,Numerical Linear Algebra

Preconditioning techniques

Maya Neytcheva, TDB, February-March 2021

Preconditioning techniques to accelerate the convergence of the iterative solution methods

Note

Many issues related to iterative solution of linear systems of equations are contradictory:

- numerical efficiency vs computational efficiency
- numerical efficiency vs parallelization

Scalability of sparse direct solvers

Example 1: Multiphace flow simulation (2D)

	Block preconditioner			Direct solver (MUMPS)		
DOF	$N1/N2$	$time(s)$	MB	$N1$	$time(s)$	MB
131 072	4/10	16.98	185	3	7.2	352
528 392	4/10	72.61	646	3	53.4	1 409
1 176 578	4/10	170	1 429	3	193.75	3 126
2 097 152	4/10	306.05	2 587	*Out of memory*		

Multiface flow: Direct vs iterative: run time and memory consumption

$$72.61/16.98 = 4.28 \qquad 53.4/7.2 = 7.4$$
$$170/72.61 = 2.34 \qquad 193.75/53.4 = 3.62$$
$$306.05/170 = 1.8 \qquad ?$$

Scalability of sparse direct solvers, cont.

Example 2: Linear elasticity, 2D

	Assembly time (s)		Solution time (s)		
N	Abaqus	Iterative	Abaqus	Iterative	
				time	iterations
		2D			
6043	1	0.2178	1.098	1.02 (0.4863)	13 (1,1)
23603	3.326	0.8857	4.718	4.225 (1.995)	12 (1,1)
93283	13.02	3.978	18.05	19.38 (9.813)	11 (2,1)
370883	50.54	17.71	72.98	89.34 (49.43)	11 (2,1)
1479043	269.1	77.7	317.5	431.8 (257.6)	12 (2,1)

4.2969	4.1422
3.8258	4.5870
4.0432	4.6099

Scalability of sparse direct solvers, cont.

Example 2: Linear elasticity, 3D

	Assembly time (s)		Solution time (s)		
N	Abaqus	Iterative	Abaqus	Iterative	
				time	iterations
		2D			
6043	1	0.2178	1.098	1.02 (0.4863)	13 (1,1)
23603	3.326	0.8857	4.718	4.225 (1.995)	12 (1,1)
93283	13.02	3.978	18.05	19.38 (9.813)	11 (2,1)
370883	50.54	17.71	72.98	89.34 (49.43)	11 (2,1)
1479043	269.1	77.7	317.5	431.8 (257.6)	12 (2,1)
		3D			
12512	1.525	1.899	3.049	8.009 (3.465)	12 (2,1)
89700	14.09	8.756	43.29	63.34 (33.08)	13 (2,1)
678116	110.3	65.8	1347	749.3 (506.8)	15 (4,1)

Hybrid solvers!

Example, what are we after?

$\varepsilon \downarrow h \rightarrow$	$2^{-4}(289)$	$2^{-5}(1089)$	$2^{-6}(4225)$	$2^{-7}(16641)$	$2^{-8}(66049)$
			$\beta = 10^{-6}$		
10^{-2}	**3(8)**	**4(11)**	**3(13)**	**4(14)**	**4(11)**
	0.03	0.1	0.212	1.185	4.442
10^{-4}	**3(10)**	**4(17)**	**5(16)**	**5(15)**	**6(11)**
	0.033	0.128	0.436	1.669	9.431
10^{-6}	**3(12)**	**4(23)**	**4(26)**	**4(29)**	**7(20)**
	0.037	0.154	0.721	3.745	23.053
10^{-8}	**3(14)**	**4(28)**	**4(33)**	**4(38)**	**6(26)***
	0.04	0.18	0.607	2.729	28.763
10^{-10}	**3(17)**	**4(33)**	**4(39)**	**4(46)**	**6(31)***
	0.043	0.208	0.723	3.328	16.242

Table 1: Performance of a nonlinear solver

Example, what are we after?

$\varepsilon \downarrow h \rightarrow$	$2^{-4}(289)$	$2^{-5}(1089)$	$2^{-6}(4225)$	$2^{-7}(16641)$	$2^{-8}(66049)$
			$\beta = 10^{-10}$		
10^{-2}	**3(8)**	**4(11)**	**3(13)**	**4(14)**	**4(11)**
	0.03	0.1	0.212	1.185	4.442
10^{-4}	**3(10)**	**4(17)**	**5(16)**	**5(15)**	**6(11)***
	0.033	0.128	0.436	1.669	9.431
10^{-6}	**3(12)**	**4(23)**	**4(26)**	**4(29)**	**7(20)***
	0.037	0.154	0.721	3.745	23.053
10^{-8}	**3(14)**	**4(28)**	**4(33)**	**4(38)**	**6(26)**
	0.04	0.18	0.607	2.729	28.763
10^{-10}	**3(17)**	**4(33)**	**4(39)**	**4(46)**	**6(31)***
	0.043	0.208	0.723	3.328	16.242

Table 2: Performance of a nonlinear solver

Preconditioning techniques - the task to combine numerical and computational efficiency

Preconditioners

Requirements:

- Numerical efficiency
  - The condition number $\varkappa(C^{-1}A)$ should be as small as possible and independent of problem, discretization and method parameters.
    Wishes: $\varkappa(C^{-1}A) = O(1)$
    Eigenvalues clusteres in small intervals on the real axes or in a few tight clusters, well separated from the origin.
- Computational efficiency
  - The construction of C should be computationally cheap
  - The solution of systems with C should be much cheaper (easier) than with A
- Papallel efficiency
  - Both the construction and the solution with the preconditioner should be parallelizable

Clearly the goals are contradicting.

Types of preconditioners

- Left preconditioning $A\mathbf{x} = \mathbf{b} \Longrightarrow C^{-1}A\mathbf{x} = C^{-1}\mathbf{b}$
  $eig(XY) = eig(YX)$ up to some zero eigenvalues.
- Right Preconditioning $A\mathbf{x} = \mathbf{b} \Longrightarrow AC^{-1}\mathbf{y} = \mathbf{b}, \mathbf{x} = C^{-1}\mathbf{y}$
- Symmetric preconditioning

$$A\mathbf{x} = \mathbf{b} \Longrightarrow C_1^{-1}AC_2^{-1}\mathbf{y} = C_1^{-1}\mathbf{b}, \mathbf{x} = C_2^{-1}\mathbf{y}$$

- Approximate inverse (multiplicative preconditioning)
  $C \approx A^{-1}$. Then no solution but only multiplication with C occur.
- Implicitly defined preconditioners $[C^{-1}]A\mathbf{x} = [C^{-1}]\mathbf{b}$
- Variable (nonlinear) preconditioners C changes from one iteration to another or a number of times through the iterative process.
  Flexible GMRES, GCG, GCR.
- Inner-outer iterations (inner stopping tolerance)

Types of preconditioners

- 'Given-the-matrix' - only the matrix is given and the origin of the problem is not known or is not to be used during the solution process.
  Very general, thus, expected to be less numerically efficient.
- 'Given-the-problem' - we are in a position to use knowledge about the mesh, discretization technique, the origin of the problem (the PDE, for instance)

$$\begin{tikzcd}
\mathcal{L} \arrow[r, Rightarrow, "discretize"] \arrow[d, Rightarrow, "(approx)"'] & A \arrow[d, Rightarrow, "(approx)"] \\
\mathcal{C} \arrow[r, Rightarrow, "discretize"] & C
\end{tikzcd}$$

Examples:
Linear elasticity (Korn's inequality)
Navier-Stokes

Include preconditioning: $C^{-1}A\mathrm{x} = C^{-1}\mathrm{b}$

Unpreconditioned CG

```
x = x0
r = A*x-b
delta0 = (r,r)
g = -r
Repeat: h = A*g
        tau = delta0/(g,h)
        x = x + tau*g
        r = r + tau*h
        delta1 = (r,r)
        if delta1 <= eps, stop
        beta = delta1/delta0
        g = -r + beta*g
```

Preconditioned CG

```
x = x0
r = A*x-b; C*h = r
delta0 = (r,h)
g = -h
Repeat: h = A*g
        tau = delta0/(g,h)
        x = x + tau*g
        r = r + tau*h; C*h = r
        delta1 = (r,h)
        if delta1 <= eps, stop
        beta = delta1/delta0
        g = -h + beta*g
```

Start with LU factorization $A(m,n)$

```
for $k = 1, 2 \cdots m-1$
  $d = 1/a_{kk}^{(k)}$
  for $i = k+1, \cdots m$
    $\ell_{ik}^{(k)} = -a_{ik}^{(k)}\, d$
    for $j = k+1, \cdots n$
      $a_{ij}^{(k+1)} = a_{ij}^{(k)} + \ell_{ik} a_{kj}^{(k)}$
    end
  end
end
```

The operational count for the LU factorization can be obtained by integrating the loops:

$$Flops_{LU} = \int_1^{m-1}\int_k^m\int_k^n d_j d_i d_k \approx n^3/3 \ (m = n)$$

Block LU factorization $A(m,n)$

```
for $k = 1, 2 \cdots m-1$
  $D = (A_{kk}^{(k)})^{-1}$
  for $i = k+1, \cdots m$
    $L_{ik}^{(k)} = -A_{ik}^{(k)} D$
    for $j = k+1, \cdots n$
      $A_{ij}^{(k+1)} = A_{ij}^{(k)} + L_{ik} A_{kj}^{(k)}$
    end
  end
end
```

The block version offers possibility to use BLAS3.

Incomplete LU factorization $A(n,n)$

```
for $k = 1, 2 \cdots m-1$
  $d = 1/a_{kk}^{(k)}$
  for $i = k+1, \cdots m$
    $\ell_{ik}^{(k)} = -a_{ik}^{(k)}\, d$
    for $j = k+1, \cdots n$
      $a_{ij}^{(k+1)} = a_{ij}^{(k)} + \ell_{ik} a_{kj}^{(k)}$
      when some condition holds true, drop $a_{ij}^{(k+1)}$
    end
  end
end
```

Most often used conditions:
- $a_{ij}^{(k)}$ too small compared to some (relative) value
- $a_{ij}^{(k)}$ does not belong to a chosen sparsity pattern
Loss of information.

Incomplete LU factorization $A(n,n)$

How much ILU can improve the convergence of an iterative process?

Example: 2D, Discrete Laplace operator L(spd).

$\lambda_{min}(A) = h^2$, $\lambda_{max}(A) = O(1)$, $\varkappa(L) = \lambda_{max}(A)/\lambda_{min}(A) = O(h^{-2})$.

The convergence estimate for the CG method is:

$$\|\mathbf{e^k}\|_L \le 2\left[\frac{\varkappa(L)+1}{\varkappa(L)-1}\right]^k \|\mathbf{e^0}\|_L$$

$$k > \frac{1}{2}\sqrt{\varkappa}\,ln(\frac{2}{\varepsilon})$$

$\Rightarrow k = O(h^{-1})$.

Let $C = LL^T$, where L is some ILU-obtained approximation of the exact Cholesky factor of L. Then

$$\varkappa(C^{-1}L) = O(h^{-1}).$$

Thus, only the constant is improved!

Incomplete Factorization Preconditioners

- pointwise and block ILU
- ILU and IC (when A is spd)
- MILU and MIC (Modified incomplete LU factorization)
  Instead of dropping $a_{ij}^{(k+1)}$, let $a_{k+1,k+1}^{(k+1)} = a_{k+1,k+1}^{(k+1)} + a_{ij}^{(k+1)}$

Reference: Ivar Gustafsson, *A class of first order factorization methods*, BIT, 18 (1978), pp. 142-156.

The information is not fully wasted; preserving positive definiteness.

Incomplete Factorization Preconditioners, cont.

- RILU (Relaxed ILU)
  Instead of dropping $a_{ij}^{(k+1)}$, let $a_{k+1,k+1}^{(k+1)} = a_{k+1,k+1}^{(k+1)} + \omega a_{ij}^{(k+1)}$
  Implemented in IFPACK (part of Trilinos, Sandia Nat. Lab.)
  User manual: *For most situations, RelaxValue should be set to zero.*
  - For certain kinds of problems, e.g., reservoir modeling, there is a conservation principle involved such that any operator should obey a **zero row-sum property**. *MILU was designed for these cases and you should set the RelaxValue to 1.*
  - For other situations, setting RelaxValue to some nonzero value **may improve the stability** *of factorization, and can be used if the computed ILU factors are poorly conditioned.*

MILU: M-matrix, nearest M-matrix to a symmetric positive matrix

ILU preconditioners, cont.

- ILU, based on apriori chosen spartity pattern (ILU(0)): the nonzero pattern of L and U coincides with that of the lower/upper part of A
- ILUT: Threshhold-based ILU
  Reference: Yousef Saad, A dual threshold incomplete LU factorization, Numerical Linear Algebra with Applications, 1 (1994), 387–402.
- ILUTP: Threshhold&Permutation-based ILU, available in Matlab.

ILU preconditioners, cont.

ILU(p)

```
For all nonzero elements $a_{ij}$ define $u_{ij} = a_{ij}, lev(u_{ij}) = 0$
For $i = 2, \cdots, n$ do
   For $k = 1, \cdots, k-1$ and if $u_{ij} \neq 0$ do
      Compute $l_{ik} = u_{ik}/u_{kk}$, set $lev(l_{ik}) = lev(u_{ik})$
      Compute $u_{i*} = u_{i*} - l_{ik}u_{k*}$
      Update the levels of $u_{i*}$ as follows
         $level(f_{ij}) = level(l_{ij}) + level(u_{kj}) + 1$
      Replace any element in row $i$ with $lev(u_{ij}) > p$ by zero.
   EndFor
EndFor
```

ILU preconditioners, cont.

ILUT (Generic ILU with threshhold), Y. Saad, 1994

```
0  row(1:n) = 0
1  do i=2:n
2     row(1:n) = a(i,1:n) % sparse copy
3     for k=1:i-1 and where row(k) is nonzero, do
4        row(k)=row(k)/a(kk)
5        apply a dropping rule to row(k)
6        if row(k)$\neq$0
7           row(k+1,n)=row(k+1,n)-row(k)*u(k,k+1:n) % sparse update
8        endif
9     enddo
10    apply a dropping rule to row(1:n)
11    l(i,1:i-1)=row(1:i-1) % sparse copy
12    u(i,i:n) = row(i:n) % sparse copy
13    row(1:n) = 0
14    enddo
15 enddo
```

ILU preconditioners, cont.

ILUT: Dropping rules

5: an element is dropped if it is less than the relative tolerance τ_i, equal to $\tau\|row(k)\|$ (using the original row)

10: Drop all entries less than τ_i, keep the largest p entries in the L- and U-part and the diagonal element, which is always kept.

Observe, that sort operations are included.

ILU preconditioners, cont.

BILUT Block-ILUT

General design of the solver: CUDA, SIMT (Single Instruction Multiple threads)

1. CRS sparse format
2. Reorder using RCM (Recursive Cuthill-McKee)
3. Use Metis to generate a balanced partitioning; reorder and repartition after long time intervals
4. FGMRES with some preconditioner per timestep.

Reference: H. Sudan, H. Klie, R. Li and Y. Saad, High Performance Manycore Solvers for Reservoir Simulation

European Association of Geoscientists & Engineers Conference Proceedings, ECMOR XII - 12th European Conference on the Mathematics of Oil Recovery, Sep 2010, cp-163-00042 ISBN: 978-90-73781-89-4

DOI: https://doi.org/10.3997/2214-4609.20144961

Block ILU preconditioners

- LDL^T and LDU preconditioners
- Let $A = D_A - L_A - U_A$.
  Symmetric Gauss-Seidel preconditioner: $C = (D_A - L_A)D_A^{-1}(D_A - U_A)$
  Note:
  $C = (D_A - L_A)D_A^{-1}(D_A - U_A) = D_A - L_A - U_A + LD_A^{-1}U = A + LD_A^{-1}U$

$$A = LU = \begin{bmatrix} I_1 & 0 & \cdots & 0 \\ L_{2,1} & I_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ L_{n,1} & L_{n,2} & \cdots & I_n \end{bmatrix} \begin{bmatrix} D_1 & U_{1,2} & \cdots & U_{1,n} \\ 0 & D_2 & \cdots & U_{2,n} \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & D_n \end{bmatrix}$$

$$\mathbf{y}_i = \mathbf{y}_i - \sum_{j=1}^{i-1} L_{ji}\mathbf{y}_j \qquad \mathbf{x}_i = D_i^{-1}(\mathbf{y}_i - \sum_{j=i+1}^{n} U_{ij}\mathbf{x}_j)$$

Block ILU preconditioners, cont.

Attractive: D - diagonal.
How do we obtain a permutation, such that we get D - diagonal?
Use multicoloring algorithm:

```
For $1 = 1 : n$
    Set color(i)=0
endfor
For $1 = 1 : n$
    Set color(i)=min($k > 0 : k \neq$color(j) for $j \in Adj(i)$)
endfor
```

where $Adj(i) = \{j \neq i : a_{ij} \neq 0\}$.

Power(q)-pattern Method, Dimitar Lukarski

Building of power(q)-pattern – ILU(p,q)

- Perform multi-coloring analysis for $|A|^q$ and obtain
  - corresponding permutation π
  - the number of colors B and local block sizes b_i
- Permute $A_\pi := \pi A \pi^{-1}$
- Apply a modified ILU(p) factorization to A_π

Central result: For $q = p + 1$ the diagonal blocks of L and U becone diagonal themselves. No fill-ins here!
p - fill-in levels
q - describes the degree of parallelism.

D. Lukarski, J.-P- Weiss, Enhanced Parallel ILU(p)-based Preconditioners for Multi-core CPUs and GPUs – The Power(q)-pattern Method, January 2011, DOI: 10.11588/emclpp.2011.08.11690

ILU(p,q)

- LU sweeps, solve in parallel $LUz = r$
- Re-formulate into a block-form
- Use fine-grained parallelism on the block level
- Parallelism $=$ $N/$Num blocks

$$x_i := D_{Li}^{-1}(r_i - \sum_{j=1}^{i-1} L_{i,j}x_j)$$

$$z_i := D_{Ui}^{-1}(x_i - \sum_{j=1}^{B-i} U_{i,j}z_{i+j})$$

ILU(p,q) constructs D_{Li}^{-1} and D_{Ui}^{-1} to be with diagonal elements only

Block-tridiagonal matrices

Let A be block-tridiagonal, and expressed as $\boxed{A = D_A + L_A + U_A}$.
One can envisage three major versions of the factorization algorithm:

(i) $\boxed{A = (D + L_A)D^{-1}(D + U_A)}$

(ii) $\boxed{A = (D^{-1} + L_A)D(D^{-1} + U_A)}$

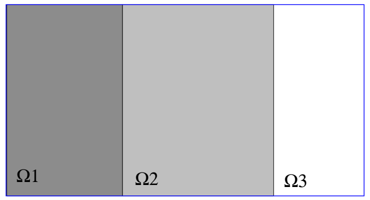
(iii) $\boxed{A = (I - \widetilde{L}_A)D^{-1}(I - \widetilde{U}_A)}$

$D_i = A_{ii} - A_{i,i-1}D_{i-1}^{-1}A_{i-1,i}, i \geq 2, D_1 = A_{11}$

$D_i = (A_{ii} - A_{i,i-1}D_{i-1}A_{i-1,i})^{-1},\ i \geq 1, D_0 = 0$ (Inverse free substitutions), where $\widetilde{L}_A = L_A D, \widetilde{U}_A = DU_A$.

Here $A^{-1} = (I - \widetilde{U}_A)^{-1}D(I - \widetilde{L}_A)^{-1}$
$(I - \widetilde{U}_A)^{-1} = (I + \widetilde{U}_A^{2^s})\ldots(I + \widetilde{U}_A^2)(I + \widetilde{U}_A)$ and similarly for $(I - \widetilde{L}_A)^{-1}$.

Block-tridiagonal matrices, cont.



Consider a two-dimensional domain partitioned in strips. Assume that points on the lines of intersection are only coupled to their nearest neighbors in the underlying mesh (and we do not have periodic boundary conditions). Hence, there is no coupling between subdomains except through the “glue” on the interfaces.

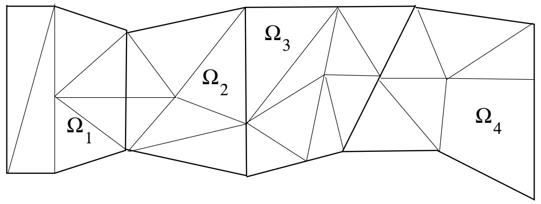
Block-tridiagonal matrices, cont.

When the subdomains are ordered lexicographically from left to right, a domain Ω_i becomes coupled only to its pre- and postdecessors Ω_{i-1} and Ω_{i+1}, respectively and the corresponding matrix takes the form of a block tridiagonal matrix
$\boxed{A = \text{tridiag}\ (A_{i,i-1}, A_{i,i}, A_{i,i+1}),}$ or

$$A = \begin{bmatrix} A_{11} & A_{12} & & 0 \\ A_{21} & A_{22} & A_{23} & \\ & \ddots & \ddots & \ddots \\ 0 & & A_{n,n-1} & A_{n,n} \end{bmatrix}$$

For definiteness we let the boundary meshline $\overline{\Omega}_i \cap \overline{\Omega}_{i+1}$ belong to Ω_i.
In order to preserve the sparsity pattern we shall factor A without use of permutations.
Naturally, the lines of intersection do not have to be straight.

Block-tridiagonal matrices, cont.



Examples of subdomain decompositions

Block-tridiagonal matrices, cont.

Paper:
Modification and compensation strategies for threshold-based incomplete factorizations
S. MacLachlan, D. Osei-Kuffuor, Yousef Saad

Examples of ILU performance (numerical)

	Precond.with $\widehat{A}_0$		ILU	
dt	$N1/N2$	$time(s)$	$N1/N2$	$time(s)$
h	$4/10$	14.58		
$h/4$	$4/10$	16.98	*no convergence*	
$h/5$	$4/10$	16.77		
$h/10$	$4/10$	14.67	$4/42$	23.66
$h/20$	$4/10$	14.62	$4/13$	13.23
$h/40$	$4/10$	14.11	$4/10$	10.55

Multiphace flow with convection: Two preconditioners: Run time and number of iterations, $Pe = 1000$

Examples of ILU performance, cont.

	Precond.with $\widehat{A}_0$		ILU	
dt	$N1/N2$	$time(s)$	$N1/N2$	$time(s)$
h	$3/10$	16.66		
$h/4$	$3/10$	16.54		
$h/5$	$3/10$	16.53	*no convergence*	
$h/10$	$3/10$	16.28		
$h/20$	$3/10$	15.82		
$h/40$	$3/10$	15.59		

Problem 2: Two preconditioners: run time and number of iterations, $Pe = 1$.

ILU - pros and cons

PROS:

- Very general, no additional knowledge of the problem is required
- Relatively easy to implement
- Available in the software packages

CONS:

- Method parameters to tune; nonlinear behaviour wrt to τ
- Not very nummerically efficient (may not converge)
- Problem-dependent behavious
- Relatively less degrees of parallelism

Complexity of linear solvers

Time to solve the Poisson model problem on a regular grid with N points

Solver	1D	2D	3D
Sparse Choleski	$O(N)$	$O(N^{1.5})$	$O(N^2)$
Unprecond. CG	$O(N^2)$	$O(N^{1.5})$	$O(N^{1.33})$
IC-precond. CG	$O(N^{1.5})$	$O(N^{1.25})$	$O(N^{1.17})$
Multigrid	$O(N)$	$O(N)$	$O(N)$

Demo ILU

Approximate inverse preconditioning

Some references

- Kolotilina, L. Yu.; Yeremin, A. Yu. Factorized sparse approximate inverse preconditionings. I. Theory. SIAM J. Matrix Anal. Appl. 14 (1993), 45-58
- Kolotilina, L. Yu.; Nikishin, A. A.; Yeremin, A. Yu. Factorized sparse approximate inverse preconditionings. IV. Simple approaches to rising efficiency. Numer. Linear Algebra Appl. 6 (1999), 515-531
- Grote, Marcus J.; Huckle, Thomas Parallel preconditioning with sparse approximate inverses. SIAM J. Sci. Comput. 18 (1997), 838-853.
- Benzi, Michele; Meyer, Carl D.; Tuma, Miroslav A sparse approximate inverse preconditioner for the conjugate gradient method. SIAM J. Sci. Comput. 17 (1996), 1135-1149

Some references

- Benzi, Michele; Tuma, Miroslav A sparse approximate inverse preconditioner for nonsymmetric linear systems. SIAM J. Sci. Comput. 19 (1998), 968-994
- Huckle, Thomas Approximate sparsity patterns for the inverse of a matrix and preconditioning. Appl. Numer. Math. 30 (1999), 291-303
- Bröker, Oliver; Grote, Marcus J. Sparse approximate inverse smoothers for geometric and algebraic multigrid. Appl. Numer. Math. 41 (2002), 61-80
- Broker, Oliver; Parallel multigrid methods using sparse approximate inverses. Thesis (Dr.sc.)ž2013Eidgenoessische Technische Hochschule Zuerich (Switzerland). 2003. 169 pp
- Holland, Ruth M.; Wathen, Andy J.; Shaw, Gareth J. Sparse approximate inverses and target matrices. SIAM J. Sci. Comput. 26 (2005),1000-1011
- Jia, Zhongxiao; Zhu, Baochen A power sparse approximate inverse preconditioning procedure for large sparse linear systems. Numer. Linear Algebra Appl. 16 (2009),259-299

Some references

- Wang, Shun; de Sturler, Eric Multilevel sparse approximate inverse preconditioners for adaptive mesh refinement. Linear Algebra Appl. 431 (2009), 409-426
- Malas, Tahir; Gürel, Levent Accelerating the multilevel fast multipole algorithm with the sparse-approximate-inverse (SAI) preconditioning. SIAM J. Sci. Comput. 31 (2009), 1968-1984
- Neytcheva, M; Bängtsson, Erik; Linnér, Elisabeth Finite-element based sparse approximate inverses for block-factorized preconditioners. Adv. Comput. Math. 35 (2011), 323-355

- Thomas Huckle,
  Modified Sparse Approximate Inverses
  http://www5.in.tum.de/wiki/index.php/MSPAI
  Factorized Sparse Approximate Inverses
  http://www5.in.tum.de/wiki/index.php/FSPAI

Approximate inverses: Explicit Methods

Given a sparse matrix $A = [a_{ij}] \in^{n\times n}$.
Let S be a sparsity pattern. We want to compute $G \in \mathcal{S}$, such that

$$(GA)_{ij} = \delta_{ij},\ (i,j) \in \mathcal{S},$$

i.e.

$$\sum_{k:(i,k)\in S} g_{ik}a_{kj} = \delta_{ij},\ (i,j) \in \mathcal{S}.$$

Some observations:

- ⊕ the elements in the ith row of G can be computed independently;
- ⊖ even if A is symmetric, G is not necessarily symmetric, because g_{ij} and g_{ji} are, in general, not equal.

How does this work?

Choose $\mathcal{S}$ to be the tridiagonal part of A,
$\mathcal{S} = \{(1,1),(1,2),\{(i,i-1),(i,i),(i,i+1)\}_{i=1}^{n},(n,n-1),(n,n)\}$.
Then, when computing the ith row of G we need only the entries of the matrix A, namely,

$$A^i = \begin{bmatrix} a_{i-1,i-1} & a_{i-1,i} & a_{i-1,i+1} \\ a_{i,i-1} & a_{i,i} & a_{i,i+1} \\ a_{i+1,i-1} & a_{i+1,i} & a_{i+1,i+1} \end{bmatrix}$$

Given $A \in R^{n\times n}$ and $\mathcal{S}$

```
for i=1:n,
        Extract from $A$ the small matrix $A^i$, needed to compute
        the entries of $G(i,:)$
        Solve with $A^i$
        Store row $G(i,:)$
end
```

Example:

We want to find G with the same sparsity pattern as A, i.e.,

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -2 & 0 \\ 0 & -2 & 4 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \qquad G = \begin{bmatrix} g_{11} & g_{12} & 0 & 0 \\ g_{21} & g_{22} & g_{23} & 0 \\ 0 & g_{32} & g_{33} & g_{34} \\ 0 & 0 & g_{43} & g_{44} \end{bmatrix}$$

$$G(1,:): \begin{array}{rcl} 2g_{11} - g_{12} & = & 1 \\ -g_{11} + 3g_{12} & = & 0 \end{array} \qquad G(2,:): \begin{array}{rcl} 2g_{21} - g_{22} & = & 0 \\ -g_{21} + 3g_{22} - 2g_{23} & = & 1 \\ -2g_{22} + 4g_{23} & = & 0 \end{array}$$

Example, cont.

$$A^{-1} = \frac{1}{19}\begin{bmatrix} 13 & 7 & 4 & 2 \\ 7 & 14 & 8 & 4 \\ 4 & 8 & 10 & 5 \\ 2 & 4 & 5 & 12 \end{bmatrix}$$

$$G = \begin{bmatrix} \frac{3}{5} & \frac{1}{5} & 0 & 0 \\ \frac{1}{3} & \frac{2}{3} & \frac{1}{3} & 0 \\ 0 & \frac{4}{13} & \frac{6}{13} & \frac{3}{13} \\ 0 & 0 & \frac{1}{7} & \frac{4}{7} \end{bmatrix} \qquad GA = \begin{bmatrix} 1 & 0 & -0.40 & 0 \\ 0 & 1 & 0 & -0.33 \\ -0.31 & 0 & 1 & 0 \\ 0 & -0.28 & 0 & 1 \end{bmatrix}$$

Example, cont.

Note: the second row of G is the second row of the matrix

$$B = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & -2 & 0 \\ 0 & -2 & 4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}, \; GB = \begin{bmatrix} 1 & 0 & -0.4 & 0 \\ 0 & 1 & 0 & 0 \\ -0.31 & 0 & 1.2308 & 0.4615 \\ 0 & -0.2857 & 0.5714 & 1.1429 \end{bmatrix}.$$

However, if we compute AG then

$$AG = \begin{bmatrix} 0.8667 & -0.2667 & -0.3333 & 0 \\ 0.4000 & 1.1846 & 0.0769 & -0.4615 \\ -0.6667 & -0.1026 & 1.0366 & 0.3516 \\ 0 & -0.3077 & -0.1758 & 0.9121 \end{bmatrix}$$

i.e., the matrix G is computed as a left-side approximate inverse of A and as such is somewhat less accurate than as a right-side approximate inverse.
The drawback of the above method is that
in general even if A is symmetric, G is not!

Implicit Methods

Let A be in a factored form.

Suppose $A = LD^{-1}U$ is a triangular matrix factorization of A. If A is a band matrix then L and U are also band matrices.

Let $\boxed{L = I - \widetilde{L};\ U = I - \widetilde{U},}$ where $\widetilde{L}$ and $\widetilde{U}$ are strictly lower and upper triangular matrices correspondingly.

Lemma 1 *Using the above notations it can be shown that*
(i) $A^{-1} = DL^{-1} + \widetilde{U}A^{-1}$, *(ii)* $A^{-1} = U^{-1}D + A^{-1}\widetilde{L}$.

Proof

$$A = LD^{-1}U \Longrightarrow A^{-1} = U^{-1}DL^{-1}$$

$$\Longrightarrow (I - \widetilde{U})A^{-1} = DL^{-1} \Longrightarrow A^{-1} = DL^{-1} + \widetilde{U}A^{-1}.$$

Also

$$A^{-1}(I - \widetilde{L}) = U^{-1}D \Longrightarrow A^{-1} = U^{-1}D + A^{-1}\widetilde{L}.$$

■

Algorithm to compute A^{-1}

```
for $r = n, n-1, \cdots, 1$
        $(A^{-1})_{r,r} = D_{r,r} + \sum_{s=1}^{min(q,n-r)} \widetilde{U}_{r,r+s}(A^{-1})_{r+s,r}$
    for $k = 1, 2, \cdots, q$
        $(A^{-1})_{r-k,r} = \sum_{s=1}^{min(q,n-r+k)} \widetilde{U}_{r-k,r-k+s}(A^{-1})_{r-k+s,r} \rightsquigarrow (i)$
        $(A^{-1})_{r,r-k} = \sum_{t=1}^{min(q,n-r+k)} (A^{-1})_{r,r-k+t}\widetilde{L}_{r-k+t,r-k} \rightsquigarrow (ii)$
    endfor
endfor
```

q is the bandwidth.

A drawback:

Consider an spd matrix

$$A = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -3 \\ 1 & -3 & 4 \end{bmatrix}. \quad \text{Then} \quad A_{band} = \begin{bmatrix} 1 & -2 & 0 \\ -2 & 5 & -3 \\ 0 & -3 & 4 \end{bmatrix}$$

is indefinite.

A general framework for computing approximate inverses

Frobenius norm minimization

$$\|A\|_I = \sqrt{\sum_{i=1}^{n}\sum_{i=1}^{n} a_{ij}^2} = \sqrt{tr(AA^H)}$$

Let a sparsity pattern $\mathcal{S}$ be given. Consider the functional

$$F_W(G) = \|I - GA\|_W^2 = tr(I - GA)W(I - GA)^T,$$

where the weight matrix W is spd If $W \equiv I$ then $\|I - GA\|_I$ is the Frobenius norm of $I - GA$.

Clearly $F_W(G) \geq 0$. If $G = A^{-1}$ then $F_W(G) = 0$. Hence, we want to compute the entries of G in order to minimize $F_W(G)$, i.e. to find $\hat{G} \in S$, such that

$$\|I - \hat{G}A\|_W \leq \|I - GA\|_W,\ \forall G \in S.$$

The following properties of $tr(\cdot)$ will be used:

$$trA = trA^T,\ tr(A + B) = trA + trB.$$

$$
\begin{aligned}
F_W(G) &= tr(I-GA)W(I-GA)^T \\
&= tr(W - GAW - W(GA)^T + GAW(GA)^T) \qquad (1)\\
&= trW - trGAW - tr(GAW)^T + trGAWA^TG^T.
\end{aligned}
$$

Minimize F_W w.r.t. G, consider the entries $g_{i,j}$ as variables. The necessary condition for a minimizing point are

$$\frac{\partial F_W(G)}{\partial g_{ij}} = 0,\ (i,j) \in \mathcal{S}. \qquad (2)$$

From (1) and (2) we get $\boxed{-2(WA^T)_{ij} + 2(GAWA^T)_{ij} = 0,}$ or

$$(GAWA^T)_{ij} = (WA^T)_{ij},\ (i,j) \in \mathcal{S}. \qquad (3)$$

The equations (3) may or may not have a solution, depending on the particular matrix A and the choice of $\mathcal{S}$ and W.

Choices of W:

Choise 1: Let A be spd Choose $W = A^{-1}$ which is also spd

$$\Longrightarrow (GA)_{ij} = \delta_{ij},\ (i,j) \in S,$$

i.e. the formula for the explicit method can be seen as a special case of the more general framework for computing approximate inverses using weighted Frobenius norms.

Choise 2: Let $W = (A^TA)^{-1}$.

$$\Longrightarrow (G)_{ij} = (A^{-1})_{ij},\ (i,j) \in S,$$

which is the formula for the implicit method. In this case the entries of G are the corresponding entries of the exact inverse.

Improvement via diagonal compensation

Let A be symmetric and five-diagonal. Suppose we know that the two of the off-diagonals contain small entries. Such matrix appears if we solve the anisotropic problem, for instance:

$$-\frac{\partial^2 u}{\partial x^2} - \varepsilon\frac{\partial^2 u}{\partial y^2} = f,$$

where $\varepsilon > 0$ is small.
We choose a tridiagonal sparsity pattern $\mathcal{S}_3$ for G, where the the two nonzero off-diagonals will correspond to the off-diagonals of A, containing bigger elements, i.e. they are not necessarily next to the main diagonal. Then we construct an approximate inverse in the following way:

- Step 1: Let $\widetilde{A}$ be A with deleted small entries, i.e. $\widetilde{A} \in \mathcal{S}_3$.
- Step 2: Compute $\widetilde{G}$: $(\widetilde{G}A)_{ij} = \delta_{ij},\ (i,j) \in S_3$.
- Step 3: Find $G = \bar{G} + D$, where $\bar{G} = \frac{1}{2}(\widetilde{G} + \widetilde{G}^T)$ and D is diagonal, computed from the following imposed condition on G, i.e.

  $$G A\mathbf{e} = \mathbf{e},$$

  and $\mathbf{e} = (1, 1, \cdots, 1)^T$.

The diagonal compensation technique prescribes the spd property of A.

Constructing an spd approximate inverse

The methods described till now do not guarantee that G will be such a matrix.

We want now to compute an spd approximate inverse of an spd matrix.

Let $\mathcal{S}$ be a symmetric sparsity pattern. We seek G of the form

$$G = L_G^T L_G, L_G \in \mathcal{S}_L.$$

Clearly G will be spd

Theorem 1 *A matrix G of the form $G = L_G^T L_G$ which is an spd approximation of A^{-1} can be computed from the following relation:*

$$min_{X\in\mathcal{S}_L} \frac{\frac{1}{n} trXAX^T}{(det(XAX^T))^{\frac{1}{n}}} = \frac{\frac{1}{n} trL_G A L_G^T}{(det(L_G A L_G^T))^{\frac{1}{n}}}. \tag{4}$$

Proof:
$X \in \mathcal{S}_L$ is lower triangular. Let $X = D(I - \widetilde{X})$, where $\widetilde{X} \in \mathcal{S}_{\widetilde{L}}$ is strictly lower triangular. Then $\widetilde{X} = I - D^{-1}X$. Let denote also $D = diag(d_1, d_2, \cdots, d_n)$. Then

$$\begin{aligned}
\frac{\frac{1}{n} trXAX^T}{(det(XAX^T))^{\frac{1}{n}}} &= \frac{\frac{1}{n}\sum_i (XAX^T)_{ii}}{(det(X)^2 det(A))^{\frac{1}{n}}} \\
= \frac{\frac{1}{n}\sum_i \left(D(I-\widetilde{X})A(I-\widetilde{X})^T D\right)_{ii}}{(det(X)^2 det(A))^{\frac{1}{n}}} &= \frac{\frac{1}{n}\sum_i d_i^2 \left((I-\widetilde{X})A(I-\widetilde{X})^T\right)_{ii}}{\left(\prod_i d_i^2\right)^{\frac{1}{n}} (det(A))^{\frac{1}{n}}} \\
&= \frac{\frac{1}{n}\sum_i \alpha^2}{\left(\prod_i \alpha^2\right)^{\frac{1}{n}}} \cdot \frac{\left(\prod_i ((I-\widetilde{X})A(I-\widetilde{X})^T)_{ii}\right)^{\frac{1}{n}}}{(det(A))^{\frac{1}{n}}} \\
&= Expression\_A \cdot Expression\_B.
\end{aligned} \tag{5}$$

In the above notations $\alpha_i^2 = d_i^2 \left((I-\widetilde{X})A(I-\widetilde{X})^T\right)_{ii}$.

$Expression\_B$ does not depend on d_i. The problem of minimizing $Expression\_B$ is a particular case of the already considered problem of minimizing the functional $F_W(G)$ with a special choice of the corresponding matrices - $W = A,\ A = I,\ G = \widetilde{X}$. In other words, the solution of the problem

$$min_{\widetilde{X}\in S_{\widetilde{L}}} \prod_i \left((I-\widetilde{X})A(I-\widetilde{X})^T\right)_{ii} = min_{\widetilde{X}\in S_{\widetilde{L}}} tr(I-\widetilde{X})A(I-\widetilde{X})^T \tag{6}$$

will be also the solution of minimizing $Expression\_B$.

Further, $Expression\_A \geq 1,\ \forall\alpha$, being the ratio of the arithmetic and geometric mean, and takes the value 1 when $\alpha_i^2 = 1$.

Thus, we minimize $Expression\_A$ computing

$$d_i = \frac{1}{\left((I-\widetilde{X})A(I-\widetilde{X})^T\right)_{ii}^{\frac{1}{2}}}. \tag{7}$$

Let $\widetilde{L}_G$ be the solution of (7). Note that it is strictly lower triangular. Let the entries d_i of D are computed from the relations (7), where instead of $\widetilde{X}$ $\widetilde{L}_G$ is used. Then the matrix $L_G^T L_G$, where $L_G = D(I - \widetilde{L}_G)$, will be the searched approximation of A^{-1}:

- $(L_G A L_G^T)_{ii} = 1$ by construction;
- The equality (4) gives a measure of the quality of the approximate inverse constructed (the K-condition number (Igor Kaporin).

Let $A = tridiag(-1,4,-1)$. Find $L_G^T L_G$ - an approximate inverse of A, where L_G is bidiagonal. Thus, $\mathcal{S}_{\widetilde{L}} = \{\{(i-1,i)\}_{i=2}^n\}$.

First we compute a strictly lower bidiagonal matrix $\widetilde{L}$ from the condition

$$(\widetilde{L}A)_{i,j} = (A)_{i,j},\ i,j \in \mathcal{S}_{\widetilde{L}},$$

which gives us

$$\widetilde{L} = \begin{bmatrix} 0 & & & & & \\ \frac{1}{4} & 0 & & & & \\ & \frac{1}{4} & 0 & & & \\ & & & \ddots & & \\ & & & \frac{1}{4} & 0 & \\ & & & & \frac{1}{4} & 0 \end{bmatrix}.$$

Then d_i are found to be

$$d_1 = \frac{1}{2}, d_i = \frac{2}{\sqrt{15}},\ i = 1,2,\cdots,n.$$

$$L_G = \begin{bmatrix} \frac{1}{2} & 0 & \cdots & 0 & 0 \\ \frac{1}{2\sqrt{15}} & \frac{2}{\sqrt{15}} & \cdots & 0 & 0 \\ & & \ddots & & \\ & & & \ddots & \\ 0 & 0 & \cdots & \frac{1}{2\sqrt{15}} & \frac{2}{\sqrt{15}} \end{bmatrix}, \quad L_G^T L_G = \begin{bmatrix} \frac{4}{15} & \frac{1}{15} & 0 & \cdots & 0 \\ \frac{1}{15} & \frac{17}{60} & \frac{1}{15} & \cdots & 0 \\ & & \ddots & & \\ & & & \ddots & \\ 0 & \cdots & & \frac{1}{15} & \frac{17}{60} \end{bmatrix},$$

and

$$L_G^T L_G A = \begin{bmatrix} 1 & 0 & -\frac{1}{15} & \cdots & 0 \\ \frac{7}{15} & 1 & \frac{7}{60} & \cdots & 0 \\ & & \ddots & & \\ & & & \ddots & \\ 0 & \cdots & & \frac{7}{60} & 1 \end{bmatrix}.$$

Extensions

- When minimizing $\|I - AG\|_F$, minimize the 2-norm of each column separately, $\|\mathbf{e}_k - A\mathbf{g}_k\|_F, k = 1,\cdots,n$
- use adaptive $\mathcal{S}$ (much more expensive)
- used the sparsity pattern of powers of A
- Modified SPAI: combines
  - Frobenius norm minimization
  - MILU
  - vector probing

MSPAI

Consider the formulation:

$$\min_G \|CG - B\|_F = \min_G \left\| \begin{bmatrix} C \\ \rho\mathbf{e}^T C \end{bmatrix} G - \begin{bmatrix} B_0 \\ \rho\mathbf{e}^T B_0 \end{bmatrix} \right\|_F$$

$\rho = 0, C_0 = A, B_0 = I$ - the original form
$C_0 = I, B_0 = A$ - explicit approximation of A
$\rho = [1,1,\cdots,1]$ - MILU
Improve existing approximations:

$$\min_U \left\| \begin{bmatrix} L \\ \rho\mathbf{e}^T L \end{bmatrix} U - \begin{bmatrix} A \\ \rho\mathbf{e}^T A \end{bmatrix} \right\|_F$$

Finite element setting:

$$A = \sum_{k=1}^{M} R_k^T A_k R_k,$$

with R_k being the Boolean matrices which prescribe the local-to-global correspondence of the numbered degrees of freedom.

Is this of interest?

$$B^{-1} = \sum_{k=1}^{M} R_k^T A_k^{-1} R_k.$$

B^{-1} and A^{-1} are spectrally equivalent, namely, for some $0 < \alpha_1 < \alpha_2$ there holds

$$\alpha_1 A_{11}^{-1} \le B_{11}^{-1} \le \alpha_2 A_{11}^{-1},$$

Finite element setting:

Consider spd matrices.

$$\min_M(\lambda_{min}(A_k)) \le \lambda(A) \le p \max_M(\lambda_{max}(A_k)),$$

where p is the maximum degree of the graph representing the discretization mesh. Similarly, there holds

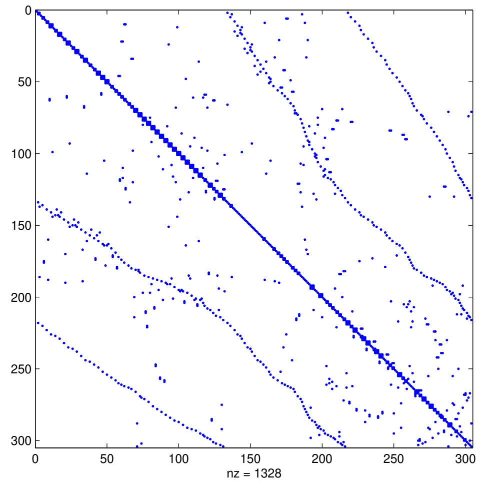
$$\min_M(\lambda_{min}(A_k)^{-1}) \le \lambda(B^{-1}) \le p \max_M(\lambda_{max}(A_k)^{-1}).$$

Then we obtain

$$\frac{\min(\lambda_{min}(A_k))}{\max(\lambda_{max}(A_k))} \le \frac{\mathbf{x}^T B^{-1}\mathbf{x}}{\mathbf{x}^T A^{-1}\mathbf{x}} \le \frac{\max(\lambda_{max}(A_k))}{\min(\lambda_{min}(A_k))}$$

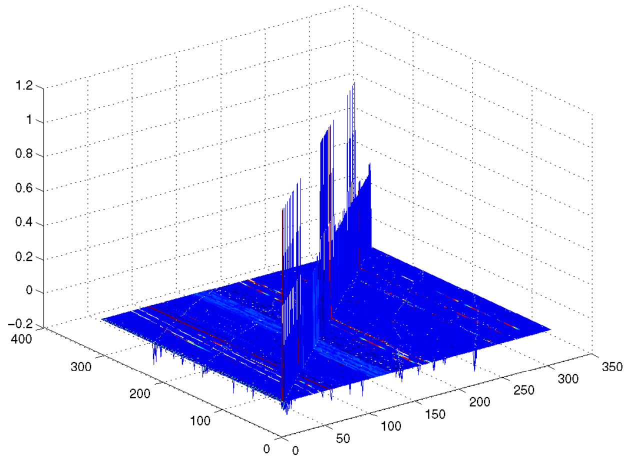
Thus, the spectral equivalence constants do not depend on the mesh parameter h but they are in general robust neither with respect to problem and mesh-anisotropies, nor to jumps in the problem coefficients as the eigenvalues of A_k depend on those.

FEM-SPAI



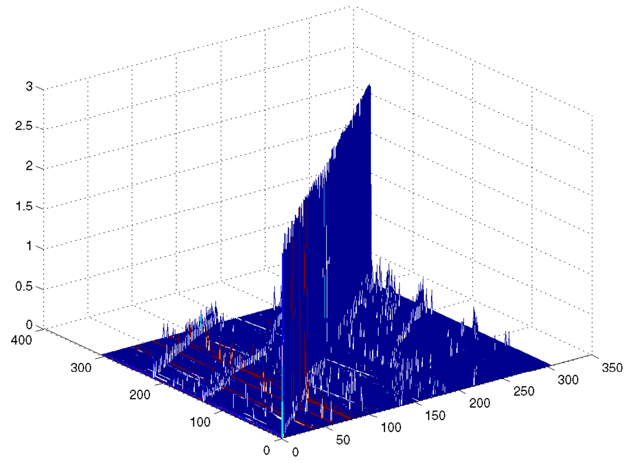
The matrix itself (`spy(A)`)

FEM-SPAI



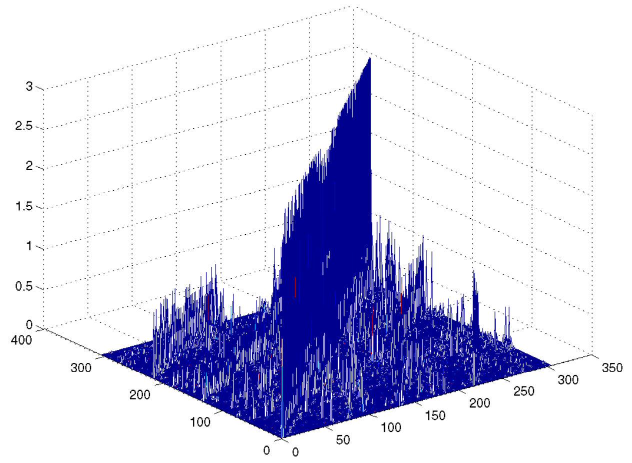
The matrix itself (`mesh(A)`)

FEM-SPAI



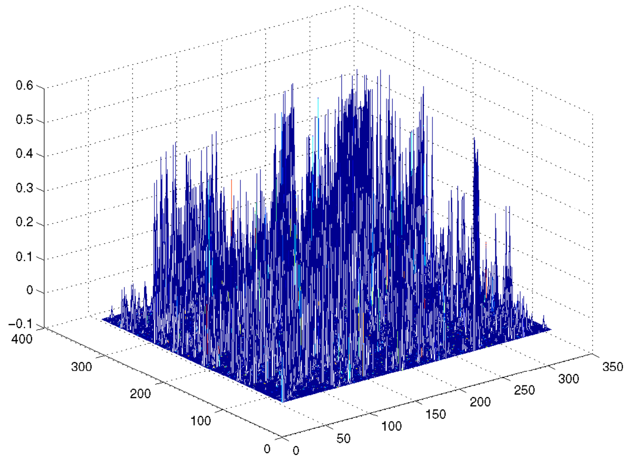
The approximate inverse (`mesh(AI)`)

FEM-SPAI



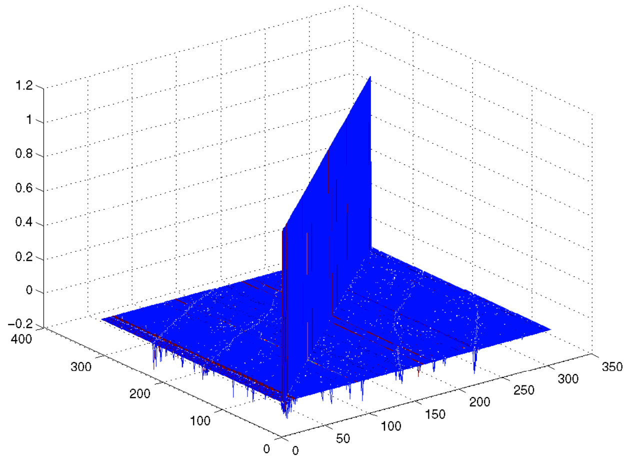
The exact inverse matrix (`mesh(inv(A))`)

FEM-SPAI



The difference (`mesh(inv(A)-AI)`)

FEM-SPAI



The product with A (`mesh(AI*A)`)

FEM-SPAI: Scalability figures: Constant problem size

$\#proc$	n_{fine}	$t_{B_{11}^{-1}}/t_A$	t_{repl} [s]	$t_{solution}$ [s]	# $iter$
4	197129	0.005	0.28	7.01	5
16	49408	0.180	0.07	0.29	5
64	12416	0.098	0.02	0.03	5

Problem size: 787456
Solution method: *PCG*
Relative stopping criterium: $< 10^{-6}$

FEM-SPAI: Scalability figures: Constant load per processor

$\#proc$	$t_{B_{11}^{-1}}/t_A$	t_{repl} [s]	$t_{solution}$ [s]	# $iter$
1	0.0050	-	0.17	5
4	0.0032	0.28	7.01	5
16	0.0035	0.24	4.55	5
64	0.0040	0.23	12.43	5

Local number of degrees of freedom: 197129
Solution method: *PCG*
Relative stopping criterium: $< 10^{-6}$

Domain decomposition methods

Many different interpretations within the PDE community:

- Parallel computing: data decomposition (independent of numerical method)
- Asymptotic analysis: separation of physical domain into regions with possibly different models
- Preconditioning methods: solution of a large linear system arising from the discretization of the PDE on the whole domain by DDM as a solver or a preconditioner:
  - Overlapping domain decomposition
  - Non-overlapping domain decomposition

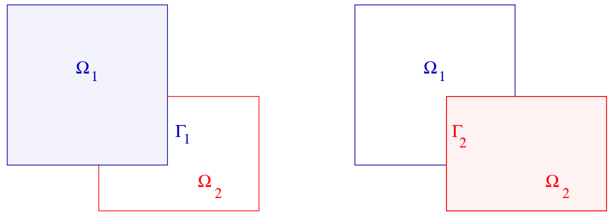
Domain decomposition:

Domain decomposition - decomposition of the spatial domain into several subdomains. Search the global true solution through (iteratively) solving subproblems while enforcing suitable continuity requirements between neighbor subdomains.

- Flexible - localized treatment of complex and irregular geometries, singularities etc.
- Efficient - often optimal convergence rate
- Easy to parallelize (coarse grain parallelization)

Schwarz 1870 (alternating method)

$A\mathbf{u} = \mathbf{f}$



Matrix form of Alternating Schwarz

Decompose A as $A_i A_{\partial\Omega_i\setminus\Gamma_i} A_{\Gamma_i}$.
Let $I_{\Omega_i\to\Gamma_j}$ be the discrete operator that interpolates the nodes in the interior of Ω_i to Γ_j. Then:

$$\begin{aligned} A_{\Omega_1}\mathbf{u}^k_{\Omega_1} &= \mathbf{f}_1 - A_{\Gamma_1} I_{\Omega_2\to\Gamma_1}\mathbf{u}^{k-1}_{\Omega_2} \\ A_{\Omega_2}\mathbf{u}^k_{\Omega_2} &= \mathbf{f}_2 - A_{\Gamma_2} I_{\Omega_1\to\Gamma_2}\mathbf{u}^{k}_{\Omega_1} \end{aligned}$$

Gauss-Seidel method for the system

$$\begin{bmatrix} A_{\Omega_1} & A_{\Gamma_1} I_{\Omega_2\to\Gamma_1} \\ A_{\Gamma_2} I_{\Omega_1\to\Gamma_2} & A_{\Omega_2} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\Omega_1} \\ \mathbf{u}_{\Omega_2} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \end{bmatrix}$$

Rearrange as a simple iteration:

$$\mathbf{u}^k_{\Omega_1} = \mathbf{u}^{k-1}_{\Omega_1} + A^{-1}_{\Omega_1}(\mathbf{f}_1 - A_{\Omega_1}\mathbf{u}^{k-1}_{\Omega_1} - A_{\Gamma_1} I_{\Omega_2\to\Gamma_1}\mathbf{u}^{k-1}_{\Omega_2})$$

$$\mathbf{u}^k_{\Omega_2} = \mathbf{u}^{k-1}_{\Omega_2} + A^{-1}_{\Omega_2}(\mathbf{f}_2 - A_{\Omega_2}\mathbf{u}^{k-1}_{\Omega_2} - A_{\Gamma_1} I_{\Omega_1\to\Gamma_2}\mathbf{u}^{k}_{\Omega_2})$$

Additive and multiplicative Schwarz methods:

$$\mathbf{u}^k_{\Omega_1} = \mathbf{u}^{k-1}_{\Omega_1} + A^{-1}_{\Omega_1}(\mathbf{f}_1 - A_{\Omega_1}\mathbf{u}^{k-1}_{\Omega_1} - A_{\Omega\setminus\overline{\Omega}_1}\mathbf{u}^{k-1}_{\Omega\setminus\overline{\Omega}_1})$$

$$\mathbf{u}^k_{\Omega_2} = \mathbf{u}^{k-1}_{\Omega_2} + A^{-1}_{\Omega_2}(\mathbf{f}_2 - A_{\Omega_2}\mathbf{u}^{k-1}_{\Omega_2} - A_{\Omega\setminus\overline{\Omega}_2}\underset{\substack{\Uparrow \\ \mathbf{u}^k_{\Omega\setminus\overline{\Omega}_2}}}{\mathbf{u}^{k-1}_{\Omega\setminus\overline{\Omega}_2}})$$

For the whole system: two-step algorithm

$$\mathbf{u}^{k+1/2} = \mathbf{u}^k + \begin{bmatrix} A_{\Omega_1}^{-1} & 0 \\ 0 & 0 \end{bmatrix} (\mathbf{f} - A\mathbf{u}^k)$$

$$\mathbf{u}^{k+1} = \mathbf{u}^{k+1/2} + \begin{bmatrix} 0 & 0 \\ 0 & A_{\Omega_2}^{-1} \end{bmatrix} (\mathbf{f} - A\mathbf{u}^{k+1/2})$$

Final form

Denote: $\mathbf{u}_{\Omega_1} = R_1\mathbf{u} = \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\Omega_1} \\ \mathbf{u}_{\Omega\setminus\Omega_1} \end{bmatrix}$

$\mathbf{u}_{\Omega_2} = R_2\mathbf{u} = \begin{bmatrix} 0 & I \end{bmatrix} \begin{bmatrix} \mathbf{u}_{\Omega\setminus\Omega_2} \\ \mathbf{u}_{\Omega_2} \end{bmatrix}$

Then $A_{\Omega_i} = R_i A R_i^T$; let $B_i = R_i^T (R_i A R_i^T)^{-1} R_i$.

$$\mathbf{u}^{k+1} = \mathbf{u}^k + (B_1 + B_2 - B_2 A B_1)(\mathbf{f} - A\mathbf{u}^k)$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k + (B_1 + B_2)(\mathbf{f} - A\mathbf{u}^k)$$

Final form, many subdomains

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \sum_{i=1}^{p} B_i(\mathbf{f} - A\mathbf{u}^k)$$

$$\begin{aligned} \mathbf{u}^{k+1/p} &= \mathbf{u}^k + B_1(\mathbf{f} - A\mathbf{u}^k) \\ \mathbf{u}^{k+2/p} &= \mathbf{u}^{k+1/p} + B_2(\mathbf{f} - A\mathbf{u}^{k+1/p}) \\ &\vdots \\ \mathbf{u}^{k+1} &= \mathbf{u}^{k+(p-1)/p} + B_p(\mathbf{f} - A\mathbf{u}^{k+(p-1)/p}) \end{aligned}$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k + (I - (I - B_p A(\cdots(I - B_A)))) A^{-1}(\mathbf{f} - A\mathbf{u}^k)$$

Problem:

Too slow Convergence deteriorates as p increases (H decreases).
Reason: The only global communication of information between subdomains are through overlapping regions. **Too slow!**
How to speed up? **Coarse grid correction.**

$$\mathbf{u}^{fine} = \mathbf{u}^{fine} + R^T A_C^{-1} R(\mathbf{f} - A\mathbf{u}^{fine})$$

Two-level additive Schwarz method:

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \left(R^T A_C^{-1} R + \sum_{i=1}^{p} R^T A_i^{-} R \right) \mathbf{r}^k$$

Summarizing DD:

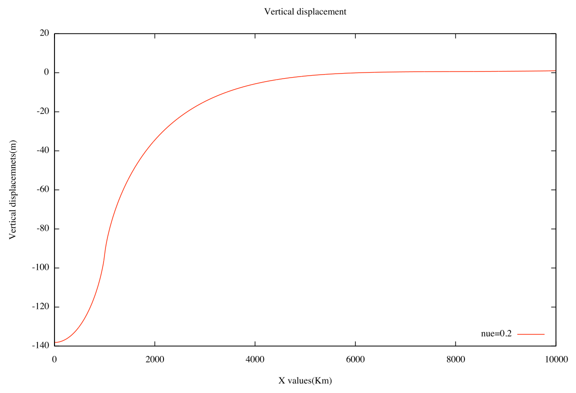
- multiplicative is faster than additive
- overlapping or nonoverlapping; large overlap is better for convergence
- deteorating convergence when increasing the number of subdomains (if implemented straightforwardly)
- stabilization with a coarse grid corrrection, nearly optimal convergence
- used as a preconditiner
- used in a Multigrid setting as a smoother
- attractive for parallel computations (FETI, BETI, ...)

Two-by-two block matrix preconditioning. Schur Complement Based Preconditioners

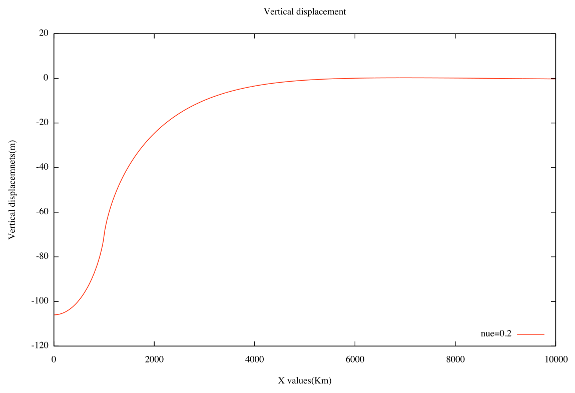
write by hand ...

two-by-two block factorizations, saddle point problems

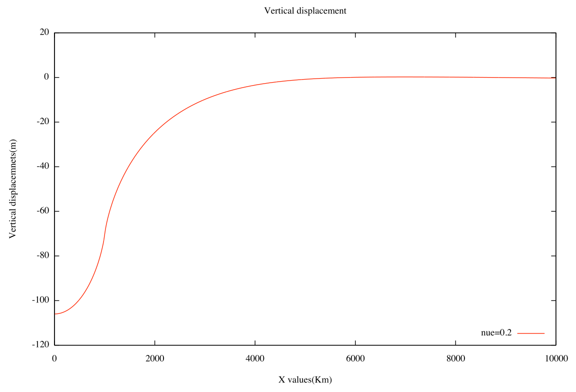
	Tolerance	Residual	Iterations	sol.time
With inner	$1e-7$	$1.224e-07$	$16(11,5)$	73.4
solvers	$1e-12$	$3.0087e-12$	$120(8,6)$	426
One AMG	$1e-7$	$2.631e-07$	28	11.9
iter	$1e-12$	$3.2777e-12$	271	138



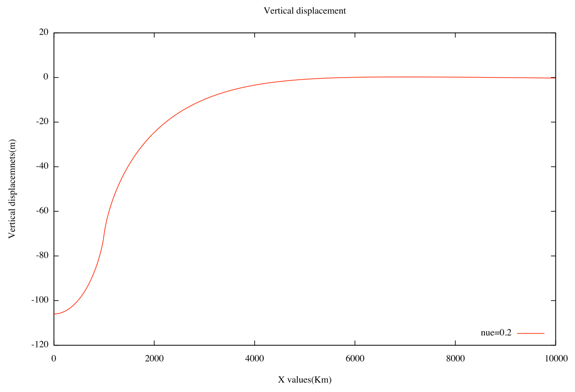
Results with solver only one AMG iteration and tolerance $1e-7$



Results with solver only one AMG iteration and tolerance $1e-12$



Results with inner solver and tolerance $1e-7$



Results with inner solver and tolerance $1e-12$