FEM and Sparse Linear System Solving

Lecture 10, Nov 24, 2017: Preconditioning

http://people.inf.ethz.ch/arbenz/FEM17

Peter Arbenz
Computer Science Department, ETH Zürich
E-mail: arbenz@inf.ethz.ch

Survey on lecture

- The finite element method
- Direct solvers for sparse systems
- Iterative solvers for sparse systems
  - Stationary iterative methods, preconditioning
  - Steepest descent and conjugate gradient methods
  - Krylov space methods, GMRES, MINRES
  - Preconditioning
    - Preconditioning by stationary and related iterations
    - Incomplete factorization preconditioning
    - Domain decomposition
  - Nonsymmetric Lanczos iteration based methods
    Bi-CG, QMR, CGS, BiCGstab
  - Multigrid preconditioning

Outline of this lecture

1. Preconditioning
2. Preconditioned GMRES
3. Preconditioning with stationary iterations
4. Flexible GMRES
5. Incomplete factorization preconditioning
6. Domain decomposition preconditioning

References

- A. Wathen: *Preconditioning*. Acta Numerica 2015, pp. 329–376.
- Y. Saad: *Iterative Methods for Sparse Linear Systems*, SIAM, 2nd edition, 2003.
- B. Smith, P. Bjørstad, W. Gropp: *Domain decomposition: Parallel multilevel methods for elliptic partial differential equations*. Cambridge University Press 1996.
- V. Dolean, P. Jolivet, F. Nataf: *An introduction to domain decomposition methods.* SIAM 2015.

Preconditioning

- Given a system of equations

$$A\mathbf{x} = \mathbf{b}, \qquad A \in \mathbb{R}^{n \times n} \text{ is nonsingular} \tag{1}$$

  that we want to solve iteratively.
- Can we improve the convergence behavior, i.e., reduce the number of iterations until convergence?
- Introduce a preconditioner M and change (1) into

$$M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b}. \tag{2}$$

  This is called left preconditiong as the preconditioner is multiplied from the left to (1).

Preconditioning (cont.)

- Right preconditioning with M:

$$AM^{-1}\boldsymbol{y} = \boldsymbol{b}, \qquad \boldsymbol{x} = M^{-1}\boldsymbol{y}$$

- Split preconditioning with $M = LU$:

$$L^{-1}AU^{-1}\boldsymbol{z} = L^{-1}\boldsymbol{b}, \qquad \boldsymbol{x} = U^{-1}\boldsymbol{z}$$

  Special case if A and M are SPD. Then use Cholesky instead of LU factorization of M, $M = LL^T$,

$$L^{-1}AL^{-T}\boldsymbol{z} = \boldsymbol{b}, \qquad \boldsymbol{x} = L^{-T}\boldsymbol{z}.$$

- *How to choose M?*
- *Relations among the three versions of preconditioning?*

Summary: Requirements for a preconditioner

- Either $\kappa(M^{-1}A) \approx 1$.
  - M is a good approximation of A
    (or M^{-1} is a good approximation of A^{-1}).
- or the spectrum of $M^{-1}A$ consists of a few eigenvalues (or eigenvalue clusters).
- $M\boldsymbol{z} = \boldsymbol{r}$ can be solved cheaply.
- M can be constructed cheaply (if necessary at all)

Note: In Krylov space methods the preconditioner M is applied once in each iteration step, more precisely: we solve the system of equations $M\boldsymbol{z} = \boldsymbol{r}$ in each iteration step.

Krylov space methods

- Without preconditioning we work in the Krylov space

$$\mathcal{K}_m(\boldsymbol{r}_0, A) = \operatorname{span}\left\{\boldsymbol{r}_0, A\boldsymbol{r}_0, A^2\boldsymbol{r}_0, \ldots, A^{m-1}\boldsymbol{r}_0\right\}$$

- With $\boldsymbol{x}_m \in \boldsymbol{x}_0 + \mathcal{K}_m(\boldsymbol{r}_0, A)$, we have

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + p_{m-1}(A)\boldsymbol{r}_0, \qquad \boldsymbol{r}_m = (I - Ap_{m-1}(A))\boldsymbol{r}_0.$$

- In GMRES we determine $\boldsymbol{x}_m = \boldsymbol{x}_0 + V_m\boldsymbol{y}_m$ such that

$$\begin{aligned}\min \|\boldsymbol{r}_m\|_2 &= \min \|\boldsymbol{b} - A\boldsymbol{x}_m\|_2 = \min \|\boldsymbol{b} - A(\boldsymbol{x}_0 + V_m\boldsymbol{y})\|_2 \\ &= \min \|\boldsymbol{r}_0 - AV_m\boldsymbol{y}\|_2 = \min \|\boldsymbol{r}_0 - V_{m+1}\bar{H}_m\boldsymbol{y}\|_2 \\ &= \min \|V_{m+1}\left(\beta\boldsymbol{e}_1 - \bar{H}_m\boldsymbol{y}\right)\|_2 = \min \|\beta\boldsymbol{e}_1 - \bar{H}_m\boldsymbol{y}\|_2.\end{aligned}$$

$\bar{H}_m$ is a $m+1 \times m$ upper Hessenberg matrix.

Residual results

Residual polynomial: Since $\boldsymbol{r}_m = \boldsymbol{b} - A\boldsymbol{x}_m \in \boldsymbol{r}_0 + A\mathcal{K}_m$, we have $\boldsymbol{r}_m = (I - A\,p_{m-1}(A))\boldsymbol{r}_0 = p_m(A)\boldsymbol{r}_0$.

p_m is a polynomial of degree m and $p_m(0) = 1$. Denote the set of all such polynomials by $\mathbb{P}'_m$.

Theorem: *Let $A = Q\Lambda Q^{-1}$ be diagonalizable. Then at step m of the GMRES iteration, the residual $\boldsymbol{r}_m$ satisfies*

$$\frac{\|\boldsymbol{r}_m\|_2}{\|\boldsymbol{r}_0\|_2} \leq \inf_{p_m \in \mathbb{P}'_m} \|p_m(A)\|_2 \leq \kappa(Q) \inf_{p_m \in \mathbb{P}'_m} \max_{\lambda \in \sigma(A)} |p_m(\lambda)|.$$

Here, λ runs through the set $\sigma(A)$ of A's eigenvalues.
Remark: A similar result holds for the symmetric case (i.e., for CG), but with A-norm, $\kappa(Q) = 1$, and all eigenvalues are real.

CG method for SPD A

We bound

$$\inf_{p_m \in \mathbb{P}'_m} \max_{\lambda \in \sigma(A)} |p_m(\lambda)| \qquad \text{by} \qquad \inf_{p_m \in \mathbb{P}'_m} \max_{\lambda \in [\lambda_{\min}, \lambda_{\max}]} |p_m(\lambda)|$$

Fiddling around with Chebyshev polynomials gives (Saad)

Theorem: *Let A be SPD, then the error of the CG method satisfies*

$$\|e_k\|_A \leq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \|e_0\|_A.$$

Here, $\kappa = \kappa(A) = \dfrac{\lambda_{\max}}{\lambda_{\min}}$.

By consequence: We choose M such that $\kappa(M^{-1}A) \ll \kappa(A)$.

Alternative view point

- The previous statement (for PCG) holds also for general matrices, but the set of eigenvalues is not so easy to manage, since the eigenvalues do not lie on a straight line.
- The polynomial p_m must be small on all eigenvalues.
- Desired, $m \ll n$.
- Sounds impossible, but eigenvalues may be equal (!) or at least clustered.
  If there are just a few eigenvalues (or eigenvalue clusters) then it may be possible that $|p_m(\lambda)|$ is small on all eigenvalues.
- By consequence: We must choose M such that eigenvalues of $M^{-1}A$ cluster.
- With both view points, $M = A$ is the ideal preconditioner.

The left preconditioned GMRES algorithm

- With left preconditioning we work in the Krylov space

$$\mathcal{K}_m(\boldsymbol{z}_0, M^{-1}A) = \operatorname{span}\left\{\boldsymbol{z}_0, M^{-1}A\boldsymbol{z}_0, \ldots, (M^{-1}A)^{m-1}\boldsymbol{z}_0\right\}$$

  where $\boldsymbol{z}_0 = M^{-1}\boldsymbol{r}_0 = M^{-1}(\boldsymbol{b} - A\boldsymbol{x}_0)$.
- We determine $\boldsymbol{x}_m \in \boldsymbol{x}_0 + \mathcal{K}_m(\boldsymbol{z}_0, M^{-1}A)$ such that

$$\|\boldsymbol{z}_m\|_2 = \|(I - M^{-1}A\, p_{m-1}(M^{-1}A))\boldsymbol{z}_0\|$$

  is as small as possible.
- So,

$$\begin{aligned}\boldsymbol{x}_m &= \boldsymbol{x}_0 + p_{m-1}(M^{-1}A)\boldsymbol{z}_0\\ &= \boldsymbol{x}_0 + p_{m-1}(M^{-1}A)M^{-1}\boldsymbol{r}_0\\ &= \boldsymbol{x}_0 + M^{-1}p_{m-1}(AM^{-1})\boldsymbol{r}_0\end{aligned}$$

The left preconditioned GMRES(m) algorithm

Choose initial guess $\boldsymbol{x}_0$.
2: Compute $\boldsymbol{z}_0 = M^{-1}(\boldsymbol{b} - A\boldsymbol{x}_0)$, $\beta = \|\boldsymbol{z}_0\|_2$, and $\boldsymbol{v}_1 = \boldsymbol{z}_0/\beta$.
for $j = 1, \ldots, m$ **do**
  Compute $\boldsymbol{w} := M^{-1} A \boldsymbol{v}_j$
  Orthogonalize $\boldsymbol{w}$ against $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_j$. (Gram–Schmidt)
  Compute $h_{j+1,j} = \|\boldsymbol{w}\|_2$ and $\boldsymbol{v}_{j+1} = \boldsymbol{w}/h_{j+1,j}$.
end for
Define $V_m := [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m]$, $\bar{H}_m = ((h_{i,j}))$
Compute $\boldsymbol{y}_m = \text{argmin}_{\boldsymbol{y}} \|\beta \boldsymbol{e}_1 - \bar{H}_m \boldsymbol{y}\|_2$ and $\boldsymbol{x}_m = \boldsymbol{x}_0 + V_m \boldsymbol{y}_m$
if converged **then**
  leave GMRES
else
  set $\boldsymbol{x}_0 := \boldsymbol{x}_m$ and goto **2**.
end if

The right preconditioned GMRES algorithm

- With right preconditioning we work in the Krylov space

$$\mathcal{K}_m(\boldsymbol{r}_0, AM^{-1}) = \mathrm{span}\left\{\boldsymbol{r}_0, AM^{-1}\boldsymbol{r}_0, \ldots, (AM^{-1})^{m-1}\boldsymbol{r}_0\right\}$$

  where $\boldsymbol{r}_0 = \boldsymbol{b} - AM^{-1}\boldsymbol{u}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, i.e., $\boldsymbol{x}_0 = M^{-1}\boldsymbol{u}_0$.
- We determine $\boldsymbol{u}_m$ such that

$$\|\boldsymbol{r}_m\|_2 = \|(I - AM^{-1}\, p_{m-1}(AM^{-1}))\boldsymbol{r}_0\|$$

  is minimal.
- So, $\boldsymbol{u}_m = \boldsymbol{u}_0 + p_{m-1}(AM^{-1})\boldsymbol{r}_0$,
  or, $\boldsymbol{x}_m = M^{-1}\boldsymbol{u}_m = \boldsymbol{x}_0 + M^{-1}p_{m-1}(AM^{-1})\boldsymbol{r}_0$.

The right preconditioned GMRES algorithm (cont.)

Theorem

The approximate solutions obtained by left- and right-preconditioned GMRES both have the form

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + M^{-1} p_{m-1}(AM^{-1})\boldsymbol{r}_0$$

where p_{m-1} is a polynomial of degree $m-1$. In right-preconditioning p_{m-1} minimizes $\|\boldsymbol{b} - A\boldsymbol{x}_m\|$ while in left-preconditioning p_{m-1} minimizes $\|M^{-1}(\boldsymbol{b} - A\boldsymbol{x}_m)\|$.

Remark (Saad, p. 272) In most practical situations, the difference in the convergence behavior is not significant. The only exception is when M is ill-conditioned, which could lead to substantial differences.

The right preconditioned GMRES(m) algorithm

Choose initial guess $\boldsymbol{x}_0 = M\boldsymbol{u}_0$.
2: Compute $\boldsymbol{r}_0 = \boldsymbol{b} - AM^{-1}\boldsymbol{u}_0$, $\beta = \|\boldsymbol{r}_0\|_2$, $\boldsymbol{v}_1 = \boldsymbol{r}_0/\beta$.
for $j = 1, \ldots, m$ **do**
  Compute $\boldsymbol{w} := AM^{-1}\boldsymbol{v}_j$
  Orthogonalize $\boldsymbol{w}$ against $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_j$. (Gram–Schmidt)
  Compute $h_{j+1,j} = \|\boldsymbol{w}\|_2$ and $\boldsymbol{v}_{j+1} = \boldsymbol{w}/h_{j+1,j}$
end for
Define $V_m := [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m]$, $\bar{H}_m = ((h_{i,j}))$
Set $\boldsymbol{y}_m = \text{argmin}_{\boldsymbol{y}} \|\beta\boldsymbol{e}_1 - \bar{H}_m\boldsymbol{y}\|_2$ and $\boldsymbol{x}_m = \boldsymbol{x}_0 + M^{-1}V_m\boldsymbol{y}_m$
if converged **then**
  leave GMRES
else
  set $\boldsymbol{x}_0 := \boldsymbol{x}_m$ and goto **2**.
end if

Preconditioning with stationary iterations

- We can choose the same preconditioners for PCG / PGMRES as in stationary iterations.
- Jacobi (= diagonal), block Jacobi, (block) Gauss–Seidel, (block) symmetric Gauss–Seidel, (block) (S)SOR.
- Usually just one iteration step.
- These preconditioners are very simple and easy to implement.
- They are often not very powerful.
  But the Jacobi preconditioner parallelizes ideally, and can make up in this way for deficiencies.

Preconditioning with stationary iterations (cont.)

- Let $A = M - N$, M nonsingular, be a matrix splitting.
- One step of the corresponding stationary iteration for solving $A\boldsymbol{z} = \boldsymbol{r}$ is

$$\boldsymbol{z}_1 = \boldsymbol{z}_0 + M^{-1}(\boldsymbol{r} - A\boldsymbol{z}_0)$$

  Let's set our first approximate to $\mathbf{0}$ (we do not know anything better anyway). Then,

$$\boldsymbol{z}_1 = M^{-1}\boldsymbol{r}$$

- If one step of a stationary iteration is executed, then the M matrix of the underlying matrix splitting is the preconditioner of GMRES, PCG, or any other Krylov space method.

Preconditioning with stationary iterations (cont.)

- Let's execute $p > 1$ steps of the stationary iteration. Then

$$
\begin{aligned}
\boldsymbol{z}_1 &= M^{-1}\boldsymbol{r} \\
\boldsymbol{z}_2 &= \boldsymbol{z}_1 + M^{-1}(\boldsymbol{r} - A\boldsymbol{z}_1) = M^{-1}\boldsymbol{r} + M^{-1}(\boldsymbol{r} - AM^{-1}\boldsymbol{r}) \\
&= GM^{-1}\boldsymbol{r} + M^{-1}\boldsymbol{r}, \qquad G = I - M^{-1}A \\
\boldsymbol{z}_3 &= \boldsymbol{z}_2 + M^{-1}(\boldsymbol{r} - A\boldsymbol{z}_2) = G^2M^{-1}\boldsymbol{r} + GM^{-1}\boldsymbol{r} + M^{-1}\boldsymbol{r} \\
\boldsymbol{z}_p &= (I + G + G^2 + \cdots + G^{p-1})M^{-1}\boldsymbol{r}
\end{aligned}
$$

- Therefore, $M_{\text{eff}} = M(I + G + G^2 + \cdots + G^{p-1})^{-1}$.
  Of course, we do not form M_{eff} but proceed as above.
- Note that
  $M(I + G + G^2 + \cdots)^{-1} = M((I - G)^{-1})^{-1} = M(M^{-1}A) = A$.

CG for the normal equations (CGNE)

Can we get the benefits of the conjugate gradient (CG) algorithm for nonsymmetric A?

Maybe: instead of $A\boldsymbol{x} = b$ solve the normal equations

$$A^T A\boldsymbol{x} = A^T \boldsymbol{b}.$$

A related approach is to solve

$$AA^T \boldsymbol{y} = \boldsymbol{b}, \qquad \boldsymbol{x} = A^T \boldsymbol{y}.$$

Two severe issues:

1. Condition number $\kappa(AA^T) = \kappa(A^T A) = \kappa^2(A)$.
2. Each iteration step requires multiplication with A and A^T.

CG for the normal equations (CGNE) (cont.)

Consensus: CGNE used only if A is well conditioned and Matvec with A^T is cheap.

Important general observation: $M^T M$ can be an arbitrarily bad preconditioner for $A^T A$ irrespective of the quality of M as a preconditioner of A, see Wathen (2015).

Flexible GMRES

- So far we have considered the preconditioner to be fixed. It does not change from step to step.
- The formalism does not allow to change it. (Construction of Krylov space!)
- We could envision, however, to solve $A\boldsymbol{z} = \boldsymbol{r}$ to a given accuracy instead of just executing a fixed number of steps of some stationary iteration method.
- What happened if we used a Krylov space method to solve $A\boldsymbol{z} = \boldsymbol{r}$ approximately? This would be an *inner iteration*.
- Then, formally, we solved $M_j\boldsymbol{z} = \boldsymbol{r}$ in the j-th (outer) iteration step.

Flexible GMRES (cont.)

- In line 9 of right-preconditioned GMRES the solution $\boldsymbol{x}_m$ is expressed as a linear combination of preconditioned vectors $\boldsymbol{z}_j = M^{-1}\boldsymbol{v}_j$, $i = 1, \ldots, m$.
- The $\boldsymbol{z}_j$ are obtained by the $\boldsymbol{v}_j$ by multiplying with the same matrix M^{-1} whence the $\boldsymbol{z}_j$ need not be stored. Instead we apply M^{-1} to the linear combination of the $\boldsymbol{v}_j$.
- If the preconditioner could change at every step, then the $\boldsymbol{z}_j$ were given by

$$\boldsymbol{z}_j = M_j^{-1}\boldsymbol{v}_j.$$

Then it would be natural to compute the approximate solution as

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + [M_1^{-1}\boldsymbol{v}_1, \ldots, M_m^{-1}\boldsymbol{v}_m]\boldsymbol{y}_m = \boldsymbol{x}_0 + Z_m\boldsymbol{y}_m,$$

with $Z_m = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m]$.

The flexible GMRES algorithm (FGMRES)

1: Compute $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$, $\beta = \|\boldsymbol{r}_0\|_2$, and $\boldsymbol{v}_1 = \boldsymbol{r}_0/\beta$.
for $j = 1, \ldots, m$ **do**
  Compute $\boldsymbol{z}_j := M_j^{-1}\boldsymbol{v}_j$
  Compute $\boldsymbol{w} := A\boldsymbol{z}_j$
  for $i = 1, \ldots, j$ **do**
    $h_{i,j} = \boldsymbol{w}^T\boldsymbol{v}_i$
    $\boldsymbol{w} = \boldsymbol{w} - h_{i,j}\boldsymbol{v}_i$
  enddo
  Compute $h_{j+1,j} = \|\boldsymbol{w}\|_2$ and $\boldsymbol{v}_{j+1} = \boldsymbol{w}/h_{j+1,j}$
enddo
Define $Z_m := [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m]$, $\bar{H}_m = ((h_{i,j}))$
Compute $\boldsymbol{y}_m = \text{argmin}_{\boldsymbol{y}}\|\beta\boldsymbol{e}_1 - \bar{H}_m\boldsymbol{y}\|_2$ and $\boldsymbol{x}_m = \boldsymbol{x}_0 + Z_m\boldsymbol{y}_m$
If converged leave GMRES else set $\boldsymbol{x}_0 := \boldsymbol{x}_m$ and goto **1**

Discussion of FGMRES

- FGMRES is quite a simple modification of GMRES
- Flexibility may cause problems as the Z_m may be badly conditioned.
- There is a relation

$$AZ_m = V_{m+1}\bar{H}_m$$

  instead of the simpler $(AM^{-1})V_m = V_{m+1}\bar{H}_m$
- Provided that H_m is nonsingular we still have

$$\boldsymbol{b} - A\boldsymbol{z} = \boldsymbol{b} - A(\boldsymbol{x}_0 + Z_m\boldsymbol{y}) = V_{m+1}[\beta\boldsymbol{e}_1 - \bar{H}_m\boldsymbol{y}]$$

- So, the approximate solution $\boldsymbol{x}_m$ obtained at step m of FGMRES minimizes the residual norm $\|\boldsymbol{b} - A\boldsymbol{x}_m\|$ over $\boldsymbol{x}_0 + \text{span}(Z_m)$.

Discussion of FGMRES (cont.)

- FGMRES can break down.
- A breakdown occurs if the vector $\boldsymbol{v}_j$ cannot be computed because $h_{j+1,j} = 0$.
- For GMRES this was a happy event. In FGMRES this is different.
- Theorem. Assume that $\beta = \|\boldsymbol{r}_0\| \neq 0$ and that $j-1$ steps of FGMRES have been successfully performed, i.e., that $h_{i+1,i} \neq 0$ for $i < j$. In addition, assume that the matrix H_j is nonsingular. Then $\boldsymbol{x}_j$ is exact if and only if $h_{j+1,j} = 0$.
- The additional cost of the flexible variant over the standard algorithm is the additional vectors that have to be stored. This may be worth it.

Incomplete factorization preconditioners

- $M = A$ would be the ideal preconditioner.
- However, to solve with A we need to compute a factorization of A,

$$LU = A$$

that introduces fill-in.

- *Want to get as close to A as we can without allowing too much fill-in.*
- A general Incomplete LU (ILU) factorization process computes a sparse lower triangular matrix L and a sparse upper triangular matrix U such that the residual

$$R = LU - A$$

satisfies certain constraints such as having zero entries at certain locations.

General static Pattern ILU

- Let
$$P \subset \{(i,j) \mid i \neq j \ ; \ 1 \leq i,j \leq n\}$$
be a so-called zero pattern.
- We want compute an ILU factorization of A such that $l_{i,j} = 0$, $u_{i,j} = 0$ for all $(i,j) \in P$.
- We assume that if $a_{i,j} \neq 0$ in the original matrix A then $(i,j) \notin P$.
- In the following algorithms L and U are stored in A.
Since we know where the nonzeros of L/U will be, the memory layout can easily be prepared before the factorization starts.

IKJ variant of Gaussian elimination

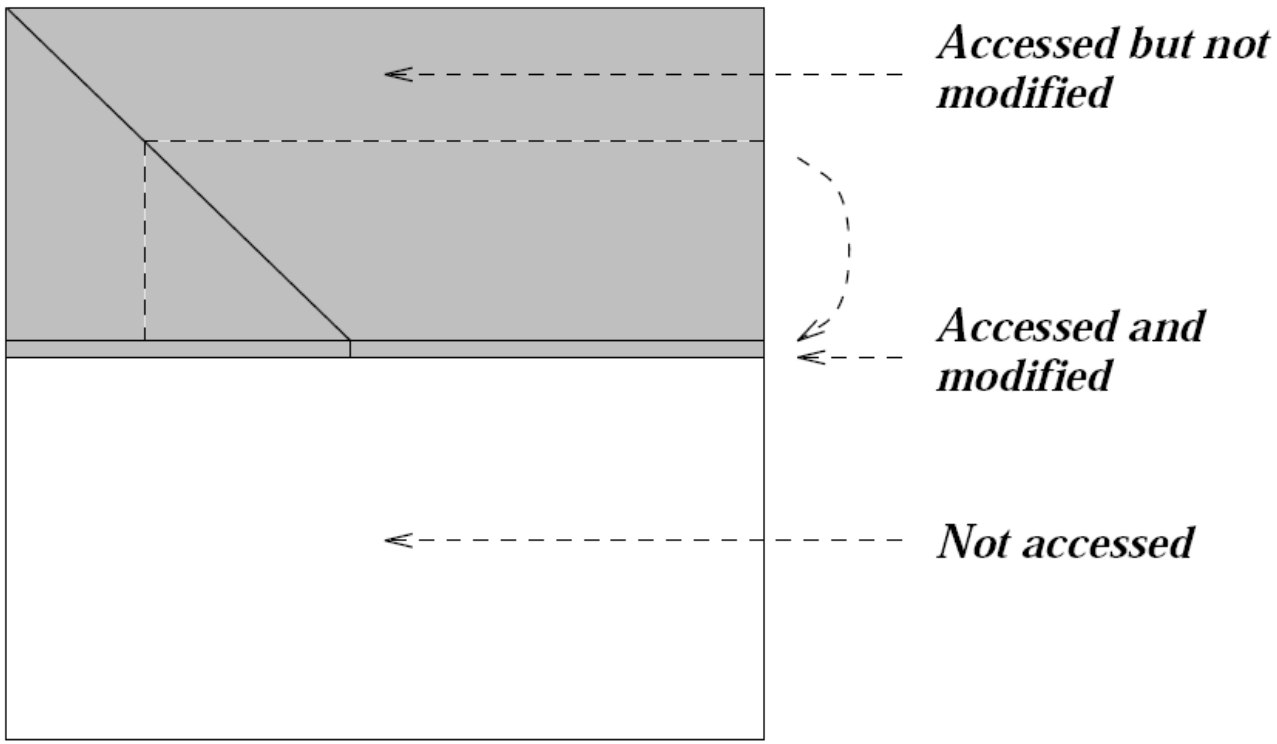


Image from Saad: Iterative methods (1st edition), p. 272.

General static ILU factorization, IKJ version

for $i = 2, \ldots, n$ **do**
  for $k = 1, \ldots, i-1$ and if $(i,k) \notin P$ **do**
    $a_{ik} = a_{ik}/a_{kk}$;
    for $j = k+1, \ldots, n$ and if $(i,j) \notin P$ **do**
      $a_{ij} = a_{ij} - a_{ik}a_{kj}$;
    enddo
  enddo
enddo

If we wanted to compute the residual matrix R there would be a statement $r_{ij} = r_{ij} + a_{ik}a_{kj}$ for $(i,j) \in P$.

ILU(0) / IC(0)

The most popular zero pattern is obtained by choosing P to be the zero pattern of the original matrix A:

$$P = \{(i,j) \mid a_{i,j} = 0\}$$

In this way, the sparsity structure of the incomplete factors is *a priori* determined to be the structure of the original matrix A. These preconditioners are called ILU(0) and IC(0) for the incomplete Cholesky variants. cg with the preconditioner IC(0) is called ICCG(0).

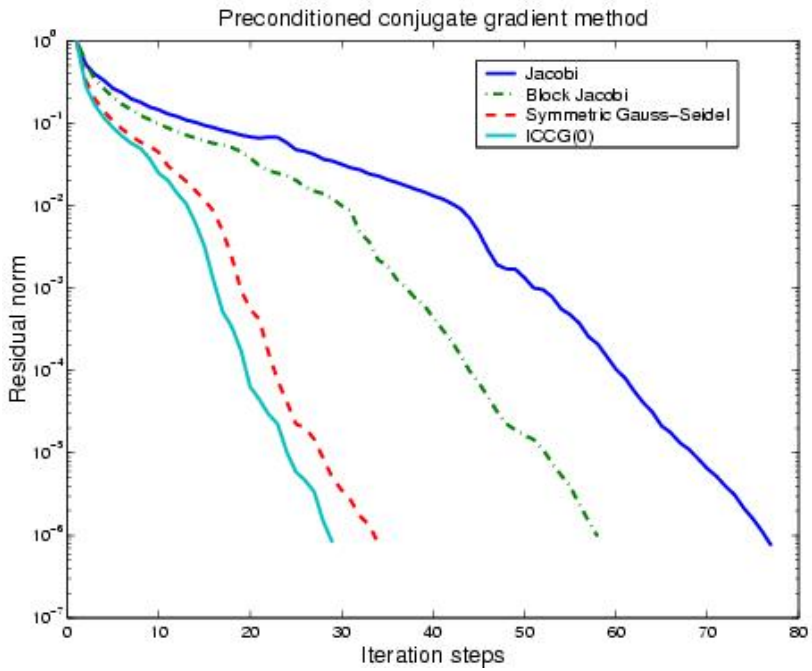
ILU(p) / IC(p)

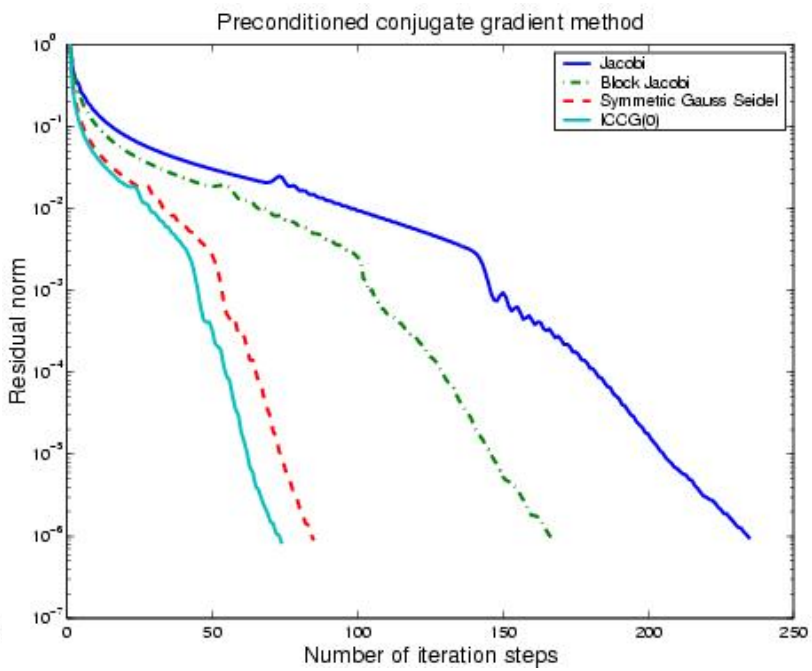
Static Incomplete LU/Cholesky factorizations with more fill-in exist. They require more computing time and more memory space.
Let L_0 be the IC factor of A. The sparse factor L_1 corresponding to IC(1) is obtained by accepting nonzeros at the nonzero positions of $L_0 L_0^T$.
ILU(p) / IC(p) are obtained in this recursive fashion. Do not use $p > 0$.
Matrices should be reordered before the incomplete factorization.

Numerical example

We solve $-\Delta u = f$ with homogeneous boundary conditions on the square by the Finite Difference method on a $m \times m$ grid, $m = 31$, $m = 101$. The iterative solver is PCG with 4 different preconditioners, see next slide.

Numerical example





Exec times (it steps)	Jacobi	Block Jacobi	Sym. GS	ICCG(0)
$m = 31$	0.45 (76)	1.23 (57)	0.34 (33)	0.28 (28)
$m = 101$	18.0 (234)	54.1 (166)	10.1 (84)	8.8 (73)

Dynamic nonzero patterns

- Incomplete factorizations that rely on the levels of fill p are blind to numerical values because elements that are dropped depend only on the structure of A.
- This can cause difficulties in realistic problems that arise in many applications.
- Alternative methods drop elements in the Gaussian elimination process according to their magnitude rather than their location.
- With these techniques the zero pattern P is determined dynamically.

The ILUT(p, τ) approach

- In the ILUT(p, τ) approach there are two strategies combined: Small elements are dropped and the number of elements per row of L and U are limited.
- The parameter τ is used to drop small elements: Set $a_{i,k} \leftarrow 0$ if it is less than tolerance $\tau_i = \tau \|\boldsymbol{a}_{i,:}\|$
- Limit the number of elements per row of L and U by keeping only the p largest (in modulus).
- Note that L and U have at most p nonzeros per row which eases the memory management considerably.
- Incomplete factorizations may not exist. Pivoting is possible ($\rightarrow$ ILUTP), for details see Saad, *Iterative methods for Sparse Linear Systems*, Chapter 10, both editions.
- MATLAB does not provide ILUT(p, τ).

Algorithm ILUT(p, τ)

for $i = 1, \ldots, n$ **do**
  $\boldsymbol{w} = \boldsymbol{a}_{i,:}$ // copy of i-th row of A
  for $k = 1, \ldots, i-1$ and if $w_k \neq 0$ **do**
    $w_k = w_k / a_{kk}$;
    Apply the dropping rule to w_k
    if $w_k \neq 0$ **then**
      $\boldsymbol{w} = \boldsymbol{w} - w_k \times \boldsymbol{u}_{k,:}$
    endif
  enddo
  Limit the number of nonzeros per row of L / U
  $l_{i,1:i-1} = w_{1:i-1}$
  $u_{i,i:n} = w_{i:n}$
enddo

Symmetric reorderings for ILU / IC

- The primary goal of reordering techniques is to reduce fill-in during Gaussian elimination.
- A good ordering for reducing fill-in may lead to factors of poor numerical quality (e.g., small pivots).
- For *incomplete* factorizations we may argue that fill-reducing permutations result in dropping fewer terms such that the sparse factors are more accurate.
- In general it is advisable to apply RCM or MD reordering before the factorization.

Reordering for ILU / IC

- In a second category of reorderings row-permutations are applied to avoid poor pivots in Gaussian elimination.
- More precisely, we are looking for a permutation π or corresponding permutation matrix Q_π such that

$$B = Q_\pi A$$

  has large entries on the diagonal. The hope is that the pivots are mostly on the diagonal and few row permutations are needed.
- More details are in Lecture 6.

Polynomial preconditioning

A preconditioner of the form

$$M^{-1} = s(A) = \sum_{j=0}^{m-1} \alpha_j A^j$$

is called a polynomial preconditioner. The polynomial $s(A)$ should approximate A^{-1}, i.e., $s(\lambda) \approx \lambda^{-1}$ for $\lambda \in \sigma(A)$.
Such a preconditioner is easy to implement, in particular, on parallel or vector processors.
In a sequential environment polynomial preconditioners are not recommended as the same work can be used to extend a Krylov subspace in a CG or GMRES iteration. (Here, work corresponds to matvec's.)

Neumann polynomials

If $\|N\| < 1$ then

$$(I - N)^{-1} = \sum_{j=0}^{\infty} N^j \qquad \text{(Neumann series)}$$

Let $\|A\| < 1/\omega$. Then $\|\omega A\| < 1$ and

$$(\omega A)^{-1} = (I - (I - \omega A))^{-1} = \sum_{j=0}^{\infty} (I - \omega A)^j$$

and

$$M^{-1} = \sum_{j=0}^{k} (I - \omega A)^j \quad \Longleftrightarrow \quad s(\lambda) = \sum_{j=0}^{k} (1 - \omega\lambda)^j.$$

The preconditioner is applied using Horner's rule.

Chebyshev polynomials

For stationary iterations

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + M^{-1}\boldsymbol{r}_k$$

the error satisfies

$$\boldsymbol{e}_{k+1} = \boldsymbol{e}_k - M^{-1}A\boldsymbol{e}_k = (I - M^{-1}A)\boldsymbol{e}_k$$

So, we may try to find a polynomial s of degree k such that

$$\max_{\lambda\in\sigma(A)} |1 - \lambda s(\lambda)|$$

is minimized. Since this problem is too hard to solve we relax it and try to find a polynomial $s \in \mathbb{P}_k$ such that

$$\max_{\lambda\in(\alpha,\beta)} |1 - \lambda s(\lambda)| = \max_{\lambda\in(\alpha,\beta),s(0)=1} |s(\lambda)| \tag{3}$$

is minimized, where $\sigma(A) \subset (\alpha, \beta)$, $0 < \alpha \leq \beta$.

This minimizing problem for p is solved by the Chebyshev polynomial $T_k(t;\alpha,\beta)$ shifted to the interval (α,β) and scaled such that $T_k(0;\alpha,\beta)=1$:

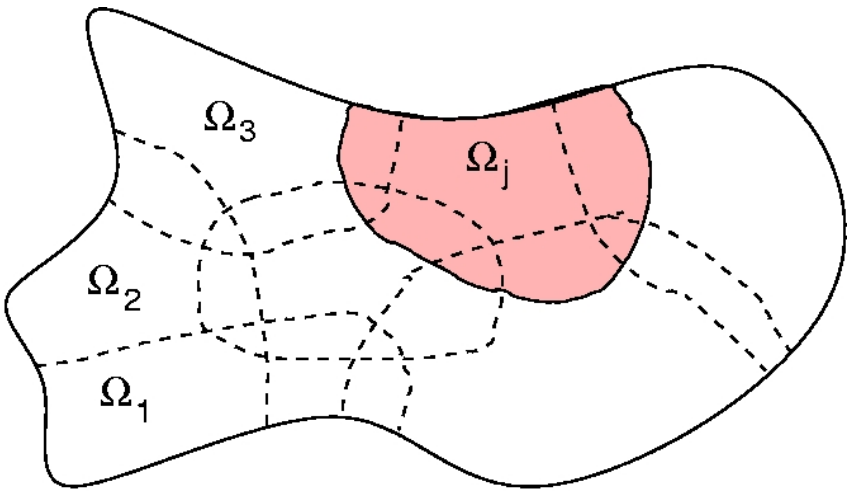
$$T_k(t;\alpha,\beta)=\frac{T_k\left(\frac{\beta+\alpha-2t}{\beta-\alpha}\right)}{T_k\left(\frac{\beta+\alpha}{\beta-\alpha}\right)}.$$

The preconditioner is applied using the 3-term recurrence for Chebyshev polynomials,

$$T_{k+1}(t)=2tT_k(t)-T_{k-1}(t),\qquad T_1(t)=t,\ T_0(t)=1.$$

Note that this is the Chebyshev iteration of last week.
Note also that a different norm in (3), e.g. $\|\cdot\|_2$ instead of $\|\cdot\|_\infty$, will lead to different polynomials.
Ref.: Saad: Iterative methods for sparse linear systems. SIAM 2003.

Domain decomposition



Let's assume that we want to solve the Poisson equation $-\Delta u(\boldsymbol{x}) = f(\boldsymbol{x})$ (with Dirichlet boundary conditions $u = g$) in some domain Ω.

Let's further assume that we have an approximation $v \approx u$ that satisfies the boundary conditions. We want to correct v by some e that is nonzero only in Ω_j such that $v + e$ better approximates u. Then we solve

$$-\Delta e = f + \Delta v, \qquad \boldsymbol{x} \in \Omega_j$$
$$e = 0 \text{ on } \partial\Omega_j.$$

Let $A\boldsymbol{x} = \boldsymbol{b}$ be a FE or FD discretization of a PDE, like on the previous slide. We decompose the underlying domain Ω in subdomains Ω_j, $j = 1, \ldots, d$, such that $\Omega = \cup\Omega_j$.
Thus, *each grid point is in at least one subdomain*
(overlapping vs. non-overlapping domains).
Let R_j^T be the projector that extracts from a vector those components that belong to subdomain Ω_j. (The columns of R_j are columns of the identity matrix.) Then we write

$$A|_{\Omega_j} = A_j = R_j^T A R_j$$
$$(\boldsymbol{b} - A\boldsymbol{x}_k)|_{\Omega_j} = R_j^T(\boldsymbol{b} - A\boldsymbol{x}_k)$$

Note. The R_j is a generalization of the topology maps that we encountered when mapping the local dof's in the reference triangle to the global dof's in the actual triangle.

If we apply the above procedure for $j = 1, \ldots, d$, i.e., improve solutions in subdomain Ω_j, one after the other, then we can written this as

$$\boldsymbol{x}_{k+\frac{j}{d}} = \boldsymbol{x}_{k+\frac{j-1}{d}} + \underbrace{R_j (R_j^T A R_j)^{-1} R_j^T}_{B_j} \underbrace{(\boldsymbol{b} - A\boldsymbol{x}_{k+\frac{j-1}{d}})}_{\boldsymbol{r}_{k+\frac{j-1}{d}}},$$

$$= \boldsymbol{x}_{k+\frac{j-1}{d}} + B_j \, \boldsymbol{r}_{k+\frac{j-1}{d}}, \qquad j = 1, \ldots, d.$$

This is called a multiplicative Schwarz procedure[1].
If there are no overlaps, it is a block Gauss–Seidel iteration and converges (as a stationary method) for SPD matrices.

[1] H.A. Schwarz: Vierteljahresschrift der Naturforschenden Gesellschaft Zürich 15 (1870), 272–286.

Let $d = 2$. Then

$$\boldsymbol{x}_{k+1/2} = \boldsymbol{x}_k + B_1 \boldsymbol{r}_k,$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_{k+1/2} + B_2 \boldsymbol{r}_{k+1/2}.$$

Combining the two steps gives,

$$\begin{aligned}
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_{k+1/2} + B_2 \boldsymbol{r}_{k+1/2} \\
&= \boldsymbol{x}_k + B_1 \boldsymbol{r}_k + B_2(\boldsymbol{b} - A\boldsymbol{x}_{k+1/2}) \\
&= \boldsymbol{x}_k + B_1 \boldsymbol{r}_k + B_2(\underbrace{\boldsymbol{b} - A\boldsymbol{x}_k}_{\boldsymbol{r}_k} + AB_1 \boldsymbol{r}_k) \\
&= \boldsymbol{x}_k + \underbrace{(B_1 + B_2 - B_2 A B_1)}_{M^{-1}} \boldsymbol{r}_k
\end{aligned}$$

For the iteration matrix we have

$$I - M^{-1}A = I - B_1 A - B_2 A + B_2 A B_1 A = (I - B_2 A)(I - B_1 A)$$

In general, for d domains, we have

$$I - M^{-1}A = (I - B_d A)(I - B_{d-1}A) \cdots (I - B_2 A)(I - B_1 A).$$

The preconditioner M is not symmetric! A simple remedy is a second sweep through the domains in reversed order. (The domain d does not have to be treated twice.)
For $d = 2$ we have

$$I - M^{-1}A = (I - B_1 A)(I - B_2 A)(I - B_1 A),$$

i.e.,

$$M^{-1} = (I - (I - B_1 A)(I - B_2 A)(I - B_1 A))A^{-1}.$$

This procedure is similar to the symmetrization of SOR or Gauss–Seidel.

Let us consider the factors $(I - B_j A)$ of the iteration matrix $I - M^{-1}A$. Let $P_j = B_j A$. Then,

$$B_j \boldsymbol{r} = R_j (R_j^T A R_j)^{-1} R_j^T \boldsymbol{r} = R_j (R_j^T A R_j)^{-1} R_j^T A A^{-1} \boldsymbol{r} = P_j \boldsymbol{e} =: \boldsymbol{e}_j.$$

P_j is a projector on $\mathcal{R}(R_j)$:

$$P_j^2 = R_j (R_j^T A R_j)^{-1} R_j^T A R_j (R_j^T A R_j)^{-1} R_j^T A = R_j (R_j^T A R_j)^{-1} R_j^T A = P_j$$

If A is SPD then $\langle \boldsymbol{x}, \boldsymbol{y} \rangle_A$ is an inner product and we have

$$\langle P_j \boldsymbol{x}, \boldsymbol{y} \rangle_A = \boldsymbol{x}^T P_j^T A \boldsymbol{y} = \boldsymbol{x}^T A B_j A \boldsymbol{y} = \langle \boldsymbol{x}, P_j \boldsymbol{y} \rangle_A.$$

So, P_j is symmetric (w.r.t. to the A-inner product).
Altogether, P_j is an A-orthogonal projector on $\mathcal{R}(R_j)$.

Likewise,

$$I - P_j = I - B_j A$$

is an A-orthogonal projector onto the A-orthogonal complement of $\mathcal{R}(R_j)$ which we denote by $\mathcal{R}(R_j)^{\perp_A}$.

We can interpret the formula

$$\boldsymbol{e}_{k+1} = (I - M^{-1}A)\boldsymbol{e}_k = (I - B_d A)\cdots(I - B_2 A)(I - B_1 A)\boldsymbol{e}_k.$$

as follows.
P_j projects the error $\boldsymbol{e}$ onto $\boldsymbol{e}_j$ which is the vector in $\mathcal{R}(R_j)$ closest to $\boldsymbol{e}$. Then, this 'local' error is subtracted from the global error.

So, $I - B_j A$ reduces the error in $\mathcal{R}(R_j)$ in the 'best possible way'.

This is done in turn for all subdomains $j = 1, \ldots, d$.

Additive Schwarz

An alternative method is the additive Schwarz procedure:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \sum_{j=1}^{d} B_j \boldsymbol{r}_k.$$

$M^{-1} = \sum_{j=1}^{d} B_j$ is clearly symmetric.

- Notice that additive Schwarz is actually block Jacobi if the domains do not overlap.
- Additive Schwarz as a stationary method with overlapping domains often does not converge.

Simple 1D example

Let's consider the 1D Poisson equation

$$-u''(x) = f(x), \qquad u(0) = u(1) = 0,$$

discretized by $P1$ finite elements on an equidistant grid with 6 interior points.

$$\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & -1 & 2 & -1 & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = A\boldsymbol{x} = \boldsymbol{b} = h^2 \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{pmatrix}.$$

Let I be the 6×6 identity matrix.

Simple 1D example (cont.)

(1) DD without overlap. Set $R_1 = I(:, 1:3)$, $R_2 = I(:, 4:6)$, $B_j = R_j(R_j^T A R_j)^{-1} R_j^T$ and $M^{-1} = B_1 + B_2$.

M is the block Jacobi preconditioner with 3×3 blocks.
We have $\rho(G) = \rho(I - M^{-1}A) < 1$ and thus convergence.

(2) DD with overlap (1). Set $R_1 = I(:, 1:4)$, $R_2 = I(:, 3:6)$ and the rest as in **(1)**.

Then, $\rho(G) = \rho(I - M^{-1}A) = 1$ and no convergence.

(3) DD with overlap (2). R_1, R_2 as in **(2)** and the rest as in **(1)**.
Set, $D_1 = \text{diag}([1, 1, 2/3, 1/3])$ and $D_2 = \text{diag}([1/3, 2/3, 1, 1])$

Then, $R_1 D_1 R_1^T + R_2 D_2 R_2^T = I$ (partition of unity)

Set $M^{-1} = R_1 D_1 (R_1^T A R_1)^{-1} R_1^T + R_2 D_2 (R_2^T A R_2)^{-1} R_2^T$.

Then, $\rho(G) < 1$ and we have convergence.

Restricted additive Schwarz (RAS) preconditioner

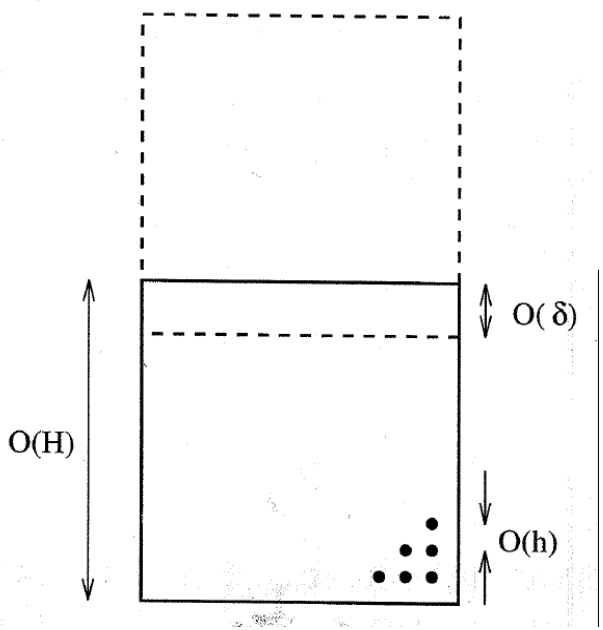
Let Ω_j, $j = 1, \ldots, d$, be a covering partition of Ω, $\Omega = \cup\Omega_j$.
Let $\mathcal{N}_j$ be the set of indices associated with degrees of freedom in Ω_j. We define the diagonal matrix D_j, $j = 1, \ldots, d$, by

$$(D_j)_{ii} = \begin{cases} 1, & i \in \mathcal{N}_j, i \notin \mathcal{N}_k \text{ for } k \neq j, \\ 1/M_i, & i \in \mathcal{N}_j, i \in \mathcal{N}_k \text{ for } M_i \text{ subdomains } \Omega_k, \\ 0, & \text{otherwise.} \end{cases}$$

Then, $\sum_{j=1}^{d} R_j D_j R_j^T = I$.
The restricted additive Schwarz (RAS) preconditioner is defined by

$$M^{-1} = \sum_{j=1}^{d} R_j D_j (R_j^T A R_j)^{-1} R_j^T.$$

Convergence



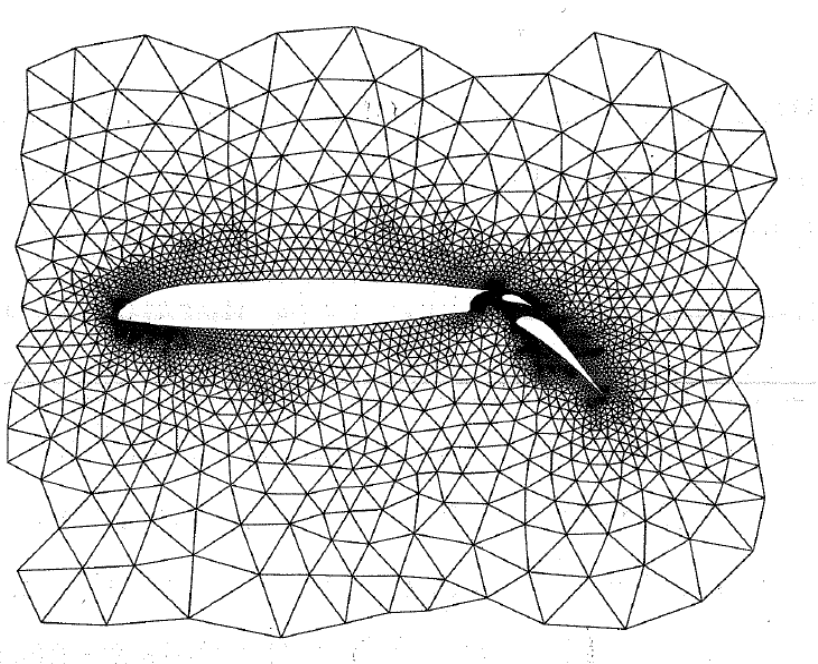
- Number of iterations (iteration count) grows with $1/H$.
- If δ proportional to H: # its bounded indept. of h and H/h.
- # its (multiplicative Schwarz) $\approx \frac{1}{2}$ # its (additive Schwarz)
- Convergence poor if $\delta = 0$ (Jacobi), increases rapidely as δ increases.

Numerical example

[taken from Smith/Bjørstad/Gropp]

Poisson equation with Dirichlet boundary conditions

$$-\Delta u = xe^y \quad \text{in } \Omega, \qquad u = -xe^y \quad \text{on } \partial\Omega,$$



Ω is either a unit square with $N \times N$ grid points or an unstructured grid.

GMRES(10) with DD preconditioner.

Comparison with ILU(τ) and SSOR preconditioner.

Partitioning by METIS.

Numerical example (cont.)

Iteration Counts for 10^{-2} Reduction in Error, for Unit Square

		Multiplicative with Overlap			Additive with Overlap		
N	Decomposition	1	2	4	1	2	4
33	2,1	3	3	2	5	5	4
33	4,1	4	4	3	7	6	5
33	8,1	8	6	4	9	8	6
33	2,2	4	3	3	7	6	4
33	ILU				10		
33	SSOR				18		
65	2,1	4	4	3	7	6	5
65	4,1	6	5	4	10	8	6
65	8,1	8	8	6	16	12	9
65	2,2	5	4	3	10	8	7
65	ILU				19		
65	SSOR				62		

Numerical example (cont.)

Iteration Counts for 10^{-2} Reduction in Error, for Unstructured Grid

Overlap	Domains	Multiplicative	Additive
1	2	4	7
1	4	4	8
1	8	5	10
1	16	5	12
2	2	3	5
2	4	3	7
2	8	4	7
2	16	4	8
4	2	2	3
4	4	2	5
4	8	3	6
4	16	3	6
ILU			13
SSOR			18

Parallelizing the multiplicative Schwarz procedure

The multiplicative Schwarz procedure is related to Gauss-Seidel in that always the most recent values are used for computing the residuals. Thus, the problems with parallelizing multiplicative Schwarz are related to parallelizing Gauss-Seidel and the solution is the same: multi-coloring

If we have q colors then

$$\boldsymbol{x}^{(k+1/q)} = \boldsymbol{x}_k + \sum_{j \in \text{color}_1} B_j \boldsymbol{r}_k$$

$$\boldsymbol{x}^{(k+2/q)} = \boldsymbol{x}^{(k+1/q)} + \sum_{j \in \text{color}_2} B_j \boldsymbol{r}^{(k+1/q)}$$

$$\vdots$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}^{(k+\frac{q-1}{q})} + \sum_{j \in \text{color}_q} B_j \boldsymbol{r}^{(k+\frac{q-1}{q})}$$

Coarse grid correction

- Domain decomposition (DD) preconditioning with domain size one is ordinary Jacobi or Gauss–Seidel preconditioning.
- *Non-overlapping* DD preconditioning corresponds to ordinary *block* Jacobi and *block* Gauss–Seidel preconditioning, resp.
- Convergence behavior of DD preconditioning is similar:
  1. The iteration count increases with problem size.
  2. With fixed problem size: the iteration count increases with increased number of subdomains (parallelism).
- DD preconditioners are sophisticated smoothers. But they do not reduce highly oszillating error components.
- Remedy: Coarse grid correction.

Coarse grid correction: the procedure

Let Z be a rectangular matrix with columns that approximate the 'slow modes' of the SPD matrix A.
We want to improve an approximation $\boldsymbol{y} \approx \boldsymbol{x}^*$ by a vector in $\mathcal{R}(Z)$.

$$\min_{\boldsymbol{d}} \|A(\boldsymbol{y} + Z\boldsymbol{d}) - \boldsymbol{b}\|_{A^{-1}}$$

$$\iff \quad \min_{\boldsymbol{d}} \boldsymbol{d}^T Z^T A Z \boldsymbol{d} + 2(A\boldsymbol{y} - \boldsymbol{b})^T Z \boldsymbol{d} + \text{const}$$

$$\implies \quad Z\boldsymbol{d} = Z(Z^T A Z)^{-1} Z^T (\boldsymbol{b} - A\boldsymbol{y}).$$

Complement a multiplicative or additive DD preconditioner by a coarse grid correction, e.g., with $R_0 = Z$

$$M^{-1}_{\text{RAS},2} = R_0 (R_0^T A R_0)^{-1} R_0^T + \sum_{j=1}^{d} R_j D_j (R_j^T A R_j)^{-1} R_j^T.$$

Coarse grid correction: the procedure (cont.)

How do we choose Z. A number of variants exist.
Main idea: approximate the lowest mode(s). For the Poisson equation this is the constant function.

Nicolaides coarse space:
Let D_j, $j = 1, \ldots, d$, be the diagonal matrices defined earlier for the restricted additive Schwarz preconditioner. We have,

$$\sum_{j=1}^{d} R_j D_j R_j^T = I.$$

Then, we we define $Z = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_d]$ columnwise by

$$\boldsymbol{z}_j := R_j D_j R_j^T \boldsymbol{e}, \qquad \boldsymbol{e} = [1, 1, \ldots, 1]^T.$$

Coarse grid correction: the procedure (cont.)

A procedure for a multiplicative DD preconditioner with a multiplicative coarse grid correction could look as follows.

$\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{x}$;
for j:=1 to d **do**
  $\boldsymbol{x} = \boldsymbol{x} + R_j(R_j^T A R_j)^{-1} R_j^T \boldsymbol{r}$;
  $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{x}$;
end for
$\boldsymbol{x} = \boldsymbol{x} + R_0(R_0^T A R_0)^{-1} R_0^T \boldsymbol{r}$;

Notice that one can combine, e.g., the coarse grid correction multiplicatively with an additive Schwarz preconditioner.

A number of variants are possible. See the book by Smith, Bjørstad, and Gropp.

Experiments: $-\Delta u = x \cdot e^y$ on unit square

problem size	overlap 0	1	2	overlap 0	1	2	overlap 0	1	2
	RAS,1			GMRES RAS,1			GMRES MS,1		
40×40	288	150	103	44	33	24	20	15	11
80×80	515	269	182	59	44	38	28	20	16
160×160	920	484	324	103	64	51	40	28	23
	RAS,2			GMRES RAS,2			GMRES MS,2		
40×40	62	40	32	17	14	12	15	11	9
80×80	113	73	57	25	20	18	20	16	13
160×160	205	133	103	36	28	25	27	21	18

4×4 domains, `tol`$=10^{-5}$, `restart`$=10$.

Multiplicative coarse grid correction

Weak scalability test: $-\Delta u = x \cdot e^y$ on unit square

Solve problem with $p \times p$ equally sized subdomains (size 21×21).

domains	n	GMRES MS,1 overlap=0		GMRES MS,1 overlap=2		GMRES MS,2 overlap=2	
		its	time	its	time	its	time
2×2	1600	11	0.018	7	0.011	7	0.013
4×4	6084	31	0.14	15	0.070	12	0.067
6×6	13456	36	0.42	25	0.30	15	0.20
8×8	23716	67	2.2	37	1.4	15	0.63
10×10	36864	90	5.8	40	3.1	16	1.6
12×12	52900	112	15	59	8.8	16	2.2
16×16	93636	175	69	88	36	16	7.4

time is in seconds, `tol`$=10^{-5}$, `restart`$=10$.