

SOLVING SPARSE LINEAR SYSTEMS OF EQUATIONS

Chao Yang

Computational Research Division

Lawrence Berkeley National Laboratory

Berkeley, CA, USA



OUTLINE

- Sparse matrix storage format
- Basic factorization algorithm
 - Left-looking
 - Right-looking
 - Multi-frontal
- Supernodes and block algorithm
- Elimination tree and symbolic factorization
- Matrix ordering
- Parallel left-looking factorization algorithm for share-memory machines
- Parallel right-looking factorization algorithm for distributed-memory machines
- Parallel triangular substitution
- Sparse solvers: PARDISOL, SuperLU, MUMPS

Sparse Matrix and storage form

- Triplet format: 3 arrays: rowind, colind, nzvals

rowind	colind	nzvals
1	1	1
3	1	2
...
4	2	5

$$A = \begin{pmatrix} 1 & & & & & & & & & \\ & 4 & & & & & & & & \\ 2 & & 6 & & & & & & & \\ 3 & & 5 & & 8 & & & & & \\ & & & 7 & & 9 & & & & \end{pmatrix}$$

- Compressed sparse column (CSC) format

nzvals	1	2	3	4	5	6	7	8	9
rowind	1	3	4	2	4	3	5	4	5
colptr	1	4	6	8	9	10			

- Compressed sparse row (CSR) format
- Other format (skyline, ELLPACK format)

Sparse matrix-vector multiplication

- $y \leftarrow Ax$

- non-symmetric version

```

for j = 1:n
    for i = colptr(j):colptr(j+1)-1
        y(rowind(i))=y(rowind(i))+nzvals(i)*x(j)
    end
end

```

- Symmetric version

```

for j = 1:n
    for i = colptr(j):colptr(j+1)-1
        y(rowind(i))=y(rowind(i))+nzvals(i)*x(j)
        if (rowind(i) != j) then
            y(j) = y(j) + nzvals(i)*x(rowind(i))
        end
    end
end
end

```

indirect addressing

Basic algorithm for sparse Cholesky

- Recall

$$A = \begin{pmatrix} \sqrt{\alpha_{11}} & \\ a/\sqrt{\alpha_{11}} & I \end{pmatrix} \begin{pmatrix} 1 & \\ & \hat{A} - \frac{aa^T}{\alpha_{11}} \end{pmatrix} \begin{pmatrix} \sqrt{\alpha_{11}} & a^T/\sqrt{\alpha_{11}} \\ & I \end{pmatrix}$$

- Left-looking:

$$l = \frac{a}{\sqrt{\alpha_{11}}} \text{ (cdiv)} \quad \hat{A}e_1 = \hat{A}e_1 - e_1^T l l^T e_1 \text{ (cmod)}$$

- Algorithm:

```

for j = 1:n
  If (j>1) then
    foreach k such that  $L_{jk} \neq 0$  do
      cmod(j,k)
    end
  end
  If (j<n) then
    cdiv(j)
  end
end
end
  
```

exploit sparsity



Non-zero fill

- The L factor can be much denser than the original matrix
- The extra nonzeros in L are called nonzero fills
- The positions of these nonzeros should be determined (quickly) by a preprocessing procedure called symbolic factorization
- The number of nonzeros in L can be reduced/minimized by properly reordering the rows and columns of the matrix

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

$$L = \begin{pmatrix} & + & & & \\ & & + & & \\ + & & + & & \\ + & + & \oplus & + & \\ & & + & & + \end{pmatrix}$$

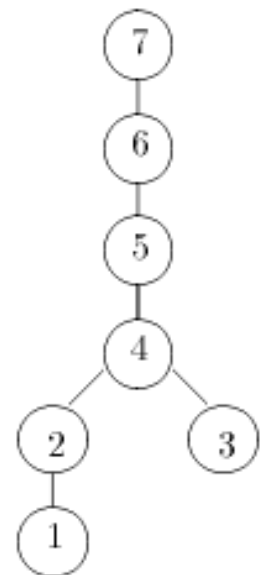
Symbolic factorization and elimination tree

- Symbolic factorization is used to determine the nonzero structure of L before the matrix A is factored numerically
- The nonzero structure of the j th column of L is determined by the nonzero structure of the k th column of L for all $k \leq j$ such that $L_{jk} \neq 0$
- The column dependency can be represented by a tree called elimination tree

$$L = \begin{pmatrix} 1 & & & & & & & \\ \times & 2 & & & & & & \\ & & 3 & & & & & \\ \times & \times & \times & 4 & & & & \\ & & \times & \times & 5 & & & \\ \times & \times & & \times & \times & 6 & & \\ & \times & & \times & \times & \times & 7 & \end{pmatrix}$$

Each node is mapped to a matrix column number

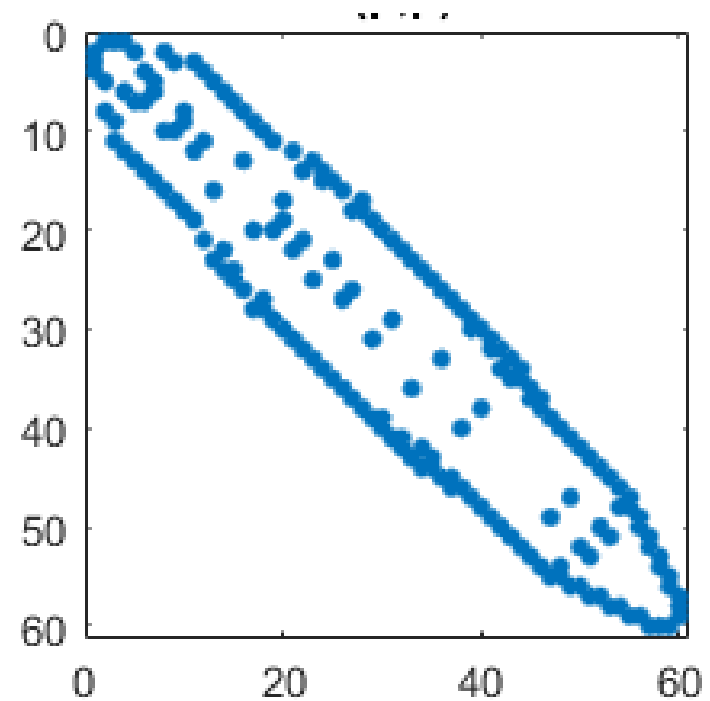
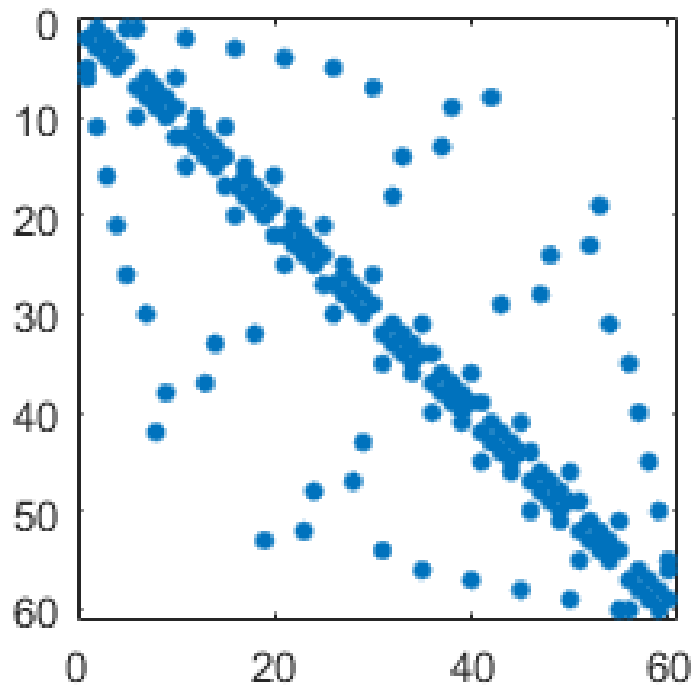
Parent(k) = the smallest row index j such that $L_{jk} \neq 0$



Matrix reordering

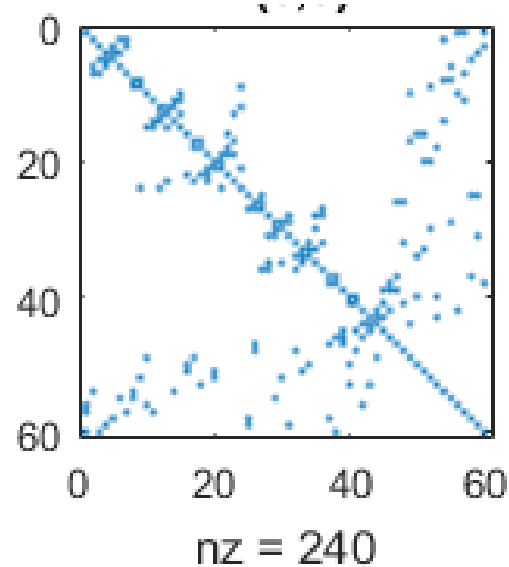
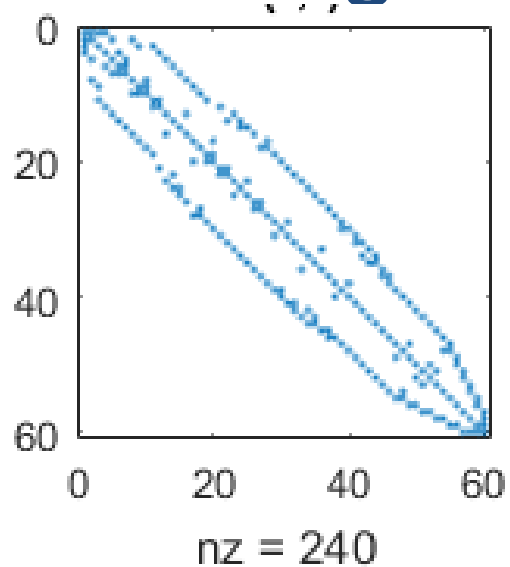
- Reverse Cuthill-McGee (make reordered matrix narrow banded)
- Minimum degree (greedy algorithm, heuristics to minimize potential nonzero-fill)
- Minimum fill
- Nested dissection (divide-and-conquer, motivated by mesh domain decomposition)

Reverse Cuthill-McKee

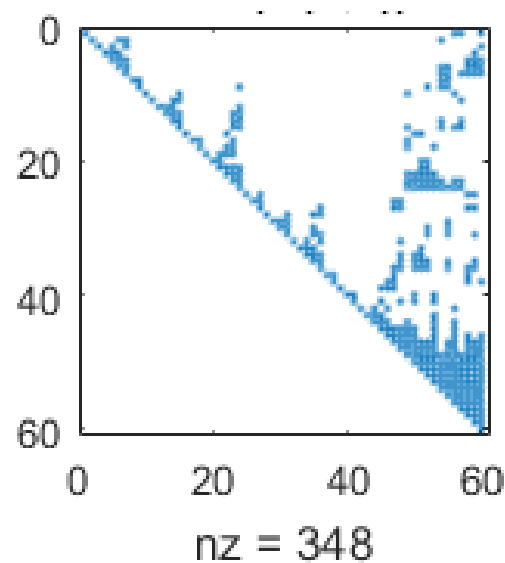
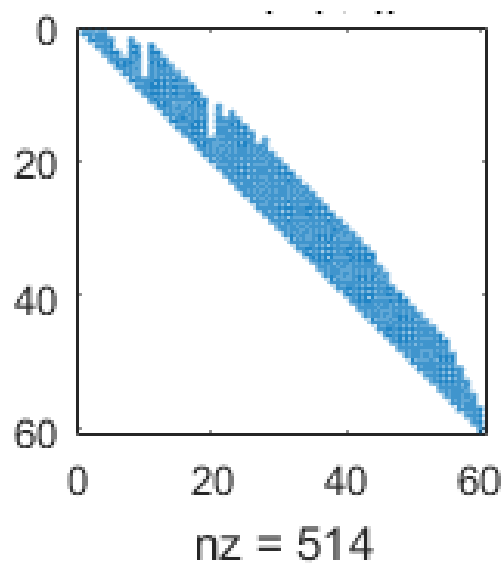


Minimum degree

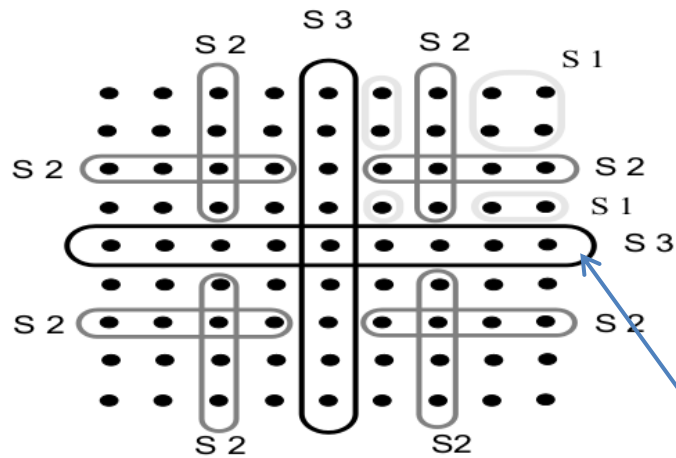
matrix A



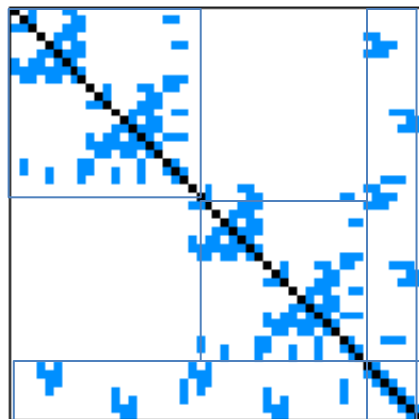
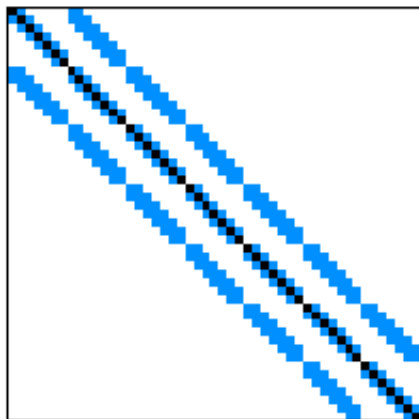
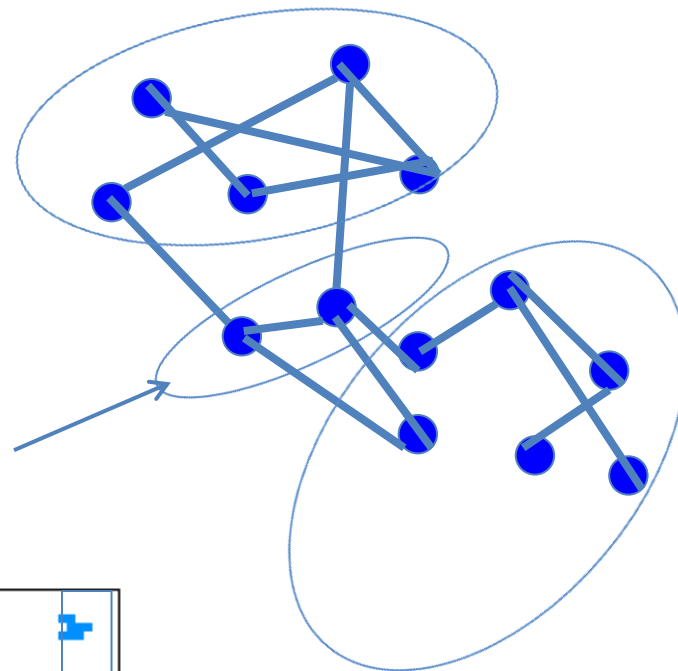
matrix L



Nested Dissection and Graph Partition

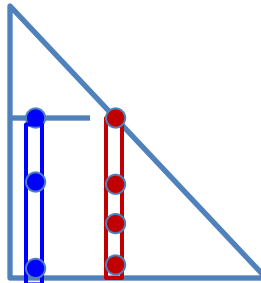


separator



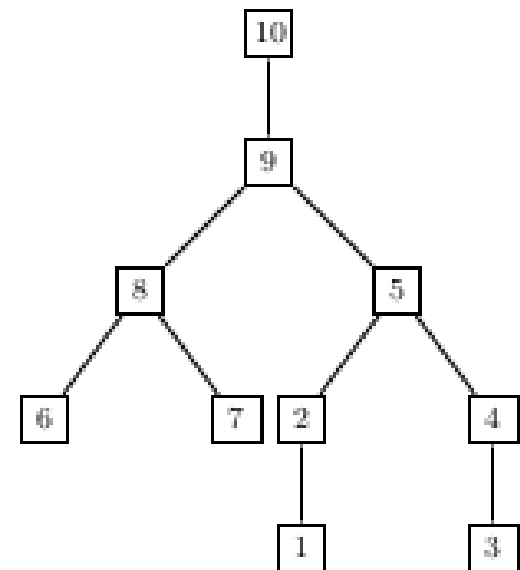
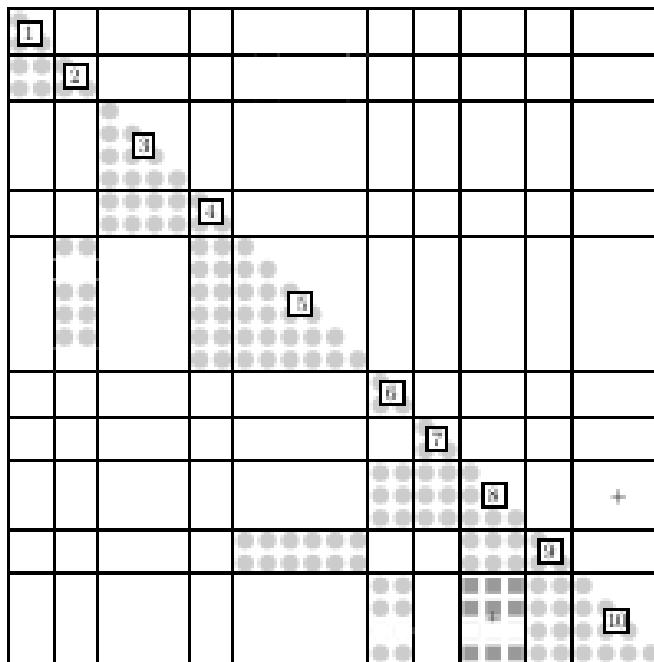
Indirect address mapping

- The challenge of a sparse factorization is in $\text{cmod}(j,k)$
 - Not all columns $k < j$ contribute to the update of column j
 - Columns k and j may have different sparsity structures
- Use symbolic factorization (to be discussed later) to construct, for each column j , a list of contributing columns ($k < j$).
 - Can be achieved by converting L from CSC to CSR format.
 - Can be done dynamically using an array of length n
- Use an index map to place nonzero update from column k in column j



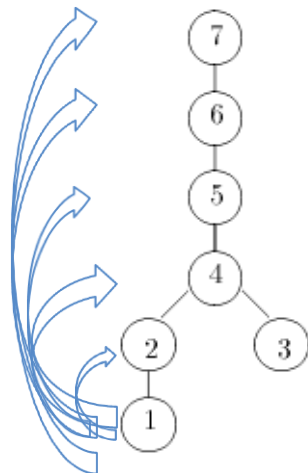
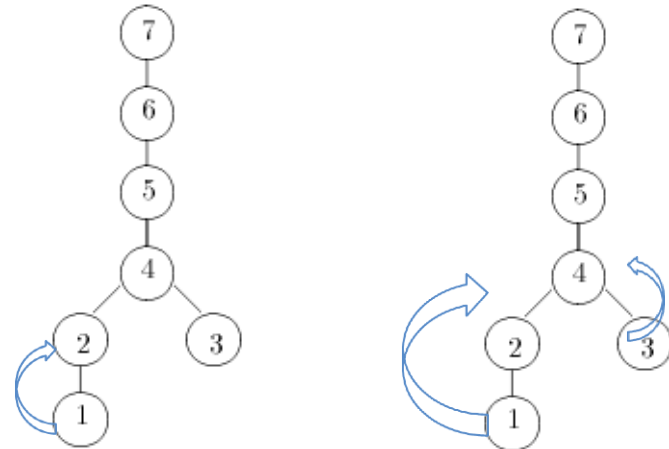
Supernodes

- A supernode is a set of adjacent columns that share identical nonzero structure below the diagonal block
- The nonzero structure of a supernode can be indexed by a single array (reduce the number of indirect addressing)
- A supernode contains dense blocks that can take advantage of BLAS3



Left-looking, right-looking, multi-frontal

- Left-looking: use the factored nodes below the node being factored in the elimination tree to update
- right-looking: the node being eliminated is used to update all of its ancestors



- Multifrontal: the node being eliminated passes its contribution to all ancestors as fronts to its parent

Parallel left-looking factorization for shared memory parallel machines

```

1. Copy the nonzero entries of  $A$  to  $L$ ;
2. do  $j = 1, 2, \dots, n$ 
  2.1 if ( $j > 1$ ) then
    2.1.1 foreach  $k$  such that  $L_{j,k} \neq 0$  do
      2.1.1.1  $cmod(j, k)$ ;
    2.1.2 endfor
  2.2 endif
  2.3. if ( $j < n$ ) then
    2.3.1  $cdiv(j)$ ;
  2.4. endif
3. enddo;

```



```

1. Copy the nonzero entries of  $A$  to  $L$ ;
2. Compute the row structure of  $L$ ;
3. do  $j = 1, 2, \dots, nsup$  in parallel
  3.1  $kusd = 0$ ;
  3.2 foreach ( $k < j$ ) such that  $L_{j,k} \neq 0$  do
    3.2.1 if  $iready(k) = 1$  then
      3.2.1.1  $nelm = nelm + 1$ ;
      3.2.1.2  $enode(nelm) = k$ ;
    3.2.2 else
      3.2.1.1 Save  $k$  in a linked list  $link$ ;
    3.2.3 endif
  3.3 endfor
  3.4 for  $p = 1, nelm$  do
    3.4.1  $k = enode(p)$ ;
    3.4.1  $cmod(j, k)$ ;
  3.5 endfor;
  3.6  $kusd = kusd + nelm$ ;
  3.7 if  $kusd < rnnz$  then
    3.7.1  $nelm = 0$ ;
    3.7.2 foreach  $k \in link$  do
      3.7.2.1 if  $iready(k) = 1$  then
        3.7.2.1.1 remove  $k$  from  $link$ ;
        3.7.2.1.2  $nelm = nelm + 1$ ;
        3.7.2.1.3  $enode(nelm) = k$ ;
      3.7.2.2 endif
    3.7.3 endfor
    3.7.4 go to 3.4;
  3.8 endif
  3.9.  $cdiv(j)$ ;
  3.10.  $iready(j) = 1$ ;
4. enddo;

```

Parallel right-looking factorization for distributed-memory machines

```

1. while some  $L_{IJ}$  with  $\text{map}[L_{IJ}] = \text{MyID}$  is not complete do
2.     receive some  $L_{IK}$ 
3.     if  $I = K$  /* diagonal block */
4.          $\text{Diag}_{K, \text{MyID}} := L_{KK}$ 
5.         foreach  $L_{JK} \in \text{Wait}_{K, \text{MyID}}$  do
6.              $L_{JK} := L_{JK} L_{KK}^{-1}$ 
7.             send  $L_{JK}$  to all P that could own blocks in
                row  $J$  or column  $J$ 
8.     else
9.          $\text{Rec}_{K, \text{MyID}} := \text{Rec}_{K, \text{MyID}} \cup \{L_{IK}\}$ 
10.        foreach  $L_{JK} \in \text{Rec}_{K, \text{MyID}}$  do
11.            if  $\text{map}[L_{IJ}] = \text{MyID}$  then
12.                Find  $L_{IJ}$ 
13.                 $L_{IJ} := L_{IJ} - L_{IK} L_{JK}^T$ 
14.                 $\text{nmod}[L_{IJ}] := \text{nmod}[L_{IJ}] - 1$ 
15.                if  $(\text{nmod}[L_{IJ}] = 0)$  then
16.                    if  $I = J$  then /* diagonal block */
17.                         $L_{JJ} := \text{Factor}(L_{JJ})$ 
18.                        send  $L_{JJ}$  to all P that could own blocks in
                                column  $J$ 
19.                    else if  $(\text{Diag}_{J, \text{MyID}} \neq \emptyset)$  then
20.                         $L_{IJ} := L_{IJ} L_{JJ}^{-1}$ 
21.                        send  $L_{IJ}$  to all P that could own blocks in
                                row  $I$  or column  $I$ 
22.                    else
23.                         $\text{Wait}_{J, \text{MyID}} := \text{Wait}_{J, \text{MyID}} \cup \{L_{IJ}\}$ 

```

Rothberg and Gupta, SISC
vol 15, pp 1413-1439, 1994

Sparse direct solver software

- MUMPS
- SuperLU, SuperLU_MT, SuperLU_DIST
- PARDISOL (Intel MKL)
- PSPASES
- CLIQUE (ELEMENTAL)

Iterative methods for solving linear systems

- Direct methods are good when the factorization does not produce too many nonzero fills
- Iterative method only require a procedure to multiply a matrix (and its transpose) with a vector
- The convergence of iterative method often depends on the condition number of the coefficient matrix A
- Preconditioner are often used to accelerate convergence
- Performance largely depends on how efficient matrix-vector multiplication can be performed
- Types of iterative methods:
 - Matrix splitting based methods (Jacobi, Gauss-Seid)
 - Krylov subspace based methods (GMRES, Conjugate Gradient)
 - Multigrid

Matrix splitting methods

- Matrix splitting: $A = M - R$
- Iterative method based on: $Mx_{k+1} = Rx_k + b$
- Error recurrence: $e_{k+1} = M^{-1}Re_k$, where
 $e_k = x_k - x$
- Convergence guaranteed if
$$|\lambda_{\max}(M^{-1}R)| < 1$$

Jacobi iteration

- Let $A = L + D + U$, L strictly lower triangular, U strictly upper triangular, D diagonal
- Recurrence defined by

$$Dx_{k+1} = (-L - U)x_k + b$$

or

$$x_{k+1} = x_k + D^{-1}(b - Ax_k)$$

Gauss-Seidel

- Let $A = L + D + U$
- Gauss-Seidel is defined by setting $M = L + D$, and $R = -U$
- Recurrence:
- $(L + D)x_{k+1} = -Ux_k + b$

Successive Overrelaxation

- Let $A = L + D + U$
- Choose $M = D + \omega L$, $R = (1 - \omega)D - \omega U$
- Recurrence:
$$x_{k+1} = x_k + \omega(D + \omega L)^{-1}(b - Ax_k)$$
- Choose ω to optimize the converge (i.e., the spectral radius of $M^{-1}R$)

Krylov subspace method

- Krylov subspace $\mathcal{K}(A, v_0) = \{v_0, Av_0, A^2v_0, \dots, A^{k-1}v_0\}$
- Approximate the solution to $Ax = b$ from the Krylov subspace $x \approx Vg$, where $\text{span}\{V\} = \mathcal{K}(A, v_0)$
- Prefer V to be an orthonormal basis. Can be generated by Gram-Schmidt: Arnoldi algorithm

$$AV_k = V_k H_k + f e_k^T, V_k^T V_k = I, V_k^T f = 0$$

where H_k is upper Hessenberg or tridiagonal when A is symmetric

- Convergence depends on the condition number of A and also the distribution of eigenvalues
- Precondition: solve $M^{-1}Ax = M^{-1}b$ or $L^{-1}AL^{-T}(L^T x) = L^{-1}b$

How to extract approximation from a Krylov subspace

- Recall $\hat{x} \approx Vg$, where V contains an orthonormal basis of a Krylov subspace
- We can choose g
 - to minimize $\|r\| = \|A\hat{x} - b\|_2$
 - to minimize $\|r\|_{A^{-1}}^2 = (A\hat{x} - b)A^{-1}(A\hat{x} - b) = \|\hat{x} - x\|_A^2$
 - to ensure $r = Ax - b$ is orthogonal to $\mathcal{K}(A, v_0)$ (Galerkin condition)

$$V^T(b - Ax) = 0$$

- To ensure $r = Ax - b$ is orthogonal to some other subspace (Petro-Galerkin condition)

GMRES (general minimum residual)


- $\hat{x} = \left(V_k, \frac{f}{\|f\|} \right) g$, choose g to minimize $\|r\| = \|A\hat{x} - b\|_2$
- Because $AV_k = V_k H_k + f e_k^T$, $V_k^T V_k = I$, $V_k^T f = 0$
- Equivalent to solving small least squares $\|\hat{H}_k g - \hat{b}\|_2$
 where $\hat{H} = \begin{pmatrix} H_k \\ \|f\| e_k^T \end{pmatrix}$, $\hat{b} = \begin{pmatrix} V_k^T \\ f^T / \|f\| \end{pmatrix} b$
- Incremental QR factorization of \hat{H}_k by Given's rotation
- Monitor the convergence, terminate when the estimated residual norm is small enough

The GMRES algorithm

Modified Gram-Schmidt 

QR factorization 

Monitor the residual 

Solve triangular system only when the residual is small enough 

```

 $\beta = \|r\|_2, v^1 = r/\beta; \hat{b} = \beta e^1$ 
 $e^1$  is the first unit vector (of length  $m + 1$ )
for  $i = 1, 2, \dots, m$ 
   $w = Av^i$ 
  for  $k = 1, \dots, i$ 
     $h_{k,i} = (v^k)^T w, w = w - h_{k,i}v^k$ 
   $h_{i+1,i} = \|w\|_2, v^{i+1} = w/h_{i+1,i}$ 
   $r_{1,i} = h_{1,i}$ 
  for  $k = 2, \dots, i$ 
     $\gamma = c_{k-1}r_{k-1,i} + s_{k-1}h_{k,i}$ 
     $r_{k,i} = -s_{k-1}r_{k-1,i} + c_{k-1}h_{k,i}$ 
     $r_{k-1,i} = \gamma$ 
   $\delta = \sqrt{r_{i,i}^2 + h_{i+1,i}^2}, c_i = r_{i,i}/\delta, s_i = h_{i+1,i}/\delta$ 
   $r_{i,i} = c_i r_{i,i} + s_i h_{i+1,i}$ 
   $\hat{b}_{i+1} = -s_i \hat{b}_i, \hat{b}_i = c_i \hat{b}_i$ 
   $\rho = |\hat{b}_{i+1}| (= \|b - Ax^{(j-1)m+i}\|_2)$ 
  if  $\rho$  is small enough then
    ( $n_r = i$ , goto SOL)
   $n_r = m, y_{n_r} = \hat{b}_{n_r}/r_{n_r,n_r}$ 
  for  $k = n_r - 1, \dots, 1$ 
     $y_k = (\hat{b}_k - \sum_{i=k+1}^{n_r} r_{k,i}y_i)/r_{k,k}$ 
   $x = \sum_{i=1}^{n_r} y_i v^i$ , if  $\rho$  small enough quit
   $r = b - Ax$ 

```

SOL:

MINRES

- When A is symmetric (but not necessarily positive definite), H_k is tridiagonal
- The orthonormal basis of $\mathcal{K}(A, v_0)$ can be generated via a 3-term recurrence
- The solution of the tridiagonal least squares problem $\min_g \|\hat{H}_k g - \beta e_1\|$ can be accumulated with a few vectors
- But the use of 3-term recurrence may quickly lead to loss of orthogonality among the columns of V_k (only three are kept at one time)
- As a result, round off error can propagate rapidly

MINRES algorithm

Compute $v_1 = b - Ax_0$ for some initial guess x_0

$$\beta_1 = \|v_1\|_2; \eta = \beta_1;$$

$$\gamma_1 = \gamma_0 = 1; \sigma_1 = \sigma_0 = 0;$$

$$v_0 = 0; w_0 = w_{-1} = 0;$$

for $i = 1, 2, \dots$

The Lanczos recurrence:

$$v_i = \frac{1}{\beta_i} v_i; \alpha_i = v_i^T A v_i;$$

$$v_{i+1} = A v_i - \alpha_i v_i - \beta_i v_{i-1}$$

$$\beta_{i+1} = \|v_{i+1}\|_2$$

QR part:

old Givens rotations on new column of T:

$$\delta = \gamma_i \alpha_i - \gamma_{i-1} \sigma_i \beta_i; \rho_1 = \sqrt{\delta^2 + \beta_{i+1}^2}$$

$$\rho_2 = \sigma_i \alpha_i + \gamma_{i-1} \gamma_i \beta_i; \rho_3 = \sigma_{i-1} \beta_i$$

New Givens rotation for subdiag element:

$$\gamma_{i+1} = \delta / \rho_1; \sigma_{i+1} = \beta_{i+1} / \rho_1$$

Update of solution (with $W_i = V_i R_{i,i}^{-1}$)

$$w_i = (v_i - \rho_3 w_{i-2} - \rho_2 w_{i-1}) / \rho_1$$

$$x_i = x_{i-1} + \gamma_{i+1} \eta w_i$$

$$\|r_i\|_2 = |\sigma_{i+1}| \|r_{i-1}\|_2$$

check convergence; continue if necessary

$$\eta = -\sigma_{i+1} \eta$$

end

Conjugate Gradient

- Let $\hat{x} = Vg$
- Choose g such that $V^T(b - A\hat{x}) = 0$ (Galerkin condition)
- Equivalent to solving

$$H_k g = V^T b$$

if $Ve_1 = b/\|b\|$, the right hand side becomes $\|b\|e_1$

- If A is nonsymmetric, this is the FOM
- If A is symmetric positive definite, this choice of g also minimizes $\|r\|_{A^{-1}} = \|e\|_A$, yields the **conjugate gradient** method:
 - $\hat{x} = VH_k^{-1}e_1\|b\| = VL^{-T}D^{-1}L^{-1}e_1\|b\| = Py$
 - $P = VL^{-T}$ satisfies $P^TAP = I$
 - Columns of P are successive A -conjugate search directions used to minimize $f(x) = \frac{1}{2}x^T Ax - x^T b$
 - 3-term recurrence follows from the fact that H_k is tridiagonal

The CG algorithm

Compute $r_0 = b - Ax_0$ for some initial guess x_0
for $i = 1, 2, \dots$
 Solve z_{i-1} from $Kz_{i-1} = r_{i-1}$
 $\rho_{i-1} = r_{i-1}^T z_{i-1}$
 if $i = 1$
 $p_1 = z_0$
 else
 $\beta_{i-1} = \frac{\rho_{i-1}}{\rho_{i-2}}$;
 $p_i = z_{i-1} + \beta_{i-1}p_{i-1}$
 endif
 $q_i = Ap_i$
 $\alpha_i = \frac{\rho_{i-1}}{p_i^T q_i}$
 $x_i = x_{i-1} + \alpha_i p_i$
 $r_i = r_{i-1} - \alpha_i q_i$
 check convergence; continue if necessary
end;

Chosen to enforce
A-conjugation



Line search to
minimize $f(x)$
along p_i



Precondition

- Suppose $M = LL^T$ is a good approximation to A
- We apply CG to $L^{-1}AL^{-T}(L^T x) = L^{-1}b$
- Gradient of the preconditioned problem
$$\tilde{r} = L^{-1}A\hat{x} - L^{-1}b$$
- $\rho = \tilde{r}^T \tilde{r} = r^T L^{-T} L^{-1} r = r^T M^{-1} r$

Convergence rate of CG

- Let $x^{(i)}$ be the approximation obtained at the i th CG iteration
- Error bound:

$$\|x - x^{(i)}\|_A \leq \left(\frac{\sqrt{\kappa(A) - 1}}{\kappa(A) + 1} \right)^i \|x - x^{(0)}\|_A$$

- The actual number of iterations required to reach convergence depends on the number of eigenvalue clusters and the right-hand side b

Bi-CG

- When A is nonsymmetric, the Arnoldi procedure does not lead to a short recurrence
- Try to construct a short (3-term) recurrence $AV_k = V_k H_k + f e_k^T$ by giving up the orthonormality constraint $V_k^T V_k = I$
- Require $W_k^T V_k = D$ for some basis W_k and diagonal matrix D
- Force $W_k^T AV_k = H_k$ to be tridiagonal
- Generate W_k from $A^T W_k = W_k H_k^T + h e_k^T$ (two-sided Lanczos)
- Make sure $W_k^T (b - A\hat{x}) = 0$ (Petro-Galerkin)
- Take $\hat{x} = V_k g$, use a LU factorization of $H_k = LU$ (both L and U are bidiagonal) to construct short recurrences
- Serious breakdown: $f^T h = 0$
- Stabilization yields the BiCGSTAB algorithm

Bi-CG algorithm

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$

Choose $\hat{r}^{(0)}$ (for instance, $\hat{r}^{(0)} = r^{(0)}$)

for $i = 1, 2, \dots$

Solve z^{i-1} from $Kz^{i-1} = r^{(i-1)}$

solve \hat{z}^{i-1} from $K^T \hat{z}^{i-1} = \hat{r}^{(i-1)}$

$$\rho_{i-1} = (\hat{r}^{(i-1)})^* z^{i-1}$$

if $i = 1$

$$p^1 = z^0$$

$$\hat{p}^1 = \hat{z}^0$$

else

$$\beta_{i-1} = \frac{\rho_{i-1}}{\rho_{i-2}};$$

$$p^i = z^{i-1} + \beta_{i-1} p^{i-1}$$

$$\hat{p}^i = \hat{z}^{i-1} + \beta_{i-1} \hat{p}^{i-1}$$

endif

$$q^i = Ap^i$$

$$\hat{q}^i = A^* \hat{p}^i$$

$$\alpha_i = \frac{\rho_{i-1}}{(\hat{p}^i)^* q^i}$$

$$x^{(i)} = x^{(i-1)} + \alpha_i p^i$$

$$r^{(i)} = r^{(i-1)} - \alpha_i q^i$$

$$\hat{r}^{(i)} = \hat{r}^{(i-1)} - \alpha_i \hat{q}^i$$

check convergence; continue if necessary

end;

QMR (Quasi-Minimal Residual)

- Try to minimize $\|r\| = \|b - A\hat{x}\|$, where $\hat{x} = V_k g$,
- Recall: V_k satisfies $AV_k = V_{k+1}\hat{H}_k$, where $\hat{H}_k = \begin{pmatrix} H_k \\ \|f\|e_k^T \end{pmatrix}$,
 $V_{k+1} = (V_k, f/\|f\|)$
- $\|r\| \neq \|W_{k+1}^T(b - A\hat{x})\| = \|\|b\|e_1 - \hat{H}_k g\|$ because
 $W_{k+1}^T W_{k+1} \neq I$
- Minimize $\|\tilde{r}\| = \|\|b\|e_1 - \hat{H}_k g\|$ anyway to yield quasi-minimal residual norm
- Can show $\|\tilde{r}\| \leq \sqrt{k}\|r\|$

Restarted GMRES

- Limit the size of the Krylov subspace
- Use the last residual vector to start a new GMRES to seek the correction to the previous approximation
 - $r = b - Ax_0$
 - While no convergence
 - $c = \text{GMRES}(A, r, k)$;
 - $x_0 = x_0 + c$
 - $r = b - Ax_0$