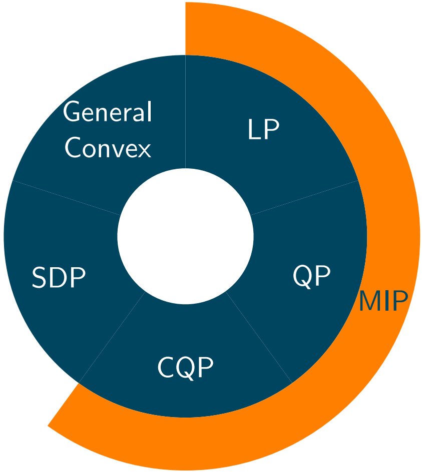
On the Linear Algebra Employed in the MOSEK Conic Optimizer

Monday Jul 13

Erling D. Andersen

www.mosek.com

MOSEK summary



- Version 8: Work in progress.

The conic optimization problem

$$
\begin{array}{llcll}
\text{min} & \sum_{j=1}^{n} c_j x_j + \sum_{j=1}^{\bar{n}} \langle \bar{C}_j, \bar{X}_j \rangle & & & \\
\text{subject to} & \sum_{j=1}^{n} a_{ij} x_j + \sum_{j=1}^{\bar{n}} \langle \bar{A}_{ij}, \bar{X}_j \rangle & = & b_i, & i = 1, \ldots, m, \\
& x \in \mathcal{K}, & & & \\
& \bar{X}_j \succeq 0, & & & j = 1, \ldots, \bar{n}.
\end{array}
$$

Explanation:

- x_j is a scalar variable.
- $\bar{X}_j$ is a square matrix variable.

- $\mathcal{K}$ represents Cartesian product of conic quadratic constraints e.g.

$$x_1 \geq \|x_{2:n}\| .$$

- $\bar{X}_j \succeq 0$ represents $\bar{X}_j = \bar{X}_j^T$ and $\bar{X}_j$ is PSD.
- $\bar{C}_j$ and $\bar{A}_j$ are required to be symmetric.
- $\langle A, B \rangle := \mathrm{tr}(A^T B)$.
- Dimensions are large.
- Data matrices are typically sparse.
    - A has ≤ 10 nonzeros per column on average usually.
    - $\bar{A}_{ij}$ contains few nonzeros and/or is low rank.

The algorithm: Simplified version

- Step 1: Setup the homogeneous and self-dual model.
- Step 2. Choose a starting point.
- Step 3: Compute Nesterov-Todd search direction.
- Step 4: Take a step.
- Step 5: Stop if the trial solution is good enough.
- Step 6: Goto 3.

Search direction computation

Requires solution of:

$$\begin{bmatrix} -(WW^T)^{-1} & 0 & A^T \\ 0 & -(\bar{W}\bar{W}^T)^{-1} & \bar{A}^T \\ A & \bar{A} & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_{\bar{x}} \\ d_y \end{bmatrix} = \begin{bmatrix} r_x \\ r_{\bar{x}} \\ r_y \end{bmatrix}$$

where

- W and $\bar{W}$ are nonsingular block diagonal matrices.
- WW^T is a diagonal matrix $+$ low rank terms.

Reduced Schur system approach

We have

$$((AW)(AW)^T + (\bar{A}\bar{W})(\bar{A}\bar{W})^T)d_y = \cdots$$

and

$$\begin{aligned} d_x &= -(WW^T)(r_x - A^T d_y), \\ d_{\bar{x}} &= -(\bar{W}\bar{W}^T)(r_{\bar{x}} - \bar{A}^T d_y). \end{aligned}$$

Cons:

- Dense columns cause issues.
- Numerical stability. Bad condition number.

Pros:

- A positive definite symmetric system.
- Use Cholesky with no pivoting.
- Employed in major commercial solvers.

Computing the Schur matrix

Assumptions:

- Let us focus at:

$$(\bar{A}\bar{W})(\bar{A}\bar{W})^T = \bar{A}\bar{W}\bar{W}^T\bar{A}^T.$$

- Only one 1 matrix variable. The general case follows easily.
- NT search direction implies

$$\bar{W} = R \otimes R \text{ and } \bar{W}^T = R^T \otimes R^T$$

where the Kronecker product $\otimes$ is defined as

$$R \otimes R = \begin{bmatrix} R_{11}R & R_{12}R & \cdots \\ R_{21}R & R_{22}R & \\ \vdots & & \end{bmatrix}.$$

Fact:

$$e_k^T \bar{A}\bar{W}\bar{W}^T\bar{A}^T e_l = vec(\bar{A}_k)^T vec(RR^T\bar{A}_l RR^T).$$

Exploiting sparsity 1

Compute the lower triangular part of

$$\bar{A}\bar{W}\bar{W}^T\bar{A}^T = \begin{bmatrix} vec(\bar{A}_1)^T \\ \vdots \\ vec(\bar{A}_m)^T \end{bmatrix} \begin{bmatrix} vec(RR^T\bar{A}_1RR^T) \\ \vdots \\ vec(RR^T\bar{A}_mRR^T) \end{bmatrix}^T$$

so the lth column is computed as

$$e_k^T\bar{A}\bar{W}\bar{W}^T\bar{A}^Te_l = vec(\bar{A}_k)^T vec(RR^T\bar{A}_lRR^T), \quad for \quad k \geq l.$$

Avoid computing

$$\begin{aligned} & e_k^T\bar{A}\bar{W}\bar{W}^T\bar{A}^Te_l \\ = \ & vec(\bar{A}_k)^T vec(RR^T\bar{A}_lRR^T) \\ = \ & 0 \end{aligned}$$

if $A_k = 0$ or $A_l = 0$.

Exploiting sparsity 2

Moreover,

- R is a dense square matrix.
- A_i is typically extremely sparse e.g.

$$A_i = e_k e_k^T.$$

  as observed by J. Sturm for instance.

- Wlog assume

$$A_i = U_i V_i^T + (U_i V_i^T)^T.$$

  because $U_i = A_i/2$ and $V_i = I$ is a valid choice.

- In practice U_i and V_i are sparse and **low** rank e.g. has few columns.
- The new idea!

Recall

$$e_k^T \bar{A} \bar{W} \bar{W}^T \bar{A}^T e_l = vec(\bar{A}_k)^T vec(RR^T \bar{A}_l RR^T)$$

must be computed for all $k \geq l$ and

$$\begin{aligned} RR^T \bar{A}_l RR^T &= RR^T (U_l (V_l)^T + (U_l (V_l)^T)^T) RR^T \\ &= \hat{U}_l \hat{V}_l^T + (\hat{U}_l \hat{V}_l^T)^T \end{aligned}$$

where

$$\begin{aligned} \hat{U}_l &:= RR^T U_l, \\ \hat{V}_l &:= RR^T V_l. \end{aligned}$$

- $\hat{U}_I$ and $\hat{V}_I$ are dense matrices.
- Sparsity in U_I and V_I are exploited.
- Low rank structure is exploited.
- Is all of $\hat{U}_I$ and $\hat{V}_I$ required?

Observe

$$e_i^T(U_k V_k^T + (U_k V_k^T)^T) = 0, \quad \forall i \notin \mathcal{I}^k$$

where

$$\mathcal{I}^k := \{i \mid U_{ki:} \neq 0 \vee V_{ki:} \neq 0\}.$$

Now

$$\begin{aligned} & vec(\bar{A}_k)^T vec(RR^T \bar{A}_l RR^T) \\ = \; & vec(U_k V_k^T + (U_k V_k^T)^T) vec(\hat{U}_l \hat{V}_l^T + (\hat{U}_l \hat{V}_l^T)^T) \\ = \; & \sum_i 2(U_k e_i)^T (\hat{U}_l \hat{V}_l^T + (\hat{U}_l \hat{V}_l^T)^T)(V_k e_i) \end{aligned}$$

Therefore, only rows $\hat{U}_l$ and $\hat{V}_l$ corresponding to

$$\bigcup_{k \geq l} \mathcal{I}^k$$

are needed.

Proposed algorithm:

- Compute

$$\bigcup_{k \geq l} \mathcal{I}^k$$

- Compute $\hat{U}_{kI^K:}$ and $\hat{V}_{kI^K:}$.
- Compute

$$\sum_i 2(U_k e_i)^T (\hat{U}_l \hat{V}_l^T + (\hat{U}_l \hat{V}_l^T)^T)(V_k e_i)$$

Possible improvements

- Exploit the special case $U_{k:j} = \alpha V_{k:j}$.
- Exploit dense computations e.g. level 3 BLAS when possible and worthwhile.

Summary:

- Exploit sparsity as done in SeDuMi by Sturm.
- Also able to exploit low rank structure.
- Not implemented yet!

Linear algebra summary

- Sparse matrix operations e.g. multiplications.
- Large sparse matrix factorization e.g. Cholesky.
  - Including ordering (AMD,GP).
  - Dense column detection and handling.
- Dense sequential level 1,2,3 BLAS operations.
  - Inside sparse Cholesky for instance.
  - Sequential INTEL Math Kernel Library is employed extensively.
- Eigenvalue computations.
- What about the parallelization?
  - Modern computers have many cores.
  - Typically from 4 to 12.
  - Recent customer example had 80.

The parallelization challenge on shared memory

- A computer has many cores.
- Parallelization using native threads is cumbersome and error prone.
- Employ a parallelization framework e.g. Cilk or OpenMP.

Other issues;

- Exploit caches.
- Do not overload the memory bus.
- Not fine grained due to threading overhead.

Cilk summary:

- Extension to C and C++.
- Has a runtime environment that execute tasks in parallel on a number of workers.
- Handles the load balancing.
- Allows nested/recursive parallelism e.g.
  - Parallel dense matrix mul. within parallelized sparse Cholesky.
  - Parallel IPM within B&B.
- Is run to run deterministic.
  - Care must be taken in floating point computatiosn.
- Supported by the Intel C compiler, Gcc, Clang.

Example parallelized dense `syrk`

The dense level 3 BLAS `syrk` operation does

$$C = AA^T.$$

Parallelized version using Cilk:
If C is small

$$C = AA^T$$

else

cilk_spawn	$C_{21} = A_{2:}A_{1:}^T$	gemm
cilk_spawn	$C_{11} = A_{1:}A_{1:}^T$	syrk
cilk_spawn	$C_{22} = A_{2:}A_{2:}^T$	syrk
cilk_sync		

Usage of recursion is allowed!

Our experience with cilk

- cilk is easy to learn i.e. 3 new keywords.
- Nested/recursive parallelism is allowed.
- Useful for both sparse and dense matrix computations.
- Efficient parallelization is nevertheless hard.

Summary and conclusions

- I am behind the schedule with MOSEK version 8.
- Proposed a new algorithm for computing the Schur matrix in the semidefinite case.
- Discussed the usage of task based parallelization framework exemplified by cilk.
- Slides url `https://mosek.com/resources/presentations`.