USN-HCMV PARIS 13

JOINT MASTER 2

High Performance Computing

Pr. Laurence Halpern and Juliette Ryan

with support of Drs. Ong Thanh Hai, and NguyenTanTrung classe 2008

Purpose : This is all about solving $Ax = b$, where A is a square matrix and b is a given righthand side, or a family of given righthand sides.

November 2016

Table des matières

Chapitre 1

Classical methods

Contents

1.1 Direct methods

1.1.1 Gauss method

Example

$$\underbrace{\begin{pmatrix} 1 & 3 & 1 \\ 1 & 1 & -1 \\ 3 & 11 & 6 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} 9 \\ 1 \\ 36 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 9 \\ 1 \\ 36 \end{pmatrix}}_{b}$$

Take the 3×4 matrix $\bar{A} = [A \,|\, b]$. Define

$$M_1 = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix}$$

and multiply on the left by M_1 to put zeros under the diagonal in the first column :

$$M_1[A \,|\, b] = \left(\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ 0 & -2 & -2 & -8 \\ 0 & 2 & 3 & 9 \end{array}\right).$$

Multiply now on the left by M_2 to put zeros under the diagonal in the second column :

$$M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

$$M_2\, M_1[A \,|\, b] = \left(\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ 0 & -2 & -2 & -8 \\ 0 & 0 & 1 & 1 \end{array}\right)$$

Define $M = M_2 M_1$. Then the column j of M is the column j of M_j :

$$M = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -3 & 1 & 1 \end{pmatrix}.$$

$$M[A \,|\, b] = [MA \,|\, Mb].$$

$$Ax = b \iff MAx = Mb : M \text{ is a } \textbf{preconditioner}.$$

The matrix $U = MA$ is upper triangular, and solving $Ux = Mb$ is simpler than solving $Ax = b$. Define $L = M^{-1}$. In the column j, the entries below the diagonal are those of M with a change of signe.

$$L := M^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & -1 & 1 \end{pmatrix}.$$

$$U = MA \iff A = LU, Ax = b \iff LUx = b \iff \begin{cases} Ly = b \\ Ux = y \end{cases}$$

Solving $Ax = b$ is then equivalent to performing the LU decomposition, and solving two triangular systems. Counting of operations :

1. LU decomposition $\mathcal{O}(\frac{2n^3}{3})$ elementary operations.
2. Solve $Ly = b$ $\quad\mathcal{O}(n^2)$ elementary operations.
3. Solve $Ux = y$ $\quad\mathcal{O}(n^2)$ elementary operations.

For P values of the righthand side, $N_{op} \sim \frac{2n^3}{3} + P \times 2n^2$.

1.1.2 Codes

```
function x=BackSubstitution(U,b)
% BACKSUBSTITUTION solves by backsubstitution a linear system
% x=BackSubstitution(U,b) solves Ux=b, U upper triangular by
% backsubstitution
n=length(b);
for k=n:-1:1
s=b(k);
for j=k+1:n
s=s-U(k,j)*x(j);
end
x(k)=s/U(k,k);
end
x=x(:);
```

```
function x=Elimination(A,b)
% ELIMINATION solves a linear system by Gaussian elimination
% x=Elimination(A,b) solves the linear system Ax=b using Gaussian
% Elimination with partial pivoting. Uses the function
% BackSubstitution
n=length(b);
norma=norm(A,1);
A=[A,b]; % augmented matrix
for i=1:n
[maximum,kmax]=max(abs(A(i:n,i))); % look for Pivot A(kmax,i)
kmax=kmax+i-1;
if maximum < 1e-14*norma; % only small pivots
error('matrix is singular')
end
if i ~= kmax % interchange rows
h=A(kmax,:); A(kmax,:)=A(i,:); A(i,:)=h;
end
A(i+1:n,i)=A(i+1:n,i)/A(i,i); % elimination step
A(i+1:n,i+1:n+1)=A(i+1:n,i+1:n+1)-A(i+1:n,i)*A(i,i+1:n+1);
end
x=BackSubstitution(A,A(:,n+1));
```

1.1.3 Theoretical results

Theorem 1.1 (Regular case) *Let A be an invertible matrix, with all principal minors $\neq 0$. Then there exists a unique matrix L lower triangular with $l_{ii} = 1$ for all i, and a unique matrix U upper triangular, such that $A = LU$. Furthermore $\det(A) = \prod_{i=1}^{n} u_{ii}$.*

Theorem 1.2 (Partial pivoting) *Let A be an invertible matrix. There exist a permutation matrix P, a matrix L lower triangular with $l_{ii} = 1$ for all i, and a matrix U upper triangular, such that*

$$PA = LU$$

1.1.4 Symmetric definite matrices : Cholewski decomposition

Theorem 1.3 *If A is symmetric definite positive, there exists a unique lower triangular matrix R with positive entries on the diagonal, such that $A = RR^T$.*

1.1.5 Elimination with Givens rotations

This is meant to avoid pivoting. It is used often in connection with the resolution of least-square problems. In the i step of the Gauss algorithm, we need to eliminate x_i in equations $i+1$ to n of the reduced system :

$$\begin{array}{rcccccc} (i): & a_{ii}x_i & +\cdots & + & a_{in}x_n & = & b_i \\ & \vdots & \vdots & & & & \\ (k): & a_{ki}x_i & +\cdots & + & a_{kn}x_n & = & b_k \\ & \vdots & \vdots & & & & \\ (i): & a_{ni}x_i & +\cdots & + & a_{nn}x_n & = & b_n \end{array}$$

If $a_{ki} = 0$, nothing needs to be done. If $a_{ki} \neq 0$, we multiply equation(i) with $\sin\alpha$ and equation (k) with $\cos\alpha$ and add. This leads to replacing equation (k) by the linear combination

$$(k)_{new} = -\sin\alpha \cdot (i) + \cos\alpha \cdot (k).$$

The idea is to choose α such that the first coefficient in the line vanishes, *i.e.*

$$-\sin\alpha \cdot a_{ii} + \cos\alpha \cdot a_{ki} = 0.$$

Since $a_{ki} \neq 0$, this defines $\text{cotg}\alpha_{ki}$, that is α_{ki} modulo π. For stability reasons, line (i) is also modified, end we end up with

$$\begin{array}{rcll} (i)_{new} & = & \cos\alpha \cdot (i) & +\sin\alpha \cdot (k) \\ (k)_{new} & = & -\sin\alpha \cdot (i) & +\cos\alpha \cdot (k) \end{array}$$

From which the sine and cosine of α_{ki} are obtained through well-known trigonometric formulas

$$\sin\alpha_{ki} = 1/\sqrt{1+\text{cotg}^2\alpha_{ki}}, \quad \cos\alpha_{ki} = \sin\alpha_{ki}\,\text{cotg}\alpha_{ki}.$$

$$\begin{array}{rcll} A_{ij\,new} & = & \cos\alpha_{ki} \cdot A_{ij} & +\sin\alpha_{ki} \cdot A_{kj} \\ A_{kj\,new} & = & -\sin\alpha_{ki} \cdot A_{ij} & +\cos\alpha_{ki} \cdot A_{kj} \end{array}$$

```
function x=BackSubstitutionSAXPY(U,b)
% BACKSUBSTITUTIONSAXPY solves linear system by backsubstitution
% x=BackSubstitutionSAXPY(U,b) solves Ux=b by backsubstitution by
% modifying the right hand side (SAXPY variant)n=length(b);
n=length(b);
for i=n:-1:1
x(i)=b(i)/U(i,i);
b(1:i-1)=b(1:i-1)-x(i)*U(1:i-1,i);
end
x=x(:);
```

```
function x=EliminationGivens(A,b);
% ELIMINATIONGIVENS solves a linear system using Givens-rotations
% x=EliminationGivens(A,b) solves Ax=b using Givens-rotations. Uses
% the function BackSubstitutionSAXPY.
n=length(A);
for i= 1:n
for k=i+1:n
if A(k,i)~=0
cot=A(i,i)/A(k,i); % rotation angle
si=1/sqrt(1+cot^2); co=si*cot;
A(i,i)=A(i,i)*co+A(k,i)*si; % rotate rows
h=A(i,i+1:n)*co+A(k,i+1:n)*si;
A(k,i+1:n)=-A(i,i+1:n)*si+A(k,i+1:n)*co;
A(i,i+1:n)=h;
h=b(i)*co+b(k)*si; % rotate right hand side
b(k)=-b(i)*si+b(k)*co; b(i)=h;
end
end;
if A(i,i)==0
error('Matrix is singular');
end;
end
x=BackSubstitutionSAXPY(A,b);
```

1.1.6 QR Decomposition

Note G^{ik} which differs from identity only on the rows i and k where

$$g_{ii} = g_{kk} = \cos\alpha, \quad g_{ik} = -g_{ki} = \sin\alpha$$

Example for $n = 5$,

$$G^{24} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Multipliying a vector b by G^{ik} changes only the components i and k,

$$G^{ik} \begin{pmatrix} \vdots \\ b_i \\ \vdots \\ b_k \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots & \\ \cos\alpha \cdot b_i & +\sin\alpha \cdot b_k \\ \vdots & \\ -\sin\alpha \cdot b_i & +\cos\alpha \cdot b_k \\ \vdots & \end{pmatrix}$$

$$G^{ik}\boldsymbol{e}_i = \cos\alpha\, \boldsymbol{e}_i - \sin\alpha\, \boldsymbol{e}_k, \quad G^{ik}\boldsymbol{e}_k = \sin\alpha\, \boldsymbol{e}_i + \cos\alpha\, \boldsymbol{e}_k.$$

G^{ik} represents the rotation of angle α in the plane generated by $\boldsymbol{e}_i$ and $\boldsymbol{e}_k$. $(G^{ik}(\alpha))^* = G^{ik}(-\alpha)$, $(G^{ik}(\alpha))^* G^{ik}(\alpha) = I$. Thus it is an orthogonal matrix. By applying successively $G_{21}, \ldots, G_{n1}$ whereever a_{k1} is not zero, we put zeros under the diagonal in the first column. We continue the process until the triangular matrix R is obtained. Then there are orthogonal matrices $G_1, \cdots, G_N$ such that Then

$$Q^* = G_N \ldots G_1, \quad QA = R.$$

Q is an orthogonal matrix,

$$Q^*Q = G_N \ldots G_1 G_1^* \ldots G_N^* = I.$$

then

$$A = QR,$$

we have reached the QR decomposition of the matrix A.

1.2 Sparse and banded matrices

The first encounter of this name seems to be due to Wilkinson in 69 : *any matrix with enough zeros that it pays to take advantage of them.*

Example : a banded matrix, with upper bandwidth $p = 3$ and lower bandwidth $q = 2$, in total $p + q + 1$ nonzero diagonals.

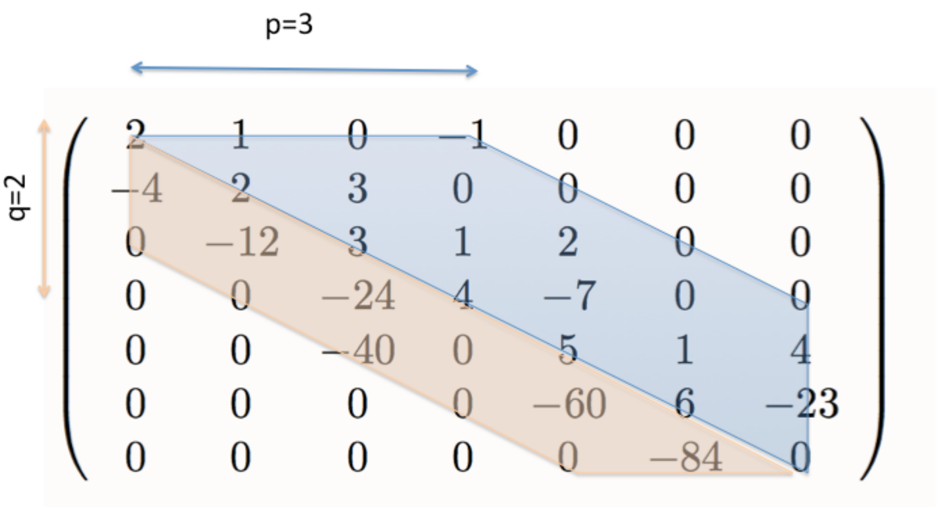


FIGURE 1.1 – A bandmatrix

Then L is lowerbanded with $q = 2$, and $U$is upperbanded with $p = 3$.

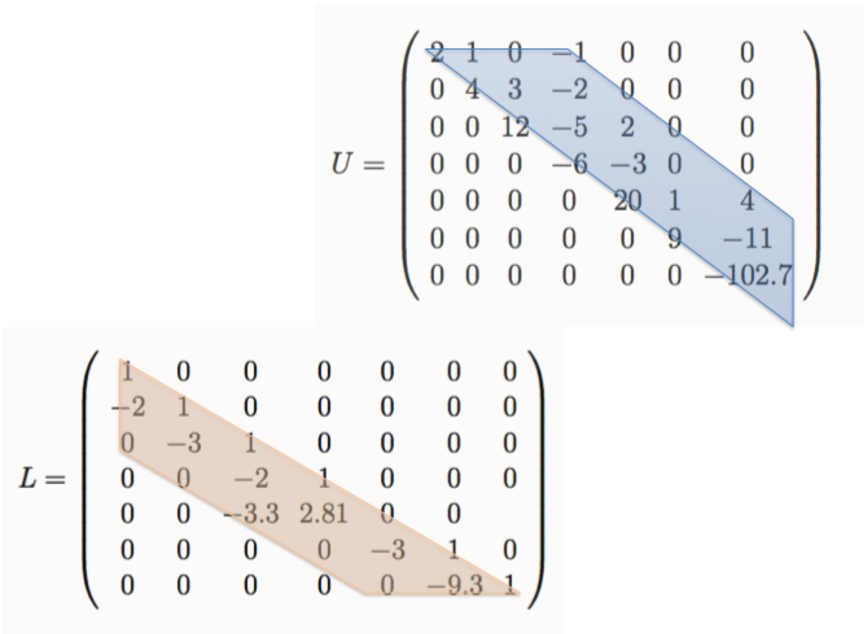


FIGURE 1.2 – LU decomposition

It is not the case anymore, when pivoting is used :

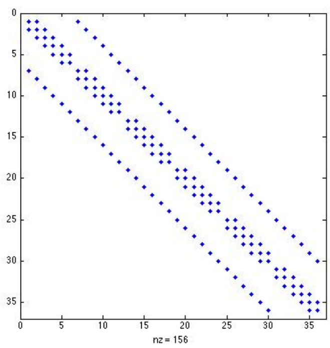
$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.6 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -0.5 & -0.17 & -0.05 & -0.21 & 0.025 & 0.0027 & 1 \end{pmatrix}$$

$$U = \begin{pmatrix} -4 & 2 & 3 & 0 & 0 & 0 & 0 \\ 0 & -12 & 3 & 1 & 2 & 0 & 0 \\ 0 & 0 & -40 & 0 & 5 & 1 & 4 \\ 0 & 0 & 0 & 4 & -10 & -0.6 & -2.4 \\ 0 & 0 & 0 & 0 & -60 & 6 & -23 \\ 0 & 0 & 0 & 0 & 0 & -84 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.275 \end{pmatrix}$$

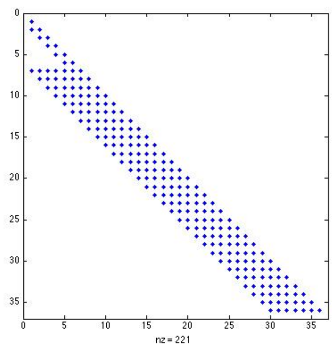
Here the permutation matrix is

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

In the Cholewsky decomposition, there is no need of permutation, unless some parameters are very small. Then if A is banded, R is banded with the same lower bandwidth, but it may be less sparse, in the sense that it can have more zeros. Consider as an example the 36×36 sparse matrix of $2-D$ finite differences in a square. With the command `spy` de matlab, the nonzero terms appear in blue :



A bandmatrix sparse matrix



Corresponding Cholewski

Even though R has the same bandwidth as A, nonzero diagonals appear.

EXERCISE Write the Gauss and Givens algorithms for a tridiagonal matrix $A = diag(c, -1) + diag(d, 0) + diag(e, 1)$.

LU factorization : verify that

$$c_k = l_k\, u_k, \;\; d_{k+1} = l_k\, f_k + u_{k+1}, \;\; e_k = f_k.$$

then it is not necessary to compute f_k, and only recursively

$$c_k = l_k\, u_k, \quad u_{k+1} = d_{k+1} - l_k\, e_k.$$

```
n=length(d);
for k=1:n-1 % LU-decomposition with no pivoting
        c(k)=c(k)/d(k);
        d(k+1)=d(k+1)-c(k)*e(k);
end
for k=2:n % forward substitution
        b(k)=b(k)-c(k-1)*b(k-1);
end
b(n)=b(n)/d(n); % backward substitution
```

```
for k=n-1:-1:1
        b(k)=(b(k)-e(k)*b(k+1))/d(k);
end
```

Givens : verify that the process inserts an extra updiagonal.

```
n=length(d);
e(n)=0;
for i=1: n-1 % elimination
        if c(i)~=0
                t=d(i)/c(i); si=1/sqrt(1+t*t); co=t*si;
                d(i)=d(i)*co+c(i)*si; h=e(i);
                e(i)=h*co+d(i+1)*si; d(i+1)=-h*si+d(i+1)*co;
                c(i)=e(i+1)*si; e(i+1)=e(i+1)*co;
                h=b(i); b(i)=h*co+b(i+1)*si;
                b(i+1)=-h*si+b(i+1)*co;
        end;
end;
b(n)=b(n)/d(n); % backsubstitution
b(n-1)=(b(n-1)-e(n-1)*b(n))/d(n-1);
for i=n-2:-1:1,
        b(i)=(b(i)-e(i)*b(i+1)-c(i)*b(i+2))/d(i);
end;
```

Creation and manipulation of sparse matrices in matlab

```
>>S=sparse([2 3 1 2],[1 1 2 3],[2 4 1 3])

 S =

   (2,1)        2
   (3,1)        4
   (1,2)        1
   (2,3)        3

>>S=speye(2,3)

S =

   (1,1)        1
   (2,2)        1

>>n=4;
>>e=ones(n,1)
e =

     1
     1
```

```
     1
     1

>>A=spdiags([e -2*e e],-1:1,n,n)
A =

   (1,1)       -2
   (2,1)        1
   (1,2)        1
   (2,2)       -2
   (3,2)        1
   (2,3)        1
   (3,3)       -2
   (4,3)        1
   (3,4)        1
   (4,4)       -2

>>full(A)
ans =

    -2     1     0     0
     1    -2     1     0
     0     1    -2     1
     0     0     1    -2


>>S=sparse([2 3 1 2],[1 1 2 3],[2 4 1 3])

 S =

   (2,1)        2
   (3,1)        4
   (1,2)        1
   (2,3)        3

>>S=speye(2,3)

S =

   (1,1)        1
   (2,2)        1

>>n=4;
>>e=ones(n,1)
e =

     1
```

```
     1
     1
     1

>>A=spdiags([e -2*e e],-1:1,n,n)
A =

   (1,1)       -2
   (2,1)        1
   (1,2)        1
   (2,2)       -2
   (3,2)        1
   (2,3)        1
   (3,3)       -2
   (4,3)        1
   (3,4)        1
   (4,4)       -2

>>full(A)
ans =

    -2     1     0     0
     1    -2     1     0
     0     1    -2     1
     0     0     1    -2
```

The direct methods first transform the original system into a triangular matrix, and then solve the simpler triangular system. Therefore a direct method leads, modulo truncation errors, to the exact solution, after a number of operations which is a function of the size of the matrix. Thereby, when the matrix is sparse, the machine performs a large number of redundant operations due to the large number of multiplication by zero it performs.

The iterative methods rely on a product matrix vector, therefore are easier to perform in a *sparse* way. They have gain a lot of popularity for sparse matrix, in conjunction with preconditioning and and domain decomposition. However their success relies on the convergence speed of the algorithm.

1.3 Stationary iterative methods

$$
\begin{aligned}
&\text{For any splitting } A = M - N, \quad \text{write} \quad && Mx = Nx + b,\\
&\text{Define the sequence} && Mx^{m+1} = Nx^m + b.
\end{aligned}
$$

$$\begin{aligned} Mx^{m+1} = Nx^m + b &\iff Mx^{m+1} = (M-A)x^m + b \\ &\iff x^{m+1} = (I - M^{-1}A)x^m + M^{-1}b \\ &\iff x^{m+1} = x^m - M^{-1}Ax^m + M^{-1}b \\ &\iff \text{fixed point algorithm to solve } x - M^{-1}Ax + M^{-1}b = x \\ &\iff \text{fixed point algorithm to solve } M^{-1}Ax = M^{-1}b. \end{aligned}$$

Again, M is a preconditioner.

Definition 1.1

- $e^m := x - x^m$ *is the error at step* m.
- $r^m := b - Ax^m = Ae^m$ *is the residual at step* m.
- $R = M^{-1}N = I - M^{-1}A$ *is the iteration matrix.*

Then the sequence of the errors satisfies

$$Me^{m+1} = Ne^m, \quad e^{m+1} = M^{-1}Ne^m$$

Stopping criterion Usually, one stops if $\frac{\|r^m\|}{\|b\|} < \varepsilon$.

1.3.1 Classical methods

Use $A = D - E - F$.

Jacobi	$M = D$	$R := J = I - D^{-1}A$
Relaxed Jacobi	$M = \frac{1}{\omega}D$	$R = I - \omega D^{-1}A$
Gauss-Seidel	$M = D - E$	$R := \mathcal{L}_1 = I - D^{-1}A$
SOR	$M = \frac{1}{\omega}D - E,$	$R := \mathcal{L}_\omega = (D - \omega E)^{-1}((1-\omega)D + \omega F)$
Richardson	$M = \frac{1}{\rho}I$	$R = I - \rho A$

The relaxed methods are obtained as follows : define $\hat{x}^m$ as an application of Jacobi or Gauss-Seidel, then take the centroid of $\hat{x}^m$ and x^m as $x^{m+1} = \omega\hat{x}^m + (1-\omega)x^m$.
For symmetric positive definite matrices A, RIchardson is a gradient method with fixed parameter. There is an optimal value for this parameter, given by $\rho_{opt} = \frac{2}{\lambda_1 + \lambda_n}$ where the λ_j are the eigenvaues of A.

1.3.2 Fundamentals tools

Define the sequence

$$e^{m+1} = Re^m, \ R = M^{-1}N.$$

Then $e^m = R^m e_0$, and for any norm

$$\|e^{m+1}\| \leq \|R\|\|e^m\|, \quad \|e^m\| \leq \|R^m\|\|e^0\|.$$

Definition 1.2

- $\rho(R) = \max\{|\lambda|, \lambda$ *eigenvalue of* $R\}$ *is the* *spectral radius* *of* R.
- $\rho_m(R) = \|R^m\|^{1/m}$ *is the* *mean convergence factor* *of* R.
- $\rho_\infty(R) = \lim_{m\to\infty} \|R^m\|^{1/m}$ *is the* *asymptotic convergence factor* *of* R.

Theorem 1.4

- *For any matrix* R*, for any norm, for any* m*,* $\rho_m(R) \geq \rho(R)$*. The sequence* $\rho_m(R)$ *has a limit, called the* *asymptotic convergence factor* *of* R *and denoted by* $\rho_\infty(R)$*.*
- *The sequence* x^m *is convergent* *for any* x^0 *if and only if* $\rho(R) < 1$*.*

To reduce the initial error by a factor ε, we need m iterations, defined by

$$\frac{\|e^m\|}{\|e^0\|} \leq (\rho_m(R))^m \sim \varepsilon.$$

So $m \sim \dfrac{\log \varepsilon}{\log \rho_m(R)}$. It is easier to use the asymptotic value relation, $m \sim \dfrac{\log \varepsilon}{\log \rho_\infty(R)}$. Then to obtain another decimal digit in the solution, one needs to choose $\varepsilon = 10^{-1}$, then $\bar{m} \sim -\dfrac{\ln(10)}{\ln(\rho(R))}$.

Definition 1.3 *The asymptotic convergence rate is* $F = -\ln(\rho(R))$*.*

Diagonally dominant matrices

Theorem 1.5

- *If* A *is a matrix, either strictly diagonally dominant, or irreducible and strongly diagonally dominant, then the Jacobi algorithm converges.*
- *If* A *is a matrix, either strictly diagonally dominant, or irreducible and strongly diagonally dominant, then for* $0 < \omega \leq 1$*, the SOR algorithm converges.*

M- matrices

Definition 1.4 $A \in \mathbb{R}^{n\times n}$ *is a* M*-matrix if*

1. $a_{ii} > 0$ *for* $i = 1, \dots, n$*,*
2. $a_{ij} \leq 0$ *for* $i \neq j$*,* $i, j = 1, \dots, n$*,*
3. A *is invertible,*
4. $A^{-1} \geq 0$*.*

Theorem 1.6 *If* A *is a M-matrix and* $A = M - N$ *is a regular splitting (*M *is invertible and both* M^{-1} *and* N *are nonnegative), then the stationary method converges.*

Symmetric positive definite matrices

Theorem 1.7 (Householder-John) *Suppose A is positive. If $M+M^T-A$ is positive definite, then $\rho(R)<1$.*

Corollary 1.1
1. *If $D+E+F$ is positive definite, then Jacobi converges.*
2. *If $\omega \in (0,2)$, then SOR converges.*

Tridiagonale matrices

Theorem 1.8
1. *$\rho(\mathcal{L}_1) = (\rho(J))^2$: Jacobi Gauss-Seidel converge or diverge simultaneously. If convergent, Gauss-Seidel is twice as fast.*
2. *Suppose the eigenvalues of J are real. Then Jacobi and SOR converge or diverge simultaneously for $\omega \in]0,2[$.*
3. *Same assumptions, SOR has an optimal parameter $\omega^* = \dfrac{2}{1+\sqrt{1-(\rho(J))^2}}$, and then $\rho(\mathcal{L}_{\omega^*}) = \omega^* - 1$.*

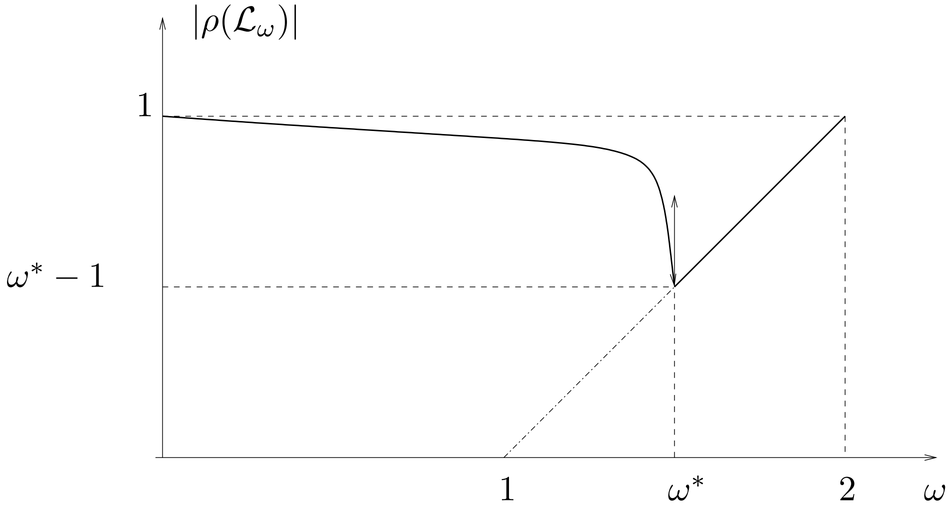


FIGURE 1.3 – Variations of $\rho(\mathcal{L}_\omega)$ as a fonction of ω

1.4 Non-Stationary iterative methods. Symmetric definite positive matrices

Descent methods

1.4.1 Definition of the iterative methods

Suppose the descent directions p_m are given beforehand. Define

$$x^{m+1} = x^m + \alpha_m p^m, \quad e^{m+1} = e^m - \alpha_m p^m, \quad r^{m+1} = r^m - \alpha_m A p^m.$$

Define the A norm : $\|y\|_A^2 = (Ay, y).$

Theorem 1.9 *x is the solution of $Ax = b \iff$ it minimizes over $\mathbb{R}^N$ the functional $J(y) = \frac{1}{2}(Ay, y) - (b, y)$.*

This is equivalent to minimizing $G(y) = \frac{1}{2}(A(y-x), y-x) = \frac{1}{2}\|y-x\|_A^2$. At step m, α_m is defined such as to minimize J in the direction of p_m. Define the quadratic function of α

$$\varphi_m(\alpha) = J(x^m + \alpha p^m) = J(x^m) - \alpha(r^m, p^m) + \frac{1}{2}\alpha^2 (Ap^m, p^m).$$

Minimizing φ_m leads to

$$\alpha_m = \frac{(p^m, r^m)}{(Ap^m, p^m)}, \quad (p^m, r^{m+1}) = 0$$

$$G(x^{m+1}) = G(x^m)(1 - \mu_m), \quad \mu_m = \frac{(r^m, p^m)^2}{(Ap^m, p^m)(A^{-1}r^m, r^m)}$$

- **Steepest descent (gradient à pas optimal)** $p^m = r^m$.

$$x^{m+1} = x^m + \alpha_m r^m, \quad e^{m+1} = e^m - \alpha_m r^m, \quad r^{m+1} = (I - \alpha_m A)p^m.$$

$$\alpha_m = \frac{\|r^m\|^2}{(Ar^m, r^m)}, \quad (r^m, r^{m+1}) = 0$$

$$G(x^{m+1}) = G(x^m)\left(1 - \frac{\|r^m\|^4}{(Ar^m, r^m)(A^{-1}r^m, r^m)}\right) \leq \left(\frac{\kappa(A) - 1}{\kappa(A) + 1}\right)^2 G(x^m)$$

- **Conjugate gradient**

$$x^{m+1} = x^m + \alpha_m p^m, \quad \alpha_m = \frac{(p^m, r^m)}{(Ap^m, p^m)}, \quad (r^m, p^{m-1}) = 0.$$

Search p^m as $p^m = r^m + \beta_m p^{m-1}$

$$G(x^{m+1}) = G(x^m)(1 - \mu_m)$$

$$\mu_m = \frac{(r^m, p^m)^2}{(Ap^m, p^m)(A^{-1}r^m, r^m)} = \frac{\|r^m\|^4}{(Ap^m, p^m)(A^{-1}r^m, r^m)}$$

Maximize μ_m, or minimize

$$(Ap^m, p^m) = \beta_m^2 (Ap^{m-1}, p^{m-1}) + 2\beta_m (Ap^{m-1}, r^m) + (Ar^m, r^m)$$

$$\beta_m = -\frac{(Ap^{m-1}, r^m)}{(Ap^{m-1}, p^{m-1})} \quad \Rightarrow (Ap^{m-1}, p^m) = 0$$

$$(r^m, r^{m+1}) = 0, \quad \beta_m = \frac{\|r^m\|^2}{\|r^{m-1}\|^2}.$$

Properties of the conjugate gradient Choose $p^0 = r^0$. Then $\forall m \geq 1$, if $r^i \neq 0$ for $i < m$.

1. $(r^m, p^i) = 0$ for $i \leq m-1$.
2. $\mathrm{vec}(r^0, \ldots, r^m) = \mathrm{vec}(r^0, Ar^0, \ldots, A^m r^0)$.
3. $\mathrm{vec}(p^0, \ldots, p^m) = \mathrm{vec}(r^0, Ar^0, \ldots, A^m r^0)$.
4. $(p^m, Ap^i) = 0$ for $i \leq m-1$.
5. $(r^m, r^i) = 0$ for $i \leq m-1$.

Definition 1.5 *Krylov space* $\mathcal{K}_m = \mathit{vec}(r^0, Ar^0, \ldots, A^{m-1} r^0)$.

Theorem 1.10 (optimality of CG) *A symétrique définie positive,*

$$\|x^m - x\|_A = \inf_{y \in x^0 + \mathcal{K}_m} \|y - x\|_A, \quad \|x\|_A = \sqrt{x^T A x}.$$

Theorem 1.11 *Convergence in at most N steps (size of the matrix). Furthermore*

$$G(x^m) \leq 4 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^2 G(x^{m-1})$$

The conjugate gradient algorithm

$$x^0 \text{chosen}, \quad p^0 = r^0 = b - Ax^0.$$

while $m < Niter$ or $\|r^m\| \geq tol$, do

$$\begin{aligned}
\alpha_m &= \frac{\|r^m\|^2}{(Ap^m, p^m)}, \\
x^{m+1} &= x^m + \alpha_m p^m, \\
r^{m+1} &= r^m - \alpha_m A p^m, \\
\beta_{m+1} &= \frac{\|r^{m+1}\|^2}{\|r^m\|^2}, \\
p^{m+1} &= r^{m+1} - \beta_{m+1} p^m.
\end{aligned}$$

end.

1.4.2 Comparison of the iterative methods

Basic example :. 1-D Poisson equation $-u'' = f$ on $(0,1)$, with Dirichlet boundary conditions $u(0) = g_g,\ u(1) = g_d$. Introduce the second order finite difference stencil.

$$(0,1) = \cup(x_j, x_{j+1}), \quad x_{j+1} - x_j = h = \frac{1}{n+1}, \quad j = 0, \ldots, n.$$

$$-\frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{h^2} \sim f(x_i), \quad i = 1, \ldots n$$

$$u_0 = g_g, \quad u_{n+1} = g_d.$$

$$|u_i - u(x_i)| \le h^2 \frac{\sup_{x\in[a,b]} |u^{(4)}(x)|}{12}$$

.

The vector of discrete unknowns is $u =^t (u_1, \ldots, u_n)$.

$$A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & 0 \\ & \ddots & \ddots & \ddots & \\ 0 & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \qquad b = \begin{pmatrix} f_1 - \frac{g_g}{h^2} \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n - \frac{g_d}{h^2} \end{pmatrix}$$

The matrix A is symmetric definite positive.

The discrete problem to be solved is

$$Au = b$$

1.4.3 Condition number and error

$$Ax = b, \quad A\hat{x} = \hat{b}$$

Define $\kappa(A) = \|A\|_2\|A^{-1}\|_2$. If A is symmetric > 0, $\kappa(A) = \frac{\max \lambda_i}{\min \lambda_i}$.

Theorem 1.12

$$\frac{\|\hat{x} - x\|_2}{\|x\|_2} \le \kappa(A)\frac{\|\hat{b} - b\|_2}{\|b\|_2}$$

and there is a b *such that it is equal.*

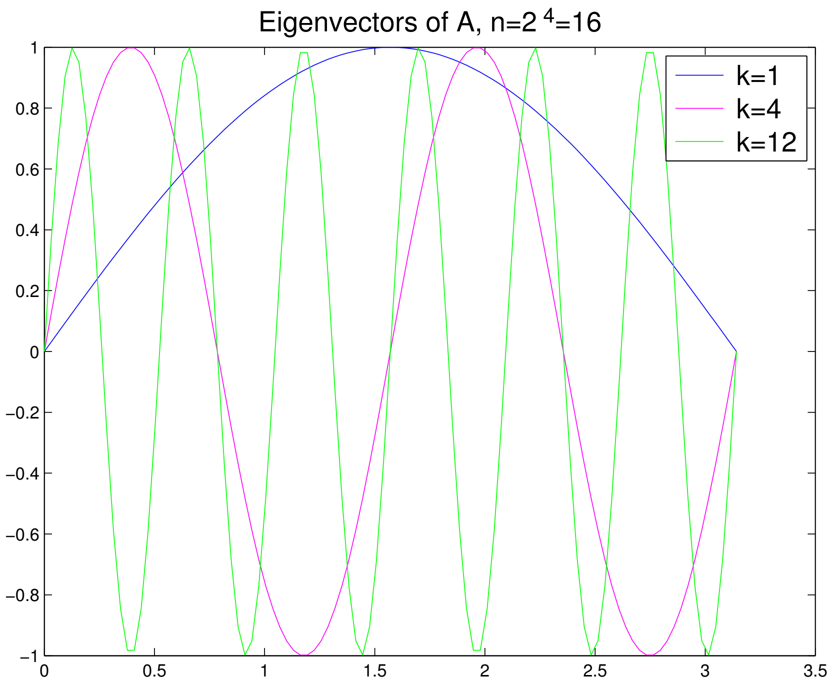


FIGURE 1.4 – Eigenvectors of A

Eigenvalues and eigenvectors of A $(h \times (n+1) = 1)$.

$$\mu_k = \frac{4}{h^2} \sin^2 \frac{k\pi h}{2}, \quad \Phi^{(k)} = \left(\sin \frac{jk\pi}{n+1} \right)_{1 \le j \le n},$$

$$\kappa(A) = \frac{\sin^2 \frac{n\pi h}{2}}{\sin^2 \frac{\pi h}{2}} = \frac{\cos^2 \frac{\pi h}{2}}{\sin^2 \frac{\pi h}{2}} \sim \frac{4}{\pi^2 h^2}$$

For any iterative method, the eigenfunctions of the iteration matrix are equal to those of A.

Algorithm	**Eigenvalues of the iteration matrix R**
Jacobi	$\lambda_k(J) = 1 - \frac{h^2}{2}\mu_k = cos(k\pi h)$
Gauss-Seidel	$\lambda_k(\mathcal{L}_1) = (\lambda_k(J))^2 = cos^2(k\pi h)$
SOR	$\eta = \lambda_k(\mathcal{L}_\omega)$ solution of $(\eta + \omega - 1)^2 = \eta\omega(\lambda_k(J))^2$.

TABLE 1.1 – Eigenvalues of the iteration matrix

Algorithm	**Convergence factor**	$n=5$	$n=30$	$n=60$
Jacobi	$\cos \pi h$	0.81	0.99	0.9987
Gauss-Seidel	$\cos^2 \pi h$	0.65	0.981	0.9973
SOR	$\frac{1-\sin \pi h}{1+\sin \pi h}$	0.26	0.74	0.9021
steepest descent	$\frac{K(A)-1}{K(A)+1} = \cos \pi h$	0.81	0.99	0.9987
conjugate gradient	$\frac{\sqrt{K(A)}-1}{\sqrt{K(A)}+1} = \frac{\cos \pi h - \sin \pi h}{\cos \pi h + \sin \pi h}$	0.51	0.86	0.9020

TABLE 1.2 – Convergence factor

Algorithm	convergence factor ρ_∞	convergence rate F
Jacobi	$1 - \frac{\varepsilon^2}{2}$	$\frac{\varepsilon^2}{2}$
Gauss-Seidel	$1 - \varepsilon^2$	ε^2
SOR	$1 - 2\varepsilon$	2ε
Steepest descent	$1 - \varepsilon^2$	$1\varepsilon^2$
conjugate gradient	$1 - 2\varepsilon$	2ε

TABLE 1.3 – Asymptotic behavior in function of $\varepsilon = \pi h$

n	Jacobi and steepest descent	Gauss-Seidel	SOR	conjugate gradient
10	56	28	4	4
100	4760	2380	38	37

TABLE 1.4 – Reduction factor for one digit $M \sim -\frac{\ln(10)}{F}$

Gauss elimination	n^2
optimal overrelaxation	$n^{3/2}$
FFT	$n \ln_2(n)$
conjugate gradient	$n^{5/4}$
multigrid	n

TABLE 1.5 – Asymptotic order of the number of elementary operations needed to solve the $1-D$ problem as a function of the number of grid points

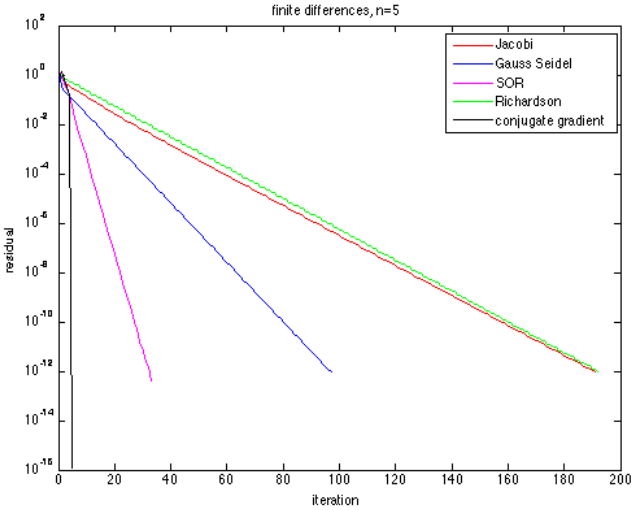


FIGURE 1.5 – Convergence history, $n = 5$

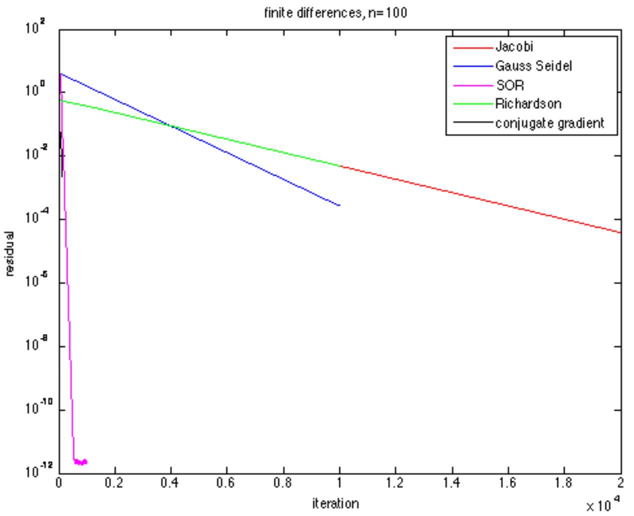


FIGURE 1.6 – Convergence history, $n = 100$

Not only the conjugate gradient is better, but the convergence rate being $\mathcal{O}(h^{1/2})$, the number of iterations necessary to increases the precision of one digit is multiplied by $\sqrt{10}$ when the mesh size is divided by 10, whereas for the Jacobi or Gauss-Seidel it is divided by 100. The miracle of multigrids, is that the convergence rate is independent of the mesh size.

1.5 Preconditioning

Preconditioning : purpose

Take the system $AX = b$, with A symmetric definite positive, and the conjugate gradient algorithm. The speed of convergence of the algorithm deteriorates when $\kappa(A)$ increases. The purpose is to replace the problem by another system, better conditioned. Let M be a symmetric regular matrix. Multiply the system on the left by M^{-1}.

$$AX = b \iff M^{-1}AX = M^{-1}b \iff (M^{-1}AM^{-1})MX = M^{-1}b$$

Define

$$\tilde{A} = M^{-1}AM^{-1}, \quad \tilde{X} = MX, \quad \tilde{b} = M^{-1}b,$$

and the new problem to solve $\tilde{A}\tilde{X} = \tilde{b}$. Since M is symmetric, $\tilde{A}$ is symmetric definite positive. Write the conjugate gradient algorithm for this "tilde" problem.

The algorithm for $\tilde{A}$

$$\tilde{X}^0 \text{ given}, \quad \tilde{p}^0 = \tilde{r}^0 = \tilde{b} - \tilde{A}\tilde{X}^0.$$

While $m < Niter$ or $\|\tilde{r}^m\| \geq tol$, do

$$\begin{aligned}
\alpha_m &= \frac{\|\tilde{r}^m\|^2}{(\tilde{A}\tilde{p}^m, \tilde{p}^m)}, \\
\tilde{X}^{m+1} &= \tilde{X}^m + \alpha_m \tilde{p}^m, \\
\tilde{r}^{m+1} &= \tilde{r}^m - \alpha_m \tilde{A}\tilde{p}^m, \\
\beta_{m+1} &= \frac{\|\tilde{r}^{m+1}\|^2}{\|\tilde{r}^m\|^2}, \\
\tilde{p}^{m+1} &= \tilde{r}^{m+1} - \beta_{m+1}\tilde{p}^m.
\end{aligned}$$

Now define

$$p^m = M^{-1}\tilde{p}^m, \quad X^m = M^{-1}\tilde{X}^m, \quad r^m = M\tilde{r}^m,$$

and replace in the algorithme above.

The algorithm for A

$$Mp^0 = M^{-1}r^0 = M^{-1}b - M^{-1}AM^{-1}MX^0 \iff \begin{cases} p^0 = M^{-2}r^0, \\ r^0 = b - AX^0. \end{cases}$$

$$\|\tilde{r}^m\|^2 = (M^{-1}r^m, M^{-1}r^m) = (M^{-2}r^m, r^m)$$

Define $\boxed{z^m = M^{-2}r^m}$. Then $\boxed{\beta_{m+1} = \dfrac{(z^{m+1}, r^{m+1})}{(z^m, r^m)}}$.

$$(\tilde{A}\tilde{p}^m, \tilde{p}^m) = (M^{-1}AM^{-1}Mp^m, Mp^m) = (Ap^m, p^m)$$

$$\Rightarrow \boxed{\alpha_m = \frac{(z^m, r^m)}{(Ap^m, p^m)}}.$$

$$MX^{m+1} = MX^m + \alpha_m Mp^m \iff \boxed{X^{m+1} = X^m + \alpha_m p^m}.$$

$$M^{-1}r^{m+1} = M^{-1}r^m - \alpha_m M^{-1}AM^{-1}Mp^m \iff \boxed{r^{m+1} = r^m - \alpha_m Ap^m}.$$

$$Mp^{m+1} = M^{-1}r^{m+1} - \beta_{m+1}Mp^m \iff \boxed{p^{m+1} = z^{m+1} - \beta_{m+1}p^m}.$$

The algorithm for A

Define $C = M^2$.

$$X^0 \text{ given}, \quad r^0 = b - AX^0, \quad \text{solve } Cz^0 = r^0, \quad p^0 = z^0.$$

While $m < Niter$ or $\|r^m\| \geq tol$, do

$$\begin{aligned} \alpha_m &= \frac{(z^m, r^m)}{(Ap^m, p^m)}, \\ X^{m+1} &= X^m + \alpha_m p^m, \\ r^{m+1} &= r^m - \alpha_m A p^m, \\ \text{solve } Cz^{m+1} &= r^{m+1}, \\ \beta_{m+1} &= \frac{(z^{m+1}, r^{m+1})}{(z^m, r^m)}, \\ p^{m+1} &= z^{m+1} - \beta_{m+1} p^m. \end{aligned}$$

How to choose C

C must be chosen such that

1. $\tilde{A}$ is better conditioned than A,
2. C is easy to invert.

Use an iterative method such that $A = C - N$ with symmetric C. For instance it can be a symmetrized version of SOR, named SSOR, defined for $\omega \in (0, 2)$ by

$$C = \frac{1}{\omega(2-\omega)}(D - \omega E)D^{-1}(D - \omega F).$$

Notice that if A is symmetric definite positive, so is D and its coefficients are positive, then its square root $\sqrt{D}$ is defined naturally as the diagonal matrix of the square roots of the coefficients. Then C can be rewritten as

$$C = SS^T, \quad \text{with } S = \frac{1}{\sqrt{\omega(2-\omega)}}(D - \omega E)D^{-1/2},$$

yielding a natural Cholewski decomposition of C.

Another possibility is to use an incomplete Cholewski decomposition of A. Even if A is sparse, that is has many zeros, the process of LU or Cholewski decomposition is very expensive, since it creates non zero values.

Example : Matrix of finite differences in a square

Poisson equation

$$-(\Delta_h u)_{i,j} = -\frac{1}{h^2}(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) - \frac{1}{h^2}(u_{i,j+1} - 2u_{i,j} + u_{i,j-1}) = f_{i,j},$$
$$1 \leq i \leq M, 1 \leq j \leq M$$

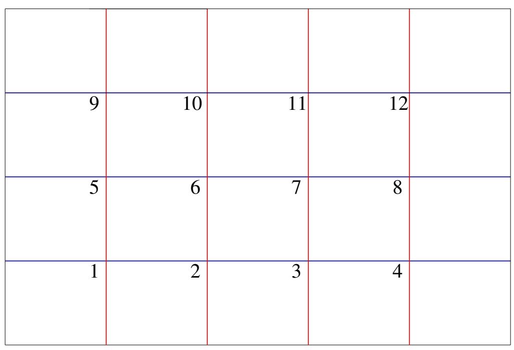


FIGURE 1.7 – Numbering by line

The point (x_i, y_j) has for number $i+(j-1)M$. A vector of all unknowns X is created :

$$Z = (u_{1,1}, u_{2,1}, u_{M,1}), (u_{1,2}, u_{2,2}, u_{M,2}), \cdots (u_{1,M}, u_{2,M}, u_{M,M})$$

with $Z_{i+(j-1)*M} = u_{i,j}$.
If the equations are numbered the same way (equation $\#k$ is the equation at point k), the matrix is block-tridiagonal :

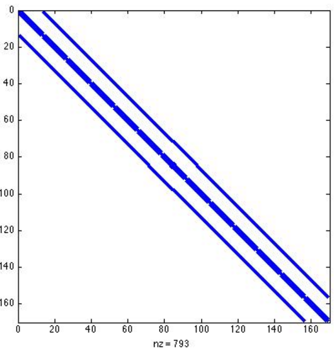
$$A = \frac{1}{h^2} \begin{pmatrix} B & -C & & 0_M & \\ -C & B & -C & & \\ & \ddots & \ddots & \ddots & \\ & & -C & B & -C \\ & 0_M & & -C & B \end{pmatrix} \tag{1.1}$$

$$C = I_M, \quad B = \begin{pmatrix} 4 & -1 & & 0 & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & 0 & & -1 & 4 \end{pmatrix}$$

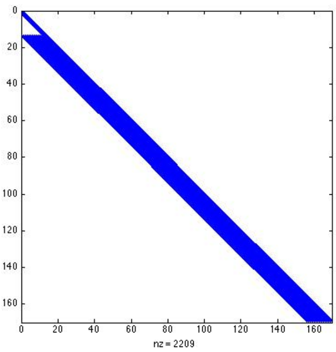
The righthand side is $b_{i+(j-1)*M} = f_{i,j}$, and the system takes the form $AZ = b$.

Cholewski decomposition of A

The block-Cholewski decomposition of A, $A = RR^T$, is block-bidiagonale, but the blocks are not tridiagonale as in A, as the `spy` command of matlab can show, in the case where $M = 15$.



spy(A)



spy(R)

However, if we look closely to the values of R between the main diagonales where A was non zero, we see that where the coefficients of A are zero, the coefficients of R are small. Therefore the incomplete Cholewski preconditioning computes only the values of R where the coefficient of A is not zero, and gains a lot of computational time.

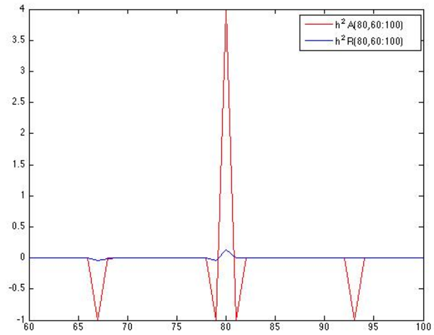


FIGURE 1.8 – Variation of the coefficients of Cholewski in the line 80 for $M = 15$

The matlab codes are as follows ([3])

Cholewski

```
Ch=tril(A);
for k=1:nn
    Ch(k,k)=sqrt(Ch(k,k));
    Ch(k+1:nn,k)=Ch(k+1:nn,k)/Ch(k,k);
    for j=k+1:nn
        Ch(j:nn,j)=Ch(j:nn,j)-Ch(j:nn,k)*Ch(j,k);
    end
end
```

Incomplete Cholewski

```
ChI=tril(A);
for k=1:nn
    ChI(k,k)=sqrt(ChI(k,k));
    for j=k+1:nn
        if ChI(j,k) ~= 0
            ChI(j,k)=ChI(j,k)/ChI(k,k);
        end
    end
    for j=k+1:nn
        for i=j:n
            if ChI(i,j) ~= 0
                ChI(i,j)=ChI(i,j)-ChI(i,k)*ChI(j,k);
            end
        end
    end
end
```

Then use $C = R * R^T$.

Comparison For the 2-D finite differences matrix and $n = 25$ internal points in each direction, we compare the convergence of the conjugate gradient and various preconditioning : Gauss-Seidel, SSOR with optimal parameter, and incomplete Cholewski. The gain even with $\omega = 1$ is striking. Furthermore Gauss-Seidel is comparable with Incomplete Cholewski.

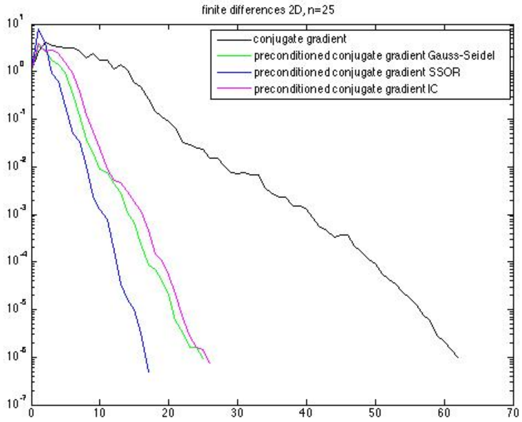


FIGURE 1.9 – Convergence history, influence of preconditioning

1.6 Krylov methods for non symmetric matrices, Arnoldi algorithm

1.6.1 Gram-Schmidt orthogonalization and QR decomposition

Starting with a free family $(v_1, \cdots, v_m, \cdots)$ in a vector space E, the process builds an orthonormal family $(w_1, \cdots, w_m, \cdots)$ recursively.

•. Define $w_1 = \dfrac{v_1}{\|v_1\|}$.

•. Note $r_{1,2} = (v_2, w_1)$, and define $u_2 = v_2 - r_{1,2}w_1$. By construction u_2 is orthogonal to w_1. It only remains to make it of norm 1 by defining $r_{2,2} = \|u_2\|$, $w_2 = \dfrac{u_2}{r_{2,2}}$.

•. Suppose we have built $(w_1, \cdots, w_j)$ orthonormal. Define $r_{i,j+1} = (v_{j+1}, w_i)$ for $1 \leq i \leq j$, and

$$u_{j+1} = v_{j+1} - \sum_{i=1}^{j} r_{i,j+1}w_i, \quad r_{j+1,j+1} = \|u_{j+1}\|, \quad w_{j+1} = \frac{u_{j+1}}{r_{j+1,j+1}}.$$

Then $(w_1, \cdots, w_j)$ is orthonormal. Furthermore, we can rewrite the previous equality as

$$v_{j+1} = r_{j+1,j+1}w_{j+1} + \sum_{i=1}^{j} r_{i,j+1}w_i,$$

which gives for each j ;

$$v_j = \sum_{i=1}^{j} r_{i,j}w_i\,. \tag{1.2}$$

Define the matrix K whose columns are the v_j, the matrix Q whose columns are the w_j, and the upper triangular matrix R whose coefficients are $r_{i,j}$ for $i \leq j$, and 0 otherwise. Then (1.2) takes the matrix form

$$K = QR \tag{1.3}$$

The matrix Q is orthogonal, so this is exactly the so-called QR decomposition of the matrix K. Note that the matrix K DOES NOT NEED TO BE SQUARE, nor the matrix Q, but the matrix R has size $m \times m$.

1.6.2 Arnoldi algorithm

The purpose is to build recursively a orthonormal basis of the Krylov space $\mathcal{K}_m = vect(r, Ar, \cdots, A^{m-1}r)$. We will take advantage of the special form of the generating family to proceed a slight modification of Gram Schmidt.

•. Define $q_1 = \frac{r}{\|r\|}$.

•. Now we must orthogonalize q_1 and Ar, or equivalently q_1 and Aq_1 :

$$h_{1,1} = (Aq_1, q_1), \quad u_2 = Aq_1 - h_{1,1}q_1, \quad h_{2,1} = \|u_2\|, \quad q_2 = \frac{u_2}{h_{2,1}}.$$

Then $q_2 \in Vec(q_1, Aq_1) = Vec(r, Ar) = \mathcal{K}_2$ and (q_1, q_2) is an orthonormal basis. All this can be rewritten as

$$Aq_1 = h_{1,1}q_1 + h_{2,1}q_2.$$

Then $\mathcal{K}_3 = Vec(q_1, q_2, A^2r) = Vec(q_1, q_2, Aq_2)$. Therefore, instead of orthonormalizing with the new vector A^2r, we can do it with the new vector Aq_2. Define

$$u_3 = Aq_2 - h_{1,2}q_1 - h_{2,2}q_2, \quad h_{2,2} = (Aq_2, q_2), \quad h_{1,2} = (Aq_2, q_1), \quad h_{3,2} = \|u_3\|, \; q_3 = \frac{u_3}{h_{3,2}}.$$

This generalizes in building an orthonormal basis of $\mathcal{K}_{j+1}$ by

$$u_{j+1} = Aq_j - \sum_{i=1}^{j} h_{i,j}q_i \;, \quad h_{i,j} = (Aq_j, q_i), \quad h_{j+1,j} = \|u_{j+1}\|, \; q_{j+1} = \frac{u_{j+1}}{h_{j+1,j}}.$$

Theorem 1.13 *If the algorithm goes through m, then $(q_1, \ldots, q_m)$ is a basis of $\mathcal{K}_m$.*

Below is the matlab script

```
for j=1:m do
                h(i,j)=(A*v(j,:),v(i,:)) , i=1:i
        w(j,:)=A*v(j,:)—sum(h(i,j)v(i,:)
        h(j+1,j)=norm(w(j,:),2)
        If h(j+1,j) == 0 stop
        v(j+1,:)= w(j,:)/h(j+1,j)
```

The definition of the q_j above can be rewritten as

$$Aq_j = \sum_{i=1}^{j+1} h_{i,j}q_i \tag{1.4}$$

Define the Hessenberg matrix $\widetilde{H}_m$ as the matrix of the $h_{i,j}$ for $i \le j+1$, and 0 otherwise. $\widetilde{H}_m$ is a matrix of size $(m+1) \times m$.

$$\widetilde{H}_m = \begin{pmatrix} h_{1,1} & & \cdots & & h_{1,m} \\ h_{2,1} & h_{2,2} & & \cdots & h_{2,m} \\ 0 & h_{3,2} & \ddots & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & h_{m,m-1} & h_{m,m} \\ 0 & 0 & 0 & 0 & h_{m+1,m} \end{pmatrix}$$

Define $V_m = [q_1, \cdots, q_m]$. H_m is the $m \times m$ matrix obtained from the $(m+1) \times m$ matrix $\widetilde{H}_m$ by deleting the last row.

Proposition 1.1

$$AV_m = V_{m+1}\widetilde{H}_m, \quad AV_m = V_{m+1}\widetilde{H}_m = V_m H_m + h_{m+1,m} q_{m+1} e_m^T, \quad V_m^T A V_m = H_m. \tag{1.5}$$

Proof The first identity is just rewriting (1.4). As for the second one, rewrite the first one in blocks as

$$V_{m+1}\widetilde{H}_m = [V_m, q_{m+1}] \begin{bmatrix} H_m \\ h_{m+1,m} e_m^T \end{bmatrix} = V_m H_m + h_{m+1,m} q_{m+1} e_m^T.$$

Use this now in the first equality to obtain

$$AV_m = V_m H_m + h_{m+1,m} q_{m+1} e_m^T.$$

Multiply on the left by V_m^T. Since V_m is orthogonal, and $V_m^T q_{m+1} = [(q_1, q_{m+1}), \cdots, (q_m, q_{m+1})]^T = 0$, we obtain

$$V_m^T A V_m = H_m.$$

■

1.6.3 Full orthogonalization method or FOM

Search for an approximate solution in $x_0 + \mathcal{K}_m(A, r_0)$ in the form $x_m = x_0 + V_m y$, and impose $r_m \perp \mathcal{K}_m(A, r_0)$. This is equivalent to $V_m^T r_m = 0$, which by

$$r_m = b - A(x_0 + V_m y) = r_0 - AV_m y$$

can be written by (1.5) as

$$V_m^T A V_m y = V_m^T r_0 \text{ or } H_m y = \|r_0\| e_1.$$

The small Hessenberg system

$$H_m y = \|r_0\| e_1 \tag{1.6}$$

can be solved at each step using a direct method : suppose all the principal minors of H_m are nonzero. Due to the special structure of H_m, the LU factorization of H_m has the form

$$L = \begin{pmatrix} 1 & & \cdots & & 0 \\ l_1 & 1 & & \cdots & 0 \\ 0 & l_2 & \ddots & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & l_{m-1} & 1 \end{pmatrix}, \quad U = \begin{pmatrix} u_{11} & & \cdots & & u_{1m} \\ 0 & u_{22} & & \cdots & u_{2m} \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & u_{mm} \end{pmatrix}$$

The following matlab code gives the LU factorization

```
 u(1,:)=h(1,:);
 for i=1:m-1
 l(i)=h(i+1,i)/u(i,i);
  for j=i+1:n
u(i+1,j)=h(i+1,j)-l(i)*u(i,j)
end
end
```

```
u(1,:)=h(1,:);
for i=1:m-1
l(i)=h(i+1,i)/u(i,i);
       for j=i+1:n
       u(i+1,j)=h(i+1,j)-l(i)*u(i,j)
       end
end
```

The computational cost is $m^2 + 2m - 1$ operations.

Theorem 1.14 *At each step m, r_m is parallel to q_{m+1}.*

Proof

$$r_m = r_0 - AV_m y = r_0 - (V_m H_m + h_{m+1,m} q_{m+1} e_m^T) y = r_0 - V_m H_m y - h_{m+1,m} y_m q_{m+1}.$$

But $H_m y = \|r_0\| e_1$, therefore $r_0 - V_m H_m y = r_0 - \|r_0\| V_m e_1 = r_0 - \|r_0\| q_1 = 0$. Therefore $r_m = -h_{m+1,m} y_m q_{m+1}$ is parallel to q_{m+1}.

■

```
function [X,R,H,Q]=FOM(A,b,x0);
% FOM full orthogonalization method
% [X,R,H,Q]=FOM(A,b,x0) computes the decomposition A=QHQ?, Q orthogonal
% and H upper Hessenberg, Q(:,1)=r/norm(r), using Arnoldi in order to
% solve the system Ax=b with the full orthogonalization method. X contains
% the iterates and R the residuals
n=length(A); X=x0;
r=b-A*x0; R=r; r0norm=norm(r);
Q(:,1)=r/r0norm;
for k=1:n
    v =A*Q(:,k);
    for j=1:k
        H(j,k)=Q(:,j)'*v; v=v-H(j,k)*Q(:,j);
    end
    e0=zeros(k,1); e0(1)=r0norm; % solve system
    y=H\e0; x= x0+Q*y;
    X=[X x];
    R=[R b-A*x];
    if k<n
        H(k+1,k)=norm(v); Q(:,k+1)=v/H(k+1,k);
    end
end
```

1.6.4 GMRES algorithm

Here we minimize at each step the residual r_m in $\mathcal{K}_m(A, r_0)$, which is equivalent to the minimization of $J(y) = \|r_0 - AV_m y\|_2$ for y in $\mathbb{R}^m$, Use the Proposition to write

$$r_0 - AV_m y = \|r_0\| q_1 - V_{m+1} \widetilde{H}_m y = V_{m+1}(\|r_0\| e_1 - \widetilde{H}_m y).$$

Since V_{m+1} is orthogonal, then

$$\|r_0 - AV_m y\| = \|\|r_0\| e_1 - \widetilde{H}_m y\|.$$

So in FOM we solve EXACTLY the square system $H_m y = \|r_0\| e_1$, while in GMRES we solve the LEAST SQUARE problem $\inf \|\|r_0\| e_1 - \widetilde{H}_m y\|$. This small minimization problem has a special form with a upper Hessenberg form, and is best solved by the Givens reflection method. Let us consider the case of $m = 3$ ($\sigma_0 = \|r_0\|$).

$$z = \widetilde{H}_3 y - \sigma_0 e_1 = \begin{pmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ 0 & h_{3,2} & h_{3,3} \\ 0 & 0 & h_{4,3} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} - \begin{pmatrix} \sigma_0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Multiply successively by the three $(m+1) \times (m+1)$ Givens matrices

$$Q_1 = \begin{pmatrix} c_1 & s_1 & 0 & 0 \\ -s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad Q_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c_2 & s_2 & 0 \\ 0 & -s_2 & c_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad Q_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & c_3 & s_3 \\ 0 & 0 & -s_3 & c_3 \end{pmatrix},$$

to make the system triangular, and even better

$$Q_3 Q_2 Q_1 z = \left(\begin{array}{ccc} \tilde{h}_{1,1} & \tilde{h}_{1,2} & \tilde{h}_{1,3} \\ 0 & \tilde{h}_{2,2} & \tilde{h}_{2,3} \\ 0 & 0 & \tilde{h}_{3,3} \\ \hline 0 & 0 & 0 \end{array}\right) \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} - \left(\begin{array}{c} c_1 \\ c_2 \\ c_3 \\ \hline c_4 \end{array}\right)$$

Therefore

$$\|z\|^2 = \|Q_3 Q_2 Q_1 z\|^2 = \|Ry - c^I\|^2 + (c_4)^2$$

where R is the upperblock of the matrix, and c^I the upperblock of the vector. Now we have a regular system, and y is THE solution of

$$Ry = c^I,$$

which is now an upper triangular system.

```
function [x,iter,resvec] = GMRES(A,b,restart,tol,maxit)
%GMRES  Generalized Minimum Residual Method for Schwarz methods
%   [x,iter]=GMRES(A,b,RESTART,TOL,MAXIT) uses gmres to solve a system
%   Ax=b where A is defined as the procedure 'A'.
%   This is an adapted copy of Matlabs GMRES.

n = length(b);

n2b = norm(b);                    % Norm of rhs vector, b

% x0=rand(n,1);
% x0 = ones(n,1);
f=1;                              % all frequencies to initialize
x0 = sin((1:n/2)'/(n/2+1)*pi*f); x0=[x0; x0];
for f=2:n/2,
  x0 = x0+[sin((1:n/2)'/(n/2+1)*pi*f); sin((1:n/2)'/(n/2+1)*pi*f)];
end;

x = x0;

% Set up for the method
flag = 1;
xmin = x;                         % Iterate which has minimal residual so far
imin = 0;                         % Outer iteration at which xmin was computed
jmin = 0;                         % Inner iteration at which xmin was computed
tolb = tol * n2b;                 % Relative tolerance
if tolb==0,
```

```
  tolb=tol;                     % use absolute error to find zero solution
end;
r = b - feval(A,x);             % Zero-th residual
normr = norm(r);                % Norm of residual

if normr <= tolb                % Initial guess is a good enough solution
  flag = 0;
  relres = normr / n2b;
  iter = 0;
  resvec = normr;
  os = sprintf(['The initial guess has relative residual %0.2g' ...
                ' which is within\nthe desired tolerance %0.2g' ...
                ' so GMRES returned it without iterating.'], ...
                relres,tol);
  disp(os);
  return;
end


resvec = zeros(restart*maxit+1,1); % Preallocate vector for norm of residuals
resvec(1) = normr;                 % resvec(1) = norm(b-A*x0)
normrmin = normr;                  % Norm of residual from xmin
rho = 1;
stag = 0;                          % stagnation of the method

% loop over maxit outer iterations (unless convergence or failure)

for i = 1 : maxit
  V = zeros(n,restart+1);          % Arnoldi vectors
  h = zeros(restart+1,1);          % upper Hessenberg st A*V = V*H ...
  QT = zeros(restart+1,restart+1); % orthogonal factor st QT*H = R
  R = zeros(restart,restart);      % upper triangular factor st H = Q*R
  f = zeros(restart,1);            % y = R\f => x = x0 + V*y
  W = zeros(n,restart);            % W = V*inv(R)
  j = 0;                           % inner iteration counter

  vh = r;
  h(1) = norm(vh);
  V(:,1) = vh / h(1);
  QT(1,1) = 1;
  phibar = h(1);

  for j = 1 : restart
    j
%   MapU(x,sqrt(n),sqrt(n));
```

```
    u = feval(A,V(:,j));         % matrix multiply
    for k = 1 : j
      h(k) = V(:,k)' * u;
      u = u - h(k) * V(:,k);
    end
    h(j+1) = norm(u);
    V(:,j+1) = u / h(j+1);
    R(1:j,j) = QT(1:j,1:j) * h(1:j);
    rt = R(j,j);

% find cos(theta) and sin(theta) of Givens rotation
    if h(j+1) == 0
      c = 1.0;                      % theta = 0
      s = 0.0;
    elseif abs(h(j+1)) > abs(rt)
      temp = rt / h(j+1);
      s = 1.0 / sqrt(1.0 + temp^2); % pi/4 < theta < 3pi/4
      c = - temp * s;
    else
      temp = h(j+1) / rt;
      c = 1.0 / sqrt(1.0 + temp^2); % -pi/4 <= theta < 0 < theta <= pi/4
      s = - temp * c;
    end

    R(j,j) = c * rt - s * h(j+1);
%   zero = s * rt + c * h(j+1);

    q = QT(j,1:j);
    QT(j,1:j) = c * q;
    QT(j+1,1:j) = s * q;
    QT(j,j+1) = -s;
    QT(j+1,j+1) = c;
    f(j) = c * phibar;
    phibar = s * phibar;

    if j < restart
      if f(j) == 0                  % stagnation of the method
        stag = 1;
      end
      W(:,j) = (V(:,j) - W(:,1:j-1) * R(1:j-1,j))/ R(j,j);
% Check for stagnation of the method
      if stag == 0
        stagtest = zeros(n,1);
        ind = (x ~= 0);
        if ~(i==1 & j==1)
          stagtest(ind) = W(ind,j) ./ x(ind);
          stagtest(~ind & W(:,j) ~= 0) = Inf;
          if abs(f(j))*norm(stagtest,inf) < eps
```

```
          stag = 1;
        end
      end
    end
    x = x + f(j) * W(:,j);         % form the new inner iterate
  else % j == restart
    vrf = V(:,1:j)*(R(1:j,1:j)\f(1:j));
% Check for stagnation of the method
    if stag == 0
      stagtest = zeros(n,1);
      ind = (x0 ~= 0);
      stagtest(ind) = vrf(ind) ./ x0(ind);
      stagtest(~ind & vrf ~= 0) = Inf;
      if norm(stagtest,inf) < eps
        stag = 1;
      end
    end
    x = x0 + vrf;                  % form the new outer iterate
  end
  normr = norm(b-feval(A,x));
  resvec((i-1)*restart+j+1) = normr;

  if normr <= tolb                 % check for convergence
    if j < restart
      y = R(1:j,1:j) \ f(1:j);
      x = x0 + V(:,1:j) * y;       % more accurate computation of xj
      r = b - feval(A,x);
      normr = norm(r);
      resvec((i-1)*restart+j+1) = normr;
    end
    if normr <= tolb               % check using more accurate xj
      flag = 0;
      iter = [i j];
      break;
    end
  end

  if stag == 1
    flag = 3;
    break;
  end

  if normr < normrmin              % update minimal norm quantities
    normrmin = normr;
    xmin = x;
    imin = i;
    jmin = j;
  end
end                                % for j = 1 : restart

```

```
  if flag == 1
    x0 = x;                           % save for the next outer iteration
    r = b - feval(A,x0);
  else
    break;
  end

end                                 % for i = 1 : maxit

% returned solution is that with minimum residual

if n2b==0,
  n2b=1;          % here we solved for the zero solution and thus show
end;              % the absolute residual !

if flag == 0
  relres = normr / n2b;
else
  x = xmin;
  iter = [imin jmin];
  relres = normrmin / n2b;
end

% truncate the zeros from resvec
if flag <= 1 | flag == 3
  resvec = resvec(1:(i-1)*restart+j+1);
else
  if j == 0
    resvec = resvec(1:(i-1)*restart+1);
  else
    resvec = resvec(1:(i-1)*restart+j);
  end
end


% only display a message if the output flag is not used
switch(flag)
    case 0,
      os = sprintf(['GMRES(%d) converged at iteration %d(%d) to a' ...
                    ' solution with relative residual %0.2g'], ...
                    restart,iter(1),iter(2),relres);

    case 1,
      os = sprintf(['GMRES(%d) stopped after the maximum %d iterations' ...
                    ' without converging to the desired tolerance %0.2g' ...
                    '\n         The iterate returned (number %d(%d))'
```

```
                                 ...
                          ' has relative residual %0.2g'], ...
                          restart,maxit,tol,iter(1),iter(2),relres);

      case 2,
        os = sprintf(['GMRES(%d) stopped at iteration %d(%d)' ...
                          ' without converging to the desired tolerance %0.2g' ...
                          '\n         because the system involving the' ...
                          ' preconditioner was ill conditioned.' ...
                          '\n         The iterate returned (number %d(%d))' ...
                          ' has relative residual %0.2g'], ...
                          restart,i,j,tol,iter(1),iter(2),relres);

      case 3,
        os = sprintf(['GMRES(%d) stopped at iteration %d(%d)' ...
                          ' without converging to the\n         desired' ...
                          ' tolerance %0.2g because the method stagnated.' ...
                          '\n         The iterate returned (number %d(%d))' ...
                          ' has relative residual %0.2g'], ...
                          restart,i,j,tol,iter(1),iter(2),relres);

end                                 % switch(flag)
if flag == 0
  disp(os);
else
  warning(os);
end

semilogy(0:length(resvec)-1,resvec);
```

Remark If A is symmetric, H_m is tridiagonale.

Restarted GMRES For reasons of storage cost, the GMRES algorithm is mostly used by restarting every M steps :

Compute $x_1, \cdots, x_M$.

If r_M is small enough, stop,

else restart with $x_0 = x_M$.