UNIVERSITY OF MINNESOTA TWIN CITIES

A tutorial on: Iterative methods for Sparse Linear Systems

Yousef Saad

University of Minnesota
Computer Science and Engineering

CIMPA - Tlemcen – May 14-21

Outline

Part 1

- Sparse matrices and sparsity
- Basic iterative techniques
- Projection methods
- Krylov subspace methods

Part 2

- Preconditioned iterations
- Preconditioning techniques
- Multigrid methods

Part 3

- Parallel implementations
- Parallel Preconditioners
- Software

INTRODUCTION TO SPARSE MATRICES

Typical Problem:

Physical Model
↓
Nonlinear PDEs
↓
Discretization
↓
Linearization (Newton)
↓
Sequence of Sparse Linear Systems $Ax = b$

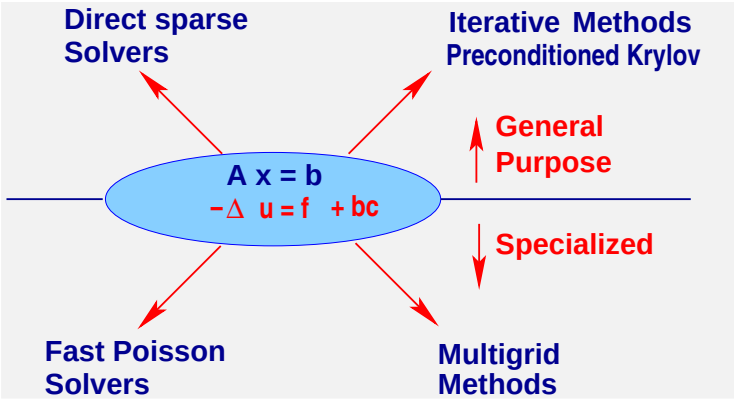
Introduction: Linear System Solvers

➤ Problem considered: Linear systems

$$Ax = b$$

➤ Can view the problem from somewhat different angles:

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



Introduction: Linear System Solvers

➤ Much of recent work on solvers has focussed on:

(1) Parallel implementation – scalable performance

(2) Improving Robustness, developing more general preconditioners

A few observations

➤ Problems are getting harder for Sparse Direct methods (more 3-D models, much bigger problems,..)

➤ Problems are also getting difficult for iterative methods Cause: more complex models - away from Poisson

➤ Researchers in iterative methods are borrowing techniques from direct methods: → preconditioners

➤ The inverse is also happening: Direct methods are being adapted for use as preconditioners

What are sparse matrices?

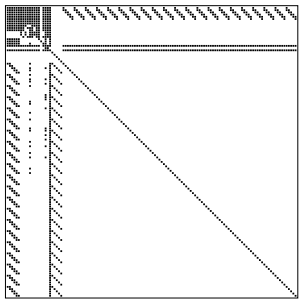
Common definition: "..matrices that allow special techniques to take advantage of the large number of zero elements and the structure."

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

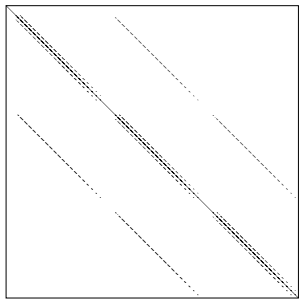
Goals: Much less storage and work than dense computations.

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

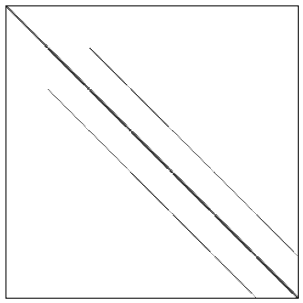
Nonzero patterns of a few sparse matrices



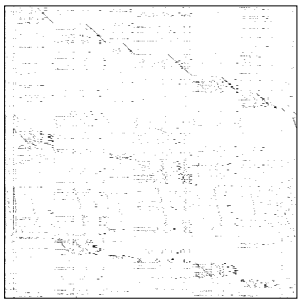
ARC130: Unsymmetric matrix from laser problem. a.r.curtis, oct 1974



SHERMAN5: fully implicit black oil simulator 16 by 23 by 3 grid, 3 unk



PORES3: Unsymmetric MATRIX FROM PORES



BP_1000: UNSYMMETRIC BASIS FROM LP PROBLEM BP

- **Two types of matrices:** structured (e.g. Sherman5) and unstructured (e.g. BP_1000)
- **Main goal of Sparse Matrix Techniques:** To perform standard matrix computations economically i.e., without storing the zeros of the matrix.
- **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 $O(n)$.

Graph Representations of Sparse Matrices

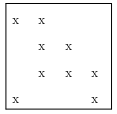
➤ **Graph theory is a fundamental tool in sparse matrix techniques.**

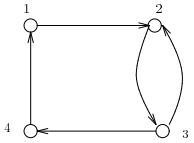
Graph $G = (V, E)$ of an $n \times n$ matrix A defined by

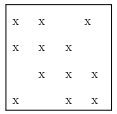
Vertices $V = \{1, 2, ..., N\}$.

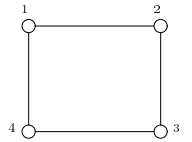
Edges $E = \{(i, j)|a_{ij} \neq 0\}$.

➤ **Graph is undirected if matrix has symmetric structure: $a_{ij} \neq 0$ iff $a_{ji} \neq 0$.**









Example: **Adjacency graph of:**

$$A = \begin{pmatrix} \star & \star & & & \star & \\ \star & \star & \star & & & \star \\ & \star & \star & & & \\ & & & \star & \star & \\ \star & & & \star & \star & \star \\ & \star & & & \star & \star \end{pmatrix}.$$

Example: **For any matrix A, what is the graph of A^2? [interpret in terms of paths in the graph of A]**

Direct versus iterative methods

Background. Two types of methods:

➤ **Direct methods : based on sparse Gaussian eimination, sparse Cholesky,..**

➤ **Iterative methods: compute a sequence of iterates which converge to the solution - preconditioned Krylov methods..**

Remark: **These two classes of methods have always been in competition.**

➤ **40 years ago solving a system with $n = 10,000$ was a challenge**

➤ **Now you can solve this in < 1 sec. on a laptop.**

➤ **Sparse direct methods made huge gains in efficiency. As a result they are very competitive for 2-D problems.**

➤ **3-D problems lead to more challenging systems [inherent to the underlying graph]**

➤ **Problems with many unknowns per grid point similar to 3-D problems**

Remarks:

- **No robust 'black-box' iterative solvers.**
- **Robustness often conflicts with efficiency**
- **However, situation improved in last $\approx$ decade**
- **Line between direct and iterative solvers blurring**

Direct Sparse Matrix Techniques

Principle of sparse matrix techniques: **Store only the nonzero elements of A. Try to minimize computations and (perhaps more importantly) storage.**

➤ **Difficulty in Gaussian elimination: Fill-in**

Trivial Example:

$$A = \begin{pmatrix} + & + & + & + & + & + \\ + & + & & & & \\ + & & + & & & \\ + & & & + & & \\ + & & & & + & \\ + & & & & & + \end{pmatrix}$$

➤ **Reorder equations and unknowns in order $N, N-1, \ldots, 1$**

➤ **A stays sparse during Gaussian eliminatin – i.e., no fill-in.**

$$A = \begin{pmatrix} + & & & & & + \\ & + & & & & + \\ & & + & & & + \\ & & & + & & + \\ & & & & + & + \\ + & + & + & + & + & + \end{pmatrix}$$

➤ **Finding the best ordering to minimize fill-in is NP-complete.**

➤ **A number of heuristics developed. Among the best known:**

- **Minimum degree ordering (Tinney Scheme 2)**
- **Nested Dissection Ordering.**
- **Approximate Minimal Degree ...**

Reorderings and graphs

➤ **Let $\pi = \{i_1, \cdots, i_n\}$ a permutation**

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

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

➤ **Define $P_\pi = I_{\pi,*}$ = "Permutation matrix" – Then:**

(1) Each row (column) of P_π consists of zeros and exactly one "1"

(2) $A_{\pi,*} = P_\pi A$

(3) $P_\pi P_\pi^T = I$

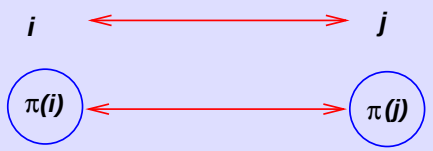
(4) $A_{*,\pi} = A P_\pi^T$

Consider now: $A' = A_{\pi,\pi} = P_\pi A P_\pi^T$

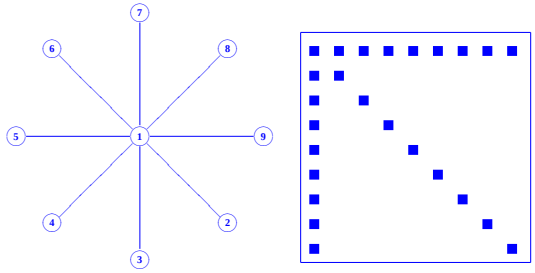
➤ **Entry (i,j) in matrix A' is exactly entry in position $(\pi(i),\pi(j))$ in A, i.e., $(a'_{ij} = a_{\pi(i),\pi(j)})$**

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

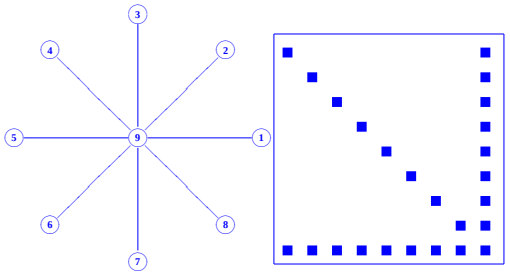
General picture :



Example **A 9 × 9 'arrow' matrix and its adjacency graph.**



Graph and matrix after permuting the nodes in reverse order.



Cuthill-McKee & reverse Cuthill-McKee

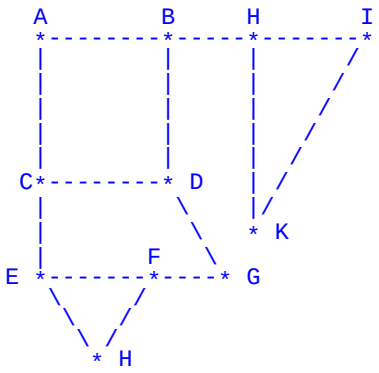
➤ **A class of reordering techniques proceeds by levels in the graph.**

➤ **Related to Breadth First Search (BFS) traversal in graph theory.**

➤ **Idea of BFS is to visit the nodes by 'levels'. Level 0 = level of starting node.**

➤ **Start with a node, visit its neighbors, then the (unmarked) neighbors of its neighbors, etc...**

Example:




```
BFS from node A:
Level 0: A
Level 1: B, C;
Level 2: E, D, H;
Level 3: I, K, E, F, G, H.
```


Implementation using levels

Algorithm $BFS(G, v)$ – by level sets –

- **Initialize $S = \{v\}$, $seen = 1$; Mark v;**
- **While $seen < n$ Do**
  - **$S_{new} = \emptyset$;**
  - **For each node v in S do**
    - **For each unmarked w in adj(v) do**
      - **Add w to S_{new};**
      - **Mark w;**
      - **$seen + +$;**
  - **$S := S_{new}$**

A few properties of Breadth-First-Search

➤ **If G is a connected undirected graph then each vertex will be visited once each edge will be inspected at least once**

➤ **Therefore, for a connected undirected graph,**

The cost of BFS is $O(|V| + |E|)$

➤ **Distance = level number;** ➤ **For each node v we have:**

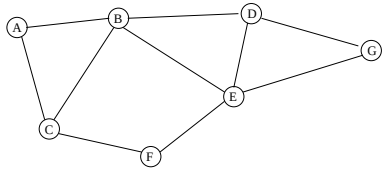
$$min\_dist(s, v) = level\_number(v) = depth_T(v)$$

➤ **Several reordering algorithms are based on variants of Breadth-First-Search**

Cuthill McKee ordering

Algorithm proceeds by levels. Same as BFS except: in each level, nodes are ordered by increasing degree

Example



Level	Nodes	Deg.	Order
0	A	2	A
1	B, C	4, 3	C, B
2	D, E, F	3, 4, 2	F, D, E
3	G	2	G

ALGORITHM : 1 ▪ *Cuthill Mc Kee ordering*

0. Find an intial node for the traversal

1. Initialize $S = \{v\}$, $seen = 1$, $\pi(seen) = v$; Mark v;

2. While $seen < n$ Do

3. $S_{new} = \emptyset$;

4. For each node v, going from lowest to highest degree, Do:

5. $\pi(++seen) = v$;

6. For each unmarked w in adj(v) do

7. Add w to S_{new};

8. Mark w;

9. EndDo

10. $S := S_{new}$

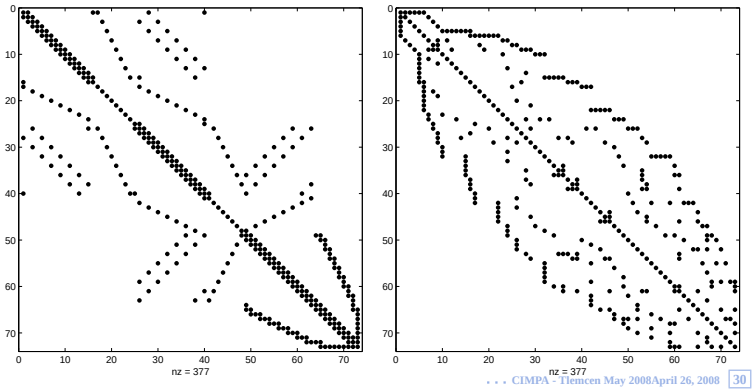
11. EndDo

12. EndWhile

Reverse Cuthill McKee ordering

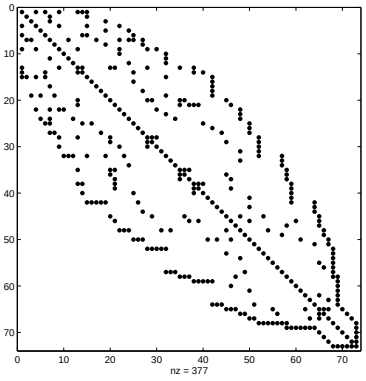
➤ **The Cuthill - Mc Kee ordering has a tendency to create small arrow matrices (going the wrong way):**

Origimal matrix **CM ordering**



➤ **Idea: Take the reverse ordering**

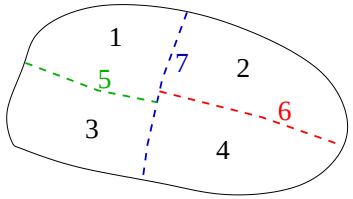
RCM ordering

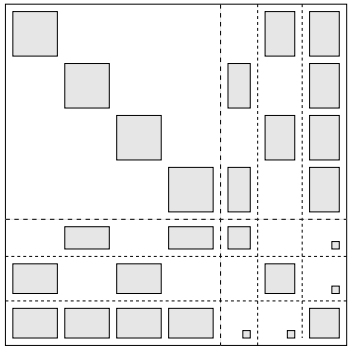


➤ **Reverse Cuthill M Kee ordering (RCM).**

Nested Dissection ordering

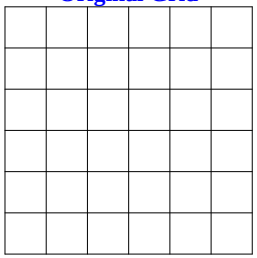
➤ **The idea of divide and conquer – recursively divide graph in two using a separator.**



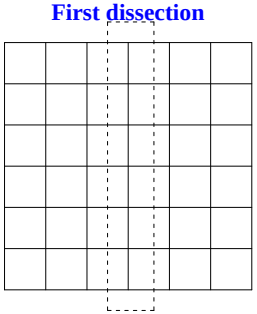


Nested dissection for a small mesh

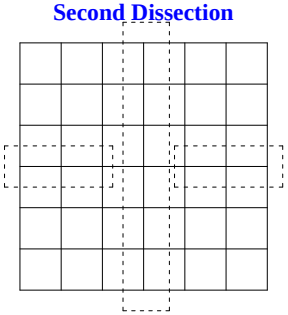
Original Grid



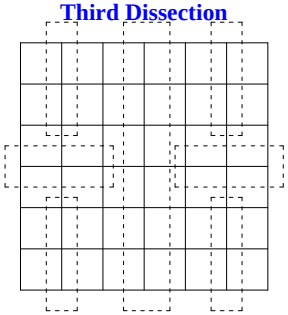
First dissection



Second Dissection



Third Dissection



Nested dissection: cost for a regular mesh

- In 2-D consider an $n \times n$ problem, $N = n^2$
- In 3-D consider an $n \times n \times n$ problem, $N = n^3$

	2-D	3-D
space (fill)	$O(N \log N)$	$O(N^{4/3})$
time (flops)	$O(N^{3/2})$	$O(N^2)$

- Significant difference in complexity between 2-D and 3-D

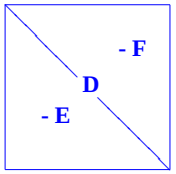
Ordering techniques for direct methods in practice

- In practice: Nested dissection (+ variants) is preferred for parallel processing
- Good implementations of Min. Degree algorithm work well in practice. Currently AMD and AMF are best known implementations/variants/
- Best practical reordering algorithms usually combine Nested dissection and min. degree algorithms.

BASIC RELAXATION METHODS

BASIC RELAXATION SCHEMES

Relaxation schemes: based on the decomposition $A = D - E - F$



D = diag(A), $-E$ = strict lower part of A and $-F$ its strict upper part.

Gauss-Seidel iteration for solving $Ax = b$**:**

$$(D - E)x^{(k+1)} = Fx^{(k)} + b$$

→ idea: correct the j**-th component of the current approximate solution,** $j = 1, 2, ..n$**, to zero the** $j - th$ **component of residual.**

Can also define a backward Gauss-Seidel Iteration:

$$(D - F)x^{(k+1)} = Ex^{(k)} + b$$

and a Symmetric Gauss-Seidel Iteration: forward sweep followed by backward sweep.

Over-relaxation is based on the decomposition:

$$\omega A = (D - \omega E) - (\omega F + (1 - \omega)D)$$

→ successive overrelaxation, (SOR):

$$(D - \omega E)x^{(k+1)} = [\omega F + (1 - \omega)D]x^{(k)} + \omega b$$

Iteration matrices

Jacobi, Gauss-Seidel, SOR, & SSOR iterations are of the form

$$x^{(k+1)} = Mx^{(k)} + f$$

- $M_{Jac} = D^{-1}(E + F) = I - D^{-1}A$
- $M_{GS}(A) = (D - E)^{-1}F == I - (D - E)^{-1}A$
- $M_{SOR}(A) = (D - \omega E)^{-1}(\omega F + (1 - \omega)D) = I - (\omega^{-1}D - E)^{-1}A$
- $M_{SSOR}(A) = I - (2\omega^{-1} - 1)(\omega^{-1}D - F)^{-1}D(\omega^{-1}D - E)^{-1}A$
  $= I - \omega(2\omega - 1)(D - \omega F)^{-1}D(D - \omega E)^{-1}A$

General convergence result

Consider the iteration: $x^{(k+1)} = Gx^{(k)} + f$

(1) Assume that $\rho(A) < 1$. Then $I - G$ is non-singular and G has a fixed point. Iteration converges to a fixed point for any f and $x^{(0)}$.

(2) If iteration converges for any f and $x^{(0)}$ then $\rho(G) < 1$.

Example: Richardson's iteration $x^{(k+1)} = x^{(k)} + \alpha(b - A^{(k)})$

$\diamondsuit$Assume $\Lambda(A) \subset \mathbb{R}$. When does the iteration converge?

- **Jacobi and Gauss-Seidel converge for diagonal dominant A**
- **SOR converges for $0 < \omega < 2$ for SPD matrices**

An observation. Introduction to Preconditioning

- **The iteration $x^{(k+1)} = Mx^{(k)} + f$ is attempting to solve $(I - M)x = f$. Since M is of the form $M = I - P^{-1}A$ this system can be rewritten as**

$$P^{-1}Ax = P^{-1}b$$

where for SSOR, we have

$$P_{SSOR} = (D - \omega E)D^{-1}(D - \omega F)$$

referred to as the SSOR 'preconditioning' matrix.

In other words:

Relaxation Scheme $\Longleftrightarrow$ Preconditioned Fixed Point Iteration

PROJECTION METHODS FOR LINEAR SYSTEMS

The Problem

We consider the linear system

$$Ax = b$$

where A is $N \times N$ and can be

- **Real symmetric positive definite**
- **Real nonsymmetric**
- **Complex**

- **Focus:**

A is large and sparse, possibly with an irregular structure

Projection Methods

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 $b - A\tilde{x} \perp L$

➤ **Leads to a small linear system ('projected problems') This is a basic projection step. Typically: sequence of such steps are applied**

➤ **With a nonzero initial guess x_0, the 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 $r_0 = b - Ax_0$. Leads to a system for δ:

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

Matrix representation:

Let

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

Then letting x be the approximate solution $\tilde{x} = x_0 + \delta \equiv x_0 + Vy$ where y is a vector of $\mathbb{R}^m$, the Petrov-Galerkin condition yields,

$$W^T(r_0 - AVy) = 0$$

and 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 $V = [v_1, \ldots, v_m]$ for K and $W = [w_1, \ldots, w_m]$ for L.

3. Compute

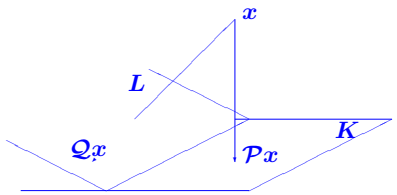
$$r \leftarrow b - Ax,$$
$$y \leftarrow (W^TAV)^{-1}W^Tr,$$
$$x \leftarrow x + Vy.$$

Operator Form Representation

Let $\mathcal{P}$ be the orthogonal projector onto K and

$\mathcal{Q}$ the (oblique) projector onto K and orthogonally to L.

$$\begin{array}{l} \mathcal{P}x \in K, \;\; x - \mathcal{P}x \perp K \\ \mathcal{Q}x \in K, \;\; x - \mathcal{Q}x \perp L \end{array}$$



The $\mathcal{P}$ and $\mathcal{Q}$ projectors

Approximate problem amounts to solving

$$\mathcal{Q}(b - Ax) = 0, \;\; x \in K$$

or in operator form

$$\mathcal{Q}(b - A\mathcal{P}x) = 0$$

Question: what accuracy can one expect?

Let x^* be the exact solution. Then

1) We cannot get better accuracy than $\|(I - \mathcal{P})x^*\|_2$, i.e.,

$$\|\tilde{x} - x^*\|_2 \geq \|(I - \mathcal{P})x^*\|_2$$

2) The residual of the *exact solution* for the *approximate problem* satisfies:

$$\|b - \mathcal{Q}A\mathcal{P}x^*\|_2 \leq \|\mathcal{Q}A(I - \mathcal{P})\|_2 \, \|(I - \mathcal{P})x^*\|_2$$

Two important particular cases.

1. $L = AK$. then $\|b - A\tilde{x}\|_2 = \min_{z \in K} \|b - Az\|_2$

   → class of minimal residual methods: CR, GCR, ORTHOMIN, GMRES, CGNR, ...

2. $L = K$ → class of Galerkin or orthogonal projection methods. When A is SPD then

$$\|x^* - \tilde{x}\|_A = \min_{z \in K} \|x^* - z\|_A.$$

One-dimensional projection processes

$$K = span\{d\} \quad \text{and} \quad L = span\{e\}$$

Then $\tilde{x} \leftarrow x + \alpha d$ and Petrov-Galerkin condition $r - A\delta \perp e$ yields

$$\alpha = \frac{(r,e)}{(Ad,e)}$$

Three popular choices:

(I) Steepest descent.

(II) Residual norm steepest descent .

(III) Minimal residual iteration.

(I) **Steepest descent.** A is SPD. Take at each step $d = r$ and $e = r$.

Iteration:

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

➤ Each step minimizes

$$f(x) = \|x - x^*\|_A^2 = (A(x - x^*), (x - x^*))$$

in direction $-\nabla f$. Convergence guaranteed if A is SPD.

(II) **Residual norm steepest descent .** A is arbitrary (nonsingular). Take at each step $d = A^T r$ and $e = Ad$.

Iteration:

$$\begin{aligned} &r \leftarrow b - Ax, d = A^T r \\ &\alpha \leftarrow \|d\|_2^2 / \|Ad\|_2^2 \\ &x \leftarrow x + \alpha d \end{aligned}$$

➤ 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.

(III) **Minimal residual iteration.** A positive definite ($A + A^T$ is SPD). Take at each step $d = r$ and $e = Ar$.

Iteration:

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

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

➤ Converges under the condition that $A + A^T$ is SPD.

Krylov Subspace Methods

Principle: Projection methods on Krylov subspaces:

$$K_m(A, v_1) = \text{span}\{v_1, Av_1, \cdots, A^{m-1}v_1\}$$

- probably the most important class of iterative methods.
- many variants exist depending on the subspace L.

Simple properties of K_m . Let μ = deg. of minimal polynomial of v

- $K_m = \{p(A)v | p = \text{polynomial of degree} \leq m - 1\}$
- $K_m = K_\mu$ for all $m \geq \mu$. Moreover, K_μ is invariant under A.
- $dim(K_m) = m$ iff $\mu \geq m$.

A little review: Gram-Schmidt process

→ **Goal: given** $X = [x_1, \ldots, x_m]$ **compute an orthonormal set** $Q = [q_1, \ldots, q_m]$ **which spans the same susbpace.**

ALGORITHM : 2 ▪ ***Classical Gram-Schmidt***

1. ***For*** $j = 1, ..., m$ ***Do:***
2. ***Compute*** $r_{ij} = (x_j, q_i)$ ***for*** $i = 1, \ldots, j-1$
3. ***Compute*** $\hat{q}_j = x_j - \sum_{i=1}^{j-1} r_{ij} q_i$
4. $r_{jj} = \|\hat{q}_j\|_2$ ***If*** $r_{jj} == 0$ ***exit***
5. $q_j = \hat{q}_j / r_{jj}$
6. ***EndDo***

ALGORITHM : 3 ▪ ***Modified Gram-Schmidt***

1. ***For*** $j = 1, ..., m$ ***Do:***
2. $\hat{q}_j := x_j$
3. ***For*** $i = 1, \ldots, j-1$ ***Do***
4. $r_{ij} = (\hat{q}_j, q_i)$
5. $\hat{q}_j := \hat{q}_j - r_{ij} q_i$
6. ***EndDo***
7. $r_{jj} = \|\hat{q}_j\|_2$. ***If*** $r_{jj} == 0$ ***exit***
8. $q_j := \hat{q}_j / r_{jj}$
9. ***EndDo***

Let:

$X = [x_1, \ldots, x_m]$ **(**$n \times m$ **matrix)**

$Q = [q_1, \ldots, q_m]$ **(**$n \times m$ **matrix)**

$R = \{r_{ij}\}$ **(**$m \times m$ **upper triangular matrix)**

➤ **At each step,**

$$x_j = \sum_{i=1}^{j} r_{ij} q_i$$

Result:

$$X = QR$$

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 $j = 1, ..., m$ **do**

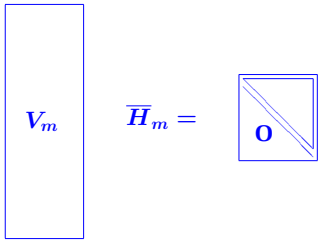
- **Compute** $w := Av_j$
- **for** $i = 1, \ldots, j$**, do** $\begin{cases} h_{i,j} := (w, v_i) \\ w := w - h_{i,j} v_i \end{cases}$
- $h_{j+1,j} = \|w\|_2$ **and** $v_{j+1} = w / h_{j+1,j}$

Result of orthogonalization process

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

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

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



Arnoldi's Method ($L_m = K_m$)

➤ Petrov-Galerkin condition when $L_m = K_m$, shows:

$$x_m = x_0 + V_m H_m^{-1} V_m^T r_0$$

➤ Select $v_1 = r_0/\|r_0\|_2 \equiv r_0/\beta$ in Arnoldi's algorithm, then:

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

Equivalent algorithms:

* FOM [YS, 1981] (above formulation)
* Young and Jea's ORTHORES [1982].
* Axelsson's projection method [1981].

Minimal residual methods ($L_m = AK_m$)

➤ When $L_m = AK_m$, we let $W_m \equiv AV_m$ and obtain:

$$x_m = x_0 + V_m[W_m^T A V_m]^{-1} W_m^T r_0$$

➤ Use again $v_1 := r_0/(\beta := \|r_0\|_2)$ and: $AV_m = V_{m+1}\bar{H}_m$

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

where y_m minimizes $\|\beta e_1 - \bar{H}_m y\|_2$ over $y \in \mathbb{R}^m$. Hence, (Generalized Minimal Residual method (GMRES) [Saad-Schultz, 1983]):

$$x_m = x_0 + V_m y_m \quad \text{where} \quad y_m : \min_y \|\beta e_1 - \bar{H}_m y\|_2$$

Equivalent methods:

- Axelsson's CGLS
- Orthomin (1980)
- Orthodir
- GCR

Restarting and Truncating

Difficulty: As m increases, storage and work per step increase fast.

First remedy: Restarting. Fix the dimension m of the subspace

ALGORITHM : 4. *Restarted GMRES (resp. Arnoldi)*

1. Start/Restart: *Compute* $r_0 = b - Ax_0$, *and* $v_1 = r_0/(\beta := \|r_0\|_2)$.
2. Arnoldi Process: *generate* $\bar{H}_m$ *and* V_m.
3. *Compute* $y_m = H_m^{-1}\beta e_1$ *(FOM), or*
   $y_m = argmin\|\beta e_1 - \bar{H}_m y\|_2$ *(GMRES)*
4. $x_m = x_0 + V_m y_m$
5. *If* $\|r_m\|_2 \leq \epsilon\|r_0\|_2$ *stop else set* $x_0 := x_m$ *and go to 1.*

Second remedy: Truncate the orthogonalization

The formula for v_{j+1} is replaced by

$$h_{j+1,j}v_{j+1} = Av_j - \sum_{i=j-k+1}^{j} h_{ij}v_i$$

→ each v_j is made orthogonal to the previous k v_i's.

→ x_m still computed as $x_0 + V_m H_m^{-1}\beta e_1$.

→ It can be shown that this is again an oblique projection process.

➤ **IOM (Incomplete Orthogonalization Method) = replace orthogonalization in FOM, by the above truncated (or 'incomplete') orthogonalization.**

The direct version of IOM [DIOM]:

Writing the LU decomposition of H_m as $H_m = L_m U_m$ we get

$$x_m = x_0 + V_m U_m^{-1} \; L_m^{-1}\beta e_1 \equiv x_0 + P_m z_m$$

➤ **Structure of L_m, U_m when $k = 3$**

$$L_m = \begin{pmatrix} 1 & & & & & \\ x & 1 & & & & \\ & x & 1 & & & \\ & & x & 1 & & \\ & & & x & 1 & \\ & & & & x & 1 \end{pmatrix} \quad U_m = \begin{pmatrix} x & x & x & & & \\ & x & x & x & & \\ & & x & x & x & \\ & & & x & x & x \\ & & & & x & x \\ & & & & & x \end{pmatrix}$$

$$p_m = u_{mm}^{-1}[v_m - \sum_{i=m-k+1}^{m-1} u_{im}p_i] \qquad z_m = \begin{bmatrix} z_{m-1} \\ \zeta_m \end{bmatrix}$$

Result: **Can update x_m at each step:**

$$x_m = x_{m-1} + \zeta_m p_m$$

Note: **Several existing pairs of methods have a similar link: they are based on the LU, or other, factorizations of the H_m matrix**

➤ **CG-like formulation of IOM called DIOM [Saad, 1982]**

➤ **ORTHORES(k) [Young & Jea '82] equivalent to DIOM(k)**

➤ **SYMMLQ [Paige and Saunders, '77] uses LQ factorization of H_m.**

➤ **Can add partial pivoting to LU factorization of H_m**

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.

In other words:

1) $h_{ij} = 0 \quad$ **for** $\quad |i - j| > 1$

2) $h_{j,j+1} = h_{j+1,j}, \quad j = 1, \ldots, m$

➤ **We can write**

$$H_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \beta_4 & \\ & & \cdot & \cdot & \cdot \\ & & & \cdot & \cdot & \cdot \\ & & & & \beta_m & \alpha_m \end{pmatrix} \quad (1)$$

The v_i's satisfy a three-term recurrence [Lanczos Algorithm]:

$$\boxed{\beta_{j+1} v_{j+1} = A v_j - \alpha_j v_j - \beta_j v_{j-1}}$$

→ simplified version of Arnoldi's algorithm for sym. systems.

Symmetric matrix + Arnoldi → Symmetric Lanczos

The Lanczos algorithm

ALGORITHM : 5 ▪ ***Lanczos***

1. ***Choose an initial vector v_1 of norm unity.***
   Set $\beta_1 \equiv 0, v_0 \equiv 0$
2. ***For $j = 1, 2, \ldots, m$ Do:***
3. $w_j := A v_j - \beta_j v_{j-1}$
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:**

- $L_m = K_m = span\{r_0, Ar_0, \ldots, A^{m-1} r_0\}$
- **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 tridigonal nature of H_m (DIOM); (3) Conjugate gradient.

ALGORITHM : 6 ▪ ***Lanczos Method for Linear Systems***

1. ***Compute $r_0 = b - A x_0$, $\beta := \|r_0\|_2$, and $v_1 := r_0/\beta$***
2. ***For $j = 1, 2, \ldots, m$ Do:***
3. $w_j = A v_j - \beta_j v_{j-1}$ ***(If $j = 1$ set $\beta_1 v_0 \equiv 0$)***
4. $\alpha_j = (w_j, v_j)$
5. $w_j := w_j - \alpha_j v_j$
6. $\beta_{j+1} = \|w_j\|_2$. ***If $\beta_{j+1} = 0$ set $m := j$ and go to 9***
7. $v_{j+1} = w_j / \beta_{j+1}$
8. ***EndDo***
9. ***Set $T_m = tridiag(\beta_i, \alpha_i, \beta_{i+1})$, and $V_m = [v_1, \ldots, v_m]$.***
10. ***Compute $y_m = T_m^{-1}(\beta e_1)$ and $x_m = x_0 + V_m y_m$***

ALGORITHM : 7 ▪ *D-Lanczos*

1. *Compute* $r_0 = b - Ax_0$, $\zeta_1 := \beta := \|r_0\|_2$, $v_1 := r_0/\beta$
2. *Set* $\lambda_1 = \beta_1 = 0$, $p_0 = 0$
3. *For* $m = 1, 2, \ldots$, *until convergence Do:*
4. *Compute* $w := Av_m - \beta_m v_{m-1}$ *and* $\alpha_m = (w, v_m)$
5. *If* $m > 1$: *Compute* $\lambda_m = \frac{\beta_m}{\eta_{m-1}}$ & $\zeta_m = -\lambda_m \zeta_{m-1}$
6. $\eta_m = \alpha_m - \lambda_m \beta_m$
7. $p_m = \eta_m^{-1}(v_m - \beta_m p_{m-1})$
8. $x_m = x_{m-1} + \zeta_m p_m$
9. *If* x_m *has converged then Stop*
10. $w := w - \alpha_m v_m$
11. $\beta_{m+1} = \|w\|_2$, $v_{m+1} = w/\beta_{m+1}$
12. *EndDo*

The Conjugate Gradient Algorithm (A S.P.D.)

- **Note: the p_i's are A-orthogonal**
- **The r_i''s are orthogonal.**
- **And we have $x_m = x_{m-1} + \xi_m p_m$**

So there must be an update of the form:

1. $p_m = r_{m-1} + \beta_m p_{m-1}$
2. $x_m = x_{m-1} + \xi_m p_m$
3. $r_m = r_{m-1} - \xi_m A p_m$

ALGORITHM : 8 ▪ *Conjugate Gradient*

Start: $r_0 := b - Ax_0$, $p_0 := r_0$.

Iterate: *Until convergence do,*

$\alpha_j := (r_j, r_j)/(Ap_j, p_j)$

$x_{j+1} := x_j + \alpha_j p_j$

$r_{j+1} := r_j - \alpha_j A p_j$

$\beta_j := (r_{j+1}, r_{j+1})/(r_j, r_j)$

$p_{j+1} := r_{j+1} + \beta_j p_j$

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 $i \neq j$.**

METHODS BASED ON LANCZOS BIORTHOGONALIZATION

ALGORITHM : 9. *Lanczos Bi-Orthogonalization*

1. *Choose two vectors* v_1, w_1 *such that* $(v_1, w_1) = 1$.
2. *Set* $\beta_1 = \delta_1 \equiv 0$, $w_0 = v_0 \equiv 0$
3. *For* $j = 1, 2, \ldots, m$ *Do:*
4. $\quad \alpha_j = (Av_j, w_j)$
5. $\quad \hat{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$
6. $\quad \hat{w}_{j+1} = A^T w_j - \alpha_j w_j - \delta_j w_{j-1}$
7. $\quad \delta_{j+1} = |(\hat{v}_{j+1}, \hat{w}_{j+1})|^{1/2}$. *If* $\delta_{j+1} = 0$ *Stop*
8. $\quad \beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1})/\delta_{j+1}$
9. $\quad w_{j+1} = \hat{w}_{j+1}/\beta_{j+1}$
10. $\quad v_{j+1} = \hat{v}_{j+1}/\delta_{j+1}$
11. *EndDo*

- **Extension of the symmetric Lanczos algorithm**
- **Builds a pair of biorthogonal bases for the two subspaces**

$$\mathcal{K}_m(A, v_1) \quad \text{and} \quad \mathcal{K}_m(A^T, w_1)$$

- **Different ways to choose $\delta_{j+1}, \beta_{j+1}$ in lines 7 and 8.**

Let

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & . & . & . & \\ & & \delta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \delta_m & \alpha_m \end{pmatrix} .$$

- $v_i \in \mathcal{K}_m(A, v_1)$ **and** $w_j \in \mathcal{K}_m(A^T, w_1)$.

If the algorithm does not break down before step m, then the vectors $v_i, i = 1, \ldots, m$, and $w_j, j = 1, \ldots, m$, are biorthogonal, i.e.,

$$(v_j, w_i) = \delta_{ij} \quad 1 \leq i, \; j \leq m \; .$$

Moreover, $\{v_i\}_{i=1,2,\ldots,m}$ is a basis of $\mathcal{K}_m(A, v_1)$ and $\{w_i\}_{i=1,2,\ldots,m}$ is a basis of $\mathcal{K}_m(A^T, w_1)$ and

$$AV_m = V_m T_m + \delta_{m+1} v_{m+1} e_m^T,$$
$$A^T W_m = W_m T_m^T + \beta_{m+1} w_{m+1} e_m^T,$$
$$W_m^T A V_m = T_m \; .$$

The Lanczos Algorithm for Linear Systems

ALGORITHM : 10. *Lanczos Alg. for Linear Systems*

1. *Compute* $r_0 = b - Ax_0$ *and* $\beta := \|r_0\|_2$
2. *Run* m *steps of the nonsymmetric Lanczos Algorithm i.e.,*
3. $\quad$ *Start with* $v_1 := r_0/\beta$, *and any* w_1 *such that*
   $(v_1, w_1) = 1$
4. $\quad$ *Generate the pair of Lanczos vectors* $v_1, \ldots, v_m$,
   and $w_1, \ldots, w_m$
5. $\quad$ *and the tridiagonal matrix* T_m *from Algorithm 9.*
6. *Compute* $y_m = T_m^{-1}(\beta e_1)$ *and* $x_m := x_0 + V_m y_m$.

- **BCG can be derived from the Lanczos Algorithm similarly to CG**

ALGORITHM : 11 ▪ *BiConjugate Gradient (BCG)*

1. *Compute* $r_0 := b - Ax_0$.
2. *Choose* r_0^* *such that* $(r_0, r_0^*) \neq 0$;
   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. $r_{j+1} := r_j - \alpha_j A p_j$
7. $r_{j+1}^* := r_j^* - \alpha_j A^T p_j^*$
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*

Quasi-Minimal Residual Algorithm

➤ Recall relation from the lanczos algorithm: $AV_m = V_{m+1}\bar{T}_m$ with $\bar{T}_m$ = $(m+1) \times m$ tridiagonal matrix $\bar{T}_m = \begin{pmatrix} T_m \\ \delta_{m+1} e_m^T \end{pmatrix}$.

➤ Let $v_1 \equiv \beta r_0$ and $x = x_0 + V_m y$. Residual norm $\|b - Ax\|_2$ equals

$$\|r_0 - AV_m y\|_2 = \|\beta v_1 - V_{m+1}\bar{T}_m y\|_2 = \|V_{m+1}\left(\beta e_1 - \bar{T}_m y\right)\|_2$$

➤ Column-vectors of V_{m+1} are not $\perp$ ($\neq$ GMRES).

➤ But: reasonable idea to minimize the function $J(y) \equiv \|\beta e_1 - \bar{T}_m y\|_2$

➤ Quasi-Minimal Residual Algorithm (Freund, 1990).

Transpose-Free Variants

➤ BCG and QMR require a matrix-by-vector product with A and A^T at each step. The products with A^T do not contribute directly to x_m. ➤ They allow to determine the scalars (α_j and β_j in BCG).

➤ QUESTION: is it possible to bypass the use of A^T?

➤ Motivation: in nonlinear equations, A is often not available explicitly but via the Frechet derivative:

$$J(u_k)v = \frac{F(u_k + \epsilon v) - F(u_k)}{\epsilon}.$$

Conjugate Gradient Squared

* Clever variant of BCG which avoids using A^T [Sonneveld, 1984].

In BCG:

$$r_i = \rho_i(A) r_0$$

where ρ_i = polynomial of degree i.

In CGS:

$$r_i = \rho_i^2(A) r_0$$

➤ Define :

$$r_j = \phi_j(A) r_0,$$

$$p_j = \pi_j(A) r_0,$$

$$r_j^* = \phi_j(A^T) r_0^*,$$
$$p_j^* = \pi_j(A^T) r_0^*$$

Scalar α_j in BCG is given by

$$\alpha_j = \frac{(\phi_j(A) r_0, \phi_j(A^T) r_0^*)}{(A\pi_j(A) r_0, \pi_j(A^T) r_0^*)} = \frac{(\phi_j^2(A) r_0, r_0^*)}{(A\pi_j^2(A) r_0, r_0^*)}$$

➤ **Possible to get a recursion for the $\phi_j^2(A) r_0$ and $\pi_j^2(A) r_0$?**

$$\begin{aligned} \phi_{j+1}(t) &= \phi_j(t) - \alpha_j t \pi_j(t), \\ \pi_{j+1}(t) &= \phi_{j+1}(t) + \beta_j \pi_j(t) \end{aligned}$$

➤ **Square these equalities**

$$\begin{aligned} \phi_{j+1}^2(t) &= \phi_j^2(t) - 2\alpha_j t \pi_j(t)\phi_j(t) + \alpha_j^2 t^2 \pi_j^2(t), \\ \pi_{j+1}^2(t) &= \phi_{j+1}^2(t) + 2\beta_j \phi_{j+1}(t)\pi_j(t) + \beta_j^2 \pi_j(t)^2. \end{aligned}$$

➤ **Problem: ...**

.. **Cross terms**

Solution: **Let $\phi_{j+1}(t)\pi_j(t)$, be a third member of the recurrence. For $\pi_j(t)\phi_j(t)$, note:**

$$\begin{aligned} \phi_j(t)\pi_j(t) &= \phi_j(t)\left(\phi_j(t) + \beta_{j-1}\pi_{j-1}(t)\right) \\ &= \phi_j^2(t) + \beta_{j-1}\phi_j(t)\pi_{j-1}(t). \end{aligned}$$

Result:

$$\begin{aligned} \phi_{j+1}^2 &= \phi_j^2 - \alpha_j t \left(2\phi_j^2 + 2\beta_{j-1}\phi_j\pi_{j-1} - \alpha_j t\, \pi_j^2\right) \\ \phi_{j+1}\pi_j &= \phi_j^2 + \beta_{j-1}\phi_j\pi_{j-1} - \alpha_j t\, \pi_j^2 \\ \pi_{j+1}^2 &= \phi_{j+1}^2 + 2\beta_j\phi_{j+1}\pi_j + \beta_j^2\pi_j^2. \end{aligned}$$

Define:

$$r_j = \phi_j^2(A) r_0, \quad p_j = \pi_j^2(A) r_0, \quad q_j = \phi_{j+1}(A)\pi_j(A) r_0$$

Recurrences become:

$$\begin{aligned} r_{j+1} &= r_j - \alpha_j A\left(2r_j + 2\beta_{j-1}q_{j-1} - \alpha_j A\, p_j\right), \\ q_j &= r_j + \beta_{j-1}q_{j-1} - \alpha_j A\, p_j, \\ p_{j+1} &= r_{j+1} + 2\beta_j q_j + \beta_j^2 p_j. \end{aligned}$$

Define auxiliary vector $d_j = 2r_j + 2\beta_{j-1}q_{j-1} - \alpha_j A p_j$

➤ **Sequence of operations to compute the approximate solution, starting with $r_0 := b - Ax_0$, $p_0 := r_0$, $q_0 := 0$, $\beta_0 := 0$.**

1. $\alpha_j = (r_j, r_0^*)/(Ap_j, r_0^*)$
2. $d_j = 2r_j + 2\beta_{j-1}q_{j-1} - \alpha_j A p_j$
3. $q_j = r_j + \beta_{j-1}q_{j-1} - \alpha_j A p_j$
4. $x_{j+1} = x_j + \alpha_j d_j$
5. $r_{j+1} = r_j - \alpha_j A d_j$
6. $\beta_j = (r_{j+1}, r_0^*)/(r_j, r_0^*)$
7. $p_{j+1} = r_{j+1} + \beta_j(2q_j + \beta_j p_j)$.

➤ **one more auxiliary vector, $u_j = r_j + \beta_{j-1}q_{j-1}$. So**

$$\begin{aligned} d_j &= u_j + q_j, \\ q_j &= u_j - \alpha_j A p_j, \\ p_{j+1} &= u_{j+1} + \beta_j(q_j + \beta_j p_j), \end{aligned}$$

➤ **vector d_j is no longer needed.**

ALGORITHM : 12 ▪ *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. $q_j = u_j - \alpha_j A p_j$
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*

➤ **Note: no matrix-by-vector products with A^T but two matrix-by-vector products with A, at each step.**

Vector: $\longleftrightarrow$ **Polynomial in BCG :**

$$q_i \longleftrightarrow \bar{r}_i(t)\bar{p}_{i-1}(t)$$
$$u_i \longleftrightarrow \bar{p}_i^2(t)$$
$$r_i \longleftrightarrow \bar{r}_i^2(t)$$

where $\bar{r}_i(t)$ = residual polynomial at step i for BCG, .i.e., $r_i = \bar{r}_i(A)r_0$, and $\bar{p}_i(t)$ = conjugate direction polynomial at step i, i.e., $p_i = \bar{p}_i(A)r_0$.

BCGSTAB (van der Vorst, 1992)

➤ **In CGS: residual polynomial of BCG is squared.** ➤ **bad behavior in case of irregular convergence.**

➤ **Bi-Conjugate Gradient Stabilized (BCGSTAB) = a variation of CGS which avoids this difficulty.** ➤ **Derivation similar to CGS.**

➤ **Residuals in BCGSTAB are of the form,**

$$r_j' = \psi_j(A)\phi_j(A)r_0$$

in which, $\phi_j(t)$ = BCG residual polynomial, and ..

➤ **.. $\psi_j(t)$ = a new polynomial defined recursively as**

$$\psi_{j+1}(t) = (1 - \omega_j t)\psi_j(t)$$

ω_i chosen to 'smooth' convergence [steepest descent step]

ALGORITHM : 13 ▪ *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. $s_j := r_j - \alpha_j A p_j$
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. $r_{j+1} := s_j - \omega_j A s_j$
9. $\beta_j := \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \times \frac{\alpha_j}{\omega_j}$
10. $p_{j+1} := r_{j+1} + \beta_j(p_j - \omega_j A p_j)$
11. *EndDo*

PRECONDITIONING

Preconditioning – Basic principles

Basic idea is to use the Krylov subspace method on a modified system such as

$$M^{-1}Ax = M^{-1}b.$$

- 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: $M = M_L M_R$.]

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:

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

Preconditioned CG (PCG)

ALGORITHM : 14 ▪ *Preconditioned Conjugate Gradient*

1. *Compute* $r_0 := b - Ax_0$, $z_0 = M^{-1}r_0$, *and* $p_0 := z_0$
2. *For* $j = 0, 1, \ldots$, *until convergence Do:*
3. $\quad \alpha_j := (r_j, z_j)/(Ap_j, p_j)$
4. $\quad x_{j+1} := x_j + \alpha_j p_j$
5. $\quad r_{j+1} := r_j - \alpha_j A p_j$
6. $\quad z_{j+1} := M^{-1} r_{j+1}$
7. $\quad \beta_j := (r_{j+1}, z_{j+1})/(r_j, z_j)$
8. $\quad p_{j+1} := z_{j+1} + \beta_j p_j$
9. *EndDo*

ALGORITHM : 15 ▪ *CG with Split Preconditioner*

1. *Compute* $r_0 := b - Ax_0$; $\hat{r}_0 = L^{-1}r_0$; *and* $p_0 := L^{-T}\hat{r}_0$.
2. *For* $j = 0, 1, \ldots$, *until convergence Do:*
3. $\quad \alpha_j := (\hat{r}_j, \hat{r}_j)/(Ap_j, p_j)$
4. $\quad x_{j+1} := x_j + \alpha_j p_j$
5. $\quad \hat{r}_{j+1} := \hat{r}_j - \alpha_j L^{-1} A p_j$
6. $\quad \beta_j := (\hat{r}_{j+1}, \hat{r}_{j+1})/(\hat{r}_j, \hat{r}_j)$
7. $\quad p_{j+1} := L^{-T}\hat{r}_{j+1} + \beta_j p_j$
8. *EndDo*

➤ The x_j's produced by the above algorithm and PCG are identical (if same initial guess is used).

Note $M^{-1}A$ is also self-adjoint with respect to $(.,.)_A$:

$$(M^{-1}Ax, y)_A = (AM^{-1}Ax, y) = (x, AM^{-1}Ay) = (x, M^{-1}Ay)_A$$

➤ Can obtain a similar algorithm

➤ Assume that M = Cholesky product $M = LL^T$.

Then, another possibility: Split preconditioning option, which applies CG to the system

$$L^{-1}AL^{-T}u = L^{-1}b, \text{ with } x = L^T u$$

➤ Notation: $\hat{A} = L^{-1}AL^{-T}$. All quantities related to the preconditioned system are indicated by $\hat{}$.

Flexible accelerators

Question: What can we do in case 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 – mixing coarse mesh / fine mesh preconditioners.

ALGORITHM : 16 ▪ *GMRES – No preconditioning*

1. **Start:** *Choose x_0 and a dimension m of the Krylov subspaces.*
2. **Arnoldi process:**
   - *Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$.*
   - *For $j = 1, ..., m$ do*
     - *Compute $w := Av_j$*
     - *for $i = 1, \ldots, j$, do* $\left\{ \begin{array}{l} h_{i,j} := (w, v_i) \\ w := w - h_{i,j} v_i \end{array} \right\}$;
     - $h_{j+1,1} = \|w\|_2$; $v_{j+1} = \frac{w}{h_{j+1,1}}$
   - *Define $V_m := [v_1, ..., v_m]$ and $\bar{H}_m = \{h_{i,j}\}$.*
3. **Form the approximate solution:** *Compute* $\boxed{x_m = x_0 + V_m y_m}$ *where* $y_m = \text{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$ *and* $e_1 = [1, 0, \ldots, 0]^T$.
4. **Restart:** *If satisfied stop, else set $x_0 \leftarrow x_m$ and goto 2.*

ALGORITHM : 17 ▪ *GMRES – (right) Preconditioning*

1. **Start:** *Choose x_0 and a dimension m*
2. **Arnoldi process:**
   - *Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$.*
   - *For $j = 1, ..., m$ do*
     - *Compute $z_j := M^{-1} v_j$*
     - *Compute $w := Az_j$*
     - *for $i = 1, \ldots, j$, do :* $\left\{ \begin{array}{l} h_{i,j} := (w, v_i) \\ w := w - h_{i,j} v_i \end{array} \right\}$
     - $h_{j+1,1} = \|w\|_2$; $v_{j+1} = w/h_{j+1,1}$
   - *Define $V_m := [v_1, ..., v_m]$ and $\bar{H}_m = \{h_{i,j}\}$.*
3. **Form the approximate solution:** $\boxed{x_m = x_0 + M^{-1} V_m y_m}$ *where* $y_m = \text{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$ *and* $e_1 = [1, 0, \ldots, 0]^T$.
4. **Restart:** *If satisfied stop, else set $x_0 \leftarrow x_m$ and goto 2.*

ALGORITHM : 18 ▪ *GMRES – variable preconditioner*

1. **Start:** *Choose x_0 and a dimension m of the Krylov subspaces.*
2. **Arnoldi process:**
   - *Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$.*
   - *For $j = 1, ..., m$ do*
     - *Compute $z_j := M_j^{-1} v_j$; Compute $w := Az_j$;*
     - *for $i = 1, \ldots, j$, do:* $\left\{ \begin{array}{l} h_{i,j} := (w, v_i) \\ w := w - h_{i,j} v_i \end{array} \right\}$;
     - $h_{j+1,1} = \|w\|_2$; $v_{j+1} = w/h_{j+1,1}$
   - *Define $Z_m := [z_1, ..., z_m]$ and $\bar{H}_m = \{h_{i,j}\}$.*
3. **Form the approximate solution:** *Compute* $\boxed{x_m = x_0 + Z_m y_m}$ *where* $y_m = \text{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$ *and* $e_1 = [1, 0, \ldots, 0]^T$.
4. **Restart:** *If satisfied stop, else set $x_0 \leftarrow x_m$ and goto 2.*

Properties

- **x_m minimizes $b - Ax_m$ over $\text{Span}\{Z_m\}$.**
- **If $Az_j = v_j$ (i.e., if preconditioning is 'exact' at step j) then approximation x_j is exact.**
- **If M_j is constant then method is $\equiv$ to Right-Preconditioned GMRES.**

Additional Costs:

- **Arithmetic: none.**
- **Memory: Must save the additional set of vectors $\{z_j\}_{j=1,...m}$**

Advantage: **Flexibility**

Standard preconditioners

- Simplest preconditioner: M = 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 Multigrid, M-level ILU, ..

An observation. Introduction to Preconditioning

➤ Take a look back at basic relaxation methods: Jacobi, Gauss-Seidel, SOR, SSOR, ...

➤ These are iterations of the form $x^{(k+1)} = Mx^{(k)} + f$ where M is of the form $M = I - P^{-1}A$. For example for SSOR,

$$P_{SSOR} = (D - \omega E)D^{-1}(D - \omega F)$$

➤ SSOR attempts to solve the equivalent system

$$P^{-1}Ax = P^{-1}b$$

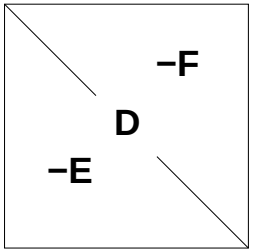
where $P \equiv P_{SSOR}$ by the fixed point iteration

$x^{(k+1)} = \underbrace{(I - P^{-1}A)}_{M} x^{(k)} + P^{-1}b$ instead of $x^{(k+1)} = (I - A)x^{(k)} + b$

In other words:

Relaxation Scheme $\Longleftrightarrow$ Preconditioned Fixed Point Iteration

The SOR/SSOR preconditioner



➤ SOR preconditioning

$$M_{SOR} = (D - \omega E)$$

➤ SSOR preconditioning

$$M_{SSOR} = (D - \omega E)D^{-1}(D - \omega F)$$

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

- k-step SOR (resp. SSOR) preconditioning:

  k steps of SOR (resp. SSOR)

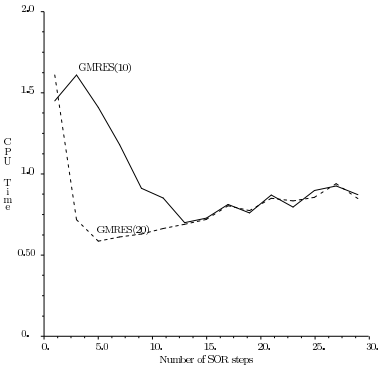
- Questions: Best ω? For preconditioning can take $\omega = 1$

  $$M = (D - E)D^{-1}(D - F)$$

Observe: $M = LU + R$ with $R = ED^{-1}F$.

- Best k? $k = 1$ is rarely the best. Substantial difference in performance.

Iteration times versus k for SOR(k) preconditioned GMRES



ILU(0) and IC(0) preconditioners

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

- Formal definition of ILU(0):

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

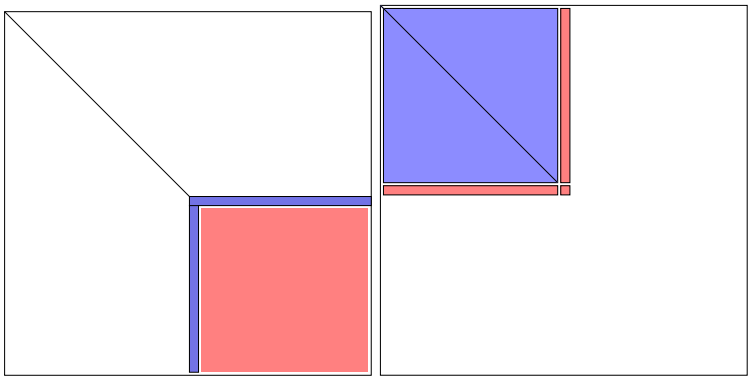
- This does not define $ILU(0)$ in a unique way.

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

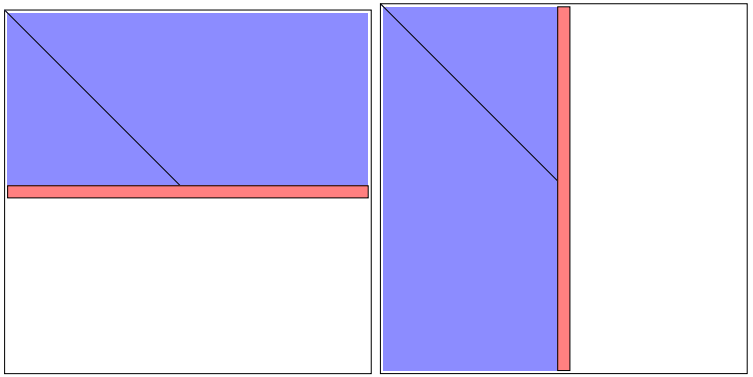
- ILU factorizations are often based on i, k, j version of GE.

What is the IKJ version of GE?

Different computational patterns for gaussian elimination



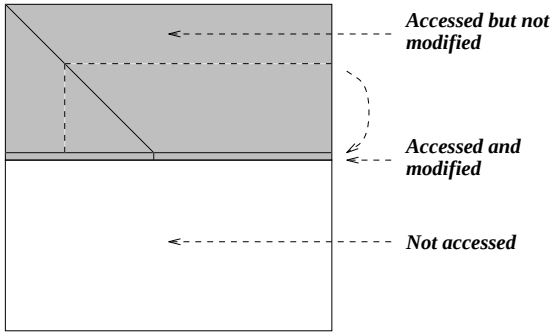
KJI,KJI IJK



IKJ **JKI**

ALGORITHM : 19 ▪ ***Gaussian Elimination – IKJ Variant***

1. ***For*** $i = 2, \ldots, n$ ***Do:***
2. ***For*** $k = 1, \ldots, i-1$ ***Do:***
3. $a_{ik} := a_{ik}/a_{kk}$
4. ***For*** $j = k+1, \ldots, n$ ***Do:***
5. $a_{ij} := a_{ij} - a_{ik} * a_{kj}$
6. ***EndDo***
7. ***EndDo***
8. ***EndDo***



ILU(0) – zero-fill ILU

ALGORITHM : 20 ▪ ***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:***

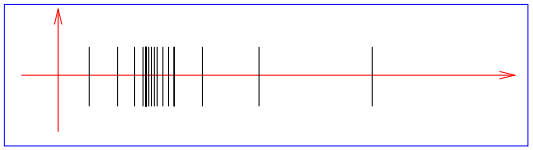
compute $a_{ij} := a_{ij} - a_{ik}a_{k,j}$.

EndFor

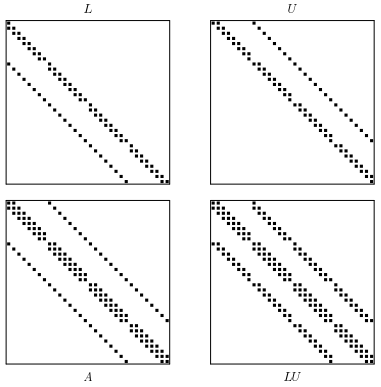
EndFor

➤ **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

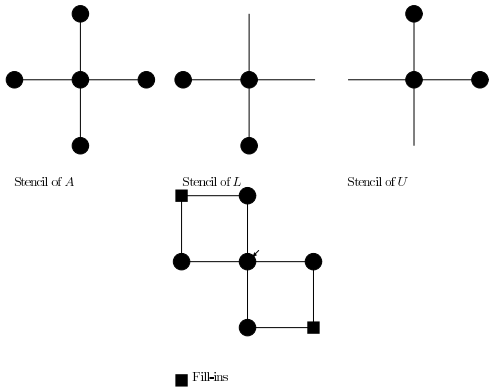


Pattern of ILU(0) for 5-point matrix



Stencils and ILU factorization

Stencils of A and the L and U parts of A:



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:**

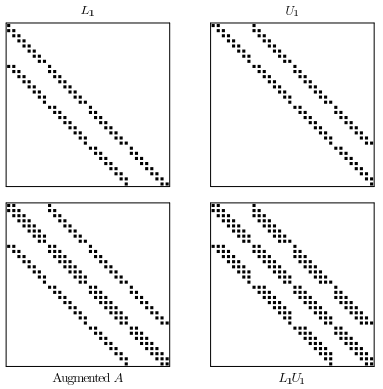
$$\boxed{Lev_{kj} = \min\{Lev_{kj}; Lev_{ki} + Lev_{ij} + 1\}}$$

➤ **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 : 21 ▪ ***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 ➤ in Symbolic phase**

ILU with threshold – generic algorithms

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

➤ **One remedy: ILU with threshold – (generic name ILUT.)**

Two broad approaches:

First approach [derived from direct solvers]: use any (direct) sparse solver and incorporate a dropping strategy. [Munksgaard (?), Osterby & Zlatev, Sameh & Zlatev[90], D. Young, & al. (Boeing) etc...]

Second approach : [derived from 'iterative solvers' viewpoint]

1. use a (row or colum) version of the (i, k, j) version of GE;
2. apply a drop strategy for the elment l_{ik} as it is computed;
3. perform the linear combinations to get a_{i*}. Use full row expansion of a_{i*};
4. apply a drop strategy to fill-ins.

ILU with threshold: ILUT(k, ϵ)

- Do the i, k, j version of Gaussian Elimination (GE).
- During each i-th step in GE, discard any pivot or fill-in whose value is below $\epsilon \|row_i(A)\|$.
- 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 –

➤ Can be made quite inexpensive.

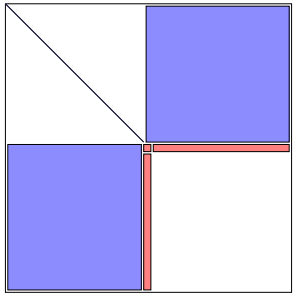
Crout-based ILUT (ILUTC)

Terminology: Crout versions of LU compute the k-th row of U and the k-th column of L at the k-th step.

Computational pattern

Black = part computed at step k

Blue = part accessed



Main advantages:

1. Less expensive than ILUT (avoids sorting)
2. Allows better techniques for dropping

References:

[1] M. Jones and P. Plassman. An improved incomplete Choleski factorization. *ACM Transactions on Mathematical Software*, 21:5–17, 1995.

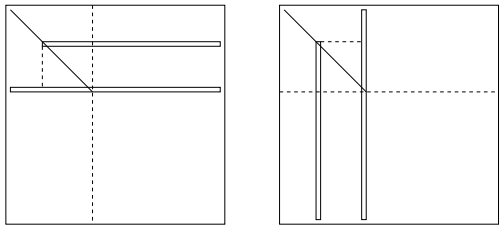
[2] S. C. Eisenstat, M. H. Schultz, and A. H. Sherman. Algorithms and data structures for sparse symmetric Gaussian elimination. *SIAM Journal on Scientific Computing*, 2:225–237, 1981.

[3] M. Bollhöfer. A robust ILU with pivoting based on monitoring the growth of the inverse factors. *Linear Algebra and its Applications*, 338(1–3):201–218, 2001.

[4] N. Li, Y. Saad, and E. Chow. Crout versions of ILU. MSI technical report, 2002.

Crout LU (dense case)

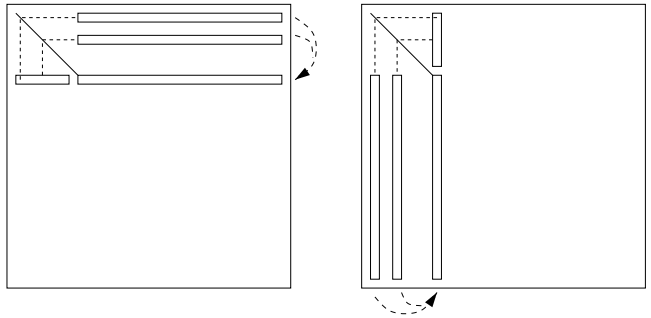
➤ Go back to delayed update algorithm (IKJ alg.) and observe: we could do both a column and a row version



➤ Left: U computed by rows. Right: L computed by columns

Note: Entries $1 : k-1$ in k-th row of figure need not be computed. Available from already computed columns of L.

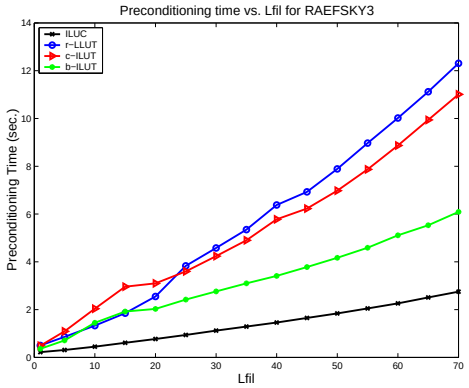
Similar observation for L (right)



ALGORITHM : 22. *Crout LU Factorization (dense case)*

1. *For* $k = 1 : n$ *Do :*
2. *For* $i = 1 : k-1$ *and if* $a_{ki} \neq 0$ *Do :*
3. $a_{k,k:n} = a_{k,k:n} - a_{ki} * a_{i,k:n}$
4. *EndDo*
5. *For* $i = 1 : k-1$ *and if* $a_{ik} \neq 0$ *Do :*
6. $a_{k+1:n,k} = a_{k+1:n,k} - a_{ik} * a_{k+1:n,i}$
7. *EndDo*
8. $a_{ik} = a_{ik}/a_{kk}$ *for* $i = k+1, ..., n$
9. *EndDo*

Comparison with standard techniques



Precondition time vs. Lfil for ILUC (solid), row-ILUT (circles), column-ILUT (triangles) and r-ILUT with Binary Search Trees (stars)

ILUM AND ARMS

Independent set orderings & ILUM (Background)

Independent set orderings permute a matrix into the form

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix}$$

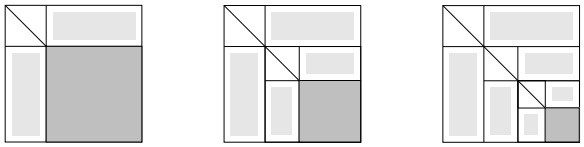
where B is a diagonal matrix.

- **Unknowns associated with the B block form an independent set (IS).**
- **IS is maximal if it cannot be augmented by other nodes to form another IS.**
- **IS ordering can be viewed as a "simplification" of multicoloring**

Main observation: Reduced system obtained by eliminating the unknowns associated with the IS, is still sparse since its coefficient matrix is the Schur complement

$$S = C - EB^{-1}F$$

- **Idea: apply IS set reduction recursively.**
- **When reduced system small enough solve by any method**
- **Can devise an ILU factorization based on this strategy.**



- **See work by [Botta-Wubbs '96, '97, YS'94, '96, (ILUM), Leuze '89, ..]**

ALGORITHM : 23. ***ILUM***

For $lev = 1, nlev$ ***Do***

a. *Get an independent set for A.*

b. *Form the reduced system associated with this set;*

c. *Apply a dropping strategy to this system;*

d. *Set A := current reduced matrix and go back to (a).*

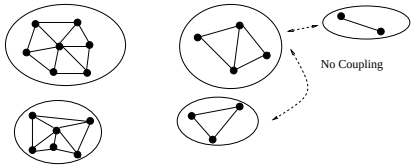
EndDo

Group Independent Sets / Aggregates

- Generalizes (common) Independent Sets

Main goal: to improve robustness

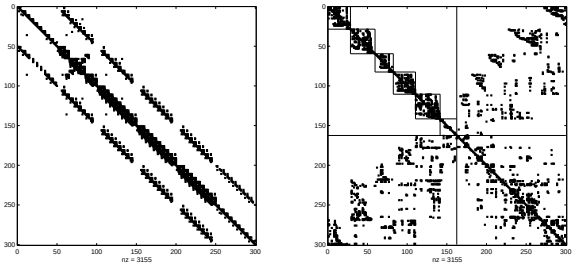
Main idea: use independent sets of "cliques", or "aggregates". There is no coupling between the aggregates.



- Reorder equations so nodes of independent sets come first

Algebraic Recursive Multilevel Solver (ARMS)

Original matrix, A , and reordered matrix, $A_0 = P_0^T A P_0$.

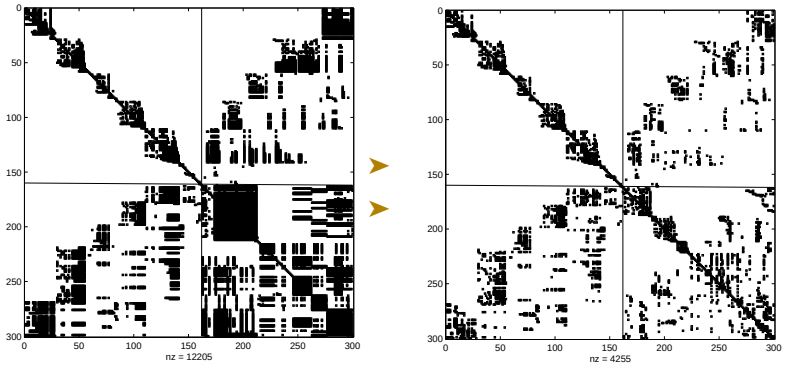


- Block ILU factorization of A_l

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

- Diagonal blocks treated as sparse

Problem: Fill-in **Remedy:** dropping strategy



- Next step: treat the Schur complement recursively

Algebraic Recursive Multilevel Solver (ARMS)

Basic step:

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \qquad \rightarrow$$

$$\begin{pmatrix} L & 0 \\ EU^{-1} & I \end{pmatrix} \times \begin{pmatrix} U & L^{-1}F \\ 0 & S \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}$$

where $S = C - EB^{-1}F$ = Schur complement.

- Perform block factorization recursively on S
- L, U Blocks: sparse
- Exploit recursivity

Factorization: at level l $P_l^T A_l P_l =$

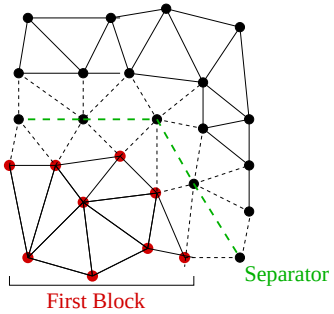
$$\begin{pmatrix} B_l & F_l \\ E_l & C_l \end{pmatrix} \approx \begin{pmatrix} L_l & 0 \\ E_l U_l^{-1} & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & A_{l+1} \end{pmatrix} \begin{pmatrix} U_l & L_l^{-1} F_l \\ 0 & I \end{pmatrix}$$

- **L-solve $\sim$ restriction. U-solve $\sim$ prolongation.**
- **Solve Last level system with, e.g., ILUT+GMRES**

ALGORITHM : 24 ▪ ***ARMS(A_{lev}) factorization***

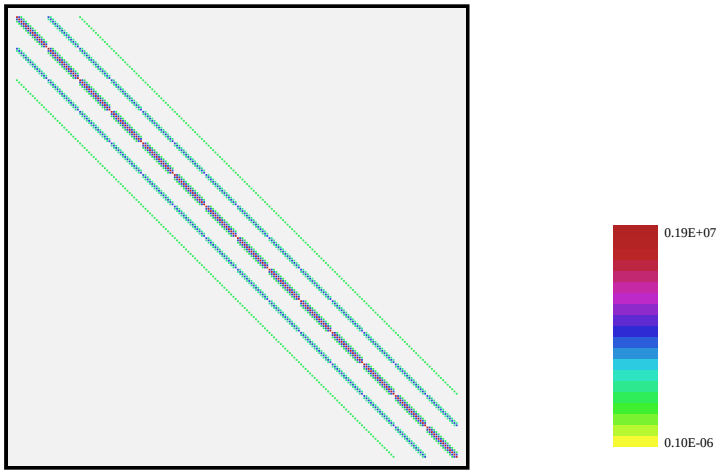
1. *If $lev = last\_lev$ then*
2. *Compute $A_{lev} \approx L_{lev} U_{lev}$*
3. *Else:*
4. *Find an independent set permutation P_{lev}*
5. *Apply permutation $A_{lev} := P_{lev}^T A_{lev} P$*
6. *Compute factorization*
7. *Call ARMS(A_{lev+1})*
8. *EndIf*

Group Independent Set reordering

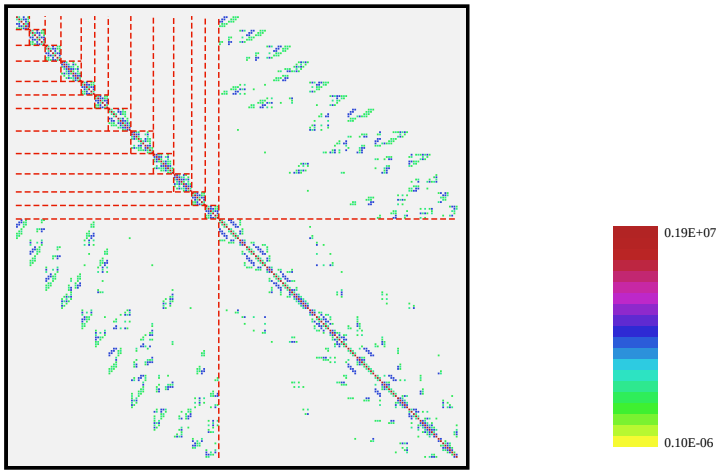


Simple strategy used: **Do a Cuthill-MKee ordering until there are enough points to make a block. Reverse ordering. Start a new block from a non-visited node. Continue until all points are visited. Add criterion for rejecting "not sufficiently diagonally dominant rows."**

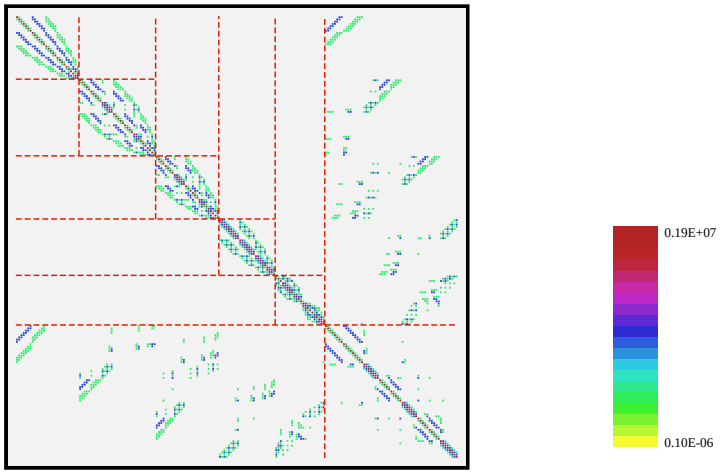
Original matrix



Block size of 6



Block size of 20



M U L T I G R I D (VERY BRIEF)

Introduction

- **Premise: we now work directly on a Partial Differential Equation e.g.**

$$-\Delta u = f, \ +\text{B.C}$$

- **Main idea of multigrid: exploit a hierarchy of grids to get good convergence from simple iterative schemes**
- **Need a good grasp of matrices and spectra of model problems**

Richardson's iteration

- Simple iterative scheme: Richardson's iteration for 1-D case
- Fixed parameter ω. Iteration:

$$u_{j+1} = u_j + \omega(b - Au_j) = (I - \omega A)u_j + \omega b\,.$$

- Iteration matrix is

$$M_\omega = I - \omega A\,.$$

Recall: convergence takes place for $0 < \omega < 2/\rho(A)$

- In practice an upper bound $\rho(A) \leq \gamma$ is often available

- take $\omega = 1/\gamma \rightarrow$ converging iteration since

$1/\gamma \leq 1/\rho(A) < 2/\rho(A)$.

- Eigenvalues of the iteration matrix are $1 - \omega\lambda_k$, where

$$\lambda_k = 2(1 - \cos\theta_k) = 4\sin^2\tfrac{\theta_k}{2}$$

- Eigenvectors are the same as those of A.
- If u_* is the exact solution, the error vector $d_j \equiv u_* - u_j$, obeys the relation,

$$d_j = M_\omega^j d_0$$

- Expand the error vector d_0 in the eigenbasis of M_ω, as

$$d_0 = \sum_{k=1}^{n} \xi_k w_k\,.$$

- Then from $d_j = M_\omega^j d_0$ and $M_\omega = I - \omega A$:

$$d_j = \sum_{k=1}^{n} \left(1 - \frac{\lambda_k}{\gamma}\right)^j \xi_k w_k\,.$$

- Each component is reduced by $(1 - \lambda_k/\gamma)^j$.
- Slowest converging component corresponds to λ_1
- Possibly very slow convergence rate when $|\lambda_1/\gamma|\lambda 1$.
- For the model problem – one-dimensional case – Gershgorin's theorem yields $\gamma = 4$, so the corresponding reduction coefficient is

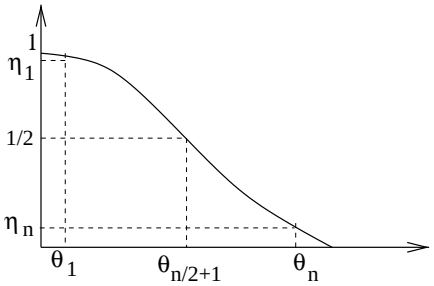
$$1 - \sin^2\frac{\pi}{2(n+1)} \approx 1 - (\pi h/2)^2 = 1 - O(h^2)\,.$$

Consequence: convergence can be quite slow for fine meshes, i.e., when h is small.

Basic observation: convergence is not similar for all components.

- Half of the error components see a very good decrease.
- This is the case for the *high frequency* components, that is, all those components corresponding to $k > n/2$. [referred to as the *oscillatory part*]
- The reduction factors for these components are

$$\eta_k = 1 - \sin^2\frac{k\pi}{2(n+1)} = \cos^2\frac{k\pi}{2(n+1)} \leq \frac{1}{2}\,.$$



Reduction coefficients for Richardson's method applied to the 1-D model problem

Observations: **Oscillatory components, undergo excellent reduction, Also reduction factor is independent of the step-size h.**

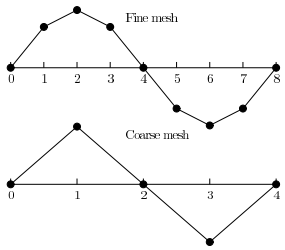
How can we reduce the other components?

➤ **Introduce a coarse grid problem. Assume n is odd. Consider the problem issued from discretizing the original PDE on a mesh Ω_{2h} with the mesh-size $2h$. Use superscripts h and $2h$ for the two meshes.**

Observation: **note that $x_i^{2h} = x_{2i}^h$ from which it follows that, for $k \leq n/2$,**

$$w_k^h(x_{2i}^h) = \sin(k\pi x_{2i}^h) = \sin(k\pi x_i^{2h}) = w_k^{2h}(x_i^{2h}) \ .$$

So: Taking a smooth mode on the fine grid (w_k^h with $k \leq n/2$) and canonically injecting it into the coarse grid, i.e., defining its values on the coarse points to be the same as those on the fine points, yields the k-th mode on the coarse grid.



The mode w_2 on a fine grid $(n = 7)$ and a coarse grid $(n = 3)$

➤ **Some of the modes which were smooth on the fine grid, become oscillatory.**

➤ **The oscillatory modes on the fine mesh are no longer represented on the coarse mesh.**

➤ **At some point iteration will fail to make progress on the fine grid: when the only components left are those associated with the smooth modes.**

Multigrid strategy: **do not attempt to eliminate these components on the fine grid. Move down to a coarser grid where smooth modes are translated into oscillatory ones. Then iterate.**

➤ **Practically, need to go back and forth between different grids.**

Inter-grid operations: Prolongation

➤ A prolongation operation takes a vector from Ω_H and defines the analogue vector in Ω_h. Common notation in use

$$I_H^h : \Omega_H \longrightarrow \Omega_h \, .$$

➤ Simplest approach : linear interpolation. Example: in 1-D. Given $(v_i^{2h})_{i=0,\ldots,(n+1)/2}$, define vector $v^h = I_{2h}^h v^{2h} \in \Omega_h$:

$$\begin{cases} v_{2j}^h &= v_j^{2h} \\ v_{2j+1}^h &= (v_j^{2h} + v_{j+1}^{2h})/2 \end{cases} \quad \text{for} \quad j = 0, \ldots, \frac{n+1}{2} \, .$$

Restriction

➤ Given a function v^h on the fine mesh, a corresponding function in Ω_H must be defined from v^h. Reverse of prolongation

➤ Simplest example canonical injection: $v_i^{2h} = v_{2i}^h$.

➤ Termed Injection operator. Obvious 2-D analogue: $v_{i,j}^{2h} = v_{2i,2j}^h$.

➤ More common restriction operator: full weighting (FW),

$$v_j^{2h} = \frac{1}{4}\left(v_{2j-1}^h + 2v_{2j}^h + v_{2j+1}^h\right) \, .$$

➤ Averages the neighboring values using the weights $0.25, 0.5, 0.25$.

Standard multigrid techniques

➤ Simplest idea: obtain an initial guess from interpolating a solution computed on a coarser grid. Repeat recursively. Interpolation from a coarser grid can be followed by a few s of a smoothing iteration.

➤ known as nested iteration

➤ MG techniques use essentially two main ingredients: a hierarchy of grid problems (restrictions and prolongations) and a smoother.

What is a smoother? Ans: any scheme which has the smoothing property of damping quickly the high frequency components of the error.

Coarse problems and smoothers

At the highest level (finest grid)

$$A_h u^h = f^h$$

➤ Can define this problem at next level (mesh-size= H) on mesh Ω_H

➤ Also common (FEM) to define the system by *Galerkin projection*,

$$A_H = I_h^H A_h I_H^h \, , \qquad f^H = I_h^H f^h \, .$$

Notation: $u_\nu^h = \texttt{smoother}^\nu(\texttt{A}_\texttt{h}, \texttt{u}_0^\texttt{h}, \texttt{f}_\texttt{h})$

u_ν^h == result of ν smoothing s for $A_h u = f_h$ starting with u_0^h.

V-cycles

ALGORITHM : 25 ▪ $u^h = \textit{V-cycle}(A_h, u_0^h, f^h)$

1. *Pre-smooth:* $u^h := \texttt{smoother}^{\nu_1}(\texttt{A}_\texttt{h}, \texttt{u}_0^\texttt{h}, \texttt{f}^\texttt{h})$
2. *Get residual:* $r^h = f^h - A_h u^h$
3. *Coarsen:* $r^H = I_h^H r^h$
4. *If* $(H == h_0)$
5. *Solve:* $A_H \delta^H = r^H$
6. *Else*
7. *Recursion:* $\delta^H = \textit{V-cycle}(A_H, 0, r^H)$
8. *EndIf*
9. *Correct:* $u^h := u^h + I_H^h \delta^H$
10. *Post-smooth:* $u^h := \texttt{smoother}^{\nu_2}(\texttt{A}_\texttt{h}, \texttt{u}^\texttt{h}, \texttt{f}^\texttt{h})$
11. *Return* u^h

➤ **V-cyle multigrid is a canonical application of recursivity - starting from the 2-grid cycle as an example. Notation:** $H = 2h$ **and** h_0 **is the coarsest mesh-size.**

➤ **Many other options avaialble**

➤ **For Poisson equation MG is a 'fast solver' – cost of order** $N \log N$**. In fact** $O(N)$ **for FMG.**

Algebraic Multigrid (AMG)

➤ **Generalizes 'geometric' or 'mesh-based' MG to general problems**

➤ **Idea: Define coarsening by looking at 'strong' couplings**

➤ **Define coarse problem from a Galerkin approach, i.e., using the restriction** $A_H = I_h^H A_h I_H^h$**...**

➤ **Generally speaking: limited success for problems with several unknowns per mesh point, of for non-PDE relatex problems..**

PARALLEL IMPLEMENTATION

Introduction

- Thrust of parallel computing techniques in most applications areas.
- Programming model: Message-passing seems (MPI) dominates
- Open MP and threads for small number of processors
- Important new reality: parallel programming has penetrated the 'applications' areas [Sciences and Engineering + industry]
- Problem 1: algorithms lagging behind somewhat
- Problem 2: Message passing is painful for large applications. 'Time to solution' up.

Parallel preconditioners: A few approaches

"Parallel matrix computation" viewpoint:

- Local preconditioners: Polynomial (in the 80s), Sparse Approximate Inverses, [M. Benzi-Tuma & al '99., E. Chow '00]
- Distributed versions of ILU [Ma & YS '94, Hysom & Pothen '00]
- Use of multicoloring to unaravel parallelism

Domain Decomposition ideas:

- Schwarz-type Preconditioners [e.g. Widlund, Bramble-Pasciak-Xu, X. Cai, D. Keyes, Smith, ...]
- Schur-complement techniques [Gropp & Smith, Ferhat et al. (FETI), T.F. Chan et al., YS and Sosonkina '97, J. Zhang '00, ...]

Multigrid / AMG viewpoint:

- Multi-level Multigrid-like preconditioners [e.g., Shadid-Tuminaro et al (Aztec project), ...]
- In practice: Variants of additive Schwarz very common (simplicity)

Intrinsically parallel preconditioners

Some alternatives

(1) Polynomial preconditioners;

(2) Approximate inverse preconditioners;

(3) Multi-coloring + independent set ordering;

(4) Domain decomposition approach.

POLYNOMIAL PRECONDITIONING

Principle: $M^{-1} = s(A)$ where s is a (low) degree polynomial:

$$s(A)Ax = s(A)b$$

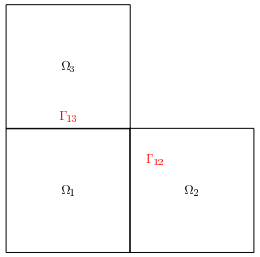
Problem: how to obtain s? Note: $s(A) \approx A^{-1}$

- Several approaches.
  - * Chebyshev polynomials
  - * Least squares polynomials
  - * Others
- Polynomial preconditioners are seldom used in practice.

Domain Decomposition

Problem:

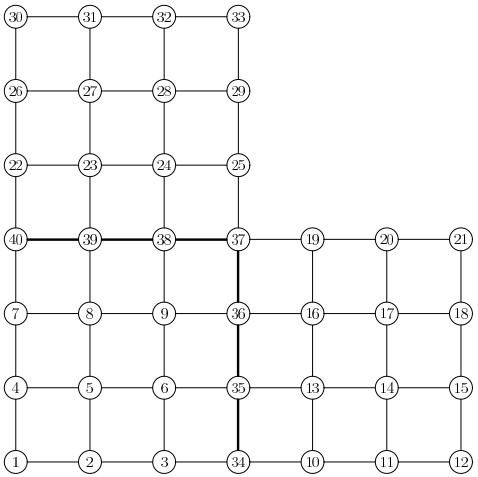
$$\begin{cases} \Delta u = f \text{ in } \Omega \\ u = u_\Gamma \text{ on } \Gamma = \partial\Omega. \end{cases}$$



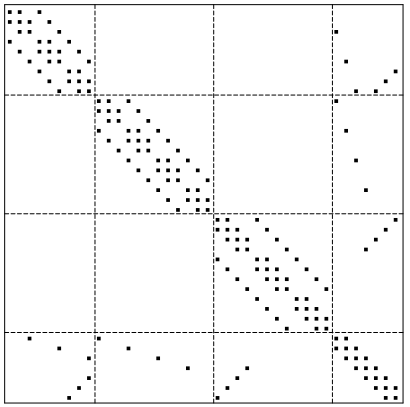
Domain:

$$\Omega = \bigcup_{i=1}^{s} \Omega_i,$$

- Domain decomposition or substructuring methods attempt to solve a PDE problem (e.g.) on the entire domain from problem solutions on the subdomains Ω_i.

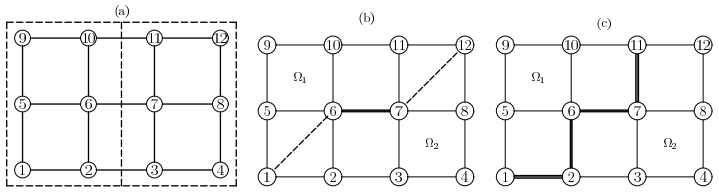


Discretization of domain



Coefficient Matrix

Types of mappings



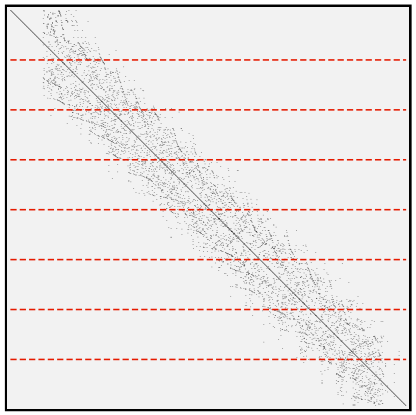
(a) Vertex-based; (b) edge-based; and (c) element-based partitioning

- **Can adapt PDE viewpoint to general sparse matrices**
- **Will use the graph representation and 'vertex-based' viewpoint –**

DISTRIBUTED SPARSE MATRICES

Generalization: Distributed Sparse Systems

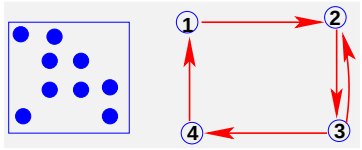
- **Simple illustration: Block assignment. Assign equation i and unknown i to a given 'process'**
- **Naive partitioning - won't work well in practice**



- **Best idea is to use the adjacency graph of A:**

Vertices = $\{1, 2, \cdots, n\}$;

Edges: $i \rightarrow j$ iff $a_{ij} \neq 0$



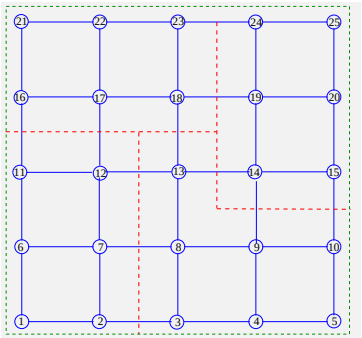
Graph partitioning problem:

- **Want a partition of the vertices of the graph so that**

(1) partitions have $\sim$ the same sizes

(2) interfaces are small in size

General Partitioning of a sparse linear system



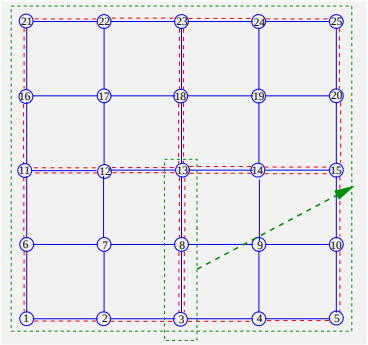
$S_1 = \{1, 2, 6, 7, 11, 12\}$: This means equations and unknowns 1, 2, 3, 6, 7, 11, 12 are assigned to Domain 1.

$S_2 = \{3, 4, 5, 8, 9, 10, 13\}$

$S_3 = \{16, 17, 18, 21, 22, 23\}$

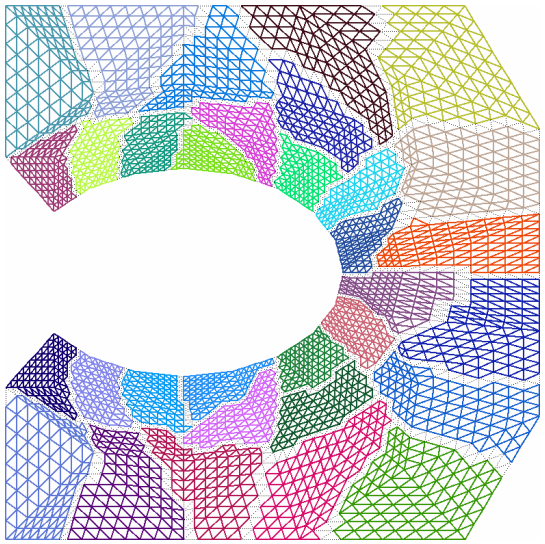
$S_4 = \{14, 15, 19, 20, 24, 25\}$

Alternative: Map elements / edges rather than vertices



Equations/unknowns 3, 8, 12 shared by 2 domains. From distributed sparse matrix viewpoint this is an overlap of one layer

- Partitioners : Metis, Chaco, Scotch, ..
- More recent: Zoltan, H-Metis, PaToH



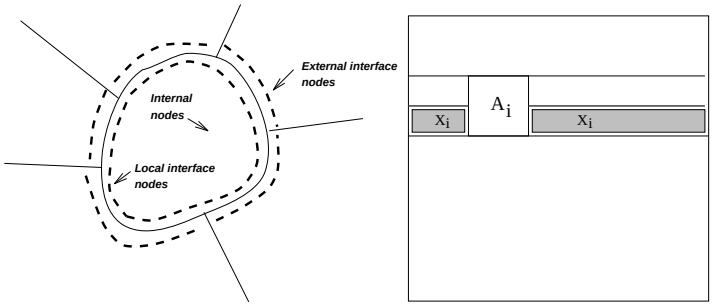
- Standard dual objective: "minimize" communication + "balance" partition sizes
- Recent trend: use of hypergraphs [PaToh, Hmetis,...]
- Hypergraphs are very general.. Ideas borrowed from VLSI work
- Main motivation: to better represent communication volumes when partitioning a graph. Standard models face many limitations
- Hypergraphs can better express complex graph partitioning problems and provide better solutions. Example: completely nonsymmetric patterns.

Distributed Sparse matrices (continued)

➤ **Once a good partitioning is found, questions are:**

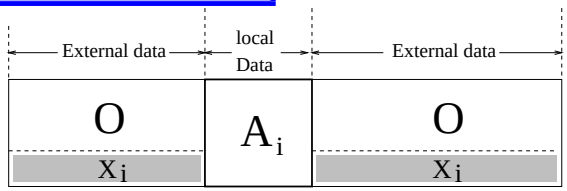
1. **How to represent this partitioning?**
2. **What is a good data structure for representing distributed sparse matrices?**
3. **How to set up the various "local objects" (matrices, vectors, ..)**
4. **What can be done to prepare for communication that will be required during execution?**

Two views of a distributed sparse matrix



➤ **Local interface variables always ordered last.**

Local view of distributed matrix:

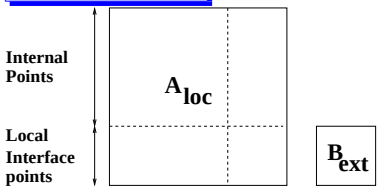


The local system:

$$\underbrace{\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix}}_{A_i} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \underbrace{\begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix}}_{y_{ext}} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}$$

➤ u_i **: Internal variables;** y_i **: Interface variables**

The local matrix:



The local matrix consists of 2 parts: a part ('A_{loc}') which acts on local data and another ('B_{ext}') which acts on remote data.

➤ **Once the partitioning is available these parts must be identified and built locally..**

➤ **In finite elements, assembly is a local process.**

➤ **How to perform a matrix vector product? [needed by iterative schemes?]**

Distributed Sparse Matrix-Vector Product Kernel

Algorithm:

1. Communicate: exchange boundary data.

Scatter x_{bound} to neighbors - Gather x_{ext} from neighbors

2. Local matrix – vector product

$$y = A_{loc}x_{loc}$$

3. External matrix – vector product

$$y = y + B_{ext}x_{ext}$$

NOTE: 1 and 2 are independent and can be overlapped.

Main Operations in (F) GMRES :

1. Saxpy's – local operation – no communication

2. Dot products – global operation

3. Matrix-vector products – local operation – local communication

4. Preconditioning operations – locality varies.

Distributed Dot Product

```
/*-------------------- call blas1 function
   tloc = DDOT(n, x, incx, y, incy);
/*-------------------- call global reduction
   MPI_Allreduce(&tloc,&ro,1,MPI_DOUBLE,MPI_SUM,comm);
```


A remark: the global viewpoint

$$\left(\begin{array}{ccc|ccccc} B_1 & & & F_1 & & & & \\ & B_2 & & & F_2 & & & \\ & & \ddots & & & \cdot & & \\ & & \ddots & & & \cdot & & \\ & & B_p & & & & & F_p \\ \hline E_1 & & & C_1 & E_{12} & \cdots & & E_{1p} \\ & E_2 & & E_{21} & C_2 & \cdots & & E_{2p} \\ & & \ddots & \vdots & \vdots & \vdots & & \\ & & E_p & E_{p1} & E_{p2} & \cdots & & C_p \end{array}\right) \left(\begin{array}{c} u_1 \\ u_2 \\ \vdots \\ \vdots \\ u_p \\ y_1 \\ y_2 \\ \vdots \\ y_p \end{array}\right) = \left(\begin{array}{c} f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_p \\ g_1 \\ g_2 \\ \vdots \\ g_p \end{array}\right)$$

← **Interior variables** → ← **Interface variables** →

PARALLEL PRECONDITIONERS

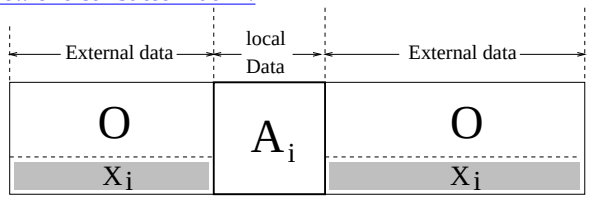
Three approaches:

- Schwarz Preconditioners
- Schur-complement based Preconditioners
- Multi-level ILU-type Preconditioners

➤ **Observation:** Often, in practical applications, Schwarz Preconditioners are used : SUB-OPTIMAL

Domain-Decomposition-Type Preconditioners

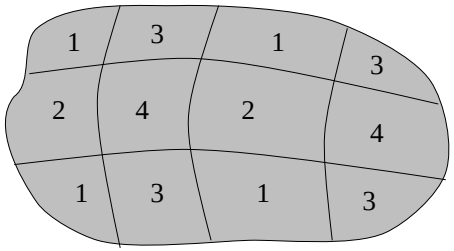
Local view of distributed matrix:



Block Jacobi Iteration (Additive Schwarz):

1. Obtain external data y_i
2. Compute (update) local residual $r_i = (b - Ax)_i = b_i - A_i x_i - B_i y_i$
3. Solve $A_i \delta_i = r_i$
4. Update solution $x_i = x_i + \delta_i$

➤ Multiplicative Schwarz. Need a coloring of the subdomains.



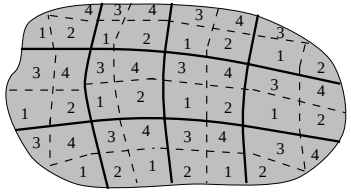
Multicolor Block SOR Iteration (Multiplicative Schwarz):

1. **Do** $col = 1, \ldots, numcols$
2. **If** $(col.eq.mycol)$ **Then**
3. **Obtain external data** y_i
4. **Update local residual** $r_i = (b - Ax)_i$
5. **Solve** $A_i \delta_i = r_i$
6. **Update solution** $x_i = x_i + \delta_i$
7. **EndIf**
8. **EndDo**

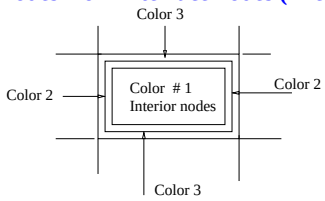
Breaking the sequential color loop

➤ **"Color" loop is sequential. Can be broken in several different ways.**

(1) Have a few subdomains per processors



(2) Separate interior nodes from interface nodes (2-level blocking)



(3) Use a block-GMRES algorithm - with Block-size = number of colors. SOR step targets a different color on each column of the block ➤ no iddle time.

Local Solves

➤ **Each local system $A_i \delta_i = r_i$ can be solved in three ways:**

1. By a (sparse) direct solver

2. Using a standard preconditioned Krylov solver

3. Doing a backward-forward solution associated with an accurate ILU (e.g. ILUT) precondioner

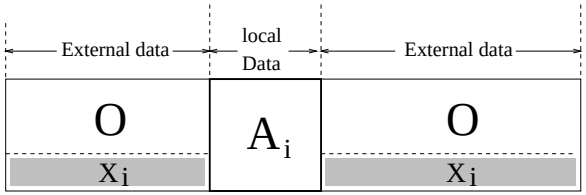
➤ **We only use (2) with a small number of inner s (up to 10) or (3).**

SCHUR COMPLEMENT-BASED PRECONDITIONERS

Schur complement system

Local system can be written as

$$A_i x_i + X_i y_{i,ext} = b_i. \qquad (2)$$



x_i= vector of local unknowns, $y_{i,ext}$ = external interface variables, and b_i = local part of RHS.

- **Local equations**

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix} \qquad (3)$$

- **eliminate u_i from the above system:**

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - E_i B_i^{-1} f_i \equiv g_i',$$

where S_i is the "local" Schur complement

$$S_i = C_i - E_i B_i^{-1} F_i. \qquad (4)$$

Structure of Schur complement system

Global Schur complement system: $Sy = g'$ **with :**

$$S = \begin{pmatrix} S_1 & E_{12} & \dots & E_{1p} \\ E_{21} & S_2 & \dots & E_{2p} \\ \vdots & & \ddots & \vdots \\ E_{p1} & E_{p-1,2} & \dots & S_p \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix} = \begin{pmatrix} g_1' \\ g_2' \\ \vdots \\ g_p' \end{pmatrix}.$$

- **E_{ij}'s are sparse = same as in the original matrix**
- **Can solve global Schur complement system iteratively. Back-substitute to recover rest of variables (internal).**
- **Can use the procedure as a preconditining to global system.**

Simplest idea: Schur Complement Iterations

$$\begin{pmatrix} u_i \\ y_i \end{pmatrix} \begin{matrix} \text{Internal variables} \\ \text{Interface variables} \end{matrix}$$

- Do a global primary iteration (e.g., block-Jacobi)
- Then accelerate only the y variables (with a Krylov method)

Still need to precondition..

Approximate Schur-LU

- Two-level method based on induced preconditioner. Global system can also be viewed as

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} \begin{pmatrix} u \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}, \quad B = \left(\begin{array}{cccc|c} B_1 & & & & F_1 \\ & B_2 & & & F_2 \\ & & \ddots & & \vdots \\ & & & B_p & F_p \\ \hline E_1 & E_2 & \cdots & E_p & C \end{array}\right)$$

Block LU factorization of A:

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} = \begin{pmatrix} B & 0 \\ E & S \end{pmatrix} \begin{pmatrix} I & B^{-1}F \\ 0 & I \end{pmatrix},$$

Preconditioning:

$$L = \begin{pmatrix} B & 0 \\ E & M_S \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} I & B^{-1}F \\ 0 & I \end{pmatrix}$$

with M_S = some approximation to S.

- Preconditioning to global system can be induced from any preconditioning on Schur complement.

Rewrite local Schur system as

$$y_i + S_i^{-1} \sum_{j \in N_i} E_{ij} y_j = S_i^{-1} \left[g_i - E_i B_i^{-1} f_i \right].$$

- equivalent to Block-Jacobi preconditioner for Schur complement.
- Solve with, e.g., a few s (e.g., 5) of GMRES

- Question: How to solve with S_i?
- Can use LU factorization of local matrix $A_i = \begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix}$

and exploit the relation:

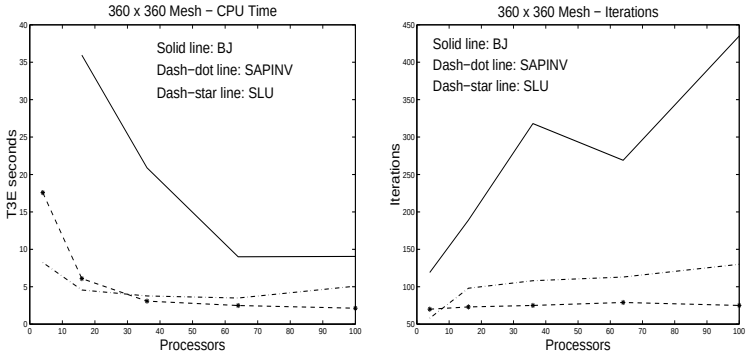
$$A_i = \begin{pmatrix} L_{B_i} & 0 \\ E_i U_{B_i}^{-1} & L_{S_i} \end{pmatrix} \begin{pmatrix} U_{B_i} & L_{B_i}^{-1} F_i \\ 0 & U_{S_i} \end{pmatrix} \quad \rightarrow \quad L_{S_i} U_{S_i} = S_i$$

- Need only the (I) LU factorization of the A_i [rest is already available]
- Very easy implementation of (parallel) Schur complement techniques for vertex-based partitioned systems : YS-Sosonkina '97; YS-Sosonkina-Zhang '99.

An experiment

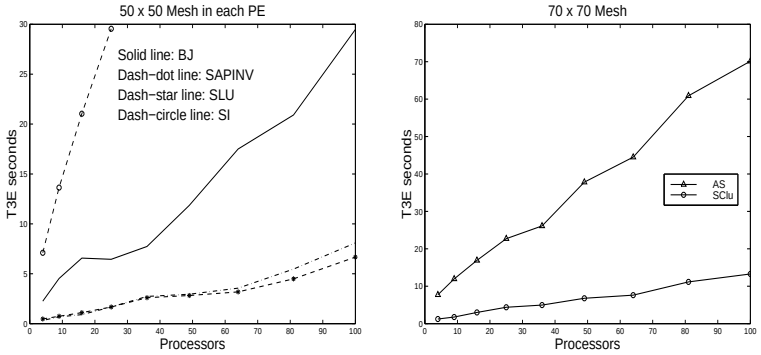
Number of FGMRES(20) iterations for the AF23560 problem.

Name	Precon	lfil	16	24	32	40	56	64	80	96
af23560	SAPINV	20	32	36	27	29	73	35	71	61
		30	32	35	23	29	46	60	33	52
	SAPINVS	20	32	35	24	29	55	35	37	59
		30	32	34	23	28	43	45	23	35
	SLU	20	81	105	94	88	90	76	85	71
		30	38	34	37	39	38	39	38	35
	BJ	20	37	153	53	60	77	80	95	*
		30	36	41	53	57	81	87	97	115



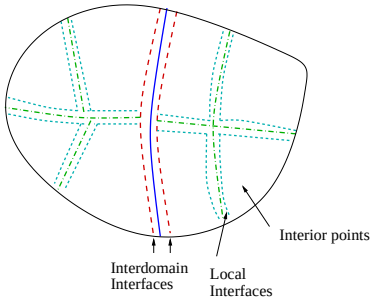
Times and iteration counts for solving a 360×360 discretized Laplacean problem with 3 different preconditioners using flexible GMRES(10).

➤ **Solution times for a Laplacean problem with various local subproblem sizes using FGMRES(10) with 3 different preconditioners (BJ, SAPINV, SLU) and the Schur complement iteration (SI).**



PARALLEL ARMS

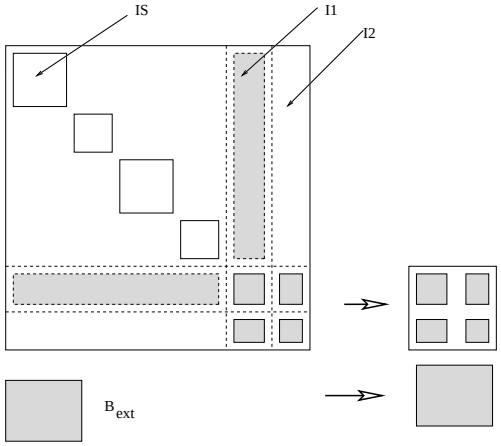
Parallel implementation of ARMS



Three types of points: interior (independent sets), local interfaces, and global interfaces

Main ideas: (1) exploit recursivity (2) distinguish two phases: elimination of interior points and then interface points.

Result: 2-part Schur complement: one corresponding to local interfaces and the other to inter-domain interfaces.



Three approaches

Method 1: Simple additive Schwarz using ILUT or ARMS locally

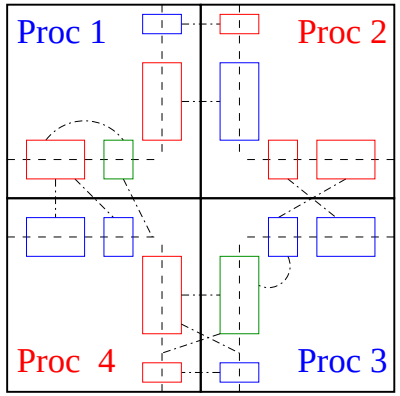
Method 2: Schur complement approach. Solve Schur complement system (both I1 and I2) with either a block Jacobi (M. Sosonkina and YS, '99) or multicolor ILU(0).

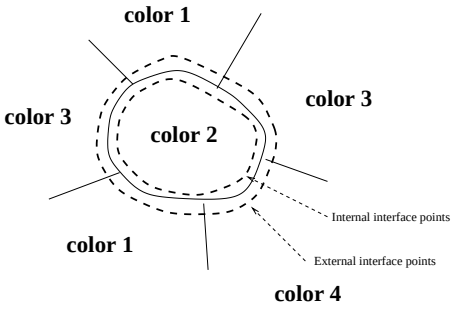
Method 3: Do independent set reduction across subdomains. Requires construction of global group independent sets.

➤ Current status Methods 1 and 2.

Construction of global group independent sets A two level strategy

1. Color subdomains
2. Find group independent sets locally
3. Color groups consistently





Algorithm: Multicolor Distributed ILU(0)

1. **Eliminate local rows,**
2. **Receive external interf. rows from PEs s.t.** $color(PE) <$ `MyColor`
3. **Process local interface rows**
4. **Send local interface rows to PEs s.t.** $color(PE) >$ `MyColor`

Methods implemented in pARMS:

add_x **Additive Schwarz with method x for subdomains. With/out overlap. x = one of ILUT, ILUK, ARMS.**

sch_x **Schur complement technique, with method x = factorization used for local submatrix. Same x as above. Equiv. to Additive Schwarz preconditioner on Schur complement.**

sch_sgs **Multicolor Multiplicative Schwarz (block Gauss-Seidel) preconditioning is used instead of additive Schwarz for Schur complement.**

sch_gilu0 **ILU(0) preconditioning to solve global Schur complement system obtained from ARMS reduction.**

Test problem

1. Scalability experiment: sample finite difference problem.

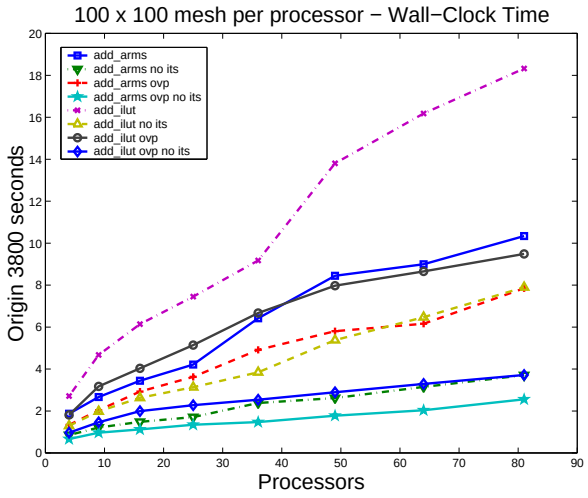
$$-\Delta u + \gamma\left(e^{xy}\frac{\partial u}{\partial x} + e^{-xy}\frac{\partial u}{\partial y}\right) + \alpha u = f\ ,$$

Dirichlet Boundary Conditions ; $\gamma = 100, \alpha = -10$**; centered differences discretization.**

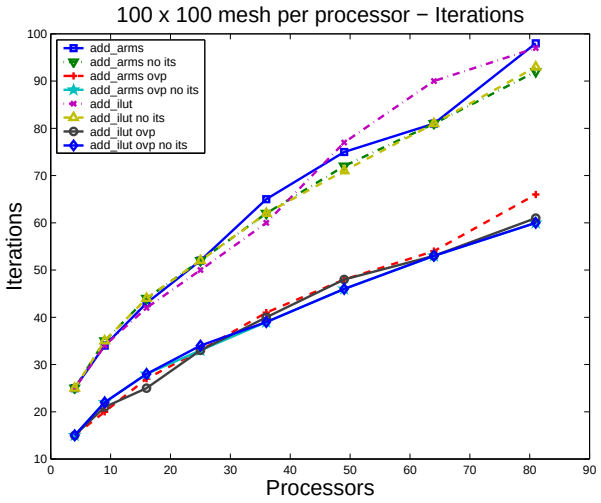
➤ **Keep size constant on each processor** $[100 \times 100]$ ➤ **Global linear system with** $10,000 * nproc$ **unknowns.**

2. Comparison with a parallel direct solver – symmetric problems

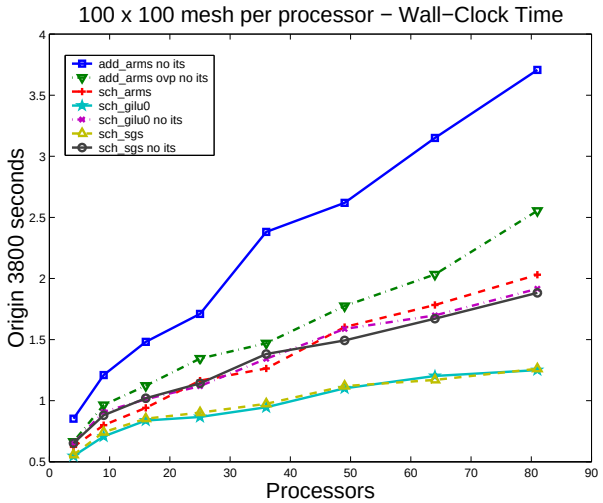
3. Large irregular matrix example arising from magneto hydrodynamics.



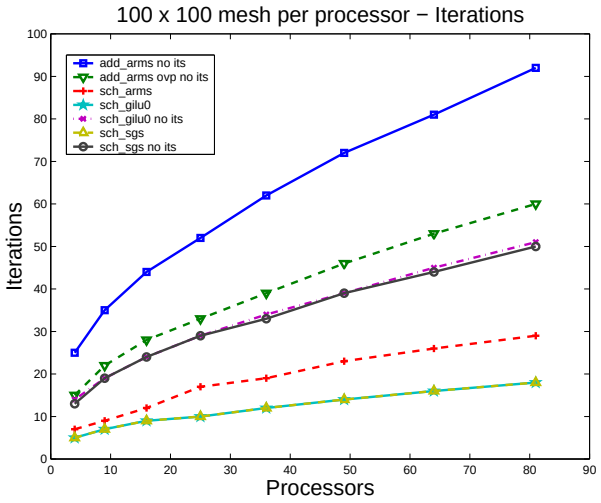
Times for 2D PDE problem with fixed subproblem size



Iterations for 2D PDE problem with fixed subproblem size



Times for 2D PDE problem with fixed subproblem size



Iterations

Software

Direct solvers:

- **SUPERLU**

  `http://crd.lbl.gov/ xiaoye/SuperLU/`

- **MUMPS: [cerfacs]**

- **Univ. Minn. / IBM's PSPASES [SPD matrices]**

  `http://www-users.cs.umn.edu/ mjoshi/pspases/`

- **UMFPACK**

Iterative solvers:

- **PETSc**

  `http://acts.nersc.gov/petsc/`

and Trilinos (more recent)

`http://trilinos.sandia.gov/`

... are very comprehensive packages..

- **PETSc includes few preconditioners...**
- **Hypre, ML, ..., all include interfaces to PETSc or trilinos**
- **pARMS:**

  `http://www.cs.umn.edu~saad/software`

is a more modest effort -