PENN NON
A Generalized Augmented Lagrangian Method for
Convex NLP and SDP

Michal Kočvara

Institute of Information Theory and Automation
Academy of Sciences of the Czech Republic
and
Czech Technical University
kocvara@utia.cas.cz
http://www.utia.cas.cz/kocvara
**PBM Method for convex NLP**

Ben-Tal, Zibulevsky, ’92, ’97

**Combination of:**
(exterio) **P**enalty meth., (interior) **B**arrier meth., Method of **M**ultipliers

**Problem:**

\[
(CP) \quad \min_{x \in \mathbb{R}^n} \{ f(x) : g_i(x) \leq 0, \quad i = 1, \ldots, m \}
\]

**Assume:**

1. \( f, g_i \ (i = 1, \ldots, m) \) convex
2. \( X^* \) nonempty and compact \( (A1) \)
3. \( \exists \hat{x} \) so that \( g_i(\hat{x}) < 0 \) for all \( i = 1, \ldots, m \) \( (A2) \)
φ possibly smooth, domφ possibly large

(φ₀) φ strictly convex, strictly monotone increasing and $C^2$
(φ₁) domφ = (−∞, b) with 0 < b ≤ ∞
(φ₂) φ(0) = 0, (φ₄) \[ \lim_{t \to b} \varphi'(t) = \infty \]
(φ₃) φ'(0) = 1, (φ₅) \[ \lim_{t \to -\infty} \varphi'(t) = 0 \]
Examples:

\[ \varphi^r_1(t) = \begin{cases} 
    c_1 \frac{1}{2} t^2 + c_2 t + c_3 & t \geq r \\
    c_4 \log(t - c_5) + c_6 & t < r 
\end{cases} \]
Examples:

\[ \varphi^r_1(t) = \begin{cases} c_1 \frac{1}{2} t^2 + c_2 t + c_3 & t \geq r \\ c_4 \log(t - c_5) + c_6 & t < r. \end{cases} \]

\[ \varphi^r_2(t) = \begin{cases} \frac{c_1}{2} t^2 + c_2 t + c_3 & t \geq r, \\ \frac{c_4}{t - c_5} + c_6 & t < r, \quad r \in \langle -1, 1 \rangle. \end{cases} \]
Examples:

\[ \varphi_r^1(t) = \begin{cases} 
  c_1 \frac{1}{2} t^2 + c_2 t + c_3 & t \geq r \\
  c_4 \log(t - c_5) + c_6 & t < r 
\end{cases} \]

\[ \varphi_r^2(t) = \begin{cases} 
  c_1 \frac{1}{2} t^2 + c_2 t + c_3 & t \geq r \\
  \frac{c_4}{t - c_5} + c_6 & t < r, \quad r \in \langle -1, 1 \rangle 
\end{cases} \]

Properties:

- \( C^2 \), bounded second derivative

  \[ \implies \text{improved behaviour of Newton’s method} \]

- composition of barrier branch (logarithmic/reciprocal) and penalty branch (quadratic)
With $p_i > 0$ for $i \in \{1, \ldots, m\}$, we have

$$g_i(x) \leq 0 \iff p_i \varphi(g_i(x)/p_i) \leq 0, \quad i = 1, \ldots, m$$

The corresponding *augmented Lagrangian*:

$$F(x, u, p) := f(x) + \sum_{i=1}^{m} u_i p_i \varphi(g_i(x)/p_i)$$

**PBM algorithm:**

$$
\begin{align*}
x^{k+1} &= \arg \min_{x \in \mathbb{R}^n} F(x, u^k, p^k) \\
u^{k+1}_i &= u^k_i \varphi'(g_i(x^{k+1})/p^k_i) \quad i = 1, \ldots, m \\
p^{k+1}_i &= \pi p^k_i \quad i = 1, \ldots, m
\end{align*}
$$
Properties of the PBM method

Theory:

- \( \{u^k\}_k \) generated by PBM is the same as for a Proximal Point algorithm applied to the dual problem (\( \rightarrow \) convergence proof)

- any cluster point of \( \{x^k\}_k \) is an optimal solution to \( (CP) \)

- \( f(x^k) \rightarrow f^* \) without \( p_k \rightarrow 0 \)
Properties of the PBM method

Theory:
- \( \{u^k\}_k \) generated by PBM is the same as for a Proximal Point algorithm applied to the dual problem (→ convergence proof)
- any cluster point of \( \{x^k\}_k \) is an optimal solution to \( (CP) \)
- \( f(x^k) \to f^* \) without \( p_k \to 0 \)

Praxis:
- fast convergence thanks to the barrier branch of \( \varphi \)
- particularly suitable for large sparse problems
- robust, typically 10–15 outer iterations and 40–80 Newton steps
Problem: \( \min_{x \in \mathbb{R}^n} \{ b^T x : \mathcal{A}(x) \preceq 0 \} \)

Question: How can the matrix constraint

\[ \mathcal{A}(x) \preceq 0 \quad (\mathcal{A} : \mathbb{R}^n \rightarrow \mathbb{S}_d \text{ convex}) \]

be treated by PBM approach?

Idea: Find an augmented Lagrangian as follows:

\[ F(x, U, p) = f(x) + \langle U, \Phi_p(\mathcal{A}(x)) \rangle_{\mathbb{S}_d} \]
PBM in semidefinite programming

Problem: \[ \min_{x \in \mathbb{R}^n} \{ b^T x : \mathcal{A}(x) \preceq 0 \} \]

Question: How can the matrix constraint

\[ \mathcal{A}(x) \preceq 0 \quad (\mathcal{A} : \mathbb{R}^n \rightarrow \mathbb{S}_d \text{ convex}) \]

be treated by PBM approach?

Idea: Find an \textit{augmented Lagrangian} as follows:

\[ F(x, U, p) = f(x) + \langle U, \Phi_p (\mathcal{A}(x)) \rangle_{\mathbb{S}_d} \]

Notation:

\[ \langle A, B \rangle_{\mathbb{S}_d} := \text{tr} (A^T B) \quad \text{inner product on } \mathbb{S}_d \]
\[ \mathbb{S}_{d+} = \{ A \in \mathbb{S}_d \mid A \text{ positive semidefinite} \} \quad \text{matrix multiplier (dual variable)} \]
\[ U \in \mathbb{S}_{d+} \quad \Phi_p \quad \text{penalty function on } \mathbb{S}_d \]
Given:
scalar valued penalty function $\varphi$ satisfying $(\varphi_0) - (\varphi_5)$
matrix $A = S^\top \Lambda S$, where $\Lambda = \text{diag} \ (\lambda_1, \lambda_2, \ldots, \lambda_d)^\top$

Define

\[
A \xrightarrow{\Phi_p} S^T \begin{pmatrix} p\varphi \left( \frac{\lambda_1}{p} \right) & 0 & \ldots & 0 \\ 0 & p\varphi \left( \frac{\lambda_2}{p} \right) & \vdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \ldots & 0 & p\varphi \left( \frac{\lambda_d}{p} \right) \end{pmatrix} S
\]

\[\rightarrow \text{any positive eigenvalue of } A \text{ is “penalized” by } \varphi\]
We have
\[ \mathcal{A}(x) \preceq 0 \iff \Phi_p(\mathcal{A}(x)) \preceq 0 \]
and the corresponding \textit{augmented Lagrangian}:
\[ F(x, U, p) := f(x) + \langle U, \Phi_p(\mathcal{A}(x)) \rangle_{S_d} \]

\textbf{PBM algorithm:}

\begin{align*}
(i) & \quad x^{k+1} = \arg \min_{x \in \mathbb{R}^n} F(x, U^k, p^k) \\
(ii) & \quad U^{k+1} = D_A \Phi_p(\mathcal{A}(x); U^k) \\
(iii) & \quad p^{k+1} < p^k
\end{align*}
The first idea may not be the best one:

The matrix function $\Phi_p$ corresponding to $\varphi$ is convex but may be nonmonotone on $\mathbb{H}_d(r, \infty)$ (right branch) $\rightarrow$

$$\langle U, \Phi_p (A(x)) \rangle_{S_d}$$

may be nonconvex.
The first idea may not be the best one:

- The matrix function $\Phi_p$ corresponding to $\varphi$ is convex but may be nonmonotone on $H_d(r, \infty)$ (right branch) \[ \langle U, \Phi_p(\mathcal{A}(x)) \rangle_{S_d} \]

may be nonconvex.

- Complexity of Hessian assembling \[ O(d^4 + d^3 n + d^2 n^2) \]

Even for a very sparse structure the complexity can be $O(d^4)$!

$n \ldots$ number of variables

d \ldots size of matrix constraint
PBM algorithm for semidefinite problems

Hessian:

\[
\begin{bmatrix}
\nabla_x x \langle U, \Phi_p (A(x)) \rangle_{S_d} \nabla \end{bmatrix}_{i,j} = \\
\sum_{k=1}^{d} (s_k(x)^\top A_i \left[ S(x) \left( \left[ \Delta^2 \varphi(\lambda_l(x), \lambda_m(x), \lambda_k(x)) \right]_{l,m=1}^n \right) \right]_{i,j} = \\
\circ[S(x)^\top U S(x)] \circ S(x)^\top A_j s_k(x)
\end{bmatrix}
\]

- \(S\) : decomposition matrix of \(A(x)\)
- \(s_k\) : \(k\)-th column of \(S\)
- \(\Delta^i\) : divided difference of \(i\)-th order
- \(A^* : S_d \rightarrow \mathbb{R}^n\) : conjugate operator to \(A\)
Find a penalty function $\varphi$ which allows “direct” computation of $\Phi$, its gradient and Hessian.
Find a penalty function \( \varphi \) which allows “direct” computation of \( \Phi \), its gradient and Hessian.

**Example:** \( \mathcal{A}(x) = \sum x_i A_i \)

\[
\varphi(x) = x^2 \implies \Phi(A) = A^2
\]

Then

\[
\frac{\partial}{\partial x_i} \Phi(\mathcal{A}(x)) = \mathcal{A}(x)A_i + A_i \mathcal{A}(x)
\]

and

\[
\frac{\partial^2}{\partial x_i \partial x_j} \Phi(\mathcal{A}(x)) = A_j A_i + A_i A_j
\]
The reciprocal barrier function in SDP: \( \mathcal{A}(x) = \sum x_i A_i \)

\[
\varphi := \frac{1}{t-1} - 1
\]

The corresponding matrix function is

\[
\Phi(A) = (A - I)^{-1} - I
\]

and we can show that

\[
\frac{\partial}{\partial x_i} \Phi(\mathcal{A}(x)) = (A - I)^{-1} A_i (A - I)^{-1}
\]

and

\[
\frac{\partial^2}{\partial x_i \partial x_j} \Phi(\mathcal{A}(x)) = (A - I)^{-1} A_i (A - I)^{-1} A_j (A - I)^{-1}
\]
Complexity of Hessian assembling:

- \( O(d^3 n + d^2 n^2) \) for dense matrices
- \( O(n^2 K^2) \) for sparse matrices
  \((K \ldots \text{max. number of nonzeros in } A_i, \ i = 1, \ldots, n)\)
- Compare to \( O(d^4 + d^3 n + d^2 n^2) \) in the general case

\[
\min_{x \in \mathbb{R}^n} \left\{ b^T x : A(x) \preceq 0 \right\} \quad A : \mathbb{R}^n \longrightarrow S_d
\]
Handling sparsity

...essential for code efficiency

\[
\min_{x \in \mathbb{R}^n} \left\{ b^T x : A(x) \preceq 0 \right\} \quad A = \sum x_i A_i
\]

Three basic sparsity types:

- many (small) blocks $\rightarrow$ sparse Hessian (multi-load truss/material)
Handling sparsity

...essential for code efficiency

\[
\min_{x \in \mathbb{R}^n} \{ b^T x : A(x) \preceq 0 \}
\]

\[ A = \sum x_i A_i \]

Three basic sparsity types:

- many (small) blocks → sparse Hessian (multi-load truss/material)

- few (large) blocks
...essential for code efficiency

$$\min_{x \in \mathbb{R}^n} \{ b^T x : A(x) \preceq 0 \} \quad A = \sum x_i A_i$$

Three basic sparsity types:

- many (small) blocks → sparse Hessian (multi-load truss/material)
- few (large) blocks
  - $A$ dense, $A_i$ sparse  (most of SDPLIB examples)
Handling sparsity

full version as inefficient as general sparse version

Recently, 3 matrix-matrix multiplication routines:

- full–full
- full–sparse
- sparse–sparse
Handling sparsity

essential for code efficiency

\[
\min_{x \in \mathbb{R}^n} \left\{ b^T x : \mathcal{A}(x) \preceq 0 \right\} \quad \mathcal{A} = \sum x_i A_i
\]

Three basic sparsity types:

- many (small) blocks → sparse Hessian (multi-load truss/material)
- few (large) blocks
  - \( \mathcal{A} \) dense, \( A_i \) sparse  (most of SDPLIB examples)
  - \( \mathcal{A} \) sparse  (truss design with buckling/vibration, maxG, . . .)

\[
(A - I)^{-1} A_i (A - I)^{-1} A_j (A - I)^{-1}
\]
Handling sparsity

Fast inverse computation of sparse matrices

\[ Z = M^{-1}N \]

Explicite inverse of \( M \): \( \mathcal{O}(n^3) \)
Assume \( M \) is sparse and Cholesky factor \( L \) of \( M \) is sparse

\[ Z_i = (L^{-1})^T L^{-1} N_i, \ i = 1, \ldots, n \]

Complexity: \( n \) times \( nK \rightarrow \mathcal{O}(n^2K) \)
New code called PENNON

Comparison with
DSDP by Benson and Ye
SDPT3 by Toh, Todd and Tütüncü
SeDuMi by Jos Sturm

SDPLIB problems: http://www.nmt.edu/~sdplib/
### Numerical results

<table>
<thead>
<tr>
<th>problem</th>
<th>variables</th>
<th>matrix</th>
<th>DSDP</th>
<th>SDPT3</th>
<th>PENNON</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch8</td>
<td>174</td>
<td>335</td>
<td>4</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>control7</td>
<td>136</td>
<td>45</td>
<td>114</td>
<td>48</td>
<td>82</td>
</tr>
<tr>
<td>control11</td>
<td>1596</td>
<td>165</td>
<td>1236</td>
<td>288</td>
<td>974</td>
</tr>
<tr>
<td>gpp500-4</td>
<td>501</td>
<td>500</td>
<td>28</td>
<td>39</td>
<td>21</td>
</tr>
<tr>
<td>mcp500-1</td>
<td>500</td>
<td>500</td>
<td>2</td>
<td>18</td>
<td>7</td>
</tr>
<tr>
<td>qap10</td>
<td>1021</td>
<td>101</td>
<td>19</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>ss30</td>
<td>132</td>
<td>426</td>
<td>10</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>theta6</td>
<td>4375</td>
<td>300</td>
<td>551</td>
<td>287</td>
<td>797</td>
</tr>
<tr>
<td>equalG11</td>
<td>801</td>
<td>801</td>
<td>139</td>
<td>156</td>
<td>102</td>
</tr>
<tr>
<td>equalG51</td>
<td>1001</td>
<td>1001</td>
<td>351</td>
<td>350</td>
<td>391</td>
</tr>
<tr>
<td>maxG11</td>
<td>800</td>
<td>800</td>
<td>6</td>
<td>54</td>
<td>25</td>
</tr>
<tr>
<td>maxG32</td>
<td>2000</td>
<td>2000</td>
<td>72</td>
<td>650</td>
<td>259</td>
</tr>
</tbody>
</table>
Multiple-load Free Material Optimization

After reformulation, discretization, further reformulation:

\[
\min_{\alpha \in \mathbb{R}, x \in (\mathbb{R}^n)^L} \left\{ \alpha - \sum_{\ell=1}^{L} (c^\ell)^T x^\ell \mid A_i(\alpha, x) \succeq 0, i = 1, \ldots, m \right\}
\]

Many (~5000) small (11–19) matrices.
Large dimension \((nL \sim 20\,000)\)

In a standard form:

\[
\min_{x \in (\mathbb{R}^n)^L} \left\{ a^T x \mid \sum_{i=1}^{nL} x_i B_i \succeq 0 \right\}
\]
### Examples from Mechanics

<table>
<thead>
<tr>
<th>problem</th>
<th>no. of var.</th>
<th>size of matrix</th>
<th>DSDP</th>
<th>SDPT3</th>
<th>SeDuMi</th>
<th>PENNON</th>
</tr>
</thead>
<tbody>
<tr>
<td>mater3</td>
<td>1439</td>
<td>3588</td>
<td>146</td>
<td>35</td>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>mater4</td>
<td>4807</td>
<td>12498</td>
<td>6269</td>
<td>295</td>
<td>97</td>
<td>29</td>
</tr>
<tr>
<td>mater5</td>
<td>10143</td>
<td>26820</td>
<td>36000</td>
<td>m</td>
<td>202</td>
<td>78</td>
</tr>
<tr>
<td>mater6</td>
<td>20463</td>
<td>56311</td>
<td>m</td>
<td>m</td>
<td>533</td>
<td>233</td>
</tr>
</tbody>
</table>
Lowest eigenfrequency of the optimal structure should be bigger than a prescribed value

\[
\min_{t,u} \sum t_i \\
\text{s.t. } A(t)u = f \\
|\sigma| \leq \sigma_{\ell} \quad (g(u) \leq c) \\
t \in T
\]

min. eigenfrequency ≥ a given value
Formulated as SDP problem:

\[
\begin{align*}
\min_t \sum t_i \\
\text{subject to} \quad & A(t) - \lambda M(t) \succeq 0 \\
& \begin{pmatrix} c & f^T \\ f & A(t) \end{pmatrix} \succeq 0 \\
& t_i \geq 0, \quad i = 1, \ldots, n
\end{align*}
\]

where

\[
A(t) = \sum t_i A_i \quad A_i = \frac{E_i}{\ell_i^2} \gamma_i \gamma_i^T
\]

\[
M(t) = \sum t_i M_i \quad M_i = c \cdot \text{diag}(\gamma_i \gamma_i^T)
\]
truss test problems

- **trto**: problems from single-load truss topology design. Normally formulated as LP, here reformulated as SDP for testing purposes.

- **vibra**: single load truss topology problems with a vibration constraint. The constraint guarantees that the minimal self-vibration frequency of the optimal structure is bigger than a given value.

- **buck**: single load truss topology problems with linearized global buckling constraint. Originally a nonlinear matrix inequality, the constraint should guarantee that the optimal structure is mechanically stable (does not buckle).

All problems characterized by sparsity of the matrix operator $\mathcal{A}$. 
### truss test problems

<table>
<thead>
<tr>
<th>problem</th>
<th>n</th>
<th>m</th>
<th>DSDP</th>
<th>SDPT3</th>
<th>PENNON</th>
</tr>
</thead>
<tbody>
<tr>
<td>trto3</td>
<td>544</td>
<td>321+544</td>
<td>11</td>
<td>19</td>
<td>17</td>
</tr>
<tr>
<td>trto4</td>
<td>1200</td>
<td>673+1200</td>
<td>134</td>
<td>124</td>
<td>106</td>
</tr>
<tr>
<td>trto5</td>
<td>3280</td>
<td>1761+3280</td>
<td>3125</td>
<td>1422</td>
<td>1484</td>
</tr>
<tr>
<td>buck3</td>
<td>544</td>
<td>641+544</td>
<td>44</td>
<td>43</td>
<td>39</td>
</tr>
<tr>
<td>buck4</td>
<td>1200</td>
<td>1345+1200</td>
<td>340</td>
<td>241</td>
<td>221</td>
</tr>
<tr>
<td>buck5</td>
<td>3280</td>
<td>3521+3280</td>
<td>10727</td>
<td>2766</td>
<td>3006</td>
</tr>
<tr>
<td>vibra3</td>
<td>544</td>
<td>641+544</td>
<td>52</td>
<td>45</td>
<td>34</td>
</tr>
<tr>
<td>vibra4</td>
<td>1200</td>
<td>1345+1200</td>
<td>596</td>
<td>294</td>
<td>191</td>
</tr>
<tr>
<td>vibra5</td>
<td>3280</td>
<td>3521+3280</td>
<td>25290</td>
<td>3601</td>
<td>2724</td>
</tr>
</tbody>
</table>
Benchmark tests by Hans Mittelmann:
http://plato.la.asu.edu/bench.html

Implemented on the NEOS server:
http://www-neos.anl.gov

Homepage:
http://www2.am.uni-erlangen.de/~kocvara/pennon/
http://www.penopt.com/

Available with MATLAB interface through TOMLAB
http://www.tomlab.biz
When PCG helps (SDP)?

Linear SDP, dense Hessian

\[ A = \sum_{i=1}^{n} A_i, \quad A_i \in \mathbb{R}^{m \times m} \]

Complexity of Hessian evaluation

- \( O(m^3_A n + m^2_A n^2) \) for dense matrices
- \( O(m^2_A n + K^2 n^2) \) for sparse matrices
  \((K \ldots \text{max. number of nonzeros in } A_i, \ i = 1, \ldots, n)\)

Complexity of Cholesky algorithm - linear SDP

- \( O(n^3) \) \((\ldots \text{from PCG we expect } O(n^2))\)

Problems with large \( n \) and small \( m \):

CG better than Cholesky (expected)
Iterative algorithms

Conjugate Gradient method for \( H d = -g, \ H \in S^n_+ \)

\[ \begin{align*}
\cdots \\
\cdots \\
y &= Hx \\
\cdots \\
\cdots
\end{align*} \]

Complexity \( O(n^2) \)

Exact arithmetics: “convergence” in \( n \) steps

\[ \rightarrow \text{overall complexity } O(n^3) \]
Iterative algorithms

Conjugate Gradient method for $Hd = -g$, $H \in S_+^n$

$\cdots$

$y = Hx$

$\cdots$

Exact arithmetics: “convergence” in $n$ steps

$\rightarrow$ overall complexity $O(n^3)$

Praxis: may be much worse (ill-conditioned problems)
Iterative algorithms

Conjugate Gradient method for $H d = -g$, $H \in S^n_+$. 

\[ y = H x \] complexity $O(n^2)$

Exact arithmetics: “convergence” in $n$ steps

$\rightarrow$ overall complexity $O(n^3)$

Praxis: may be much worse (ill-conditioned problems)

may be much better $\rightarrow$ preconditioning
Iterative algorithms

Conjugate Gradient method for \( H d = -g, \quad H \in S_+^n \)

\[
\ldots
\]

\[
y = H x
\]

\[
\ldots
\]

Exact arithmetics: “convergence” in \( n \) steps

\[
\rightarrow \text{overall complexity } O(n^3)
\]

Praxis: may be much worse (ill-conditioned problems)

may be much better \( \rightarrow \) preconditioning

Convergence theory: number of iterations depends on

- condition number
- distribution of eigenvalues
Iterative algorithms

Conjugate Gradient method for $Hd = -g$, $H \in \mathbb{S}_+^n$

\[ \ldots \]
\[ \ldots \]
\[ y = Hx \quad \text{complexity } O(n^2) \]
\[ \ldots \]
\[ \ldots \]

Exact arithmetics: “convergence” in $n$ steps

$\rightarrow$ overall complexity $O(n^3)$

Praxis: may be much worse (ill-conditioned problems)

may be much better $\rightarrow$ preconditioning

Convergence theory: number of iterations depends on

- condition number
- distribution of eigenvalues

Preconditioning: solve $M^{-1}Hd = M^{-1}g$ with $M \approx H^{-1}$
Solve \( H d = -g \), \( H \) Hessian of

\[
F(x, u, U, p, P) = f(x) + \langle U, \Phi_P (A(x)) \rangle_{S_{mA}}
\]

Condition number depends on \( P \)

Example: problem Theta2 from SDPLIB (\( n = 498 \))

\[
\kappa_0 = 394 \quad \kappa_{opt} = 4.9 \cdot 10^7
\]
Theta2 from SDPLIB \( (n = 498) \)

Behaviour of CG: testing \( \|Hd + g\|/\|g\| \)
**Theta2 from SDPLIB ($n = 498$)**

**Behaviour of QMR: testing $\|Hd + g\|/\|g\|$**
**Theta2 from SDPLIB \( (n = 498) \)**

QMR: effect of preconditioning (for small \( P \))

\[
\begin{align*}
\text{Graph 1:} & \quad x \text{ vs. } y, \\
\text{Graph 2:} & \quad x \text{ vs. } y.
\end{align*}
\]
Control3 from SDPLIB ($n = 136$)

\[ \kappa_0 = 3.1 \cdot 10^8 \]

\[ \kappa_{\text{opt}} = 7.3 \cdot 10^{12} \]
Control3 from SDPLIB \( (n = 136) \)

Behaviour of CG: testing \( \|Hd + g\|/\|g\| \)
Behaviour of QMR: testing $\|Hd + g\|/\|g\|$
Preconditioners

Should be:
- efficient (obvious but often difficult to reach)
- simple (low complexity)
- only use Hessian-vector product (NOT Hessian elements)
Preconditioners

Should be:

– efficient (obvious but often difficult to reach)
– simple (low complexity)
– only use Hessian-vector product (NOT Hessian elements)

- Diagonal
- Symmetric Gauss-Seidel
- L-BFGS (Morales-Nocedal, SIOPT 2000)
- A-inv (approximate inverse) (Benzi-Collum-Tuma, SISC 2000)

“Improves the CG performance on extremely ill-conditioned systems.”

preconditioner:

\[ M = C_k C_k^T, \quad C_{k+1} \leftarrow \alpha C_k + \beta C_k p_k p_k^T, \quad C_1 = \gamma I \]

\( \alpha, \beta, p_k \ldots \) by matrix-vector products

VERY preliminary results (MATLAB implementation)
Example: problem Theta2 from SDPLIB ($n = 498$)
Example: problem Theta2 from SDPLIB ($n = 498$)
Use finite difference formula for Hessian-vector products:

\[ \nabla^2 F(x_k)v \approx \frac{\nabla F(x_k + hv) - \nabla F(x_k)}{h} \]

with \( h = (1 + \|x_k\|_2 \sqrt{\epsilon}) \)

Complexity: Hessian-vector product = gradient evaluation
need for Hessian-vector-product type preconditioner

Limited accuracy (4–5 digits)
Test results: linear SDP, dense Hessian

Stopping criterium for PENNON

Exact Hessian: $10^{-7}$ (7–8 digits in objective function)
Approximate Hessian: $10^{-4}$ (4–5 digits in objective function)

Stopping criterium for CG/QMR ???

\[ H d = -g, \text{ stop when } \| H d + g \| / \| g \| \leq \epsilon \]
Test results: linear SDP, dense Hessian

Stopping criterium for PENNON

Exact Hessian: $10^{-7}$ (7–8 digits in objective function)
Approximate Hessian: $10^{-4}$ (4–5 digits in objective function)

Stopping criterium for CG/QMR ???

$$Hd = -g, \text{ stop when } \|Hd + g\|/\|g\| \leq \epsilon$$

Experiments: $\epsilon = 10^{-2}$ sufficient.
→ often very low (average) number of CG iterations

Complexity: $n^3 \rightarrow kn^2$, $k \approx 4 - 8$

Practice: effect not that strong, due to other complexity issues
Problems with large $n$ and small $m$

Library of examples with large $n$ and small $m$
(courtesy of Kim Toh)

CG-exact much better than Cholesky
CG-approx much better than CG-exact
<table>
<thead>
<tr>
<th>problem</th>
<th>n</th>
<th>m</th>
<th>PENSDP CPU</th>
<th>PENSDP (APCG) CPU</th>
<th>PPRDL</th>
<th>iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>ham_7_5_6</td>
<td>1793</td>
<td>128</td>
<td>126</td>
<td>4</td>
<td>52</td>
<td>1</td>
</tr>
<tr>
<td>ham_9_8</td>
<td>2305</td>
<td>512</td>
<td>423</td>
<td>210</td>
<td>66</td>
<td>46</td>
</tr>
<tr>
<td>ham_8_3_4</td>
<td>16129</td>
<td>256</td>
<td>81274</td>
<td>104</td>
<td>52</td>
<td>21</td>
</tr>
<tr>
<td>ham_9_5_6</td>
<td>53761</td>
<td>512</td>
<td>1984</td>
<td>71</td>
<td>71</td>
<td>102</td>
</tr>
<tr>
<td>theta42</td>
<td>200</td>
<td>5986</td>
<td>4722</td>
<td>51</td>
<td>269</td>
<td>393</td>
</tr>
<tr>
<td>theta6</td>
<td>4375</td>
<td>300</td>
<td>2327</td>
<td>108</td>
<td>308</td>
<td>1221</td>
</tr>
<tr>
<td>theta62</td>
<td>13390</td>
<td>300</td>
<td>68374</td>
<td>196</td>
<td>240</td>
<td>1749</td>
</tr>
<tr>
<td>theta8</td>
<td>7905</td>
<td>400</td>
<td>11947</td>
<td>263</td>
<td>311</td>
<td>1854</td>
</tr>
<tr>
<td>theta82</td>
<td>23872</td>
<td>400</td>
<td>m</td>
<td>650</td>
<td>267</td>
<td>4650</td>
</tr>
<tr>
<td>theta83</td>
<td>39862</td>
<td>400</td>
<td>m</td>
<td>1715</td>
<td>277</td>
<td>7301</td>
</tr>
<tr>
<td>theta10</td>
<td>12470</td>
<td>500</td>
<td>57516</td>
<td>492</td>
<td>278</td>
<td>4636</td>
</tr>
<tr>
<td>theta102</td>
<td>37467</td>
<td>500</td>
<td>m</td>
<td>1948</td>
<td>340</td>
<td>12275</td>
</tr>
<tr>
<td>theta103</td>
<td>62516</td>
<td>500</td>
<td>m</td>
<td>6149</td>
<td>421</td>
<td>17687</td>
</tr>
<tr>
<td>theta104</td>
<td>87845</td>
<td>500</td>
<td>m</td>
<td>8400</td>
<td>269</td>
<td></td>
</tr>
<tr>
<td>theta12</td>
<td>17979</td>
<td>600</td>
<td>t</td>
<td>843</td>
<td>240</td>
<td>8081</td>
</tr>
<tr>
<td>keller4</td>
<td>5101</td>
<td>171</td>
<td>3264</td>
<td>52</td>
<td>432</td>
<td>244</td>
</tr>
<tr>
<td>sanr200-0.7</td>
<td>6033</td>
<td>200</td>
<td>6664</td>
<td>52</td>
<td>278</td>
<td>405</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>problem</th>
<th>n</th>
<th>m</th>
<th>PENSDP (APCG) CPU</th>
<th>RENDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>theta83</td>
<td>39862</td>
<td>400</td>
<td>460</td>
<td>345</td>
</tr>
<tr>
<td>theta103</td>
<td>62516</td>
<td>500</td>
<td>1440</td>
<td>491</td>
</tr>
<tr>
<td>theta123</td>
<td>90020</td>
<td>600</td>
<td>5286</td>
<td>1062</td>
</tr>
</tbody>
</table>