Recent progress in PENNON

Michal Kočvara and Michael Stingl

UTIA AV ČR Prague and University of Erlangen-Nürnberg

Outline

1. PENNON — what is it ?
2. basic algorithm
3. what's new
   - PCG
   - modified Newton + TR
4. what's missing (initialization!)
5. results
   - NLP (COPS3)
   - SDP (TOH collection)
   - BMI

PENNON collection

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

– one algorithm to rule them all –

PENNON collection

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

– one algorithm to rule them all –

READY

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

PENNON collection

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

– one algorithm to rule them all –

READY

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

TO COME

- PENPMI polynomial matrix inequalities
- PENNON NLP + polynomial matrix inequalities

The NLP-SDP problem

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \ & f(x) \\ \text{s.t.} \quad & g_i(x) \le 0, \qquad i = 1,\ldots,m_g \\ & \mathcal{A}(x) \preccurlyeq 0 \end{aligned} \qquad \text{(NLP-SDP)}$$

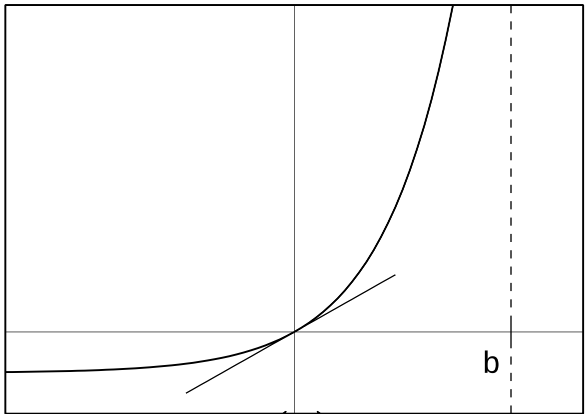
$f, \ g_i : \mathbb{R}^n \to \mathbb{R}$ smooth

$\mathcal{A} : \mathbb{R}^n \to \mathbb{S}^{m_A}$ generally nonconvex

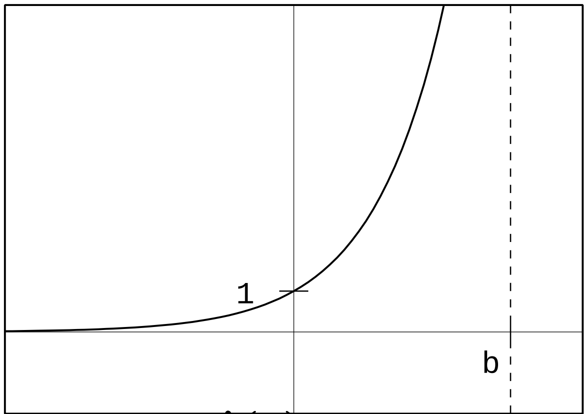
Based on the PBM method:

R. Polyak ’87
Ben-Tal, Zibulevsky ’92, ’97
Breitfeld, Shanno ’94

“Penalty/barrier function:”



$\varphi(t)$



$\varphi'(t)$

With $p_i > 0$ for $i \in \{1, \ldots, m\}$, we have

$$\boxed{g_i(x) \le 0 \iff p_i\varphi(g_i(x)/p_i) \le 0, \quad i = 1, \ldots, m_g}$$

and

$$\boxed{\mathcal{A}(x) \preccurlyeq 0 \iff \Phi_P(\mathcal{A}(x)) \preccurlyeq 0\,.}$$

PENNON algorithm

With $p_i > 0$ for $i \in \{1, \ldots, m\}$, we have

$$\boxed{g_i(x) \le 0 \iff p_i \varphi(g_i(x)/p_i) \le 0, \quad i = 1, \ldots, m_g}$$

and

$$\boxed{\mathcal{A}(x) \preccurlyeq 0 \iff \Phi_P(\mathcal{A}(x)) \preccurlyeq 0\,.}$$

The corresponding *augmented Lagrangian*:

$$\boxed{F(x, u, U, p, P) = f(x) + \sum_{i=1}^{m_g} u_i p_i \varphi_g(g_i(x)/p_i) + \langle U, \Phi_P\left(\mathcal{A}(x)\right)\rangle_{\mathbb{S}_{m_A}}}$$

PENNON algorithm

Augmented Lagrangian:

$$F(x,u,U,p,P) = f(x) + \sum_{i=1}^{m_g} u_i p_i \varphi_g(g_i(x)/p_i) + \langle U, \Phi_P\left(\mathcal{A}(x)\right)\rangle_{\mathbb{S}_{m_A}}$$

PENNON algorithm:

$$\begin{aligned}
&(i) && \text{Find } x^{k+1} = \text{satisfying } \|\nabla_x F(x,u^k,U^k,p^k,P^k)\| \le \varepsilon^k \\
&(ii) && u_i^{k+1} = u_i^k \varphi_g'(g_i(x^{k+1})/p_i^k), \quad i = 1,\ldots,m_g \\
& && U^{k+1} = D_{\mathcal{A}}\Phi_p(\mathcal{A}(x); U^k) \\
&(iii) && p_i^{k+1} < p_i^k, \; i = 1,\ldots,m_g \\
& && P^{k+1} < P^k
\end{aligned}$$

Augmented Lagrangian:

$$F(x,u,U,p,P) = f(x) + \sum_{i=1}^{m_g} u_i p_i \varphi_g(g_i(x)/p_i) + \langle U, \Phi_P(\mathcal{A}(x)) \rangle_{\mathbb{S}_{m_A}}$$

PENNON algorithm:

$$\begin{aligned}
&(i) && \text{Find } x^{k+1} = \text{satisfying } \|\nabla_x F(x,u^k,U^k,p^k,P^k)\| \le \varepsilon^k \\
&(ii) && u_i^{k+1} = u_i^k \varphi_g'(g_i(x^{k+1})/p_i^k), \quad i = 1,\ldots,m_g \\
& && U^{k+1} = D_{\mathcal{A}} \Phi_p(\mathcal{A}(x); U^k) \\
&(iii) && p_i^{k+1} < p_i^k, \; i = 1,\ldots,m_g \\
& && P^{k+1} < P^k
\end{aligned}$$

Step (i): (modified) Newton's method (or TR)

Experience so far. . .

(. . . about a year ago)

NLP

- very good for convex problems
- mixed experience with nonconvex problems (many fails)

SDP (linear)

- one of the fastest codes, in average
- troubles with large-scale problems (typical for IP codes)

What's new:

- implementation of PCG for the Newton system
- modified Newton + TR
- many small modifications to increase robustness

When PCG helps (NLP) ?

NLP: sparse problems with dense columns in ∇g

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

Hessian:

$$H(x) = \ldots + \nabla g_i(x) \varphi'' \nabla g_i(x)^T + \varphi(x)' \nabla^2 g_i(x) + \ldots$$

Solve $H(x)d = -\nabla g(x)$ by preconditioned CG method.
At each iteration need the product Hz.

When PCG helps (NLP) ?

NLP: sparse problems with dense columns in ∇g

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

Hessian:

$$H(x) = \ldots + \nabla g_i(x) \varphi'' \nabla g_i(x)^T + \varphi(x)' \nabla^2 g_i(x) + \ldots$$

Solve $H(x)d = -\nabla g(x)$ by preconditioned CG method.
At each iteration need the product Hz.

PENNON: assemble only "sparse part" of H, the rest by direct computation:

$$Hz = H_{\mathrm{sp}} z + \sum_{\mathrm{dense}} \gamma_i \gamma_i^T z$$

When PCG helps (NLP) ?

NLP: sparse problems with dense columns in ∇g

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

Hessian:

$$H(x) = \ldots + \nabla g_i(x)\varphi''\nabla g_i(x)^T + \varphi(x)'\nabla^2 g_i(x) + \ldots$$

Solve $H(x)d = -\nabla g(x)$ by preconditioned CG method.
At each iteration need the product Hz.

PENNON: assemble only "sparse part" of H, the rest by direct computation:

$$Hz = H_{\mathrm{sp}}z + \sum_{\mathrm{dense}} \gamma_i \gamma_i^T z$$

Consequence: preconditioner can only use Hessian-vector products

When PCG helps (NLP) ?

NLP: sparse problems with dense columns in ∇g

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

Hessian:

$$H(x) = \ldots + \nabla g_i(x)\varphi''\nabla g_i(x)^T + \varphi(x)'\nabla^2 g_i(x) + \ldots$$

Solve $H(x)d = -\nabla g(x)$ by preconditioned CG method.
At each iteration need the product Hz.

PENNON: assemble only “sparse part” of H, the rest by direct computation:

$$Hz = H_{\mathrm{sp}}z + \sum_{\mathrm{dense}} \gamma_i \gamma_i^T z$$

Example: `lane-emden40` from COPS3 (19241 vars, 81 nl $\leq$)

Cholesky: 1600 MB, 26 min

PCG: 300 MB, 1 min 40 sec

PENNON

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)

Preconditioners

Monteiro, O'Neil, Nemirovski (2004): Adaptive Ellipsoid Preconditioner

"Improves the CG performance on extremely ill-conditioned systems."

preconditioner:

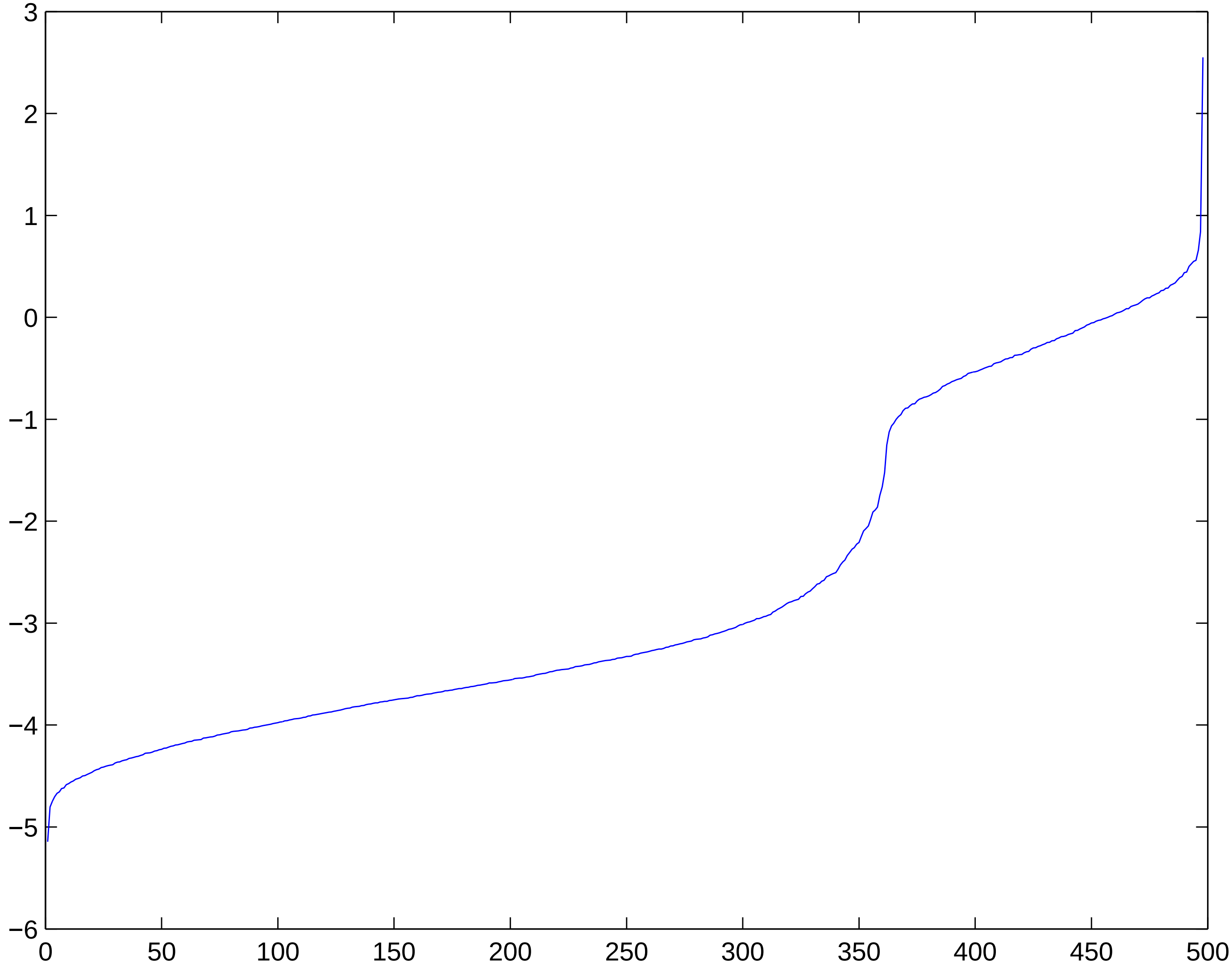
$$M = C_k C_k^T, \quad C_{k+1} \leftarrow \alpha C_k + \beta C_k p_k p_k^T, \quad C_1 = \gamma I$$

$\alpha, \beta, p_k \ldots$ by matrix-vector products

VERY preliminary results (MATLAB implementation)

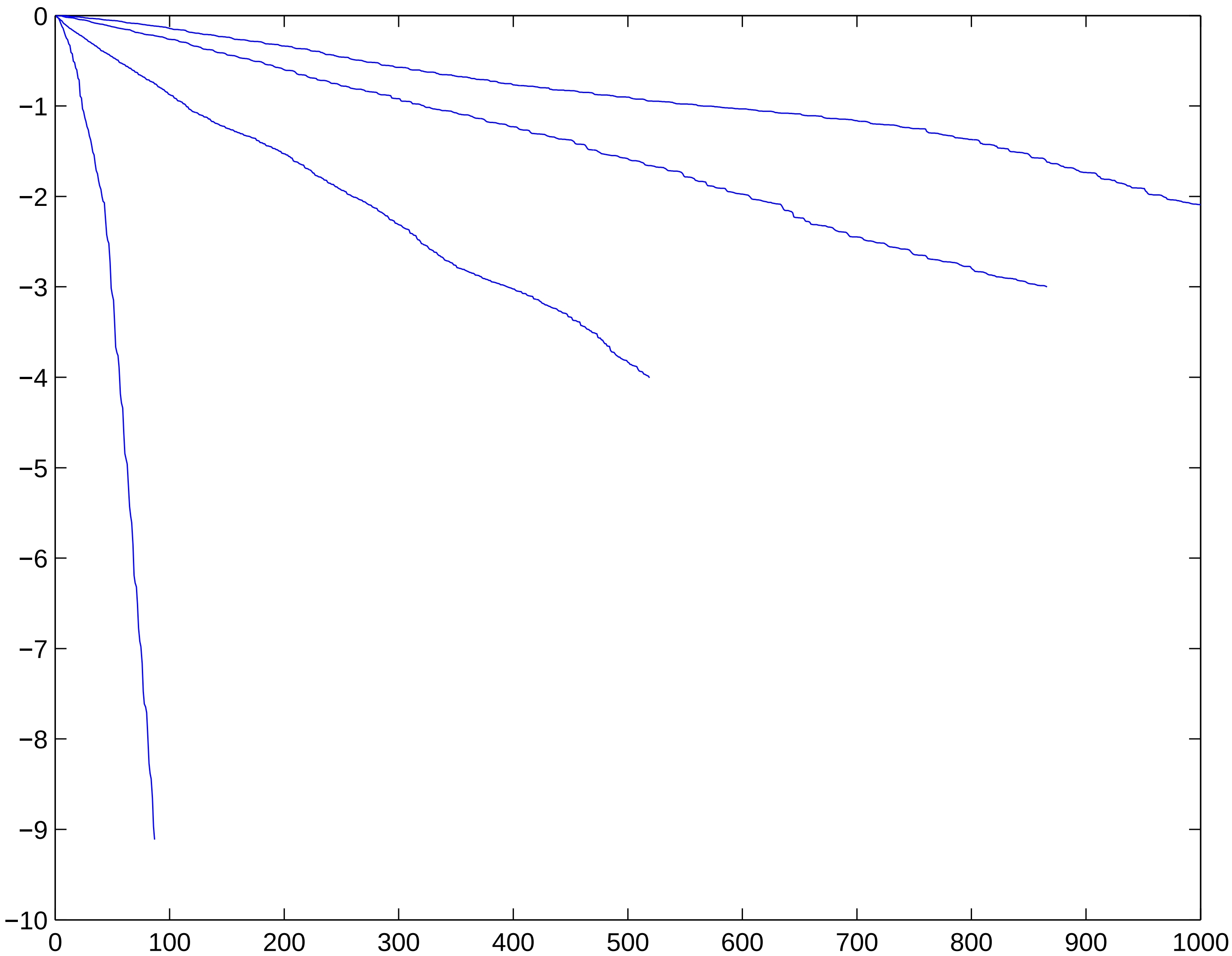
Preconditioners — example

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



Preconditioners — example

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



CG

CG-diag

CG-SGS

CG-MON

.

Modified Newton + Trust-Region

1. Given a current iterate (x, U, p), compute the gradient g and Hessian H of F at x.
2. Try to factorize H by Cholesky decomposition. If H is factorizable, set $\widehat{H} = H$ and go to Step 4.
3. Compute $\beta \in [-\lambda_{\min}, -2\lambda_{\min}]$, where $\lambda_{\min}$ is the minimal eigenvalue of H and set
$$\widehat{H} = H + \beta I.$$
4. Compute the search direction
$$d = -\widehat{H}^{-1} g.$$
5. Perform line-search in direction d. Denote the step-length by s.
6. Set
$$x_{\text{new}} = x + sd.$$

Modified Newton + Trust-Region

Choose initial $\widehat{\beta} > 0$. Perform Cholesky factorization of $H + \widehat{\beta}I$. If the factorization fails, go to Step (i); otherwise go to Step (iii).

(i) Set $\widehat{\beta} \leftarrow 2\widehat{\beta}$.

(ii) Perform Cholesky factorization of $H + \widehat{\beta}I$. If the factorization fails, go to Step (i); otherwise stop and return $\beta = \widehat{\beta}$.

(iii) Set $\widehat{\beta} \leftarrow \widehat{\beta}/2$.

(iv) Perform Cholesky factorization of $H + \widehat{\beta}I$. If the factorization fails stop and return $\beta = 2\widehat{\beta}$; otherwise go to Step (iii).

Modified Newton + Trust-Region

Numerical experience:
linesearch often fails when close to the solution $\longrightarrow$ low accuracy

Remedy:
When stopping crit. low $(\mathbf{10^{-4} - 10^{-6}})$ switch to Trust-Region

In this way we obtain solutions with very high accuracy.

Modified Newton + Trust-Region

Example `minsurf100` from COPS3 (5000 vars., box constr.)

it	obj	<U,G(x)>	||dF||	feas	pmin	Nwt	Fact
0	2.32e+00	1.2e+00	4.5e-02	2.8e-01	1.0e+00	0	0
1	2.63e+00	9.6e+02	3.8e-03	8.5e-02	1.0e+00	3	3
2	2.42e+00	1.5e+02	3.2e-03	1.7e-01	1.0e-01	2	2
....							
14	2.50e+00	1.2e-03	2.8e-05	1.7e-09	1.0e-06	1	1
15	2.50e+00	1.2e-03	2.5e-07	9.1e-11	1.0e-06	1	0
16	2.50e+00	1.2e-04	1.6e-05	7.8e-11	1.0e-07	1	0
17	2.50e+00	1.2e-05	7.2e-07	4.7e-13	1.0e-08	1	0
18	2.50e+00	1.2e-05	2.9e-08	5.3e-14	1.0e-08	1	0
19	2.50e+00	1.2e-05	2.8e-09	8.7e-14	1.0e-08	2	0
20	2.50e+00	1.2e-05	1.5e-09	1.8e-13	1.0e-08	4	0
21	2.50e+00	1.2e-05	4.2e-10	2.2e-14	1.0e-08	3	0

Test results: COPS 3.0

COPS 3.0 (E. Dolan, J. Moré, T. Munson, Argonne Nat. Lab.)

selection of difficult nonlinearly constrained optimization problems from applications in

- optimal design
- fluid dynamics
- parameter estimation
- optimal control
- etc.

Report compares FILTER, KNITRO, LOQO, MINOS, SNOPT.

Test results: COPS 3.0

COPS 3.0 (E. Dolan, J. Moré, T. Munson, Argonne Nat. Lab.)

selection of difficult nonlinearly constrained optimization problems from applications in

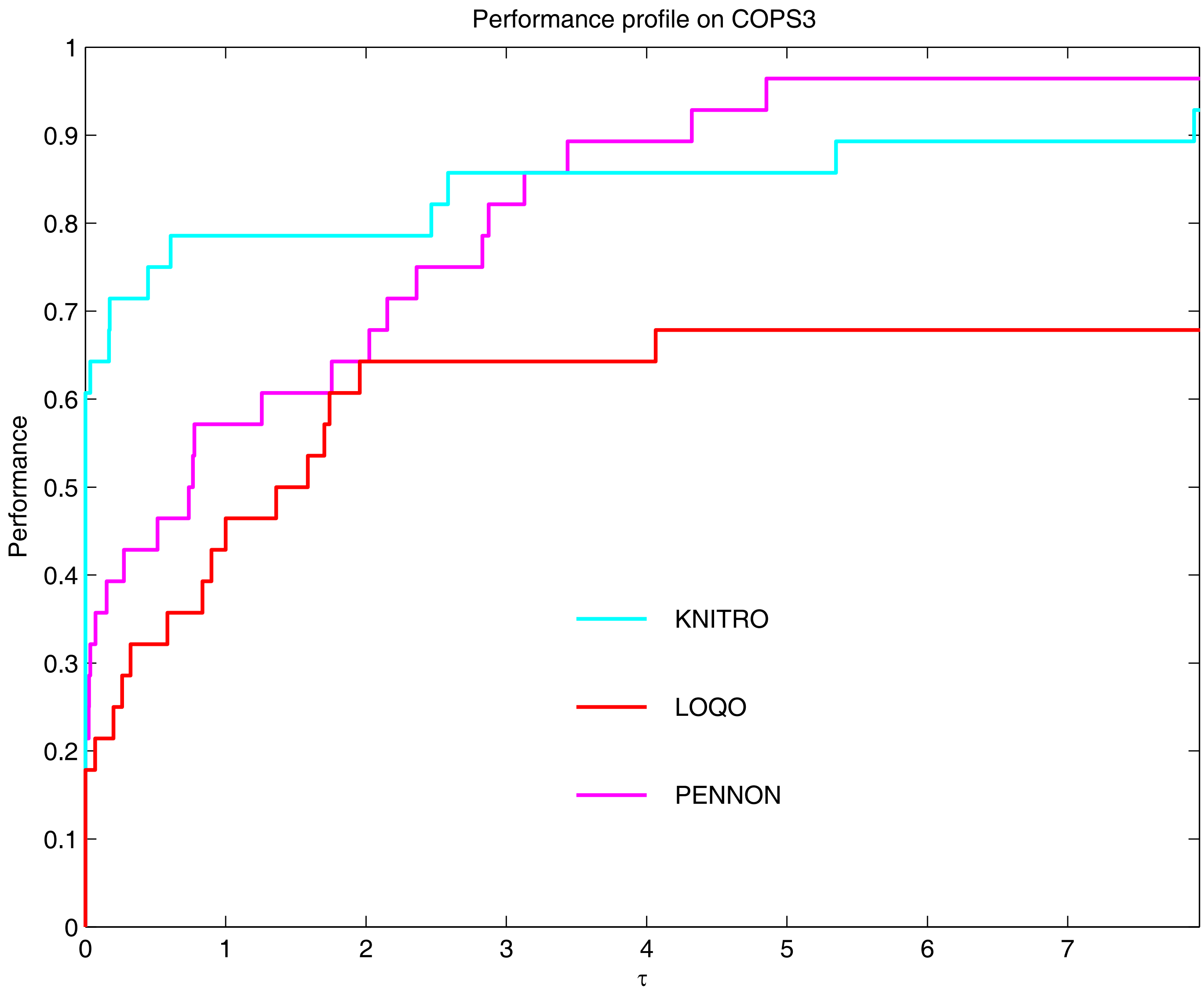
- optimal design
- fluid dynamics
- parameter estimation
- optimal control
- etc.

Report compares FILTER, KNITRO, LOQO, MINOS, SNOPT.

PENNON 1.0
can solve about 50% of the problems for default parameter setting
parameter tuning $\longrightarrow$ 80–90% of the problems (cf. Polyak-Griva)

PENNON 2.0
default parameters $\longrightarrow$ all problems but one (solved with different pars.)

Test results: COPS 3.0



Test results: COPS 3.0

problem	FILTER	KNITRO	LOQO	SNOPT	PENNON
polygon200	‡	59.5	‡	‡	44
elec200	‡	21	‡	103	36
chain800	125	0.6	1.7	126	12
pinene400	1296	2.4	4.8	96	26
channel800	343	2.7	8.8	58	12
robot800	182	6.3	‡	102	184
rocket1600	264	310	7.6	143	70
gasoil400	69	18	3	25	22
torsion100	545	‡	2	91	2
bearing100	1058	478	2	68	2
minsurf100	424	‡	‡	‡	4
tri_turtle	137	4	‡	‡	4
duct15	199	16	28	1768	16
lane_emd40	‡	82	1372	‡	196
dirichlet40	‡	329	1276	‡	396
glider400	‡	578	‡	‡	‡(46)

When PCG helps (SDP) ?

Linear SDP, dense Hessian $\qquad A = \sum_{i=1}^{n} A_i, \; A_i \in \mathbb{R}^{m \times m}$

Complexity of Hessian evaluation

- $O(m_A^3 n + m_A^2 n^2)$ for dense matrices
- $O(m_A^2 n + K^2 n^2)$ for sparse matrices
  ($K \ldots$ max. number of nonzeros in $A_i, \; i = 1, \ldots, n$)

Complexity of Cholesky algorithm - linear SDP

- $O(n^3)$ $\qquad$ (…from PCG we expect $O(n^2)$)

Problems with large n and small m:

CG better than Cholesky (expected)

Hessian free methods

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 $h = (1 + \|x_k\|_2 \sqrt{\varepsilon})$

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

Stopping criterium for CG/QMR ???

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

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.
$\longrightarrow$ often very low (average) number of CG iterations

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

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

problem	n	m	PENSDP	PENSDP (APCG)		SDPLR	
			CPU	CPU	CG	CPU	iter
ham_7_5_6	1793	128	126	4	52	1	113
ham_9_8	2305	512	423	210	66	46	222
ham_8_3_4	16129	256	81274	104	52	21	195
ham_9_5_6	53761	512		1984	71	71	102
theta42	200	5986	4722	51	269	393	11548
theta6	4375	300	2327	108	308	1221	20781
theta62	13390	300	68374	196	240	1749	16784
theta8	7905	400	11947	263	311	1854	15257
theta82	23872	400	m	650	267	4650	20653
theta83	39862	400	m	1715	277	7301	23017
theta10	12470	500	57516	492	278	4636	18814
theta102	37467	500	m	1948	340	12275	29083
theta103	62516	500	m	6149	421	17687	29483
theta104	87845	500	m	8400	269		
theta12	17979	600	t	843	240	8081	21338
keller4	5101	171	3264	52	432	244	8586
sanr200-0.7	6033	200	6664	52	278	405	12139

problem	n	m	PENSDP	PENSDP (APCG)		SDPLR	
			CPU	CPU	CG	CPU	iter
ham_7_5_6	1793	128	126	4	52	1	113
ham_9_8	2305	512	423	210	66	46	222
ham_8_3_4	16129	256	81274	104	52	21	195
ham_9_5_6	53761	512		1984	71	71	102
theta42	200	5986	4722	51	269	393	11548
theta6	4375	300	2327	108	308	1221	20781
theta62	13390	300	68374	196	240	1749	16784
theta8	7905	400	11947	263	311	1854	15257
theta82	23872	400	m	650	267	4650	20653
theta83	39862	400	m	1715	277	7301	23017
theta10	12470	500	57516	492	278	4636	18814
theta102	37467	500	m	1948	340	12275	29083
theta103	62516	500	m	6149	421	17687	29483
theta104	87845	500	m	8400	269		
theta12	17979	600	t	843	240	8081	21338
keller4	5101	171	3264	52	432	244	8586
sanr200-0.7	6033	200	6664	52	278	405	12139

problem	n	m	PENSDP (APCG)		RENDL
theta83	39862	400	460	345	440
theta103	62516	500	1440	491	850
theta123	90020	600	5286	1062	1530