_{ij} \bullet X_j = b_i \\ & X_j \in \mathcal{S}_+^{n_j}, \end{array} \qquad \begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & \sum_{i=1}^{m} y_i A_{ij} + S_j = C_j \\ & S_j \in \mathcal{S}_+^{n_j}, \end{array}$$

  where $A_{ij}, C_j \in \mathcal{S}^{n_j}$.

- Optimality conditions:

$$\sum_{j=1}^{p} A_{ij} \bullet X_j = b_i, \quad \sum_{i=1}^{m} y_i A_{ij} + S_j = C_j, \quad X_j S_j = 0, \quad X_j, S_j \in \mathcal{S}_+^{n_j}.$$

- Equivalent complementary conditions if $X_j, S_j \in \mathcal{S}_+^{n_j}$:

$$X_j \bullet S_j = 0 \quad \Longleftrightarrow \quad X_j S_j = 0 \quad \Longleftrightarrow \quad X_j S_j + S_j X_j = 0.$$

Algorithms

Notation

For symmetric matrices $U \in \mathcal{S}^n$ we define

$$\mathbf{svec}(U) = (U_{11}, \sqrt{2}U_{21}, \ldots, \sqrt{2}U_{n1}, U_{22}, \sqrt{2}U_{32}, \ldots, \sqrt{2}U_{n2}, \ldots, U_{nn})^T$$

with inverse operation

$$\mathbf{smat}(u) = \begin{bmatrix} u_1 & u_2/\sqrt{2} & \cdots & u_n/\sqrt{2} \\ u_2/\sqrt{2} & u_{n+1} & \cdots & u_{2n-1}/\sqrt{2} \\ \vdots & \vdots & & \vdots \\ u_n/\sqrt{2} & u_{n-1}/\sqrt{2} & \cdots & u_{n(n+1)/2} \end{bmatrix},$$

and a symmetric product

$$u \circ v := (1/2)(\mathbf{smat}(u)\mathbf{smat}(v) + \mathbf{smat}(v)\mathbf{smat}(u)).$$

Notation

Let

$$a_{ij} = \mathbf{svec}(A_{ij}), \quad c_j := \mathbf{svec}(C_j), \quad x_j := \mathbf{svec}(X_j), \quad s_j := \mathbf{svec}(S_j),$$

and

$$A := \begin{bmatrix} a_{11}^T & \dots & a_{1p}^T \\ \vdots & & \vdots \\ a_{m1}^T & \dots & a_{mp}^T \end{bmatrix}, \quad c := \begin{bmatrix} c_1 \\ \vdots \\ c_p \end{bmatrix}, \quad x := \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}, \quad s := \begin{bmatrix} s_1 \\ \vdots \\ s_p \end{bmatrix}.$$

We then have primal and dual problems:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \succeq 0, \end{array} \qquad \begin{array}{ll} \text{maximize} & b^T y \\ \text{subject to} & A^T y + s = c \\ & s \succeq 0. \end{array}$$

The homogeneous model

Simplified homogeneous model:

$$\begin{bmatrix} s \\ 0 \\ \kappa \end{bmatrix} + \begin{bmatrix} 0 & A^T & -c \\ A & 0 & -b \\ c^T & -b^T & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \tau \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad x, s \succeq 0,\ \tau, \kappa \geq 0.$$

Note that $(x, y, s, \kappa, \tau) = 0$ is feasible, and $x^T s + \kappa\tau = 0$.

- If $\tau > 0,\ \kappa = 0$ then $(x, y, s)/\tau$ is a primal-dual optimal solution,

$$Ax = b\tau, \quad A^T y + s = c\tau, \quad x^T s = 0.$$

- If $\kappa > 0,\ \tau = 0$ then either primal or dual is infeasible,

$$Ax = 0, \quad A^T y + s = 0, \quad c^T x - b^T y < 0,$$

  Primal infeasible if $b^T y > 0$, dual infeasible if $c^T x < 0$.

The central path

We define the central path as:

$$\begin{bmatrix} s \\ 0 \\ \kappa \end{bmatrix} + \begin{bmatrix} 0 & A^T & -c \\ A & 0 & -b \\ c^T & -b^T & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ \tau \end{bmatrix} = \gamma \begin{bmatrix} A^T y^{(0)} + s^{(0)} - c\tau^{(0)} \\ Ax^{(0)} - b\tau^{(0)} \\ c^T x^{(0)} - b^T y^{(0)} + \kappa^{(0)} \end{bmatrix}$$

$$x \circ s = \gamma \mu^{(0)} e, \quad \tau\kappa = \gamma\mu^{(0)},$$

where $\gamma \in [0,1]$, $\mu^{(0)} = \frac{(x^{(0)})^T s^{(0)} + \kappa^{(0)}\tau^{(0)}}{n+1} > 0$ and

$$e = \begin{bmatrix} \mathbf{svec}(I_{n_1}) \\ \vdots \\ \mathbf{svec}(I_{n_p}) \end{bmatrix}.$$

Primal-dual scaling

A primal-dual scaling is defined by a non-singular R_k as

$$W_k(x_k) := \mathbf{svec}(R_k^{-T} X_k R_k^{-1}), \quad W_k^{-T}(s_k) := \mathbf{svec}(R_k S_k R_k^T),$$

- preserves the semidefinite cone,

$$x_k \succ 0,\ s_k \succ 0 \quad \Longleftrightarrow \quad W_k(x_k) \succ 0,\ W_k^{-T}(s_k) \succ 0,$$

- the central path,

$$x_k \circ s_k = \gamma\mu e_k \quad \Longleftrightarrow \quad W_k(x_k) \circ W_k^{-T}(s_k) = \gamma\mu e_k,$$

- and the complementarity: $x_k^T s_k = W_k(x_k)^T W_k^{-T}(s_k)$.

Two popular primal-dual scalings

- Helmberg-Kojima-Monteiro (HKM) $R_k^{-1} = S_k^{1/2}$:

$$W_k(x_k) = \mathbf{svec}(S_k^{1/2} X_k S_k^{1/2}), \quad W_k^{-T}(s_k) = e_k.$$

- Nesterov-Todd (NT):

$$R_k^{-1} = \left(S_k^{-1/2} (S_k^{1/2} X_k S_k^{1/2})^{1/2} S_k^{-1/2} \right)^{1/2},$$

Satisfies $R_k^{-1} X_k R_k^{-1} = R_k S_k R_k$. Computed as:

$$R_k^{-1} = \Lambda^{1/4} Q^T L^{-1}, \quad LL^T = X_k, \quad Q\Lambda Q^T = L^T S_k L$$

which satisfies $R_k^{-T} X_k R_k^{-1} = R_k S_k R_k^T = \Lambda_k$.

The search direction

Linearizing the scaled centrality condition

$$W_k(x_k) \circ W_k^{-T}(s_k) = \gamma\mu e_k$$

and discarding higher-order terms leads to

$$\Delta x_k + \Pi_k \Delta s_k = \gamma\mu s_k^{-1} - x_k,$$

where

- HKM scaling: $\Pi_k z_k = \mathbf{svec}(X_k Z_k S_k^{-1} + S_k^{-1} Z_k X_k)/2$,
- NT scaling: $\Pi_k z_k = \mathbf{svec}(R_k^T R_k Z_k R_k^T R_k)$.

The search direction

Most expensive part of interior-point optimizer is solving:

$$\begin{aligned}
A\Delta x - b\Delta\tau &= (\gamma-1)(Ax - b\tau) &&= r_1\\
A^T\Delta y + \Delta s - c\Delta\tau &= (\gamma-1)(A^T y + s - c\tau) &&= r_2\\
c^T\Delta x - b^T\Delta y + \Delta\kappa &= (\gamma-1)(c^T x - b^T y + \kappa) &&= r_3\\
\Delta x + \Pi\Delta s &= \gamma\mu s^{-1} - x &&= r_4\\
\Delta\kappa + \kappa/\tau\Delta\tau &= \gamma\mu/\tau - \kappa &&= r_5
\end{aligned}$$

Gives constant decrease in residuals and complementarity:

$$\begin{aligned}
A(x+\alpha\Delta x) - b(\tau+\alpha\Delta\tau) &= (1-\alpha(1-\gamma))(Ax - b\tau)\\
(x+\alpha\Delta x)^T(s+\alpha\Delta s) + (\kappa+\alpha\Delta\kappa)(\tau+\alpha\Delta\tau) &= \underbrace{(1-\alpha(1-\gamma))}_{<1}(x^T s + \kappa\tau).
\end{aligned}$$

Polynomial complexity for γ chosen properly.

Solving for the search direction

After block-elimination, we get a 2×2 system:

$$A\Pi A^T \Delta y - (A\Pi c + b)\Delta\tau = r_y$$
$$(A\Pi c - b)\Delta y + (c^T \Pi c + \kappa/\tau)\Delta\tau = r_\tau.$$

We factor $A\Pi A^T = LL^T$ and eliminate

$$\Delta y = L^{-T}L^{-1}\left((A\Pi c + b)\Delta\tau + r_y\right).$$

Then

$$\left((A\Pi c - b)^T L^{-T}L^{-1}(A\Pi c + b) + (c^T \Pi c + \kappa/\tau)\right)\Delta\tau = r_\tau - (A\Pi c - b)^T L^{-T}L^{-1} r_y.$$

An extra (insignificant) solve compared to an infeasible method.

Forming the Schur complement

Schur complement computed as a sum of outer products:

$$\begin{aligned} A\Pi A^T &= \begin{bmatrix} a_{11}^T & \dots & a_{1p}^T \\ \vdots & & \vdots \\ a_{m1}^T & \dots & a_{mp}^T \end{bmatrix} \begin{bmatrix} \Pi_1 & & \\ & \ddots & \\ & & \Pi_p \end{bmatrix} \begin{bmatrix} a_{11} & \dots & a_{m1} \\ \vdots & & \vdots \\ a_{1p} & \dots & a_{mp} \end{bmatrix} \\ &= \sum_{k=1}^{p} P_k^T \left(P_k A_{:,k} \Pi_k A_{:,k}^T P_k^T \right) P_k, \end{aligned}$$

where P_k is a permutation of $A_{:,k}$. Complexity depends on choice of P_k.

We compute only $\mathbf{tril}(A\Pi A^T)$, so by

$$\operatorname{nnz}(P_k A_{:,k})_1 \geq \operatorname{nnz}(P_k A_{:,k})_2 \geq \cdots \geq \operatorname{nnz}(P_k A_{:,k})_p,$$

get a good reduction in complexity. More general than SDPA.

Forming the Schur complement

To evaluate each term

$$a_{ik}^T \Pi_k a_{jk} = (R_k^T A_{ik} R_k) \bullet (R_k^T A_{jk} R_k) = A_{ik} \bullet (R_k R_k^T A_{jk} R_k R_k^T).$$

we follow SeDuMi. Rewrite

$$R_k R_k^T A_{jk} R_k R_k^T = \hat{M} M + M^T \hat{M}^T,$$

i.e., a symmetric (low-rank) product.

- A_{ik} sparse: evaluate $\hat{M} M + M^T \hat{M}^T$ element by element.
- A_{ik} dense: evaluate $\hat{M} M + M^T \hat{M}^T$ using level 3 BLAS.

May exploit low-rank structure later.

Practical details

Additionally, the MOSEK solver

- uses Mehrotra's predictor-corrector step.
- solves mixed linear, conic quadratic and semidefinite problems.
- employs a presolve step for linear variables, which can reduce problem size and complexity significantly.
- scales constraints trying to improving conditioning of a problem.

Results and examples

SDPLIB benchmark

- We use the subset of SDPLIB used by H. D. Mittelmann for his SDP solver comparison.
- For brevity we report only iteration count, computation time, and relative gap at termination.
- Our SDPA-format converter detects block-diagonal structure within the semidefinite blocks (SeDuMi and SDPA does not), so for a few problems our solution time is much smaller.
- All solvers run on a single thread on an Intel Xeon E32270 with 4 cores at 3.4GHz and 32GB memory using Linux.

SDPLIB benchmark

	MOSEK 7.0			SeDuMi 1.3			SDPA 7.3.1		
Problem	it	time	relgap	it	time	relgap	it	time	relgap
arch8	25	1	5.4e-07	28	2	-2.2e-12	25	1	7.8e-09
control7	49	5	6.9e-07	38	7	-7.2e-09	44	7	6.2e-07
control10	47	21	9.3e-07	43	37	-7.4e-08	46	45	6.9e-06
control11	50	38	8.4e-07	45	66	-1.6e-08	47	74	4.5e-06
equalG11	22	45	-1.0e-07	15	75	-6.3e-07	18	19	4.5e-07
equalG51	31	125	8.7e-08	16	145	-1.7e-05	19	35	1.3e-08
gpp250-4	28	3	-2.0e-07	33	8	-3.7e-06	18	1	4.9e-09
gpp500-4	30	18	-5.7e-08	21	30	-1.4e-06	21	5	1.7e-09
hinf15	18	0	2.8e-03	16	1	-1.1e-02	13	0	-1.8e-02
maxG11	11	16	-2.2e-09	13	45	-5.4e-12	16	23	1.7e-08
maxG32	12	262	3.3e-10	14	789	-1.0e-12	17	200	1.0e-08
maxG51	20	57	8.3e-08	16	98	-9.6e-12	16	23	2.1e-08
mcp250-1	17	1	2.3e-08	15	2	-6.3e-12	15	0	3.0e-08
mcp500-1	18	5	3.7e-08	16	15	-2.8e-12	16	3	7.9e-09
qap9	23	1	1.1e-07	23	2	-1.3e-07	16	1	8.0e-05
qap10	17	1	8.0e-07	27	5	-1.8e-06	17	1	3.0e-04
qpG11	11	16	1.1e-09	14	385	-1.4e-10	16	88	1.7e-08
qpG51	17	41	2.8e-11	22	1139	-3.8e-13	19	196	1.9e-08
ss30	33	4	3.4e-06	28	15	-8.4e-12	22	5	5.5e-08
theta3	13	1	8.9e-10	15	5	-1.2e-10	17	2	7.8e-09
theta4	14	5	4.8e-09	16	24	1.6e-10	18	10	1.1e-08
theta5	13	11	3.2e-09	16	77	-6.5e-10	18	25	1.2e-08
theta6	14	27	6.2e-09	16	232	1.7e-10	18	59	1.2e-08
thetaG11	11	25	2.0e-08	15	94	-2.7e-13	22	37	1.1e-08
thetaG51	16	137	3.7e-08	20	1401	-3.4e-12	28	363	4.7e-07
truss7	24	0	8.8e-10	26	13	-4.5e-11	26	0	-2.5e-08
truss8	21	1	2.0e-10	23	2	-4.8e-12	20	1	6.7e-09

$$\text{relgap} = (c^T x - b^T y)/(1 + |c^T x| + |b^T y|)$$

Profiling

	Profiling results [%]				
Problem	Schur	factor	stepsize	search dir	update
arch8	25.7	1.2	9.8	8.0	46.2
control7	**80.8**	**12.3**	**0.7**	**1.4**	**3.0**
control10	79.1	17.3	0.3	0.9	1.5
control11	78.8	18.3	0.2	0.8	1.0
equalG11	22.0	0.9	15.1	13.8	35.0
equalG51	22.0	0.9	13.7	12.2	39.5
gpp250-4	19.3	0.9	10.6	8.8	51.2
gpp500-4	20.1	0.9	13.3	12.5	41.1
hinf15	48.0	7.7	4.0	5.4	19.3
maxG11	**2.7**	**1.3**	**18.9**	**15.4**	**45.0**
maxG32	1.2	1.1	21.6	15.5	44.0
maxG51	2.4	1.1	19.1	17.4	41.9
mcp250-1	5.4	1.5	16.1	13.4	46.5
mcp500-1	4.0	1.2	17.7	15.7	44.4
qap9	52.0	34.6	1.6	2.4	6.3
qap10	50.1	39.1	1.2	2.0	5.0
qpG11	2.0	1.2	19.6	17.0	42.9
qpG51	2.6	1.2	20.4	14.9	44.4
ss30	12.3	0.2	13.0	11.2	52.3
theta3	42.7	39.7	2.4	2.9	8.7
theta4	33.4	56.0	1.4	2.1	4.7
theta5	26.5	67.2	0.8	1.4	2.5
theta6	**21.4**	**74.5**	**0.6**	**1.0**	**1.5**
thetaG11	15.3	21.5	13.1	9.9	28.0
thetaG51	17.1	65.6	3.2	3.0	7.8
truss7	29.8	2.0	9.4	8.4	34.6
truss8	74.2	8.2	1.9	2.1	11.0

Profiling conclusions

- Forming $A\Pi A^T$ (Schur) is not always most expensive step (as perhaps believed).
- A good parallelization speedup is possible for computing $A\Pi A^T$. (Factor is already parallelized).
- The update step (updating variables, neighborhood check, new NT scaling) is very cheap for LP and SOCP, but expensive for SDP, mainly due to NT scaling; HKM scaling would give an improvement.
- Stepsize computations (e.g., stepsize to boundary) are moderately expensive, but hard to speed up.

Conic modeling

For $A \in \mathcal{S}^n$, the nearest correlation matrix is

$$X^\star = \arg \min_{X \in \mathcal{S}^n_+, \mathbf{diag}(X) = e} \|A - X\|_F.$$

Conic formulation exploiting symmetry of $A - X$:

$$\begin{array}{lcl} \text{minimize} & t & \\ \text{subject to} & \|z\|_2 & \leq t \\ & \mathbf{svec}(A - X) & = z \\ & \mathbf{diag}(X) & = e \\ & X \succeq 0. & \end{array}$$

Simple formulation on paper, but complicated to formulate in "standard form" as in SeDuMi, SDPA, ...

Conic modeling

- For conic modeling you need a modeling tool!
- Nice MATLAB toolboxes exist (Yalmip, CVX, ...), but not for other languages.
- We developed a modeling API called `MOSEK Fusion`:
  - A tool for self-dual conic modeling; no QP or general convex optimization.
  - Idea: create and manipulate linear expressions, and assign them to different cones. Simple but flexible design; scales well for large problems.
  - Available for Python, Java, .NET. Planned versions include MATLAB (soon), C++ (later).
  - Syntax almost identical across platforms.

Python/Fusion listing for nearest correlation

```
def svec(e):
    N = e.get_shape().dim(0)
    S = Matrix.sparse(N * (N+1) / 2,
                      N * N,
                      range(N * (N+1) / 2),
                      [ (i+j*N) for j in xrange(N) for i in xrange(j,N) ],
                      [ (1.0 if i == j else 2**(0.5)) for j in xrange(N) for i in xrange(j,N) ])

    return Expr.mul(S,Expr.reshape( e, N * N ))

def nearestcorr(A):
    M = Model("NearestCorrelation")
    N = len(A)

    # Setting up the variables
    X = M.variable("X",Domain.inPSDCone(N))

    # t > || z ||_2
    tz = M.variable("tz", Domain.inQCone(N*(N+1)/2+1))
    t  = tz.index(0)
    z  = tz.slice(1,N*(N+1)/2+1)

    # svec (A-X) = z
    M.constraint( Expr.sub(svec(Expr.sub(DenseMatrix(A),X)), z), Domain.equalsTo(0.0) )

    # diag(X) = e
    for i in range(N):
        M.constraint( X.index(i,i), Domain.equalsTo(1.0) )

    # Objective: Minimize t
    M.objective(ObjectiveSense.Minimize, t)
    M.solve()
```

Summary

Summary

- SDP solver in MOSEK 7.0 based on self-dual homogeneous embedding and NT scaling.
- Faster than SeDuMi, comparable with SDPA.
- Includes `Fusion`, a powerful tool for self-dual conic modeling.
- Future directions:
  - Incorporate multithreading.
  - Exploit low-rank structure in data.
  - Possibly implement the HKM direction.

Want to try it?
Contact `support@mosek.com` for a beta version.

References

[1] E.D. Andersen, C. Roos, and T. Terlaky. On implementing a primal-dual interior-point method for conic quadratic optimization. *Mathematical Programming*, 95(2):249–277, 2003.

[2] Y.E. Nesterov and M.J. Todd. Primal-dual interior-point methods for self-scaled cones. *SIAM Journal on Optimization*, 8(2):324–364, 1998.

[3] J.F. Sturm. Implementation of interior point methods for mixed semidefinite and second order cone optimization problems. *Optimization Methods and Software*, 17(6):1105–1154, 2002.

[4] M. Yamashita, K. Fujisawa, and M. Kojima. Implementation and evaluation of sdpa 6.0 (semidefinite programming algorithm 6.0). *Optimization Methods and Software*, 18(4):491–505, 2003.

[5] Y. Ye, M.J. Todd, and S. Mizuno. An $O(\sqrt{nl})$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, pages 53–67, 1994.