Iterative methods: from theory to practice (A tutorial)

Yousef Saad, Ruipeng Li, Yuanzhe Xi
(Minnesota) (LLNL) (Emory)

Copper Mountain Conference on Iterative Methods
March 31, 2022
**Schedule**

- In short: Two ‘core’ lectures of 50mn each, two supplemental lectures of 35mn each, and a 10mn break in middle.

<table>
<thead>
<tr>
<th>Time</th>
<th>Speaker</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>10:50–11:25</td>
<td>Y. Xi</td>
<td>Application: GMRES/Anderson mixing for GANs; Polynomial filtering - Eigenvalue Pbs</td>
</tr>
<tr>
<td>11:25-11:35</td>
<td></td>
<td>Break [coffee / quick lunch time]</td>
</tr>
</tbody>
</table>

- All times are in MST time zone – [Same as in official program]
Introduction: Linear System Solvers

Problem considered:
Linear systems of equations

Can view the problem from somewhat different angles:

• Discretized problem coming from a PDE
• An algebraic system of equations [ignore origin]
• System of equations where $A$ is not explicitly available

We consider: Second viewpoint + $A$ is Sparse

Problem can be seen in virtually every scientific or engineering application: Fluid Dynamics, Chemical reactions, Equilibrium models (economics), circuit/device simulation, .....
Solving sparse systems today

Direct sparse Solvers

A x = b

-Δ u = f + bc

Iterative Methods
Preconditioned Krylov

General Purpose

Specialized

Fast Poisson Solvers

Multigrid Methods
Background. Three types of methods:

- Direct methods: based on sparse Gaussian elimination, sparse Cholesky,
- Iterative methods: compute a sequence of iterates which converge to the solution - preconditioned Krylov methods,
- Special purpose methods: Multigrid, Fast-Poisson solvers, ...

Remark: The first 2 classes of methods have always been in competition.
Quotation from R. Varga’s book on iterative methods [1962]

“As an example of the magnitude of problems that have been successfully solved by cyclic iterative methods, the Bettis Atomic Power Laboratory of the Westinghouse Electric Corporation had in daily use in 1960 a 2-dimensional program which would treat as a special case Laplacean-type matrix equations of order 20,000.”

He adds in footnote: (paraphrase) the program was written for the Philco-2000 computer which had 32,000 words of core storage (!). “Even more staggering”: Bettis had a 3-D code which could treat coupled matrix equations of order 108,000.

➢ Today: tens of millions is common, hundreds of millions, to billions not too uncommon
Long standing debate: direct vs. iterative

- Starting in the 1970’s: huge progress of sparse direct solvers
- Iterative methods - much older - not designed for ‘general systems’. Big push in the 1980s with help from ‘preconditioning’
- General consensus now: Direct methods do well for 2-D problems and some specific applications [e.g., structures, ...]
- Usually too expensive for 3-D problems
- Huge difference between 2-D and 3-D case
- Test: Two Laplacean matrices of same dimension $n = 122, 500$. First: on a $350 \times 350$ grid (2D); Second: on a $50 \times 50 \times 49$ grid (3D)
Pattern of a similar [much smaller] coefficient matrix

Finite Diff. Laplacean 30x30

Finite Diff. Laplacean 10x10x9

demo_2Dvs3D.m
SPARSE MATRICES ; DATA STRUCTURES
What are sparse matrices?

Common definition: “..matrices that allow special techniques to take advantage of the large number of zero elements and the structure.”

A few applications of sparse matrices: Structural Engineering, Reservoir simulation, Electrical Networks, optimization problems, ...

Goals: Much less storage and work than dense computations.

Observation: $A^{-1}$ is usually dense, but $L$ and $U$ in the LU factorization may be reasonably sparse (if a good technique is used).
Sample sparsity patterns

ARC130: Unsymmetric matrix from laser problem. a.r.curtis, oct 1974

SHERMAN5: fully implicit black oil simulator 16 by 23 by 3 grid, 3 unk
Sparse matrices in Matlab

- Explore the scripts Lap2D, mark (provided in matlab suite) for generating sparse matrices.
- Explore the commands spy, sparse
- demo_sparse0 and demo_mark
- Load a matrix can_445 from the SuiteSparse collection. Show its pattern
Main goal of Sparse Matrix Techniques: To perform standard matrix computations economically, i.e., without storing the zeros.

Example: To add two square dense matrices of size $n$ requires $O(n^2)$ operations. To add two sparse matrices $A$ and $B$ requires $O(nnz(A) + nnz(B))$ where $nnz(X)$ = number of nonzero elements of a matrix $X$.

For typical Finite Element /Finite difference matrices, number of nonzero elements is $O(n)$. 
**Data structures: The coordinate format (COO)**

\[
A = \begin{pmatrix}
1. & 0. & 0. & 2. & 0. \\
3. & 4. & 0. & 5. & 0. \\
6. & 0. & 7. & 8. & 9. \\
0. & 0. & 10. & 11. & 0. \\
0. & 0. & 0. & 0. & 12.
\end{pmatrix}
\]

- Also known as ‘triplet format’
- Simple data structure - Often used as 'entry' format in packages
- Variant used in matlab
- Note: order of entries is arbitrary [in matlab: sorted by columns]
**Compressed Sparse Row (CSR) format**

\[ A = \begin{pmatrix} 12. & 0. & 0. & 11. & 0. \\ 10. & 9. & 0. & 8. & 0. \\ 7. & 0. & 6. & 5. & 4. \\ 0. & 0. & 3. & 2. & 0. \\ 0. & 0. & 0. & 0. & 1. \end{pmatrix} \]

- IA(j) points to beginning or row j in arrays AA, JA
- Related: Compressed Sparse Column format, Modified Sparse Row format (MSR).
- Used predominantly in Fortran & portable codes [e.g. Metis] – what about C?
CSR (CSC) format - C-style

* CSR: Collection of pointers of rows & array of row lengths

typedef struct SpaFmt {
    int n; /* size of matrix */
    int *nzcount; /* length of each row */
    int **ja; /* to store column indices */
    double **ma; /* to store nonzero entries */
} SparMat;

aa[i][*]  == entries of i-th row (col.);
ja[i][*]  == col. (row) indices,
nzcount[i] == number of nonzero elmts in row (col.) i
Data structure used in C sparse [T. Davis’ SuiteSparse code]

typedef struct cs_sparse
{ /* matrix in compressed-column or triplet form */
    int nzmax ; /* maximum number of entries */
    int m ;     /* number of rows */
    int n ;     /* number of columns */
    int *p ;    /* column pointers (size n+1) or 
                 col indices (size nzmax) */
    int *i ;    /* row indices, size nzmax */
    double *x ; /* numerical values, size nzmax */
    int nz ;    /* # of entries in triplet matrix, 
                 -1 for compressed-col */
} cs ;

- Can be used for CSR, CSC, and COO (triplet) storage
- Easy to use from Fortran
Computing $y = Ax$ – row and column storage

**Row-form:**
Dot product of $A(i,:)$ and $x$ gives $y_i$

**Column-form:**
Linear combination of columns $A(:,j)$ with coefficients $x_j$ yields $y$
void matvec( csptr mata, double *x, double *y )
{
    int i, k, *ki;
    double *kr;
    for (i=0; i<mata->n; i++) {
        y[i] = 0.0;
        kr = mata->ma[i];
        ki = mata->ja[i];
        for (k=0; k<mata->nzcount[i]; k++)
            y[i] += kr[k] * x[ki[k]];
    }
}

- Uses sparse dot products (sparse SDOTS)

- Operation count
Matvec – Column version

```c
void matvecC( csptr mata, double *x, double *y )
{
    int n = mata->n, i, k, *ki;
    double *kr;
    for (i=0; i<n; i++)
        y[i] = 0.0;
    for (i=0; i<n; i++) {
        kr = mata->ma[i];
        ki = mata->ja[i];
        for (k=0; k<mata->nzcount[i]; k++)
            y[k+ki] += kr[k] * x[i];
    }
}
```

Uses sparse vector combinations (sparse SAXPY)

Operation count
Using the CS data structure from Suite-Sparse:

```c
int cs_gaxpy (cs *A, double *x, double *y) {
    n = A->n; Ap = A-> p; Ai = A->i; Ax = A->x;
    for (j=0; j<n; j++) {
        for (p=Ap[j]; p<Ap[j+1];p++)
            y[Ai[p]] += Ax[p]*x[j];
    }
    return(1)
}
```
GRAPH MODELS
Graph Representations of Sparse Matrices. Recall:

Adjacency Graph $G = (V, E)$ of an $n \times n$ matrix $A$:

- $V = \{1, 2, \ldots, N\}$
- $E = \{(i, j) | a_{ij} \neq 0\}$

$G$ is undirected if $A$ has a symmetric pattern.

Example:

Matrix:
\[
\begin{bmatrix}
\ast & \ast & \ast & \ast \\
\ast & \ast & \ast & \ast \\
\ast & \ast & \ast & \ast \\
\ast & \ast & \ast & \ast \\
\end{bmatrix}
\]

Graph:

\[1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1\]
Reorderings and graphs

Let \( \pi = \{i_1, \cdots, i_n\} \) a permutation

\[ A_{\pi,*} = \{a_{\pi(i),j}\}_{i,j=1,...,n} = \text{matrix } A \text{ with its } i\text{-th row replaced by row number } \pi(i). \]

\[ A_{*,\pi} = \text{matrix } A \text{ with its } j\text{-th column replaced by column } \pi(j). \]

Define \( P_{\pi} = I_{\pi,*} \) = “Permutation matrix” – Then:

1. Each row (column) of \( P_{\pi} \) consists of zeros and exactly one “1”
2. \( A_{\pi,*} = P_{\pi}A \)
3. \( P_{\pi}P_{\pi}^T = I \)
4. \( A_{*,\pi} = AP_{\pi}^T \)
Consider now:  
\[ A' = A_{\pi,\pi} = P_{\pi} A P_{\pi}^T \]

- Element in position \((i, j)\) in matrix \(A'\) is exactly element in position \((\pi(i), \pi(j))\) in \(A\).  
  \[ a'_{ij} = a_{\pi(i),\pi(j)} \]

\[(i, j) \in E_{A'} \iff (\pi(i), \pi(j)) \in E_A\]

General picture:

\[ \pi(i) \quad \text{‘Old labels’} \quad \pi(j) \quad \text{‘New labels’} \]

\[ i \quad \text{‘Old labels’} \quad j \]
Example: A $9 \times 9$ 'arrow' matrix and its adjacency graph.

Fill-in?
Graph and matrix after swapping nodes 1 and 9:

Fill-in?
BASIC RELAXATION METHODS
Basic Relaxation Schemes

Relaxation schemes: based on the decomposition \( A = D - E - F \)

\[ D = \text{diag}(A), \quad -E = \text{strict lower part of } A \]

\[ -F = \text{its strict upper part}. \]

- For example, Gauss-Seidel iteration:
  \[
  (D - E)x^{(k+1)} = Fx^{(k)} + b
  \]

- Most common techniques 60 years ago.

- Now: used as smoothers in Multigrid or as preconditioners

Note: If \( \rho_i^{(k)} = i\text{th component of current residual } b - Ax \) then ‘relaxation’ form of GS is:

\[
\xi_i^{(k+1)} = \xi_i^{(k)} + \frac{\rho_i^{(k)}}{a_{ii}}
\]

for \( i = 1, \ldots, n \)
Jacobi, Gauss-Seidel, SOR, & SSOR iterations are of the form

\[ x^{(k+1)} = M x^{(k)} + f \]

- \( M_{Jac} = D^{-1}(E + F) = I - D^{-1}A \)
- \( M_{GS}(A) = (D - E)^{-1}F = I - (D - E)^{-1}A \)
- SOR relaxation: \( \xi_i^{(k+1)} = \omega \xi_i^{(GS,k+1)} + (1 - \omega)\xi_i^{(k)} \)
- \( M_{SOR}(A) = (D - \omega E)^{-1}(\omega F + (1 - \omega)D) = I - (\omega^{-1}D - E)^{-1}A \)

Matlab: take a look at: \( gs.m, sor.m, \) and \( sorRelax.m \) in iters/
Projection Methods

- The main idea of projection methods is to extract an approximate solution from a subspace.

- We define a subspace of approximants of dimension $m$ and a set of $m$ conditions to extract the solution.

- These conditions are typically expressed by orthogonality constraints.

- This defines one basic step which is repeated until convergence (alternatively the dimension of the subspace is increased until convergence).

Example: Each relaxation step in Gauss-Seidel can be viewed as a projection step.
Initial Problem: $b - Ax = 0$

Given two subspaces $K$ and $L$ of $\mathbb{R}^N$ define the approximate problem:

Find $\tilde{x} \in K$ such that $b - A\tilde{x} \perp L$

Petrov-Galerkin condition

$\mathbf{m}$ degrees of freedom ($K$) + $\mathbf{m}$ constraints ($L$) →

a small linear system (‘projected problem’)

This is a basic projection step. Typically a sequence of such steps are applied
With a nonzero initial guess $x_0$, approximate problem is

Find $\tilde{x} \in x_0 + K$ such that $b - A\tilde{x} \perp L$

Write $\tilde{x} = x_0 + \delta$ and $r_0 = b - Ax_0$. → system for $\delta$:

Find $\delta \in K$ such that $r_0 - A\delta \perp L$

Formulate Gauss-Seidel as a projection method -

Generalize Gauss-Seidel by defining subspaces consisting of ‘blocks’ of coordinates $\text{span}\{e_i, e_{i+1}, \ldots, e_{i+p}\}$
Matrix representation:

Let

- \( V = [v_1, \ldots, v_m] \) a basis of \( K \) &
- \( W = [w_1, \ldots, w_m] \) a basis of \( L \)

Write approximate solution as \( \tilde{x} = x_0 + \delta \equiv x_0 + Vy \) where \( y \in \mathbb{R}^m \). Then Petrov-Galerkin condition yields:

\[
W^T(r_0 - AVy) = 0
\]

Therefore,

\[
\tilde{x} = x_0 + V[W^TAV]^{-1}W^Tr_0
\]

Remark: In practice \( W^TAV \) is known from algorithm and has a simple structure [tridiagonal, Hessenberg,..]
Prototype Projection Method

Until Convergence Do:

1. Select a pair of subspaces $K$, and $L$;
2. Choose bases: 
   \[ V = [v_1, \ldots, v_m] \text{ for } K \text{ and } W = [w_1, \ldots, w_m] \text{ for } L. \]
3. Compute:
   \[ r \leftarrow b - Ax, \]
   \[ y \leftarrow (W^T AV)^{-1} W^T r, \]
   \[ x \leftarrow x +Vy. \]
Two Important Particular Cases.

1. \( L = K \)
   - When \( A \) is SPD then \( \| x^* - \tilde{x} \|_A = \min_{z \in K} \| x^* - z \|_A \).
   - Class of Galerkin or Orthogonal projection methods
   - Important member of this class: Conjugate Gradient (CG) method

2. \( L = AK \)
   In this case \( \| b - A\tilde{x} \|_2 = \min_{z \in K} \| b - Az \|_2 \)
   - Class of Minimal Residual Methods: CR, GCR, ORTHOMIN, GMRES, CGNR, ...
One-dimensional projection processes

\[ K = \text{span}\{d\} \]
and
\[ L = \text{span}\{e\} \]

Then \( \tilde{x} = x + \alpha d \). Condition \( r - A\delta \perp e \) yields
\[ \alpha = \frac{(r,e)}{(Ad,e)} \]

- Three popular choices:
  1. Steepest descent
  2. Minimal residual iteration
  3. Residual norm steepest descent
1. Steepest descent.

A is SPD. Take at each step $d = r$ and $e = r$.

Iteration:

$r \leftarrow b - Ax,$
$\alpha \leftarrow (r, r) / (Ar, r)$
$x \leftarrow x + \alpha r$

- Each step minimizes $f(x) = \|x - x^*\|_A^2 = (A(x - x^*), (x - x^*))$ in direction $-\nabla f$.

- Convergence guaranteed if $A$ is SPD.

⚠️ As is formulated, the above algorithm requires 2 ‘matvecs’ per step. Reformulate it so only one is needed.
Convergence based on the Kantorovitch inequality: Let $B$ be an SPD matrix, $\lambda_{\text{max}}, \lambda_{\text{min}}$ its largest and smallest eigenvalues. Then,

$$
\frac{(Bx, x)(B^{-1}x, x)}{(x, x)^2} \leq \frac{(\lambda_{\text{max}} + \lambda_{\text{min}})^2}{4 \lambda_{\text{max}} \lambda_{\text{min}}}, \quad \forall x \neq 0.
$$

This helps establish the convergence result

Let $A$ an SPD matrix. Then, the $A$-norms of the error vectors $d_k = x^* - x_k$ generated by steepest descent satisfy:

$$
\|d_{k+1}\|_A \leq \frac{\lambda_{\text{max}} - \lambda_{\text{min}}}{\lambda_{\text{max}} + \lambda_{\text{min}}} \|d_k\|_A
$$

Algorithm converges for any initial guess $x_0$. 

Copper Mountain Conf. 03-31-2022  

p. 40
Proof: Observe \( \| d_{k+1} \|^2_A = (Ad_{k+1}, d_{k+1}) = (r_{k+1}, d_{k+1}) \)

- by substitution,

\[
\| d_{k+1} \|^2_A = (r_{k+1}, d_k - \alpha_k r_k)
\]

- By construction \( r_{k+1} \perp r_k \) so we get \( \| d_{k+1} \|^2_A = (r_{k+1}, d_k) \).

Now:

\[
\| d_{k+1} \|^2_A = (r_k - \alpha_k A r_k, d_k) \\
= (r_k, A^{-1} r_k) - \alpha_k (r_k, r_k) \\
= \| d_k \|^2_A \left( 1 - \frac{(r_k, r_k)}{(r_k, A r_k)} \times \frac{(r_k, r_k)}{(r_k, A^{-1} r_k)} \right).
\]

Result follows by applying the Kantorovich inequality.
2. Minimal residual iteration.

A positive definite \((A + A^T)\) is SPD). Take at each step \(d = r\) and \(e = Ar\).

\[
\begin{align*}
    r &\leftarrow b - Ax, \\
    \alpha &\leftarrow (Ar, r)/(Ar, Ar) \\
    x &\leftarrow x + \alpha r
\end{align*}
\]

Each step minimizes \(f(x) = \|b - Ax\|_2^2\) in direction \(r\).

Converges under the condition that \(A + A^T\) is SPD.

As is formulated, the above algorithm would require 2 'matvecs' at each step. Reformulate it so that only one matvec is required.
Let $A$ be a real positive definite matrix, and let

$$\mu = \lambda_{min}(A + A^T)/2, \quad \sigma = \|A\|_2.$$ 

Then the residual vectors generated by the Min. Res. Algorithm satisfy:

$$\|r_{k+1}\|_2 \leq \left(1 - \frac{\mu^2}{\sigma^2}\right)^{1/2} \|r_k\|_2$$

In this case Min. Res. converges for any initial guess $x_0$. 
Proof: Similar to steepest descent. Start with

\[ \| r_{k+1} \|_2^2 = (r_{k+1}, r_k - \alpha_k Ar_k) \]

\[ = (r_{k+1}, r_k) - \alpha_k (r_{k+1}, Ar_k). \]

By construction, \( r_{k+1} = r_k - \alpha_k Ar_k \) is \( \perp Ar_k \), so:

\[ \| r_{k+1} \|_2^2 = (r_{k+1}, r_k) = (r_k - \alpha_k Ar_k, r_k). \] Then:

\[ \| r_{k+1} \|_2^2 = (r_k, r_k) - \alpha_k (Ar_k, r_k) \]

\[ = \| r_k \|_2^2 \left( 1 - \frac{(Ar_k, r_k)}{(r_k, r_k)} \frac{(Ar_k, r_k)}{(Ar_k, Ar_k)} \right) \]

\[ = \| r_k \|_2^2 \left( 1 - \frac{(Ar_k, r_k)^2}{(r_k, r_k)^2} \frac{\| r_k \|_2^2}{\| Ar_k \|_2^2} \right). \]

Result follows from the inequalities \((Ax, x)/(x, x) \geq \mu > 0\) and \(\| Ar_k \|_2 \leq \| A \|_2 \| r_k \|_2\).
3. Residual norm steepest descent.

A is arbitrary (nonsingular). Take at each step \( d = A^T r \) and \( e = Ad \).

\[
\begin{align*}
\text{Iteration:} & & \quad r & \leftarrow b - Ax, \quad d = A^T r \\
& & \quad \alpha & \leftarrow \frac{\|d\|_2^2}{\|Ad\|_2^2} \\
& & \quad x & \leftarrow x + \alpha d
\end{align*}
\]

- Each step minimizes \( f(x) = \|b - Ax\|_2^2 \) in direction \( -\nabla f \).
- Important Note: equivalent to usual steepest descent applied to normal equations \( A^T Ax = A^T b \).
- Converges under the condition that \( A \) is nonsingular.

Demos: run \textit{demo1Dproj}
KRYLOV SUBSPACE METHODS
Motivation

- Common feature of one-dimensional projection techniques:
  \[ x_{\text{new}} = x + \alpha d \]

  where \( d \) = a certain direction.

- \( \alpha \) is defined to optimize a certain function.

- Equivalently: determine \( \alpha \) by an orthogonality constraint

  Example

  In MR:
  \[ x(\alpha) = x + \alpha d, \text{ with } d = b - Ax. \]
  \[ \min_{\alpha} \| b - Ax(\alpha) \|_2 \text{ reached iff } b - Ax(\alpha) \perp r \]

- One-dimensional projection methods are greedy methods. They are 'short-sighted'.
Example:

Recall in Steepest Descent: New direction of search $\tilde{r}$ is $\perp$ to old direction of search $r$.

$\begin{align*}
  r &\leftarrow b - Ax, \\
  \alpha &\leftarrow (r, r)/(Ar, r) \\
  x &\leftarrow x + \alpha r
\end{align*}$

Question: can we do better by combining successive iterates?

- Yes: Krylov subspace methods..
Consider MR (or steepest descent). At each iteration:

\[ r_{k+1} = b - A(x^{(k)} + \alpha_k r_k) = r_k - \alpha_k A r_k = (I - \alpha_k A) r_k \]

In the end:

\[ r_{k+1} = (I - \alpha_k A)(I - \alpha_{k-1} A) \cdots (I - \alpha_0 A) r_0 = p_{k+1}(A) r_0 \]

where \( p_{k+1}(t) \) is a polynomial of degree \( k + 1 \) of the form

\[ p_{k+1}(t) = 1 - t q_k(t) \]

Show that: \[ x^{(k+1)} = x^{(0)} + q_k(A) r_0 \], with \( \text{deg} (q_k) = k \)

Krylov subspace methods: iterations of this form that are 'optimal' [from \( m \)-dimensional projection methods]
**Krylov subspace methods**

**Principle:** Projection methods on Krylov subspaces:

\[ K_m(A, v_1) = \text{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\} \]

- The most important class of iterative methods.
- Many variants exist depending on the subspace \( L \).

**Simple properties of \( K_m \)**

- Notation: \( \mu = \text{deg. of minimal polynomial of } v_1 \). Then:
  - \( K_m = \{p(A)v_1|p = \text{polynomial of degree } \leq m - 1\} \)
  - \( K_m = K_\mu \) for all \( m \geq \mu \). Moreover, \( K_\mu \) is invariant under \( A \).
  - \( \text{dim}(K_m) = m \) iff \( \mu \geq m \).
Arnoldi’s algorithm

- Goal: to compute an orthogonal basis of $K_m$.
- Input: Initial vector $v_1$, with $\|v_1\|_2 = 1$ and $m$.

For $j = 1, \ldots, m$ Do:
- Compute $w := Av_j$
  For $i = 1, \ldots, j$ Do:
    - $h_{i,j} := (w, v_i)$
    - $w := w - h_{i,j}v_i$
  EndDo
  Compute: $h_{j+1,j} = \|w\|_2$ and $v_{j+1} = w / h_{j+1,j}$
EndDo
Result of orthogonalization process (Arnoldi):

1. $V_m = [v_1, v_2, \ldots, v_m]$ orthonormal basis of $K_m$.

2. $AV_m = V_{m+1} \overline{H}_m$

3. $V_m^T AV_m = H_m \equiv \overline{H}_m$ — last row.

\[
V_m = \begin{bmatrix} \vdots & \vdots & \vdots \\
\end{bmatrix}
\]

\[
AV_m = V_{m+1} \overline{H}_m
\]

\[
\overline{H}_m = \begin{bmatrix} \vdots & \vdots & \vdots \\
\end{bmatrix}
\]

\[
V_{m+1} = [V_m, v_{m+1}]
\]
Arnoldi’s Method for linear systems \((L_m = K_m)\)

From Petrov-Galerkin condition when \(L_m = K_m\), we get

\[ x_m = x_0 + V_m H_m^{-1} V_m^T r_0 \]

Select \(v_1 = r_0 / \|r_0\|_2 \equiv r_0 / \beta\) in Arnoldi’s. Then

\[ x_m = x_0 + \beta V_m H_m^{-1} e_1 \]

What is the residual vector \(r_m = b - Ax_m\)?

Several algorithms mathematically equivalent to this approach:

* FOM [Y. Saad, 1981] (above formulation), Young and Jea’s OR-THORES [1982], Axelsson’s projection method [1981],...

* Also Conjugate Gradient method [see later]
Minimal residual methods \((L_m = AK_m)\)

When \(L_m = AK_m\), we let \(W_m \equiv AV_m\) and obtain relation

\[
x_m = x_0 + V_m [W_m^T AV_m]^{-1} W_m^T r_0
\]
\[
= x_0 + V_m [(AV_m)^T AV_m]^{-1} (AV_m)^T r_0.
\]

Use again \(v_1 := r_0 / (\beta := \|r_0\|_2)\) and the relation

\[
AV_m = V_{m+1} \bar{H}_m
\]

\[
x_m = x_0 + V_m [\bar{H}_m^T \bar{H}_m]^{-1} \bar{H}_m^T \beta e_1 = x_0 + V_m y_m
\]

where \(y_m\) minimizes \(\|\beta e_1 - \bar{H}_m y\|_2\) over \(y \in \mathbb{R}^m\).
Gives the Generalized Minimal Residual method (GMRES) ([Saad-Schultz, 1986]):

\[ x_m = x_0 + V_m y_m \]
\[ y_m = \min_y \| \beta e_1 - \bar{H}_m y \|_2 \]

Several Mathematically equivalent methods:
- Axelsson’s CGLS
- Orthomin (1980)
- Orthodir
- GCR
The symmetric case: Observation

Observe: When $A$ is real symmetric then in Arnoldi’s method:

$$H_m = V_m^T A V_m$$

must be symmetric. Therefore

Theorem. When Arnoldi’s algorithm is applied to a (real) symmetric matrix then the matrix $H_m$ is symmetric tridiagonal:

$$h_{i,j} = 0 \quad 1 \leq i < j - 1; \quad \text{and}$$
$$h_{j,j+1} = h_{j+1,j}, \quad j = 1, \ldots, m$$
We can write

\[ H_m = \begin{bmatrix}
\alpha_1 & \beta_2 \\
\beta_2 & \alpha_2 & \beta_3 \\
\beta_3 & \alpha_3 & \beta_4 \\
\vdots & \vdots & \ddots & \ddots \\
\beta_m & \alpha_m
\end{bmatrix} \]  \quad (1)  

The \( v_i \)'s satisfy a 3-term recurrence [Lanczos Algorithm]:

\[ \beta_{j+1} v_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1} \]

Simplified version of Arnoldi's algorithm for sym. systems.

Symmetric matrix + Arnoldi \( \rightarrow \) Symmetric Lanczos
The Lanczos algorithm

ALGORITHM : 1. Lanczos

1. Choose an initial vector $v_1$, s.t. $\|v_1\|_2 = 1$
   Set $\beta_1 \equiv 0$, $v_0 \equiv 0$
2. For $j = 1, 2, \ldots, m$ Do:
3. $w_j := Av_j - \beta_j v_{j-1}$
4. $\alpha_j := (w_j, v_j)$
5. $w_j := w_j - \alpha_j v_j$
6. $\beta_{j+1} := \|w_j\|_2$. If $\beta_{j+1} = 0$ then Stop
7. $v_{j+1} := w_j / \beta_{j+1}$
8. EndDo
Lanczos algorithm for linear systems

• Usual orthogonal projection method setting:
  
  \[ L_m = K_m = \text{span}\{r_0, Ar_0, \ldots, A^{m-1}r_0\} \]

  • Basis \( V_m = [v_1, \ldots, v_m] \) of \( K_m \) generated by the Lanczos algorithm

• Three different possible implementations.

  (1) Arnoldi-like;

  (2) Exploit tridiagonal nature of \( H_m \) (DIOM);

  (3) Conjugate gradient (CG) - derived from (2)

• We will skip details and just show the algorithm
**The Conjugate Gradient Algorithm (A S.P.D.)**

**ALGORITHM : 2. Conjugate gradient algorithm**

1. **Start:** \( r_0 := b - Ax_0, \ p_0 := r_0. \)
2. **Iterate:** Until convergence Do:
3. \( \alpha_j := (r_j, r_j)/(Ap_j, p_j) \)
4. \( x_{j+1} := x_j + \alpha_j p_j \)
5. \( r_{j+1} := r_j - \alpha_j A p_j \)
6. \( \beta_j := (r_{j+1}, r_{j+1})/(r_j, r_j) \)
7. \( p_{j+1} := r_{j+1} + \beta_j p_j \)
8. EndDo

- \( r_j = scaling \times v_{j+1}. \) The \( r_j \)'s are orthogonal.
- The \( p_j \)'s are \( A \)-conjugate, i.e., \((Ap_i, p_j) = 0\) for \( i \neq j \).
IN BRIEF: METHODS BASED ON BI-ORTHOGONALIZATION
**BiCG and related methods**

**ALGORITHM : 3. BiConjugate Gradient (BCG)**

1. Compute $r_0 := b - Ax_0$. Choose $r_0^*$ such that $(r_0, r_0^*) \neq 0$.
2. Set, $p_0 := r_0$, $p_0^* := r_0^*$
3. For $j = 0, 1, \ldots$, until convergence Do:
4.  $\alpha_j := (r_j, r_j^*)/(Ap_j, p_j^*)$
5.  $x_{j+1} := x_j + \alpha_j p_j$
6.  $r_{j+1} := r_j - \alpha_j Ap_j$
7.  $r_{j+1}^* := r_{j+1}^* - \alpha_j A^T p_j^*$
8.  $\beta_j := (r_{j+1}, r_{j+1}^*)/(r_j, r_j^*)$
9.  $p_{j+1} := r_{j+1} + \beta_j p_j$
10. $p_{j+1}^* := r_{j+1}^* + \beta_j p_j^*$
11. EndDo
**ALGORITHM : 4. Conjugate Gradient Squared**

1. Compute \( r_0 := b - Ax_0; \) \( r_0^* \) arbitrary.
2. Set \( p_0 := u_0 := r_0 \).
3. For \( j = 0, 1, 2 \ldots \), until convergence Do:
   4. \( \alpha_j = (r_j, r_0^*)/(Ap_j, r_0^*) \)
   5. \( q_j = u_j - \alpha_j Ap_j \)
   6. \( x_{j+1} = x_j + \alpha_j(u_j + q_j) \)
   7. \( r_{j+1} = r_j - \alpha_j A(u_j + q_j) \)
   8. \( \beta_j = (r_{j+1}, r_0^*)/(r_j, r_0^*) \)
   9. \( u_{j+1} = r_{j+1} + \beta_j q_j \)
  10. \( p_{j+1} = u_{j+1} + \beta_j (q_j + \beta_j p_j) \)
  11. EndDo
ALGORITHM : 5. **BCGSTAB**

1. Compute $r_0 := b - Ax_0$; $r_0^*$ arbitrary;
2. $p_0 := r_0$.
3. For $j = 0, 1, \ldots$, until convergence Do:
   4. $\alpha_j := (r_j, r_0^*)/(Ap_j, r_0^*)$
   5. $s_j := r_j - \alpha_j Ap_j$
   6. $\omega_j := (As_j, s_j)/(As_j, As_j)$
   7. $x_{j+1} := x_j + \alpha_j p_j + \omega_j s_j$
   8. $r_{j+1} := s_j - \omega_j As_j$
   9. $\beta_j := \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \times \frac{\alpha_j}{\omega_j}$
   10. $p_{j+1} := r_{j+1} + \beta_j(p_j - \omega_j Ap_j)$
11. EndDo

DemoKrylov
PRECONDITIONING
Preconditioning – Basic principles

Basic idea

use Krylov subspace method on a modified system, e.g.:  

\[ M^{-1}Ax = M^{-1}b. \]

- The matrix \( M^{-1}A \) need not be formed explicitly; only need to solve \( Mw = v \) whenever needed.

- Consequence: fundamental requirement is that it should be easy to compute \( M^{-1}v \) for an arbitrary vector \( v \).
Left, Right, and Split preconditioning

**Left preconditioning:** \( M^{-1}Ax = M^{-1}b \)

**Right preconditioning:** \( AM^{-1}u = b, \) with \( x = M^{-1}u \)

**Split preconditioning:** \( M_L^{-1}AM_R^{-1}u = M_L^{-1}b, \) with \( x = M_R^{-1}u \)

[Assume \( M \) is factored: \( M = M_LM_R. \) ]
Preconditioned CG (PCG)

- Assume: $A$ and $M$ are both SPD.
- Applying CG directly to $M^{-1}Ax = M^{-1}b$ or $AM^{-1}u = b$ won’t work because coefficient matrices are not symmetric.
- Alternative: when $M = LL^T$ use split preconditioner option
- Second alternative: Observe that $M^{-1}A$ is self-adjoint wrt $M$ inner product:

$$\left(M^{-1}Ax, y\right)_M = (Ax, y) = (x, Ay) = (x, M^{-1}Ay)_M$$

- Can now use CG on $M^{-1}Ax = M^{-1}b$ with M-inner products. Details omitted.
**Flexible accelerators**

**Question:** What can we do in case $\mathcal{M}$ is defined only approximately? i.e., if it can vary from one step to the other?

**Applications:**

- Iterative techniques as preconditioners: Block-SOR, SSOR, Multi-grid, etc..
- Chaotic relaxation type preconditioners (e.g., in a parallel computing environment)
- Mixing Preconditioners; ... etc.

**Answer:** Flexible accelerator - e.g. FGMRES. Details skipped.
Standard preconditioners

- Simplest preconditioner: $M = \text{Diag}(A)$ ➤ poor convergence.
- Next to simplest: SSOR
  \[ M = (D - \omega E)D^{-1}(D - \omega F) \]
- Still simple but often more efficient: ILU(0).
- ILU(p) – ILU with level of fill p – more complex.
- Class of ILU preconditioners with threshold
- Class of approximate inverse preconditioners
- Class of Multilevel ILU preconditioners: Multigrid, Algebraic Multi-grid, M-level ILU, ..
The SOR/SSOR preconditioner

- SOR preconditioning
  \[ M_{\text{SOR}} = (D - \omega E) \]

- SSOR preconditioning
  \[ M_{\text{SSOR}} = (D - \omega E)D^{-1}(D - \omega F) \]

- \( M_{\text{SSOR}} = LU \), \( L \) = lower unit matrix, \( U \) = upper triangular. One solve with \( M_{\text{SSOR}} \approx \) same cost as a MAT-VEC.

Q: Choice of \( \omega \); Can use \( k \) steps instead of 1 step \( \rightarrow \) best \( k \)?

demo_effect_of_prec

Write matlab script for \( k \)-step SSOR preconditioner – using relaxation, i.e., start from iters/sorRelax.m.
**ILU(0) and IC(0) preconditioners**

- **Notation:** \( \text{NZ}(X) = \{(i, j) \mid X_{i,j} \neq 0\} \)

- **Formal definition of ILU(0):**

\[
A = LU + R \\
\text{NZ}(L) \cup \text{NZ}(U) = \text{NZ}(A) \\
r_{ij} = 0 \text{ for } (i, j) \in \text{NZ}(A)
\]

- This does not define **ILU(0)** in a unique way.

**Constructive definition:** Compute the LU factorization of \( A \) but drop any fill-in in \( L \) and \( U \) outside of \( \text{Struct}(A) \).

- ILU factorizations are often based on \( i, k, j \) version of GE.
ILU(0) – zero-fill ILU

ALGORITHM : 6. ILU(0)

For $i = 1, \ldots, N$ Do:
   For $k = 1, \ldots, i - 1$ and if $(i, k) \in NZ(A)$ Do:
      Compute $a_{ik} := a_{ik}/a_{kj}$
      For $j = k + 1, \ldots$ and if $(i, j) \in NZ(A)$, Do:
         compute $a_{ij} := a_{ij} - a_{ik}a_{k,j}$.
   EndFor
EndFor

- When $A$ is SPD then the ILU factorization = Incomplete Cholesky factorization – IC(0). Meijerink and Van der Vorst [1977].
Typical eigenvalue distribution of preconditioned matrix
Pattern of ILU(0) for 5-point matrix

\[ L \]
\[ U \]
\[ A \]
\[ LU \]
Higher order ILU factorization

- Higher accuracy incomplete Cholesky: for regularly structured problems, $IC(p)$ allows $p$ additional diagonals in $L$.

- Can be generalized to irregular sparse matrices using the notion of level of fill-in [Watts III, 1979]

- Initially $Lev_{ij} = \begin{cases} 
0 & \text{for } a_{ij} \neq 0 \\
\infty & \text{for } a_{ij} == 0 
\end{cases}$

- At a given step $i$ of Gaussian elimination:

$$Lev_{kj} = \min\{Lev_{kj}; Lev_{ki} + Lev_{ij} + 1\}$$
ILU(p) Strategy = drop anything with level of fill-in exceeding \( p \).

* Increasing level of fill-in usually results in more accurate ILU and...
* ...typically in fewer steps and fewer arithmetic operations.
$ILU(1)$
ALGORITHM : 7. \textit{ILU}(p)

For $i = 2, N$ Do

For each $k = 1, \ldots, i - 1$ and if $a_{ij} \neq 0$ do

Compute $a_{ik} := a_{ik}/a_{jj}$

Compute $a_{i,*} := a_{i,*} - a_{ik}a_{k,*}$.

Update the levels of $a_{i,*}$

Replace any element in row $i$ with $lev(a_{ij}) > p$ by zero.

EndFor

EndFor

- The algorithm can be split into a symbolic and a numerical phase.
  Level-of-fill set up in symbolic phase.
**ILU with threshold: ILUT(k, \( \epsilon \))**

ILU(p) factorizations are based on structure only and not numerical values ➤ potential problems for non M-matrices.

Alternative: ILU with Threshold, ILUT

- During each \( i \)-th step in GE (\( i, k, j \) version), discard pivots or fill-ins whose value is below \( \epsilon \| \text{row}_i(A) \| \).
- Once the \( i \)-th row of \( L + U \), (L-part + U-part) is computed retain only the \( k \) largest elements in both parts.

➤ Advantages: controlled fill-in. Smaller memory overhead.

➤ Easy to implement and can be made quite inexpensive.
Typical curve of CPU time versus numerical threshold

demoPrec
MULTI-LEVEL PRECONDITIONERS
**Group Independent Sets / Aggregates**

**Main goal:** generalize independent sets to improve robustness

**Main idea:** use “cliques”, or “aggregates”. No coupling between the aggregates.

- Label nodes of independent sets first

---

No Coupling
Simple strategy used: Do a Cuthill-MKee ordering until there are enough points to make a block. Reverse ordering. Start a new block from a non-visited node. Continue until all points are visited. Add criterion for rejecting “not sufficiently diagonally dominant rows.”
Original matrix
Block size of 20
Shape of reordered matrix:

\[ P A P^T = \begin{pmatrix} B & F \\ E & C \end{pmatrix} \]

- Block factorize:

\[
\begin{pmatrix} B & F \\ E & C \end{pmatrix} = \begin{pmatrix} L & 0 \\ E U^{-1} & I \end{pmatrix} \begin{pmatrix} U & L^{-1} F \\ 0 & S \end{pmatrix}
\]

- \( S = C - E B^{-1} F \) = Schur complement + dropping to reduce fill

- Next step: treat the Schur complement recursively
**Algebraic Recursive Multilevel Solver (ARMS)**

**Level l Factorization:**

\[
\begin{pmatrix}
B_l & F_l \\
E_l & C_l
\end{pmatrix}
\approx
\begin{pmatrix}
L_l & 0 \\
E_l U_l^{-1} & I
\end{pmatrix}
\begin{pmatrix}
I & 0 \\
0 & A_{l+1}
\end{pmatrix}
\begin{pmatrix}
U_l & L_l^{-1} F_l \\
0 & I
\end{pmatrix}
\]

- \( B_l \approx L_l U_l \); \( A_{l+1} \approx S_l = C_l - E_l U_l^{-1} L_l^{-1} F_l \)
- L-solve \( \sim \) restriction; U-solve \( \sim \) prolongation.
- Perform above block factorization recursively on \( A_{l+1} \)
- Blocks in \( B_l \) treated as sparse. Can be large or small.
- Algorithm is fully recursive
- Stability criterion in block independent sets algorithm
Algebraic Recursive Multilevel Solver (ARMS)

Original matrix, \( A \), and reordered matrix, \( A_0 = P_0^T A P_0 \).
**Problem:** Fill-in

**Remedy:** dropping strategy

- Treat the Schur complement recursively
- Solve last Schur complement system with ILUT-GMRES.
ALGORITHM : 8. \textit{ARMS}(A_{lev}) factorization

1. If $\text{lev} = \text{last\_lev}$ then
2. \quad Compute $A_{lev} \approx L_{lev}U_{lev}$
3. Else:
4. \quad Find an independent set permutation $P_{lev}$
5. \quad Apply permutation $A_{lev} := P_{lev}^{T}A_{lev}P$
6. \quad Compute factorization
7. \quad Call ARMS($A_{lev+1}$)
8. EndIf
Time for a Matlab demo

Look at the armsC part of the matlab suite. arms2.m builds the arms preconditioner – compare with the algorithm given earlier. [really recursive?]

Run test driver is demoArms.m -
USE OF COMPLEX SHIFTS
Several papers promoted the use of complex shifts [or very similar approaches] for Helmholtz


Illustration with an experiment: finite difference discretization of $-\Delta$ on a $25 \times 20$ grid.

Add a negative shift of $-1$ to resulting matrix.

Do an ILU factorization of $A$ and plot eigs of $L^{-1}AU^{-1}$.

Used LUINC from matlab - no-pivoting and threshold $= 0.1$. 
Terrible spectrum:
Now plot eigs of $L^{-1}AU^{-1}$ where $L, U$ are inc. LU factors of $B = A + 0.25 * i$

Much better! Observed by many [PDE viewpoint]

Idea:
Adapt technique to ILU:
Add complex shifts before ILU
**Question:**
What if we do an exact factorization \([\text{droptol} = 0]\)?

\[
\Lambda(L^{-1}AU^{-1}) = \Lambda[(A + \alpha i I)^{-1} A]
\]

\[
\Lambda = \left\{ \frac{\lambda_j}{\lambda_j + i\alpha} \right\}
\]

- Located on a circle – with a cluster at one.
- Figure shows situation on the same example
- Next figures approximate spectra for previous (real) example
Spectrum of the Helmholtz operator matrix, $A$

Spectrum of $AM^{-1}$, $M = LU$ on $A$

Spectrum of $AM^{-1}$, $M = LU$ on shifted $A$ (dd–based scheme)

Spectrum of $AM^{-1}$, $M = LU$ on shifted $A$ ($\tau$–based scheme)
Helmholtz equation example


- Problem is set in the open domain \( \Omega_e \) of \( \mathbb{R}^d \)

\[
\begin{cases}
\Delta u + k^2 u = f \quad \text{in} \quad \Omega \\
\quad u = -u_{inc} \quad \text{on} \quad \Gamma \\
\text{or} \quad \frac{\partial u}{\partial n} = -\frac{\partial u_{inc}}{\partial n} \quad \text{on} \quad \Gamma
\end{cases}
\]

\[
limit_{r \to \infty} r^{(d-1)/2} \left( \frac{\partial u}{\partial \vec{n}} - iku \right) = 0 \quad \text{Sommerfeld cond.}
\]

where: \( u \) the wave diffracted by \( \Gamma \), \( f = \) source function = zero outside domain
Issue: non-reflective boundary conditions when making the domain finite.

Artificial boundary $\Gamma_{art}$ added – Need non-absorbing BCs.

For high frequencies, linear systems become very ‘indefinite’ – [eigenvalues on both sides of the imaginary axis]

Not very good for iterative methods
**Test Problem** Soft obstacle $= \text{disk of radius } r_0 = 0.5\, m$. Incident plane wave with a wavelength $\lambda$; propagates along the $x$-axis.

2nd order Bayliss-Turkel boundary conditions used on $\Gamma_{\text{art}}$, located at a distance $2r_0$ from obstacle.

Discretization: isoparametric elements with 4 nodes. Analytic solution known.
Comparisons

Test problem seen earlier. Mesh size $1/h = 160 \rightarrow n = 28, 980$, $nnz = 260, 280$

Convergence profiles of ARMS with different shifting schemes

Convergence profiles of ILUT with different shifting schemes

ARMS & shifted variants

ILUT & shifted variants
### Wavenumber varied - tests with ILUT

<table>
<thead>
<tr>
<th>Preconditioner</th>
<th>$k$</th>
<th>$\frac{\lambda}{h}$</th>
<th>Iters.</th>
<th>Fill Factor</th>
<th>$| (LU)^{-1} e |_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ILUT (no shift)</strong></td>
<td>$4\pi$</td>
<td>60</td>
<td>134</td>
<td>2.32</td>
<td>3.65e + 03</td>
</tr>
<tr>
<td></td>
<td>$8\pi$</td>
<td>30</td>
<td>263</td>
<td>2.25</td>
<td>1.23e+04</td>
</tr>
<tr>
<td></td>
<td>$16\pi$</td>
<td>15</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td></td>
<td>$24\pi$</td>
<td>10</td>
<td>−</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><strong>ILUT (dd-based)</strong></td>
<td>$4\pi$</td>
<td>60</td>
<td>267</td>
<td>2.24</td>
<td>2.29e + 03</td>
</tr>
<tr>
<td></td>
<td>$8\pi$</td>
<td>30</td>
<td>255</td>
<td>2.23</td>
<td>4.73e+03</td>
</tr>
<tr>
<td></td>
<td>$16\pi$</td>
<td>15</td>
<td>101</td>
<td>3.14</td>
<td>6.60e+02</td>
</tr>
<tr>
<td></td>
<td>$24\pi$</td>
<td>10</td>
<td>100</td>
<td>3.92</td>
<td>2.89e+02</td>
</tr>
<tr>
<td><strong>ILUT ($\tau$-based)</strong></td>
<td>$4\pi$</td>
<td>60</td>
<td>132</td>
<td>2.31</td>
<td>2.98e + 03</td>
</tr>
<tr>
<td></td>
<td>$8\pi$</td>
<td>30</td>
<td>195</td>
<td>2.19</td>
<td>4.12e+03</td>
</tr>
<tr>
<td></td>
<td>$16\pi$</td>
<td>15</td>
<td>75</td>
<td>3.11</td>
<td>7.46e+02</td>
</tr>
<tr>
<td></td>
<td>$24\pi$</td>
<td>10</td>
<td>86</td>
<td>3.85</td>
<td>2.73e+02</td>
</tr>
</tbody>
</table>
Wavenumber varied - tests with ARMS

<table>
<thead>
<tr>
<th>Preconditioner</th>
<th>$k$</th>
<th>$\frac{\lambda}{h}$</th>
<th>Iters.</th>
<th>Fill Factor</th>
<th>$| (LU)^{-1} e |_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARMS (no shift)</td>
<td>4$\pi$</td>
<td>60</td>
<td>120</td>
<td>3.50</td>
<td>$7.48e + 03$</td>
</tr>
<tr>
<td></td>
<td>8$\pi$</td>
<td>30</td>
<td>169</td>
<td>4.03</td>
<td>1.66e+04</td>
</tr>
<tr>
<td></td>
<td>16$\pi$</td>
<td>15</td>
<td>282</td>
<td>4.50</td>
<td>2.44e+03</td>
</tr>
<tr>
<td></td>
<td>24$\pi$</td>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ARMS (dd-based)</td>
<td>4$\pi$</td>
<td>60</td>
<td>411</td>
<td>3.83</td>
<td>$5.12e + 02$</td>
</tr>
<tr>
<td></td>
<td>8$\pi$</td>
<td>30</td>
<td>311</td>
<td>4.37</td>
<td>5.67e+02</td>
</tr>
<tr>
<td></td>
<td>16$\pi$</td>
<td>15</td>
<td>187</td>
<td>4.71</td>
<td>3.92e+02</td>
</tr>
<tr>
<td></td>
<td>24$\pi$</td>
<td>10</td>
<td>185</td>
<td>3.00</td>
<td>2.54e+02</td>
</tr>
<tr>
<td>ARMS ($\tau$-based)</td>
<td>4$\pi$</td>
<td>60</td>
<td>106</td>
<td>3.45</td>
<td>$7.56e + 03$</td>
</tr>
<tr>
<td></td>
<td>8$\pi$</td>
<td>30</td>
<td>79</td>
<td>3.84</td>
<td>6.41e+03</td>
</tr>
<tr>
<td></td>
<td>16$\pi$</td>
<td>15</td>
<td>39</td>
<td>3.95</td>
<td>1.26e+03</td>
</tr>
<tr>
<td></td>
<td>24$\pi$</td>
<td>10</td>
<td>94</td>
<td>3.02</td>
<td>4.71e+02</td>
</tr>
</tbody>
</table>
‘ALGEBRAIC’ DOMAIN DECOMPOSITION METHODS
Preconditioners in ‘algebraic’ DD context

Common framework: Partition mesh, ‘distribute’ matrix, then exploit a form of Schwarz technique ...

... or a form of ‘approximate’ Schur complement technique

- In recent years: many researchers have discovered the importance of some form of ‘low-rank correction’
- Related methods: ‘deflation’, ‘Smoothed Aggregation (SA)’, ...
- Next: Our work in LR correction techniques
Schur complement + low-rank correction techniques

- Algebraic DD: Partition graph using ‘edge separation’:

  Edge Separation:

  ![Diagram of edge separation]

  **Local view:**

  **Local Equations**

  $$
  \begin{bmatrix}
  B_i & E_i \\
  E_i^T & C_i
  \end{bmatrix} \begin{bmatrix}
  u_i \\
  y_i
  \end{bmatrix} + \sum_{j \in N_i} 0 \begin{bmatrix}
  E_{ij} y_j
  \end{bmatrix} = \begin{bmatrix}
  f_i \\
  g_i
  \end{bmatrix}
  $$

- Assume (for now) $A$ is Symmetric Positive Definite (SPD)
Recall: The global system

Global matrix has the form \[
\begin{pmatrix}
B & E \\
E^T & C
\end{pmatrix}
\]
**Schur Complement System**

**Background:**

\[
\begin{pmatrix} B & E \\ E^T & C \end{pmatrix} = \begin{pmatrix} I & B \\ E^T B^{-1} & I \end{pmatrix} \begin{pmatrix} B & E \\ S \end{pmatrix}
\]

\[S = C - E^T B^{-1} E\]

- \(S \in \mathbb{R}^{s \times s}\) == ‘Schur complement’ matrix
- Solution obtained from two solves with \(B\), one with \(S\)

**Next:** Find approximate inverse of \(S\).

- Assume \(C\) is SPD and let \(C = LL^T\). Then:

\[S = L \left( I - L^{-1} E^T B^{-1} E L^{-T} \right) L^T \equiv L(I - H)L^T.\]

- Define:

\[H = L^{-1} E^T B^{-1} E L^{-T}\]

- Can show:

\[\lambda_j(H) \in [0, 1)\]
Decay properties of $S^{-1} - C^{-1}$

- We have: 
  $$S^{-1} = L^{-T}(I - H)^{-1}L^{-1}$$

- Can we write: 
  $$S^{-1} = C^{-1} + \text{Low rank correction?}$$

  $$S^{-1} - C^{-1} = L^{-T}(I - (I - H)^{-1})L^{-1} \equiv L^{-T}XL^{-1}$$

- Thus, 
  $$S^{-1} = C^{-1} + L^{-T}XL^{-1}.$$  Note:

  $$\lambda_k(X) = \frac{\lambda_k(H)}{1 - \lambda_k(H)}$$

- Well separated when $\lambda_k \to 1$. 
Decay properties of $S^{-1} - C^{-1}$

- Example: 2-D Laplacian, $n_x = n_y = 32$, 4 subdomains
- $\Lambda(X)$ and $\Lambda(S^{-1} - C^{-1}) = \Lambda(L^{-T}XL^{-1})$

5 eigenvectors:
- 82.5% of $X$, 85.1% of $L^{-T}XL^{-1}$

10 eigenvectors:
- 89.7% of $X$, 91.4% of $L^{-T}XL^{-1}$

- Closed form analysis available for 2D Laplaceans
**Low-rank approximation**

- Preconditioner for $A$:

  $$M = \begin{pmatrix} I & \bar{E} \\ E^T B^{-1} & I \end{pmatrix} \begin{pmatrix} B & E \\ \bar{S} \end{pmatrix}$$

- $(n - s)$ of $\lambda_i(AM^{-1}) = 1$, the other $s \rightarrow \lambda_i(S\tilde{S}^{-1})$

- Eigendecomposition $H = U\Lambda U^T$. Replace $\Lambda$ with $\tilde{\Lambda}$

- Recall $S^{-1} = L^{-T}(I - H)^{-1}L^{-1}$, and rewrite

  $$S^{-1} = L^{-T}U(I - \Lambda)^{-1}U^TL^{-1}$$

  $$\tilde{S}^{-1} = L^{-T}U(I - \tilde{\Lambda})^{-1}U^TL^{-1}$$

- Can show: $\lambda(S\tilde{S}^{-1}) = \frac{1 - \lambda_i}{1 - \tilde{\lambda}_i}$, $i = 1, \ldots, s$
Numerical Experiments

- Intel Xeon X5675 (12 MB Cache, 3.06 GHz, 6-core), Xeon X5560 (8 MB Cache, 2.8 GHz, 4-core) at MSI
- Written in C/C++, MKL; OpenMP parallelism
- Accelerators: CG, GMRES(40)
- Partitioning with METIS

Details:

- $\Delta$ shifted by $-sI$. 2D: $s = 0.01$, 3D: $s = 0.05$

<table>
<thead>
<tr>
<th>Grid</th>
<th>ILDLT-GMRES</th>
<th>RAS-GMRES</th>
<th>SLR-GMRES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fill p-t its i-t</td>
<td>fill p-t its i-t</td>
<td>nd rk fill p-t its i-t</td>
</tr>
<tr>
<td>$256^2$</td>
<td>8.2 .17 F</td>
<td>6.3 .13 F</td>
<td>8 32 6.4 .21 33 .125</td>
</tr>
<tr>
<td>$512^2$</td>
<td>8.4 .70 F</td>
<td>8.4 .72 F</td>
<td>16 64 7.6 2.1 93 1.50</td>
</tr>
<tr>
<td>$1024^2$</td>
<td>13 5.1 F</td>
<td>19 22 F</td>
<td>8 128 11 25 50 4.81</td>
</tr>
<tr>
<td>$40^3$</td>
<td>6.9 .25 54 .54</td>
<td>6.7 .25 99 .30</td>
<td>64 32 6.7 .49 23 .123</td>
</tr>
<tr>
<td>$64^3$</td>
<td>9.0 1.4 F</td>
<td>11.8 2.2 F</td>
<td>128 64 9.1 3.9 45 1.16</td>
</tr>
<tr>
<td>$100^3$</td>
<td>15 11 F</td>
<td>12 15 F</td>
<td>128 180 15 63 88 13.9</td>
</tr>
</tbody>
</table>
‘Non-standard’ DD framework: HID ordering

- Issue: Schur complement can become large (3D Pbs)

- Remedy: Use Hierarchical Interface Decomposition (HID) - Henon and YS’05

**Goal:** Define a method that descends into interface variables in a hierarchical way → need a hierarchy of 'interfaces'.

- Ideas of this type in the Domain Decomposition context (PDEs) by Smith and Widlund (89) – [“Wirebasket” techniques]
The hierarchical decomposition of a graph - example

Graph

- $C^1$ = subdomain interiors; $C^2$ = sets of edges; $C^3$ = cross-points

- Label by levels → block-diagonal structure at each level

Matrix pattern
Easy way to get an HID: Nested Dissection ordering

Up: 3-level partition of a 2-D domain. An HID tree with connector level information.
Right: Non-zero pattern of the reordered matrix.
Recursive preconditioner

\[ A_l = \begin{pmatrix} B_l & E_l \\ E_l^T & C_l \end{pmatrix} \quad \text{and} \quad C_l = A_{l+1} \quad \text{for} \quad l = 0 : L - 1, \]

\[ A_0 = \text{HID-reordered matrix } A \]

\[ A_l = \text{matrix } C_{l-1} \quad \text{for} \quad l = 1, 2, \ldots, L \]

\[ A_L = \text{submatrix associated with the top-level connector.} \]

- Each leading block \( B_l \) in \( A_l \) has a block-diagonal structure

**Goal:** Explore multilevel strategies to approximate the factorization of \( A_l \)
Recall factorization:

\[
A_l = \begin{pmatrix} I & E_l^T B_l^{-1} & I \\ \end{pmatrix} \begin{pmatrix} B_l & S_l \\ \end{pmatrix} \begin{pmatrix} I & B_l^{-1} E_l \\ \end{pmatrix}
\]

\[
S_l = C_l - E_l^T B_l^{-1} E_l
\]

Main Observation: \( S_l^{-1} - C_l^{-1} \) nearly small rank

Rank bounded by number of cross-points (connectors at level \( l \) that intersect with connectors of higher levels).
Idea: Write

\[ A_l^{-1} = \left( I - B_l^{-1} E_l \right) \left( B_l^{-1} S_l^{-1} \right) \left( \begin{array}{cc} I \\ -E_l^T B_l^{-1} I \end{array} \right). \]

- Approximate \( S_l^{-1} \) as \( S_l^{-1} \approx C_l^{-1} - W_l H_l W_l^T \)
- Next: set \( C_l = A_{l+1} \) → exploit recursivity
- Last level: use (incomplete) Cholesky.
- Next: illustration for 3 levels.
At levels $l = 0, 1, 2$ express $A_l^{-1}$ as:

$$A_l^{-1} = \left( I - B_l^{-1} E_l \right) \left( B_l^{-1} S_l^{-1} \right) \left( -E_l^T B_l^{-1} I \right).$$

$S_l^{-1}$ needed $\rightarrow$ Approximate as $S_l^{-1} \approx C_l^{-1} + W_l H_l W_l^T$

$C_l^{-1}$ needed $\rightarrow$ if $l == 2$ get $C_2 \approx L_2 L_2^T$,
else set $A_{l+1} = C_l$ & go to next level

\[
S_l = C_l - E_l^T B_l^{-1} E_l
\]

\[
\begin{pmatrix}
B_0 & \ \\
E_0^T & C_0
\end{pmatrix}
\]

\[
\begin{pmatrix}
B_1 & \ \\
E_1^T & C_1
\end{pmatrix}
\]

\[
\begin{pmatrix}
B_2 & \\
E_2^T & C_2
\end{pmatrix}
\]

\[
L_2
\]
Computing the low-rank correction

Let \( C = LL^T \) and
\[
G = L^{-1}(C - S)L^{-T}
\]
We have \( S = L(I - G)L^T \rightarrow \)
\[
S^{-1} - C^{-1} = L^{-T} [(I - G)^{-1} - I] L^{-1}
= L^{-T} [G(I - G)^{-1}] L^{-1}.
\]

Use Lanczos algorithm to get a few of the largest eigenvalues of \( G \) with associated eigenvectors:
\[
[W_l, \Sigma_l] = \text{eigs}(C_l^{-1} E_l^T B_l^{-1} E_l, k) \rightarrow
\]
\[
S_l^{-1} - C_l^{-1} \approx W_l H_l W_l^T, \quad \text{with} \quad H_l = \Sigma_l(I - \Sigma_l)^{-1}.
\]

Need to solve with \( C_l \rightarrow \) exploit recursivity
Recent work: the GeMSLR package

- Thanks: Tianshi Xu, Yuanzhe Xi, Ruipeng Li, Vasilis Kalantzis, Geoffrey Dillon,

- Extension to nonsymmetric case + full parallel implementation

- Generalized Multilevel Schur-complement, Low-Rank preconditioner (GeMSLR)

- Parallel code called GeMSLR developed in C++

- Complex version available

- Details skipped – Ruipeng will provide illustrations
Resources (url links are ‘clickable’)

- PDF of ‘Iterative methods for sparse linear systems, 2nd Ed/’
  https://www-users.cse.umn.edu/~saad/IterMethBook_2ndEd.pdf

- Links to software packages:
  https://www-users.cse.umn.edu/~saad/software/

- There you will find (for example)
  parGeMSLR
  EVSL
  pARMS

...