Introducing PENLAB
a MATLAB code for NLP-SDP

Michal Kočvara
School of Mathematics, The University of Birmingham

jointly with

Jan Fiala
Numerical Algorithms Group

Michael Stingl
University of Erlangen-Nürnberg

Toulouse, July, 2013
PENNON collection

PENNON (PENalty methods for NONlinear optimization) a collection of codes for NLP, SDP and BMI

– one algorithm to rule them all –

READY

- PENNLP AMPL, MATLAB, C/Fortran
- PENSUDP MATLAB/YALMIP, SDPA, C/Fortran
- PENBMI MATLAB/YALMIP, C/Fortran

(relatively) NEW

- PENNON (NLP + SDP) extended AMPL, MATLAB, C/Fortran
The problem

Optimization problems with nonlinear objective subject to nonlinear inequality and equality constraints and semidefinite bound constraints:

\[
\min_{x \in \mathbb{R}^n, Y_1 \in \mathbb{S}^{p_1}, \ldots, Y_k \in \mathbb{S}^{p_k}} f(x, Y)
\]

subject to
\[
g_i(x, Y) \leq 0, \quad i = 1, \ldots, m_g
\]
\[
h_i(x, Y) = 0, \quad i = 1, \ldots, m_h
\]
\[
\lambda_i I \preccurlyeq Y_i \preccurlyeq \lambda_i I, \quad i = 1, \ldots, k.
\]

(NLP-SDP)
The algorithm

Based on penalty/barrier functions \( \varphi_g : \mathbb{R} \to \mathbb{R} \) and \( \Phi_P : \mathbb{S}^p \to \mathbb{S}^p \):

\[
g_i(x) \leq 0 \iff p_i \varphi_g(g_i(x)/p_i) \leq 0, \quad i = 1, \ldots, m
\]
\[
Z \preceq 0 \iff \Phi_P(Z) \preceq 0, \quad Z \in \mathbb{S}^p.
\]

Augmented Lagrangian of (NLP-SDP):

\[
F(x, Y, u, U, \bar{U}, p) = f(x, Y) + \sum_{i=1}^{m_g} u_i p_i \varphi_g(g_i(x, Y)/p_i)
\]
\[
+ \sum_{i=1}^{k} \langle U_i, \Phi_P(Y_i - \lambda_i I) \rangle + \sum_{i=1}^{k} \langle \bar{U}_i, \Phi_P(Y_i - \bar{\lambda}_i I) \rangle;
\]

here \( u \in \mathbb{R}^{m_g} \) and \( U_i, \bar{U}_i \) are Lagrange multipliers.
The algorithm


Given $x^1$, $Y^1$, $u^1$, $\underline{U}^1$, $\overline{U}^1$; $p_i^1 > 0$, $i = 1, \ldots, m_g$ and $P > 0$. For $k = 1, 2, \ldots$ repeat till a stopping criterium is reached:

(i) Find $x^{k+1}$ and $Y^{k+1}$ s.t. $\|\nabla_x F(x^{k+1}, Y^{k+1}, u^k, \underline{U}^k, \overline{U}^k, p^k)\| \leq K$

(ii) $u_i^{k+1} = u_i^k \varphi'_g(g_i(x^{k+1})/p_i^k), \quad i = 1, \ldots, m_g$

$\underline{U}_i^{k+1} = D_A \Phi_P((\underline{\lambda}_i I - Y_i); \underline{U}_i^k), \quad i = 1, \ldots, k$

$\overline{U}_i^{k+1} = D_A \Phi_P((Y_i - \overline{\lambda}_i I); \overline{U}_i^k), \quad i = 1, \ldots, k$

(iii) $p_i^{k+1} < p_i^k, \quad i = 1, \ldots, m_g$

$P_{k+1} < P_k$. 
Interfaces

How to enter the data – the functions and their derivatives?

- Matlab interface
- AMPL interface
- c/Fortran interface

Key point: Matrix variables are treated as vectors
What’s new

PENNON being implemented in NAG (The Numerical Algorithms Group) library

The first routines should appear in the NAG Fortran Library, Mark 24 (Autumn 2013)

By-product:
PENLAB — free, open, fully functional version of PENNON coded in MATLAB
PENLAB

PENLAB — free, open, fully functional version of PENNON coded in Matlab

• Open source, all in MATLAB (one MEX function)
• The basic algorithm is identical
• Some data handling routines not (yet?) implemented
• PENLAB runs just as PENNON but is slower

Pre-programmed procedures for

• standard NLP (with AMPL input!)
• linear SDP (reading SDPA input files)
• bilinear SDP (=BMI)
• SDP with polynomial MI (PMI)
• easy to add more (QP, robust QP, SOF, TTO. . . )
The problem

\[ \min_{x \in \mathbb{R}^n, Y_1 \in \mathbb{S}^{p_1}, \ldots, Y_k \in \mathbb{S}^{p_k}} f(x, Y) \]

subject to

\[ g_i(x, Y) \leq 0, \quad i = 1, \ldots, m_g \]
\[ h_i(x, Y) = 0, \quad i = 1, \ldots, m_h \]
\[ A_i(x, Y) \succeq 0, \quad i = 1, \ldots, m_A \]
\[ \lambda_i l \preceq Y_i \preceq \lambda_i l, \quad i = 1, \ldots, k \]

\( A_i(x, Y) \)… nonlinear matrix operators
Solving a problem:

- prepare a structure `penm` containing basic problem data
- `>> prob = penlab(penm)`; MATLAB class containing all data
- `>> solve(prob)`;
- results in class `prob`

The user has to provide MATLAB functions for

- function values
- gradients
- Hessians (for nonlinear functions)

of all $f, g, A$. 
Structure `penm` and f/g/h functions

Example: \[ \min x_1 + x_2 \quad \text{s.t.} \quad x_1^2 + x_2^2 \leq 1, \quad x_1 \geq -0.5 \]

```matlab
penm = []; penm.Nx = 2; penm.lbx = [-0.5 ; -Inf]; penm.NgNLN = 1; penm.ubg = [1]; penm.objfun = @(x,Y) deal(x(1) + x(2)); penm.objgrad = @(x,Y) deal([1 ; 1]); penm.confun = @(x,Y) deal([x(1)^2 + x(2)^2]); penm.congrad = @(x,Y) deal([2*x(1) ; 2*x(2)]); penm.conhess = @(x,Y) deal([2 0 ; 0 2]); % set starting point penm.xinit = [2,1];
```
Toy NLP-SDP example 1

\[
\min_{x \in \mathbb{R}^2} \frac{1}{2}(x_1^2 + x_2^2)
\]

subject to \( B + A_1 x_1 + A_2 x_2 := \begin{pmatrix}
1 & x_1 - 1 & 0 \\
x_1 - 1 & 1 & x_2 \\
0 & x_2 & 1
\end{pmatrix} \succeq 0 \)

D. Noll, 2007
Structure penm and f/g/h functions

\[
B = \begin{bmatrix} 1 & -1 & 0; & -1 & 1 & 0; & 0 & 0 & 1 \end{bmatrix};
\]
\[
A\{1\} = \begin{bmatrix} 0 & 1 & 0; & 1 & 0 & 0; & 0 & 0 & 0 \end{bmatrix};
\]
\[
A\{2\} = \begin{bmatrix} 0 & 0 & 0; & 0 & 0 & 1; & 0 & 1 & 0 \end{bmatrix};
\]

penm = [];
penm.Nx=2;
penm.NALIN=1;
penm.lbA=zeros(1,1);

penm.objfun = @(x,Y) deal(-.5*(x(1)^2+x(2)^2));
penm.objgrad = @(x,Y) deal([-x(1);x(2)]);
penm.objhess = @(x,Y) deal(-eye(2,2));

penm.mconfun=@(x,Y,k)deal(B+A\{1\}*x(1)+A\{2\}*x(2));
penm.mcongrad=@(x,Y,k,i)deal(A\{i\});
Example: nearest correlation matrix

Find a nearest correlation matrix:

\[
\min_X \sum_{i,j=1}^{n} (X_{ij} - H_{ij})^2
\]  

subject to

\[
X_{ii} = 1, \quad i = 1, \ldots, n \\
X \succeq 0
\]  

(1)
Example: nearest correlation matrix

The condition number of the nearest correlation matrix must be bounded by $\kappa$.

Using the transformation of the variable $X$:

$$\tilde{z}\tilde{X} = X$$

The new problem:

$$\min_{z,\tilde{X}} \sum_{i,j=1}^{n} (z\tilde{X}_{ij} - H_{ij})^2$$

subject to

$$z\tilde{X}_{ii} = 1, \quad i = 1, \ldots, n$$

$$I \preceq \tilde{X} \preceq \kappa I$$
function [f,userdata] = objfun(x,Y,userdata)
    YH = svec2(x(1).*Y{1}-userdata.H);
    f = YH(:)’*YH(:);

function [df, userdata]=objgrad(x,Y,userdata)
    YH=svec2(x(1).*Y{1}-userdata.H);
    df(1) = sum(2*svec2(Y{1})).*YH);
    df(2:length(YH)+1) = 2*x(1).*YH;

function [ddf, userdata] = objhess(x,Y,userdata)
    YH=svec2(x(1).*Y{1}-userdata.H);
    yy = svec2(Y{1});
    n = length(yy);  ddf = zeros(n+1,n+1);
    ddf(1,1) = 2*sum(yy.^2);
    ddf(1,2:n+1) = 2.*(x(1).*yy+YH);
    ddf(2:n+1,1) = 2.*(x(1).*yy’+YH’);
    for i=1:n, ddf(i+1,i+1) = 2*x(1).^2; end
NLP with AMPL input

Pre-programmed. All you need to do:

```plaintext
>> penm=nlp_define('datafiles/chain100.nl');
>> prob=penlab(penm);
>> prob.solve();
```
## NLP with AMPL input

<table>
<thead>
<tr>
<th>problem</th>
<th>vars</th>
<th>constr.</th>
<th>constr.</th>
<th>PENNON</th>
<th>PENLAB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>type</td>
<td>sec</td>
<td>iter.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>chain800</td>
<td>3199</td>
<td>2400</td>
<td>=</td>
<td>1</td>
<td>14/23</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>24/56</td>
</tr>
<tr>
<td>pinene400</td>
<td>8000</td>
<td>7995</td>
<td>=</td>
<td>1</td>
<td>7/7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td>17/17</td>
</tr>
<tr>
<td>channel800</td>
<td>6398</td>
<td>6398</td>
<td>=</td>
<td>3</td>
<td>3/3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>3/3</td>
</tr>
<tr>
<td>torsion100</td>
<td>5000</td>
<td>10000</td>
<td>≤</td>
<td>1</td>
<td>17/17</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>17</td>
<td>26/26</td>
</tr>
<tr>
<td>lane_emd10</td>
<td>4811</td>
<td>21</td>
<td>≤</td>
<td>217</td>
<td>30/86</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>64</td>
<td>25/49</td>
</tr>
<tr>
<td>dirichlet10</td>
<td>4491</td>
<td>21</td>
<td>≤</td>
<td>151</td>
<td>33/71</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>73</td>
<td>32/68</td>
</tr>
<tr>
<td>henon10</td>
<td>2701</td>
<td>21</td>
<td>≤</td>
<td>57</td>
<td>49/128</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>63</td>
<td>76/158</td>
</tr>
<tr>
<td>minsurf100</td>
<td>5000</td>
<td>5000</td>
<td>box</td>
<td>1</td>
<td>20/20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>97</td>
<td>203/203</td>
</tr>
<tr>
<td>gasoil400</td>
<td>4001</td>
<td>3998</td>
<td>= &amp; b</td>
<td>3</td>
<td>34/34</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13</td>
<td>59/71</td>
</tr>
<tr>
<td>duct15</td>
<td>2895</td>
<td>8601</td>
<td>= &amp; ≤</td>
<td>6</td>
<td>19/19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td>11/11</td>
</tr>
<tr>
<td>marine400</td>
<td>6415</td>
<td>6392</td>
<td>≤ &amp; b</td>
<td>2</td>
<td>39/39</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>22</td>
<td>35/35</td>
</tr>
<tr>
<td>steering800</td>
<td>3999</td>
<td>3200</td>
<td>≤ &amp; b</td>
<td>1</td>
<td>9/9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>19/40</td>
</tr>
<tr>
<td>methanol400</td>
<td>4802</td>
<td>4797</td>
<td>≤ &amp; b</td>
<td>2</td>
<td>24/24</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16</td>
<td>47/67</td>
</tr>
</tbody>
</table>
Pre-programmed. All you need to do:

```
>> sdpdata=readsdpa('datafiles/arch0.dat-s');
>> penm=sdp_define(sdpdata);
>> prob=penlab(penm);
>> prob.solve();
```
Bilinear matrix inequalities (BMI)

Pre-programmed. All you need to do:

```matlab
>> bmidata=define_my_problem; %matrices A, K, ...
>> penm=bmi_define(bmidata);
>> prob=penlab(penm);
>> prob.solve();
```

\[
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad c^T x \\
\text{s.t.} & \quad A_0^i + \sum_{k=1}^{n} x_k A_k^i + \sum_{k=1}^{n} \sum_{\ell=1}^{n} x_k x_\ell K_{k\ell}^i \succeq 0, \quad i = 1, \ldots, m
\end{align*}
\]
Polynomial matrix inequalities (PMI)

Pre-programmed. All you need to do:

```matlab
>> load datafiles/example_pmidata;
>> penm = pmi_define(pmidata);
>> problem = penlab(penm);
>> problem.solve();
```

\[
\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T H x + c^T x
\]

subject to \( b_{\text{low}} \leq B x \leq b_{\text{up}} \)

\( A_i(x) \succeq 0, \quad i = 1, \ldots, m \)

with

\[
A(x) = \sum_i x_{\kappa(i)} Q_i
\]

where \( \kappa(i) \) is a multi-index with possibly repeated entries and \( x_{\kappa(i)} \) is a product of elements with indices in \( \kappa(i) \).
Other pre-programmed modules

- Nearest correlation matrix
- Truss topology optimization (stability constraints)
- Static output feedback (input from COMPlib, formulated as PMI)
- Robust QP
Availability

PENNON: Free time-limited academic version of the code available

PENLAB: Free open MATLAB version available from NAG
What’s missing?

SOCP (Second-Order Conic Programming) - nonlinear, integrated in PENLAB (and PENNON)

Postdoctoral research position in Birmingham (sponsored by NAG)

- development of NL-SOCP algorithm (compatible with PENNON algorithm)
- implementation in PENLAB and PENNON
- Alain Zemkoho, started April 2013
Sensor network localization
(Euclidean distance matrix completion, Graph realization)

We have (in $\mathbb{R}^2$ (or $\mathbb{R}^d$))

- $n$ points $a_i$, anchors with known location
- $m$ points $x_i$, sensors with unknown location
- $d_{ij}$ known Euclidean distance between “close” points

\[ d_{ij} = \|x_i - x_j\|, \quad (i, j) \in I_x \]
\[ d_{kj} = \|a_k - x_j\|, \quad (k, j) \in I_a \]

Goal: Find the positions of the sensors!

Find $x \in \mathbb{R}^{2 \times m}$ such that

\[ \|x_i - x_j\|^2 = d_{ij}^2, \quad (i, j) \in I_x \]
\[ \|a_k - x_j\|^2 = d_{kj}^2, \quad (k, j) \in I_a \]
Sensor network localization

Example, 4 anchors, 36 sensors
Sensor network localization

Applications

- Wireless sensor network localization
  - habitat monitoring system in the Great Duck Island
  - detecting volcano eruptions
  - industrial control in semiconductor manufacturing plants
  - structural health monitoring
  - military and civilian surveillance
  - moving object tracking
  - asset location

- Molecule conformation

- ...
Sensor network localization

Solve the least-square problem

$$\min_{x_1, \ldots, x_m} \sum_{(i,j) \in \mathcal{I}_x} \left( \|x_i - x_j\|^2 - d_{ij}^2 \right)^2 + \sum_{(i,j) \in \mathcal{I}_a} \left( |a_k - x_j|^2 - \overline{d}_{kj}^2 \right)^2$$

to global minimum. This is an NP-hard problem.
SDP relaxation

(P. Biswas and Y. Ye, ’04)

Let $X = [x_1 \ x_2 \ \ldots \ x_n]$ be a $d \times n$ unknown matrix. Then

$$\|x_i - x_j\|^2 = (e_i - e_j)^T X^T X (e_i - e_j)$$

$$\|a_k - x_j\|^2 = (a_k; -e_j)^T \begin{bmatrix} I_d \\ X^T \end{bmatrix} [I_d \ X](a_k; -e_j)$$

and the problem becomes

$$(e_i - e_j)^T X^T X (e_i - e_j) = d_{ij}^2$$

$$(a_k; -e_j)^T Z(a_k; -e_j) = \bar{d}_{kj}^2$$

$$Z = \begin{pmatrix} I_d \\ X^T \\ X^T X \end{pmatrix}$$

$Z_{1:d,1:d} = I_d, \quad Z \succeq 0, \quad Z \text{ has rank } d$$
SDP relaxation

Now relax

\[ Z_{1:d,1:d} = I_d, \quad Z \succeq 0, \quad \text{\text{Z has rank} } d \]

to

\[ Z_{1:d,1:d} = I_d, \quad Z \succeq 0 \]

Relaxed problem:

\[
\begin{align*}
\min & \quad 0 \\
\text{subject to} & \\
(0; e_i - e_j)^T Z (0; e_i - e_j) &= d_{ij}^2 \quad \forall (i, j) \in \mathcal{I}_x \\
(a_k; -e_j)^T Z (a_k; -e_j) &= \bar{d}_{kj}^2 \quad \forall (k, j) \in \mathcal{I}_a \\
Z_{1:d,1:d} &= I_d \\
Z &\succeq 0
\end{align*}
\]

Full SDP relaxation, FSDP (linear SDP)
Equivalent formulation:

\[
\min \sum_{(i,j) \in I_x} \left( (0; e_i - e_j)^T Z (0; e_i - e_j) - d_{ij}^2 \right)^2 \\
+ \sum_{(k,j) \in I_a} \left( (a_k; -e_j)^T Z (a_k; -e_j) - \overline{d}_{kj}^2 \right)^2
\]

subject to

\[
Z_{1:d,1:d} = I_d \\
Z \succeq 0
\]

Full SDP relaxation, FSDP (nonlinear SDP)
SDP relaxation

For larger problems, FSDP is not solvable numerically:

- many variables (number of sensors)
- large and full matrix constraint (although low-rank)

Can we exploit sparsity of $\mathcal{I}_x$ and $\mathcal{I}_a$ at the relaxation modelling level?

Recently several approaches:

- Wolkowicz
- Toh
- Kojima
- Su

Example, 16 anchors, 455 sensors
Example, 16 anchors, 455 sensors
Example, 16 anchors, 455 sensors
Example, 16 anchors, 455 sensors

<table>
<thead>
<tr>
<th>problem</th>
<th>rmsd</th>
<th>out-3</th>
<th>out-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-linear</td>
<td>0.0191</td>
<td>307</td>
<td>147</td>
</tr>
<tr>
<td>E-quadratic</td>
<td>0.0105</td>
<td>156</td>
<td>85</td>
</tr>
</tbody>
</table>

SDP: 6714 variables, 5349 (4 × 4) LMIs
Solution refinement

Take the SDP solution as initial approximation for the original unconstrained nonconvex problem. Solve both by PENNON.
Example, 16 anchors, 455 sensors

<table>
<thead>
<tr>
<th>problem</th>
<th>rmsd</th>
<th>out-3</th>
<th>out-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-linear</td>
<td>0.0191</td>
<td>307</td>
<td>147</td>
</tr>
<tr>
<td>orig from lin</td>
<td>0.0083</td>
<td>10</td>
<td>7</td>
</tr>
</tbody>
</table>
Example, 16 anchors, 455 sensors

<table>
<thead>
<tr>
<th>problem</th>
<th>rmsd</th>
<th>out-3</th>
<th>out-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-linear</td>
<td>0.0191</td>
<td>307</td>
<td>147</td>
</tr>
<tr>
<td>E-quadratic</td>
<td>0.0105</td>
<td>156</td>
<td>85</td>
</tr>
<tr>
<td>orig from lin</td>
<td>0.0083</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>orig from qua</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Happy Birthday, PhT