Tutorial: Algorithms for Large-Scale Semidefinite Programming

Alexandre d’Aspremont, CNRS & Ecole Polytechnique.

Support from NSF, ERC (project SIPA) and Google.
A semidefinite program (SDP) is written

\[
\begin{align*}
\text{minimize} & \quad \text{Tr}(CX) \\
\text{subject to} & \quad \text{Tr}(A_iX) = b_i, \quad i = 1, \ldots, m \\
& \quad X \succeq 0,
\end{align*}
\]

where \( X \succeq 0 \) means that the matrix variable \( X \in \mathcal{S}_n \) is positive semidefinite.

Its dual can be written

\[
\begin{align*}
\text{maximize} & \quad b^T y \\
\text{subject to} & \quad C - \sum_{i=1}^{m} y_i A_i \succeq 0,
\end{align*}
\]

which is another semidefinite program in the variables \( y \).
Introduction

Classical algorithms for semidefinite programming

- Following [Nesterov and Nemirovskii, 1994], most of the attention was focused on interior point methods.
- Basic idea: Newton’s method, with efficient linear algebra to compute the Newton step (or solve the KKT system).
- Fast, and robust on small problems ($n \sim 500$).
- Computing the Hessian is too hard on larger problems. Exploiting structure (sparsity, etc.) is hard too.

Solvers

- Open source solvers: SDPT3, SEDUMI, SDPA, CSDP, ...
- Very powerful modeling systems: CVX
Solving a MaxCut relaxation using CVX

\[
\begin{align*}
\text{max.} & \quad \text{Tr}(XC') \\
\text{s.t.} & \quad \text{diag}(X) = 1 \\
& \quad X \succeq 0,
\end{align*}
\]

is written as follows in CVX/MATLAB

```matlab
cvx_begin
  .  variable X(n,n) symmetric
  .  maximize trace(C*X)
  .  subject to
  .    diag(X)==1
  .    X==semidefinite(n)
cvx_end
```
Introduction

Algorithms for **large-scale** semidefinite programming.

*Structure ⇒ algorithmic choices*

Examples:

- SDPs with constant trace cast as max. eigenvalue minimization problems.
- Fast projection steps.
- Fast prox or affine minimization subproblems.
- Closed-form or efficiently solvable block minimization subproblems.
- Etc. . .
Example. In many semidefinite relaxations of combinatorial problems, we can impose $\text{Tr}(X) = 1$ and solve

$$
\begin{align*}
\text{maximize} & \quad \text{Tr}(CX) \\
\text{subject to} & \quad \text{Tr}(A_iX) = b_i, \quad i = 1, \ldots, m \\
& \quad \text{Tr}(X) = 1, \; X \succeq 0,
\end{align*}
$$

The dual can be written as a \textit{maximum eigenvalue minimization} problem

$$
\min_x \lambda_{\text{max}} \left( C + \sum_{i=1}^m x_i A_i \right) - b^T x
$$

in the variable $x \in \mathbb{R}^m$. 
Outline

- Introduction
- First-order methods
  - Subgradient methods
  - Smoothing & accelerated algorithms
  - Improving iteration complexity
- Exploiting structure
  - Frank-Wolfe
  - Block coordinate descent
  - Dykstra, alternating projection
  - Localization, cutting-plane methods
Subgradient methods

Solve

$$\min_{x \in Q} \lambda_{\max}(A(x)) + c^T x$$

where $A(x) = C + \sum_{i=1}^{m} x_i A_i$, using the projected subgradient method.

**Input:** A starting point $x_0 \in \mathbb{R}^m$.
1. **for** $t = 0$ to $N - 1$ **do**
2. Set
   $$x_{t+1} = P_Q(x_t - \gamma \partial \lambda_{\max}(A(x))).$$
3. **end for**

**Output:** A point $x = (1/N)\sum_{t=1}^{N} x_t$.

Here, $\gamma > 0$ and $P_Q(\cdot)$ is the Euclidean projection on $Q$. 

A. d'Aspremont HPOPT, Delft, June 2012. 8/51
Subgradient methods

- The **number of iterations** required to reach a target precision $\epsilon$ is

$$N = \frac{D_Q^2 M^2}{\epsilon^2}$$

where $D_Q$ is the diameter of $Q$ and $\|\partial \lambda_{\text{max}}(A(x))\| \leq M$ on $Q$.

- The **cost per iteration** is the sum of
  
  - The cost $p_Q$ of computing the Euclidean projection on $Q$.
  - The cost of computing $\partial \lambda_{\text{max}}(A(x))$ which is e.g. $v_1 v_1^T$ where $v_1$ is a leading eigenvector of $A(x)$.

Computing one leading eigenvector of a dense matrix $X$ with relative precision $\epsilon$, using a randomly started Lanczos method, with probability of failure $1 - \delta$, costs

$$O \left( \frac{n^2 \log(n/\delta^2)}{\sqrt{\epsilon}} \right)$$

flops [Kuczynski and Wozniakowski, 1992, Th.4.2].
Subgradient methods

Solving \( \min_{X \in Q} \lambda_{\text{max}}(A(x)) \) using projected subgradient.

- Easy to implement.
- Very poor performance in practice. The \( 1/\epsilon^2 \) dependence is somewhat punishing.

Example below on MAXCUT.
Smoothing & accelerated algorithms
[Nesterov, 2007] We can **regularize** the objective and solve

\[
\min_{x \in Q} f_\mu(x) \triangleq \mu \log \text{Tr} \left( \exp \left( \frac{A(x)}{\mu} \right) \right)
\]

for some regularization parameter \( \mu > 0 \) (\( \exp(\cdot) \) is the **matrix** exponential here).

- If we set \( \mu = \epsilon / \log n \) we get

\[
\lambda_{\text{max}}(A(x)) \leq f_\mu(x) \leq \lambda_{\text{max}}(A(x)) + \epsilon
\]

- The gradient \( \nabla f_\mu(x) \) is Lipschitz continuous with constant

\[
\frac{\|A\|^2 \log n}{\epsilon}
\]

where \( \|A\| = \sup_{\|h\| \leq 1} \|A(h)\|_2 \).
The number of iterations required to get an $\epsilon$ solution using the smooth minimization algorithm in Nesterov [1983] grows as

$$\frac{\|A\| \sqrt{\log n}}{\epsilon} \sqrt{\frac{d(x^*)}{\sigma}}$$

where $d(\cdot)$ is strongly convex with parameter $\sigma > 0$.

The cost per iteration is (usually) dominated by the cost of forming the matrix exponential

$$\exp\left(\frac{A(x)}{\mu}\right)$$

which is $O(n^3)$ flops [Moler and Van Loan, 2003].

Much better empirical performance.
Smoothing & accelerated algorithms

This means that the two classical complexity options for solving

$$\min_{X \in Q} \lambda_{\text{max}}(A(x))$$

(assuming $A(x)$ cheap)

- **Subgradient methods**

  $$O \left( \frac{D_Q^2(n^2 \log n + p_Q)}{\epsilon^2} \right)$$

- **Smooth optimization**

  $$O \left( \frac{D_Q \sqrt{\log n(n^3 + p_Q)}}{\epsilon} \right)$$

  if we pick $\| \cdot \|^2_2$ in the prox term.
Improving iteration complexity
Approximate gradients

Approximate gradient is often enough. This means computing only a few leading eigenvectors.

Spectrum of $\exp((X - \lambda_{\text{max}}(X)I)/0.1)$ at the MAXCUT solution.
Approximate gradients

Convergence guarantees using approximate gradients: if \( \tilde{\nabla} f(x) \) is the approximate gradient oracle, we require

\[
|\langle \tilde{\nabla} f(x) - \nabla f(x), y - z \rangle| \leq \delta \quad x, y, z \in Q,
\]

(the condition depends on the diameter of \( Q \)). For example, to solve

\[
\begin{align*}
\text{minimize} & \quad \lambda_{\text{max}}(A + X) \\
\text{subject to} & \quad |X_{ij}| \leq \rho
\end{align*}
\]

we only need to compute the \( j \) largest eigenvalues of \( A + X \), with \( j \) such that

\[
\frac{(n - j)e^{\lambda_j} \sqrt{\sum_{i=1}^{j} e^{2\lambda_i}}}{(\sum_{i=1}^{j} e^{\lambda_i})^2} + \frac{\sqrt{n - j} e^{\lambda_j}}{\sum_{i=1}^{j} e^{\lambda_i}} \leq \frac{\delta}{\rho n}.
\]

The impact of the diameter makes these conditions quite conservative.
Approximate gradients

Other possible conditions (often less stringent), when solving

\[
\min_{x \in Q} \max_{u \in U} \Psi(x, u)
\]

If \(u_x\) is an approximate solution to \(\max_{u \in U} \Psi(x, u)\), we can check \(V_i(u_x) \leq \delta\)

\[
V_1(u_x) = \max_{u \in U} \nabla_2 \Psi(x, u_x)^T(u - u_x)
\]

\[
V_2(u_x) = \max_{u \in U} \left\{ \Psi(x, u) - \Psi(x, u_x) + \kappa \|u - u_x\|_2^2 / 2 \right\}
\]

\[
V_3(u_x) = \max_{u \in U} \Psi(x, u) - \Psi(x, u_x)
\]

where

\[
V_1(u_x) \leq V_2(u_x) \leq V_3(u_x) \leq \delta
\]

The target accuracy \(\delta\) on the oracle is a function of the target accuracy \(\epsilon\).

See [d'Aspremont, 2008a], [Devolder, Glineur, and Nesterov, 2011] for further details.
Stochastic Smoothing

Max-rank one Gaussian smoothing. Suppose we pick $u_i \in \mathbb{R}^n$ with i.i.d. $u_{ij} \sim \mathcal{N}(0, 1)$ and define

$$f(X) = \mathbb{E} \left[ \max_{i=1,\ldots,k} \lambda_{\max}(X + (\epsilon/n)u_i u_i^T) \right]$$

- Approximation results are preserved up to a constant $c_k > 0$

$$\lambda_{\max}(X) \leq \mathbb{E}[\lambda_{\max}(X + (\epsilon/n)u u^T)] \leq \lambda_{\max}(X) + c_k \epsilon$$

- The function $f(X)$ is smooth and the Lipschitz constant of its gradient is bounded by

$$L_f \leq \mathbb{E} \left[ \frac{n}{2\epsilon} \left( \min_{i=1,\ldots,k} \frac{1}{w_{i,1}^2} \right) \right] \leq C_k \frac{n}{\epsilon}$$

where $C_k = \frac{1}{\sqrt{2}k-2}$, is finite when $k \geq 3$.

- Computing $\max_{i=1,\ldots,k} \lambda_{\max}(X + (\epsilon/n)u_i u_i^T)$ costs $O(kn^2 \log n)$. 
**Stochastic Smoothing**

**Optimal Stochastic Composite Optimization.** The algorithm in Lan [2009] solves

\[
\min_{x \in Q} \Psi(x) \triangleq f(x) + h(x)
\]

with the following assumptions

- \( f(x) \) has Lipschitz gradient with constant \( L \) and \( h(x) \) is Lipschitz with constant \( M \),
- we have a **stochastic oracle** \( G(x, \xi_t) \) for the gradient, which satisfies

\[
\mathbb{E}[G(x, \xi_t)] = g(x) \in \partial \Psi(x) \quad \text{and} \quad \mathbb{E}[\|G(x, \xi_t) - g(x)\|_*^2] \leq \sigma^2
\]

After \( N \) iterations, the iterate \( x_{N+1} \) satisfies

\[
\mathbb{E} \left[ \Psi(x_{N+1}) - \Psi^* \right] \leq \frac{8LD^2_{\omega,Q}}{N^2} + \frac{4D_{\omega,Q} \sqrt{4M^2 + \sigma^2}}{\sqrt{N}}
\]

which is optimal. Additional assumptions guarantee convergence w.h.p.
Maximum Eigenvalue Minimization

For maximum eigenvalue minimization

- We have $\sigma \leq 1$, but we can reduce this by averaging $q$ gradients, to control the tradeoff between smooth and non-smooth terms.
- If we set $q = \max\{1, D_Q/(\epsilon \sqrt{n})\}$ and $N = 2D_Q\sqrt{n}/\epsilon$ we get the following complexity picture

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Num. of Iterations</th>
<th>Cost per Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonsmooth alg.</td>
<td>$O\left(\frac{D_Q^2}{\epsilon^2}\right)$</td>
<td>$O(p_Q + n^2 \log n)$</td>
</tr>
<tr>
<td>Smooth stochastic alg.</td>
<td>$O\left(\frac{D_Q\sqrt{n}}{\epsilon}\right)$</td>
<td>$O\left(p_Q + \max\left{1, \frac{D_Q}{\epsilon \sqrt{n}}\right} n^2 \log n\right)$</td>
</tr>
<tr>
<td>Smoothing alg.</td>
<td>$O\left(\frac{D_Q\log n}{\epsilon}\right)$</td>
<td>$O(p_Q + n^3)$</td>
</tr>
</tbody>
</table>
Stochastic Smoothing

- Approximate gradients reduce empirical complexity. No a priori bounds on iteration cost.
- More efficient to run a lot of cheaper iterations, everything else being equal.

Many open questions. . .

- Not clear if rank one perturbations achieve the optimal complexity/smoothness tradeoff. Can we replicate the exponential smoothing stochastically?
- Non monotonic line search for stochastic optimization?
Outline

- Introduction
- First-order methods
  - Subgradient methods
  - Smoothing & accelerated algorithms
  - Improving iteration complexity
- Exploiting structure
  - Frank-Wolfe
  - Block coordinate descent
  - Dykstra, alternating projection
  - Localization, cutting-plane methods
Frank-Wolfe

- Classical first order methods for solving

\[
\begin{align*}
\text{minimize} & \quad f(x) \\
\text{subject to} & \quad x \in C,
\end{align*}
\]

in \( x \in \mathbb{R}^n \), with \( C \subset \mathbb{R}^n \) convex, relied on the assumption that the following \textbf{prox subproblem} could be solved efficiently

\[
\begin{align*}
\text{minimize} & \quad y^T x + d(x) \\
\text{subject to} & \quad x \in C,
\end{align*}
\]

in the variable \( x \in \mathbb{R}^n \), where \( d(x) \) is a strongly convex function.

- The Franke-Wolfe alg. assumes that the \textbf{affine minimization subproblem}

\[
\begin{align*}
\text{minimize} & \quad d^T x \\
\text{subject to} & \quad x \in C
\end{align*}
\]

can be solved efficiently for any \( y \in \mathbb{R}^n \).
Frank-Wolfe

Frank and Wolfe [1956] algorithm. See also [Jaggi, 2011].

**Input:** A starting point $x_0 \in C$.

1. **for** $t = 0$ to $N - 1$ **do**
2. Compute $\nabla f(y_k)$
3. Solve the affine minimization subproblem

$$
\begin{align*}
\text{minimize} & \quad x^T \nabla f(x_k) \\
\text{subject to} & \quad x \in C
\end{align*}
$$

in $x \in \mathbb{R}^n$, call the solution $x_d$.

4. Update the current point

$$
    x_{k+1} = x_k + \frac{2}{k+2} (x_d - x_k)
$$

5. **end for**

**Output:** A point $x_N$.

Note that all iterates are feasible.
**Complexity.** Assume that $f$ is differentiable. Define the curvature $C_f$ of the function $f(x)$ as

$$C_f \triangleq \sup_{s,x \in M, \alpha \in [0,1], y = x + \alpha(s-x)} \frac{1}{\alpha^2} \left( f(y) - f(x) - \langle y - x, \nabla f(x) \rangle \right).$$

The Franke-Wolfe algorithm will then produce an $\epsilon$ solution after

$$N_{\text{max}} = \frac{4C_f}{\epsilon}$$

iterations.

- Can use line search at each iteration to improve convergence.
Stopping criterion. At each iteration, we get a lower bound on the optimum as a byproduct of the affine minimization step. By convexity,

\[ f(x_k) + \nabla f(x_k)^T(x_d - x_k) \leq f(x), \quad \text{for all } x \in C \]

and finally, calling \( f^* \) the optimal value of problem, we obtain

\[ f(x_k) - f^* \leq \nabla f(x_k)^T(x_k - x_d). \]

This allows us to bound the suboptimality of iterate at no additional cost.
Example. Semidefinite optimization with bounded trace.

\[
\begin{align*}
\text{minimize} & \quad f(X) \\
\text{subject to} & \quad \text{Tr}(X) = 1, \; X \succeq 0,
\end{align*}
\] in the variable \( X \in S_n \).

The affine minimization subproblem is written

\[
\begin{align*}
\text{minimize} & \quad \text{Tr}(\nabla f(X) Y) \\
\text{subject to} & \quad \text{Tr}(Y) = 1, \; Y \succeq 0,
\end{align*}
\] in the variable \( Y \in S_n \), and can be solved by a partial eigenvalue decomposition, with the optimum value equal to \( \lambda_{\min}(\nabla f(X)) \) [cf. Jaggi, 2011]. Each iteration is a rank one update.
Block coordinate descent methods
Coordinate Descent

We seek to solve

\[
\begin{align*}
\text{minimize} & \quad f(x) \\
\text{subject to} & \quad x \in C
\end{align*}
\]

in the variable \( x \in \mathbb{R}^n \), with \( C \subset \mathbb{R}^n \) convex.

- Our main assumption here is that \( C \) is a product of simpler sets. We rewrite the problem

\[
\begin{align*}
\text{minimize} & \quad f(x_1, \ldots, x_p) \\
\text{subject to} & \quad x_i \in C_i, \quad i = 1, \ldots, p
\end{align*}
\]

where \( C = C_1 \times \ldots \times C_p \).

- This helps if the minimization subproblems

\[
\min_{x_i \in C_i} f(x_1, \ldots, x_i, \ldots, x_p)
\]

can be solved very efficiently (or in closed-form).
Coordinate Descent

Algorithm. The algorithm simply computes the iterates $x^{(k+1)}$ as

$$
x_i^{(k+1)} = \underset{x_i \in C_i}{\text{argmin}} \ f(x_1^{(k)}, \ldots, x_i^{(k)}, \ldots, x_p^{(k)})
$$

$$
x_j^{(k+1)} = x_j^{(k)}, \quad j \neq i
$$

for a certain $i \in [1, p]$, cycling over all indices in $[1, p]$.

Convergence.

- Complexity analysis similar to coordinate-wise gradient descent (or steepest descent in $\ell_1$ norm).
- Need $f(x)$ strongly convex to get explicit complexity bound [Nesterov, 2010].
Example. **Covariance selection** [d’Aspremont et al., 2006]. The dual of the covariance selection problem is written

\[
\begin{align*}
\text{maximize} & \quad \log \det (S + U) \\
\text{subject to} & \quad \|U\|_\infty \leq \rho \\
& \quad S + U \succ 0
\end{align*}
\]

Let \( C = S + U \) be the current iterate, after permutation we can always assume that we optimize over the last column

\[
\begin{align*}
\text{maximize} & \quad \log \det \begin{pmatrix} C_{11}^{11} & C_{12}^{11} + u \\ C_{21}^{11} + u^T & C_{22}^{11} \end{pmatrix} \\
\text{subject to} & \quad \|u\|_\infty \leq \rho
\end{align*}
\]

where \( C_{12}^{11} \) is the last column of \( C \) (off-diag.).
Coordinate Descent

We can use the block determinant formula

$$\det \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \det(A) \det(D - CA^{-1}B)$$

to show that each row/column iteration reduces to a simple box-constrained QP

$$\begin{align*}
\text{minimize} & \quad u^T(C^{11})^{-1}u \\
\text{subject to} & \quad \|u\|_\infty \leq \rho
\end{align*}$$

the dual of this last problem is a LASSO optimization problem.
Dykstra, alternating projection
We focus on a simple **feasibility problem**

\[ \text{find } x \in C_1 \cap C_2 \]

in the variable \( x \in \mathbb{R}^n \) with \( C_1, C_2 \subset \mathbb{R}^n \) two convex sets.

We assume now that the **projection problems on** \( C_i \) are easier to solve

\[
\begin{align*}
\text{minimize} & \quad \| x - y \|_2 \\
\text{subject to} & \quad x \in C_i
\end{align*}
\]

in \( x \in \mathbb{R}^n \).
Dykstra, alternating projection

Algorithm (alternating projection)

- Choose $x_0 \in \mathbb{R}^n$.
- For $k = 1, \ldots, k_{\text{max}}$ iterate
  1. Project on $C_1$
     $$x_{k+1/2} = \arg\min_{x \in C_1} \| x - x_k \|_2$$
  2. Project on $C_2$
     $$x_{k+1} = \arg\min_{x \in C_2} \| x - x_{k+1/2} \|_2$$

Convergence. We can show $\text{dist}(x_k, C_1 \cap C_2) \to 0$. Linear convergence provided some additional regularity assumptions. See e.g. [Lewis, Malick, et al., 2008]
Dykstra, alternating projection

Algorithm (Dykstra)

- Choose \( x_0, z_0 \in \mathbb{R}^n \).
- For \( k = 1, \ldots, k^{\text{max}} \) iterate
  1. Project on \( C_1 \)
     \[
     x_{k+1/2} = \underset{x \in C_1}{\text{argmin}} \| x - z_k \|_2
     \]
  2. Update
     \[
     z_{k+1/2} = 2x_{k+1/2} - z_k
     \]
  3. Project on \( C_2 \)
     \[
     x_{k+1} = \underset{x \in C_2}{\text{argmin}} \| x - z_{k+1/2} \|_2
     \]
  4. Update
     \[
     z_{k+1} = z_k + x_{k+1} - x_{k+1/2}
     \]

Convergence. Usually faster than simple alternating projection.
Dykstra, alternating projection

Example. **Matrix completion** problem, given coefficients $b_{ij}$ for $(i,j) \in S$

Find $X$ such that

$$X_{ij} = b_{ij}, \quad (i,j) \in S$$

$$X \succeq 0,$$

in the variable $X \in S_n$.

Blue: alternating projection. Red: Dykstra. (from EE364B)
Dykstra, alternating projection

Countless variations.

- Proximal point algorithm
- Douglas-Rachford splitting
- Operator splitting methods
- Bregman iterative methods
- ...
Localization methods
Localization methods

From EE364B course at Stanford... 

- **Function** $f : \mathbb{R}^n \rightarrow \mathbb{R}$ convex (and for now, differentiable)
- **Problem:** minimize $f$
- **Oracle model:** for any $x$ we can evaluate $f$ and $\nabla f(x)$ (at some cost)

Main assumption: **evaluating the gradient is very expensive.**

Convexity means $f(x) \geq f(x_0) + \nabla f(x_0)^T(x - x_0)$, so

$$\nabla f(x_0)^T(x - x_0) \geq 0 \implies f(x) \geq f(x_0)$$

*i.e., all points in halfspace $\nabla f(x_0)^T(x - x_0) \geq 0$ are worse than $x_0$*
Localization methods

- $\mathcal{P}_k$ gives our uncertainty of $x^*$ at iteration $k$
- want to pick $x^{(k)}$ so that $\mathcal{P}_{k+1}$ is as small as possible
- clearly want $x^{(k)}$ near center of $C^{(k)}$
Localization methods

**analytic center** of polyhedron $\mathcal{P} = \{ z \mid a_i^T z \leq b_i, \ i = 1, \ldots, m \}$ is

$$\text{AC}(\mathcal{P}) = \arg\min_z - \sum_{i=1}^{m} \log(b_i - a_i^T z)$$

**ACCPM** is localization method with next query point $x^{(k+1)} = \text{AC}(\mathcal{P}_k)$ (found by Newton’s method)
Localization methods

- Let $x^*$ be analytic center of $\mathcal{P} = \{ z \mid a_i^T z \preceq b_i, \ i = 1, \ldots, m \}$
- Let $H^*$ be Hessian of barrier at $x^*$,

$$H^* = -\nabla^2 \sum_{i=1}^{m} \log(b_i - a_i^T z) \bigg|_{z=x^*} = \sum_{i=1}^{m} \frac{a_i a_i^T}{(b_i - a_i^T x^*)^2}$$

- Then, $\mathcal{P} \subseteq \mathcal{E} = \{ z \mid (z - x^*)^T H^* (z - x^*) \leq m^2 \}$ (not hard to show)
- Let $\mathcal{E}^{(k)}$ be outer ellipsoid associated with $x^{(k)}$
- A lower bound on optimal value $p^*$ is

$$p^* \geq \inf_{z \in \mathcal{E}^{(k)}} \left( f(x^{(k)}) + g^{(k)}^T (z - x^{(k)}) \right)$$

$$= f(x^{(k)}) - m_k \sqrt{g^{(k)}^T H^{(k)} - 1} g^{(k)}$$

($m_k$ is number of inequalities in $\mathcal{P}_k$)
- Gives simple stopping criterion $\sqrt{g^{(k)}^T H^{(k)} - 1} g^{(k)} \leq \epsilon / m_k$
Localization methods

ACCPM algorithm.

**Input:** Polyhedron $\mathcal{P}$ containing $x^*$. 
1. **for** $t = 0$ to $N - 1$ **do**
2. Compute $x^*$, the analytic center of $\mathcal{P}$, and the Hessian $H^*$. 
3. Compute $f(x^*)$ and $g \in \partial f(x^*)$. 
4. Set $u := \min\{u, f(x^*)\}$ and $l := \max\{l, f(x^*) - m\sqrt{g^T H^*^{-1} g}\}$. 
5. Add inequality $g^T(z - x^*) \leq 0$ to $\mathcal{P}$. 
6. **end for**

**Output:** A localization set $\mathcal{P}$. 

A. d’Aspremont
Localization methods

ACCPM adds an inequality to \( \mathcal{P} \) each iteration, so centering gets harder, more storage as algorithm progresses.

Schemes for **dropping constraints** from \( \mathcal{P}^{(k)} \):

- remove all redundant constraints (expensive)
- remove some constraints known to be redundant
- remove constraints based on some relevance ranking
Localization methods

Example. **Classification with indefinite kernels.** [Luss and d’Aspremont, 2008]

Solve

\[
\min_{\{K \succeq 0, \|K-K_0\|_F^2 \leq \beta\}} \max_{\{\alpha^T y = 0, \ 0 \leq \alpha \leq C\}} \alpha^T e - \frac{1}{2} \text{Tr}(K(Y\alpha)(Y\alpha)^T)
\]

in the variables \( K \in \mathbb{S}^n \) and \( \alpha \in \mathbb{R}^n \). This can be written

\[
\text{maximize} \quad \alpha^T e - \frac{1}{2} \sum_i \max(0, \lambda_i(K_0 + (Y\alpha)(Y\alpha)^T/4\rho)) (\alpha^T Y v_i)^2 \\
+ \rho \sum_i \max(0, \lambda_i(K_0 + (Y\alpha)(Y\alpha)^T/4\rho))^2 + \rho \text{Tr}(K_0K_0) \\
- 2\rho \sum_i \text{Tr}((v_i v_i^T)K_0) \max(0, \lambda_i(K_0 + (Y\alpha)(Y\alpha)^T/4\rho))
\]

subject to \( \alpha^T y = 0, \ 0 \leq \alpha \leq C \)

in the variable \( \alpha \in \mathbb{R}^n \).

Computing the gradient at each iteration is expensive, but the feasible set is a Polyhedron.
Convergence plots for ACCPM (left) and projected gradient method (right) on random subsets of the USPS-SS-3-5 data set (average gap versus iteration number, dashed lines at plus and minus one standard deviation).
Conclusion

Countless other methods not discussed here. Some with no convergence guarantees.

- Low Rank semidefinite programming. (Choose a factorization $X = VV^T$ and solve in $V$). [Burer and Monteiro, 2003, Journée et al., 2008]
- Row by row methods. Some solver variants for MAXCUT require only matrix vector products. [Wen et al., 2009]

Some recent activity on subsampling.

- Variational inequality formulation [Juditsky et al., 2008, Baes et al., 2011].
- Columnwise or elementwise matrix subsampling [d’Aspremont, 2008b].
Large-scale semidefinite programs.

- First-order algorithms for solving (mostly) generic problems.
- For more specialized problems

Structure $\Rightarrow$ algorithmic choices

What subproblem can you solve easily? Which algorithm exploits it best?
References


