



USN-HCMV

PARIS 13

JOINT MASTER 2

High Performance Computing

Pr. Laurence Halpern and Juliette Ryan

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, and the size of the system is huge.

December 2017

Table des matières

1	Classical methods	5
1.1	Direct methods	5
1.2	Stationary iterative methods	10
1.3	Sparse and banded matrices	14
2	Nonstationary methods	25
2.1	Non-Stationary iterative methods. Symmetric definite positive matrices	25
2.2	Krylov methods for non symmetric matrices, Arnoldi algorithm	31
3	Preconditioning	43
3.1	Introduction	43
3.2	Deflation method for GMRES	47
3.3	Fast methods using Fast Fourier Transform	50

Chapitre 1

Classical methods

Contents

1.1 Direct methods	5
1.1.1 Gauss method	5
1.1.2 Codes	7
1.1.3 Theoretical results	7
1.1.4 Symmetric definite matrices : Cholewski decomposition	8
1.1.5 Elimination with Givens rotations	8
1.1.6 QR Decomposition	9
1.2 Stationary iterative methods	10
1.2.1 Classical methods	11
1.2.2 Fundamentals tools	11
1.3 Sparse and banded matrices	14
1.3.1 Direct methods	14
1.3.2 Iterative methods	19
1.3.3 Implementation issues	19

1.1 Direct methods

1.1.1 Gauss method

Example

$$\underbrace{\begin{pmatrix} 1 & 3 & 1 \\ 1 & 1 & -1 \\ 3 & 11 & 6 \end{pmatrix}}_A \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_x = \underbrace{\begin{pmatrix} 9 \\ 1 \\ 36 \end{pmatrix}}_b$$

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 :

$$M_1 = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

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

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

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

The matrix $U = MA$ is upper triangular, and solving $Ux = Mb$ is simpler than solving $Ax = b$.

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

Define $L_j = M_j^{-1}$. In the column j , the entries below the diagonal are those of M_j with a change of sign.

$$L_1 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix}, \quad L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}$$

$$L := M^{-1} = (M_2 M_1)^{-1} = M_1^{-1} M_2^{-1} = L_1 L_2 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & -1 & 1 \end{pmatrix}.$$

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

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

1.1.2 Codes

```
1 function x=BackSubstitution(U,b)
2 % BACKSUBSTITUTION solves by backsubstitution a linear system
3 % x=BackSubstitution(U,b) solves  $Ux=b$ ,  $U$  upper triangular by
4 % backsubstitution
5 n=length(b);
6 for k=n:-1:1
7 s=b(k);
8 for j=k+1:n
9 s=s-U(k,j)*x(j);
10 end
11 x(k)=s/U(k,k);
12 end
13 x=x(:);

1 function x=Elimination(A,b)
2 % ELIMINATION solves a linear system by Gaussian elimination
3 % x=Elimination(A,b) solves the linear system  $Ax=b$  using Gaussian
4 % Elimination with partial pivoting. Uses the function
5 % BackSubstitution
6 n=length(b);
7 norma=norm(A,1);
8 A=[A,b]; % augmented matrix
9 for i=1:n
10 [maximum,kmax]=max(abs(A(i:n,i))); % look for Pivot  $A(kmax,i)$ 
11 kmax=kmax+i-1;
12 if maximum < 1e-14*norma; % only small pivots
13 error('matrix is singular')
14 end
15 if i ~= kmax % interchange rows
16 h=A(kmax,:); A(kmax,:)=A(i,:); A(i,:)=h;
17 end
18 A(i+1:n,i)=A(i+1:n,i)/A(i,i); % elimination step
19 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);
20 end
21 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 :

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

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 $\cotg \alpha_{ki}$, that is α_{ki} modulo π . For stability reasons, line (i) is also modified, and we end up with

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

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

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

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


```

1 function x=BackSubstitutionSAXPY(U,b)
2 % BACKSUBSTITUTIONSAXPY solves linear system by backsubstitution
3 % x=BackSubstitutionSAXPY(U,b) solves Ux=b by backsubstitution by
4 % modifying the right hand side (SAXPY variant)n=length(b);
5 n=length(b);
6 for i=n:-1:1
7 x(i)=b(i)/U(i,i);
8 b(1:i-1)=b(1:i-1)-x(i)*U(1:i-1,i);
9 end
10 x=x(:);

```

```

1 function x=EliminationGivens(A,b);
2 % ELIMINATIONGIVENS solves a linear system using Givens-rotations
3 % x=EliminationGivens(A,b) solves Ax=b using Givens-rotations. Uses
4 % the function BackSubstitutionSAXPY.
5 n=length(A);
6 for i= 1:n
7 for k=i+1:n
8 if A(k,i)~=0
9 cot=A(i,i)/A(k,i); % rotation angle
10 si=1/sqrt(1+cot^2); co=si*cot;
11 A(i,i)=A(i,i)*co+A(k,i)*si; % rotate rows
12 h=A(i,i+1:n)*co+A(k,i+1:n)*si;
13 A(k,i+1:n)=-A(i,i+1:n)*si+A(k,i+1:n)*co;
14 A(i,i+1:n)=h;
15 h=b(i)*co+b(k)*si; % rotate right hand side
16 b(k)=-b(i)*si+b(k)*co; b(i)=h;
17 end
18 end;
19 if A(i,i)==0
20 error('Matrix is singular');
21 end;
22 end
23 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 $n = 5$,

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

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

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

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

G^{ik} represents the rotation of angle α in the plane generated by \mathbf{e}_i and \mathbf{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}, \dots, G_{n1} wherever 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, \dots, G_N such that Then

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

Q is an orthogonal matrix,

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

then

$$A = QR,$$

we have reached the QR decomposition of the matrix A .

1.2 Stationary iterative methods

For any splitting $A = M - N$, write $Mx = Nx + b$,

Define the sequence $Mx^{m+1} = Nx^m + b$.

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

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 $\frac{\|r^m\|}{\|b\|} < \varepsilon$.

1.2.1 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 eigenvalues of A .

1.2.2 Fundamentals tools

Define the sequence

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

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

- $\rho(R) = \max\{|\lambda|, \lambda \text{ eigenvalue of } R\}$ is the *spectral radius* of R .
- $\rho_m(R) = \|R^m\|^{1/m}$ is the *mean convergence factor* of R .
- $\rho_\infty(R) = \lim_{m \rightarrow \infty} \|R^m\|^{1/m}$ 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 $m \sim \frac{\log \varepsilon}{\log \rho_m(R)}$. It is easier to use the asymptotic value relation, $m \sim$

$\frac{\log \varepsilon}{\log \rho_\infty(R)}$. Then to obtain another decimal digit in the solution, one needs

to choose $\varepsilon = 10^{-1}$, then $\bar{m} \sim -\frac{\ln(10)}{\ln(\rho(R))}$.

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 $\omega^* = \frac{2}{1 + \sqrt{1 - (\rho(J))^2}}$, and then $\rho(\mathcal{L}_{\omega^*}) = \omega^* - 1$.

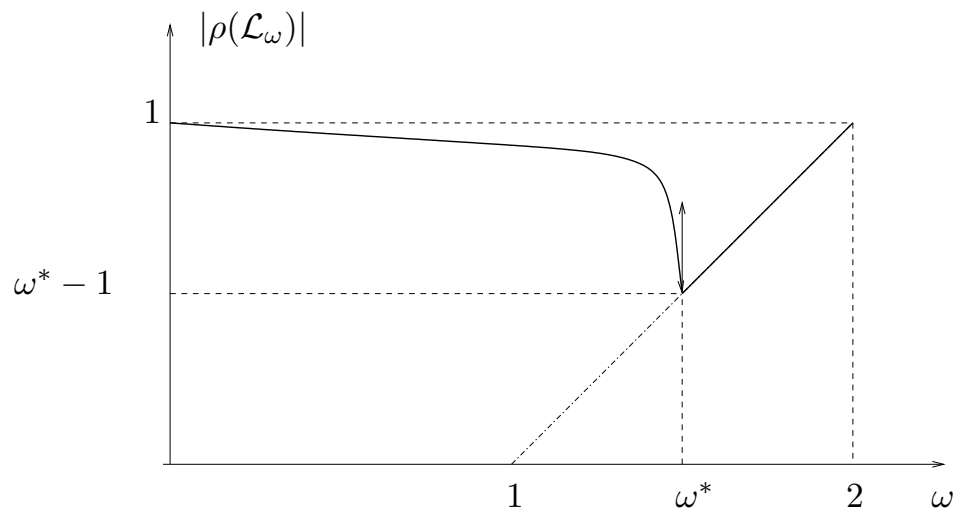


FIGURE 1.1 – Variations of $\rho(\mathcal{L}_\omega)$ as a fonction of ω

1.3 Sparse and banded matrices

1.3.1 Direct methods

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.

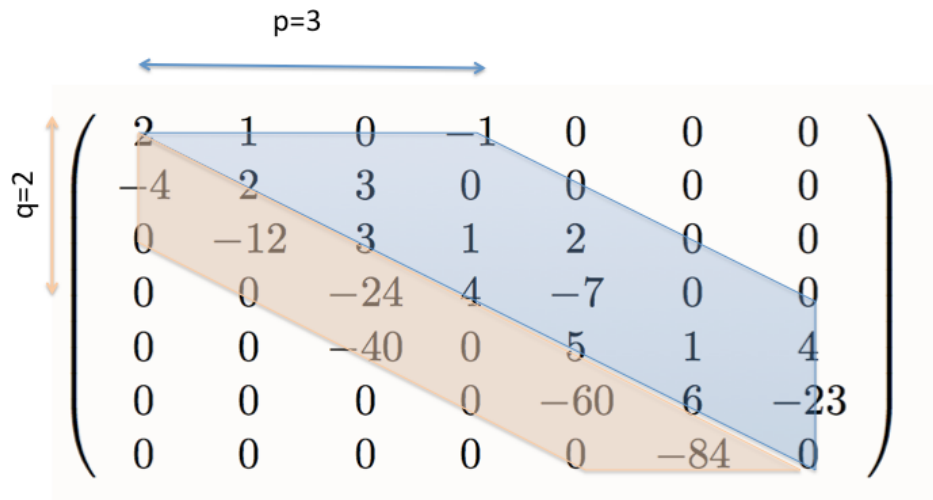


FIGURE 1.2 – A bandmatrix

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

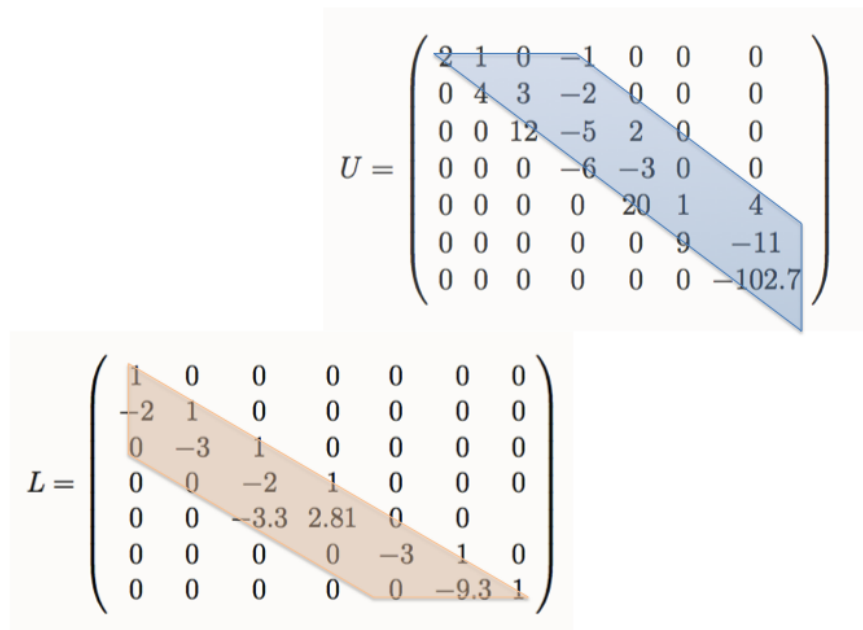


FIGURE 1.3 – LU decomposition

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

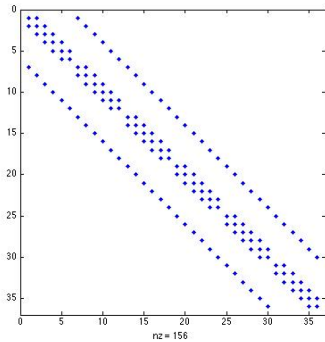
$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.6 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -0.5 & -0.17 & -0.05 & -0.21 & 0.025 & 0.0027 & 1 \end{pmatrix}$$

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

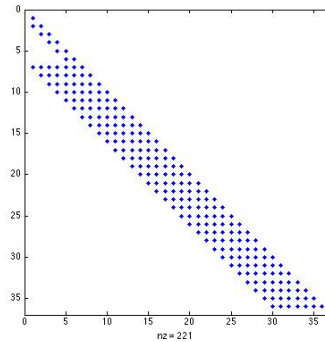
Here the permutation matrix is

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

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 $A = \text{diag}(c, -1) + \text{diag}(d, 0) + \text{diag}(e, 1)$.

LU factorization : verify that

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

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

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

```

1 n=length(d);
2 for k=1:n-1 % LU-decomposition with no pivoting
3     c(k)=c(k)/d(k);
4     d(k+1)=d(k+1)-c(k)*e(k);
5 end
6 for k=2:n % forward substitution
7     b(k)=b(k)-c(k-1)*b(k-1);
8 end
9 b(n)=b(n)/d(n); % backward substitution
10 for k=n-1:-1:1
11     b(k)=(b(k)-e(k)*b(k+1))/d(k);
12 end

```

Givens : verify that the process inserts an extra updiagonal.

```

1 n=length(d);
2 e(n)=0;
3 for i=1:n-1 % elimination
4     if c(i)~=0
5         t=d(i)/c(i); si=1/sqrt(1+t*t); co=t*si;
6         d(i)=d(i)*co+c(i)*si; h=e(i);
7         e(i)=h*co+d(i+1)*si; d(i+1)=-h*si+d(i+1)*co;
8         c(i)=e(i+1)*si; e(i+1)=e(i+1)*co;
9         h=b(i); b(i)=h*co+b(i+1)*si;
10        b(i+1)=-h*si+b(i+1)*co;
11    end;
12 end;
13 b(n)=b(n)/d(n); % backsubstitution
14 b(n-1)=(b(n-1)-e(n-1)*b(n))/d(n-1);
15 for i=n-2:-1:1,
16     b(i)=(b(i)-e(i)*b(i+1)-c(i)*b(i+2))/d(i);
17 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.

1.3.2 Iterative methods

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

1.3.3 Implementation issues

To minimize computing costs and storage of a sparse matrix, it can be useful to renumber the matrix coefficients. There are (for the moment) no absolute ideal renumbering algorithms but one of the most efficient is the Reverse Cuthill Mackee algorithm.

It is also called the bandwidth reduction problem, also known in the field of sparse matrix applications as the bandwidth minimization problem (or BMP in short) :

For a given symmetric sparse matrix, $A(n \times n)$, the problem is to reduce its bandwidth B by permuting rows and columns so as to move all the non-zero elements of A in a band as close as possible to the diagonal.

In other words, the problem consists in transforming through successive row and column permutations as for example matrix $A1$ (8×8 input matrix) into $A2$:

$$\begin{array}{c}
 \begin{pmatrix}
 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1
 \end{pmatrix}
 \quad
 \begin{pmatrix}
 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
 \end{pmatrix} \\
 A1 \qquad \qquad \qquad A2
 \end{array}$$

Notions of Graph

The graph $G(A)$ corresponding to the matrix A we will have n nodes labelled $i = 1, 2, \dots, n$. For each non-zero element a_{ij} , $i < j$ of A there will be an edge connecting nodes i and j . From the graph of A we can determine the position of all off-diagonal non-zero elements of A .

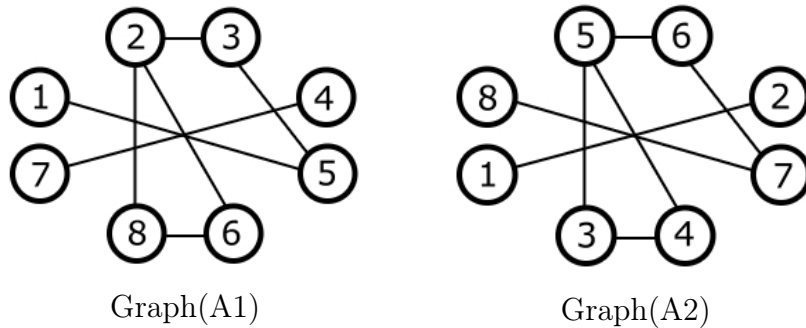
Two nodes of $G(A)$ are said to be adjacent if they are connected by an edge.

Two nodes of $G(A)$ are said to be connected if there is a sequence of edges joining them such that consecutive edges have a common end point. A graph is said to be connected if every pair of nodes of the graph are connected. If $G(A)$ is connected, the corresponding matrix is irreducible.

A component of a graph is a connected subgraph which is not contained in a larger connected subgraph.

The degree of a node i of $G(A)$ is the number of edges meeting at i . For the corresponding matrix, this is the number of non-zero off diagonal elements in row i .

For example, the corresponding graphs of $A1$ and $A2$ are



The two graph structures are identical, the only thing that is different is the node (vertex) labelling. In other words the bandwidth reduction problem can also be viewed as a graph labelling problem :

Find the node labelling that minimizes the bandwidth B of the adjacency matrix of the graph $G(A)$, where we can formally define : $B = \max_{i,j=1..n} |L_i - L_j|$, $i, j = 1..n$ and L_i is the label of node i , L_j is the label of node j and nodes i and j are adjacent.

The Reverse Cuthill Mackee algorithm (RCM)

This algorithm was presented by E. Cuthill and J. McKee in 1969 in REDUCING THE BANDWIDTH OF SPARSE SYMMETRIC MATRICES and improved by A. George

Algorithm RCM

Step 0 : Prepare an empty queue Q and an empty result array R. ;
Step 1 : Select the node in $G(A)$ with the lowest degree (ties are broken arbitrarily) that hasn't previously been inserted in the result array. Let us name it P (for Parent). ;
Step 2 : Add P in the first free position of R. ;
Step 3 : Add to the queue all the nodes adjacent with P in the increasing order of their degree. ;
Step 4.1 : Extract the first node from the queue and examine it. Let us name it C (for Child). ;
Step 4.2 : If C hasn't previously been inserted in R, add it in the first free position and add to Q all the neighbours of C that are not in R in the increasing order of their degree. ;
Step 5 : If Q is not empty repeat from Step 4.1 . ;
Step 6 : If there are unexplored nodes (the graph is not connected) repeat from Step 1 . ;
Step 7 : Reverse the order of the elements in R. Element $R[i]$ is swapped with element $R[n+1-i]$. ;

The result array will be interpreted like this : $R[L] = i$ means that the new label of node i (the one that had the initial label of i) will be L.

Nodes are explored in the increasing order of their degree. Step 7 is not mandatory, it is the modification introduced by George to the initial algorithm (it has the purpose of further reducing the profile of a matrix).

Such a renumbering is also a good technique to reduce computing costs and storage space.

Storage schemes

The main goal is to represent only the non zero elements, and to be able to perform the common matrix operations. In the following, N_Z denotes the total number of non zero elements. Only the most popular schemes are covered here.

— Compressed Sparse Row (CSR)

A real array AA that contains the real non zero values a_{ij} stored row by row, from row 1 to n. The length of AA is N_Z

An integer array JA that contains the column indices of elements a_{ij} as stored in AA. The length of JA is N_Z .

An integer array IA that contains the pointers to the beginning of each row in the arrays AA and JA. $IA(1) = 0$, $IA(2) =$ number of

- non zero elements in row 1, $IA(ii+1) = IA(ii) + \text{number of non zero elements in row } ii$. The length of IA is $n+1$, and $IA(n+1) = N_Z$
- Compressed Sparse Column (CSC)
A variation of CSR but based on storing columns instead of rows.

For example , matrix

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

FIGURE 1.4 – Matrix A

will be stored as follows/

AA	1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12.
JA	1 4 1 2 4 1 3 4 5 3 4 5
IA	1 3 6 10 12 13

FIGURE 1.5 – Sparse Matrix A storage

The case of a CSR storage leads to an efficient matrix vector product. The following Fortran 90 segment shows the main loop of the matrix-by-vector operation for matrices stored in the Compressed Sparse Row stored format.

```

DO I=1, N
  K1 = IA(I)
  K2 = IA(I+1)-1
  Y(I) = DOTPRODUCT(A(K1:K2),X(JA(K1:K2)))
ENDDO

```

FIGURE 1.6 – Sparse Matrix vector product

Notice that each iteration of the loop computes a different component of the resulting vector. This is advantageous because each of these components can be computed independently.

Solving a lower or upper triangular system is another important kernel in sparse matrix computations. The following segment of code shows a simple and parallel routine for solving $LX = Y$ for the CSR storage format.

```
X(1) = Y(1)
DO I = 2, N
  K1 = IAL(I)
  K2 = IAL(I+1)-1
  X(I)=Y(I)-DOTPRODUCT(AL(K1:K2),X(JAL(K1:K2)))
ENDDO
```

FIGURE 1.7 – Computing $LX = Y$

Chapitre 2

Nonstationary methods

Contents

2.1 Non-Stationary iterative methods. Symmetric definite positive matrices	25
2.1.1 Definition of the iterative methods	25
2.1.2 Comparison of the iterative methods	27
2.1.3 Condition number and error	28
2.2 Krylov methods for non symmetric matrices, Arnoldi algorithm	31
2.2.1 Gram-Schmidt orthogonalization and QR decomposition	31
2.2.2 Arnoldi algorithm	32
2.2.3 Full orthogonalization method or FOM	33
2.2.4 GMRES algorithm	35

2.1 Non-Stationary iterative methods. Symmetric definite positive matrices

Descent methods

2.1.1 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 2.1 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

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

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

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

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

Definition 2.1 Krylov space $\mathcal{K}_m = \text{vec}(r^0, Ar^0, \dots, A^{m-1}r^0)$.

Theorem 2.2 (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 2.3 *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

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

while $m < N_{\text{iter}}$ or $\|r^m\| \geq \text{tol}$, do

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

end.

2.1.2 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.

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

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

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

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

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

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

The matrix A is symmetric definite positive.

The discrete problem to be solved is

$$Au = b$$

2.1.3 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 2.4

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

and there is a b such that it is equal.

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

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

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

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

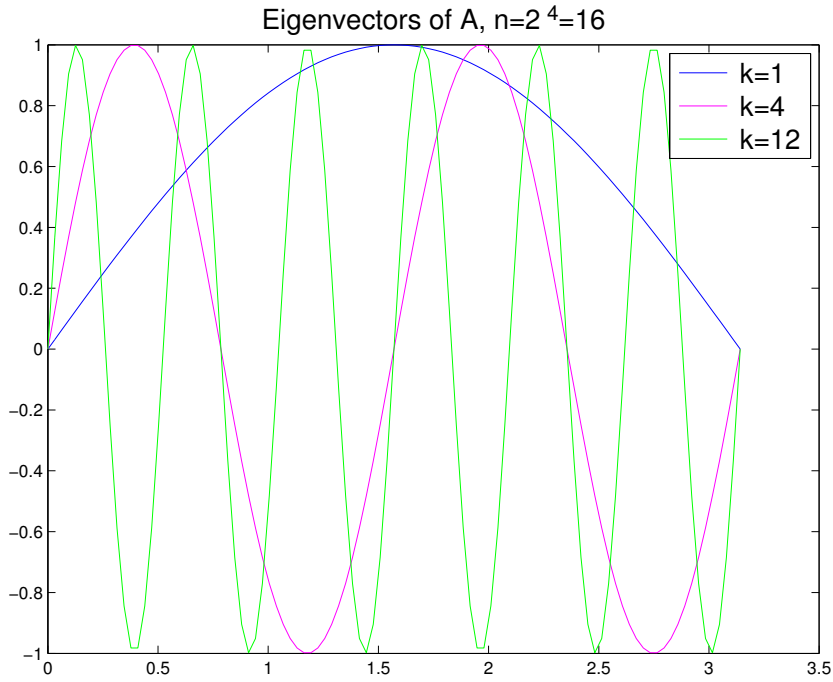


FIGURE 2.1 – Eigenvectors of A

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 2.1 – 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 2.2 – 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 2.3 – 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 2.4 – Reduction factor for one digit $M \sim -\frac{\ln(10)}{F}$

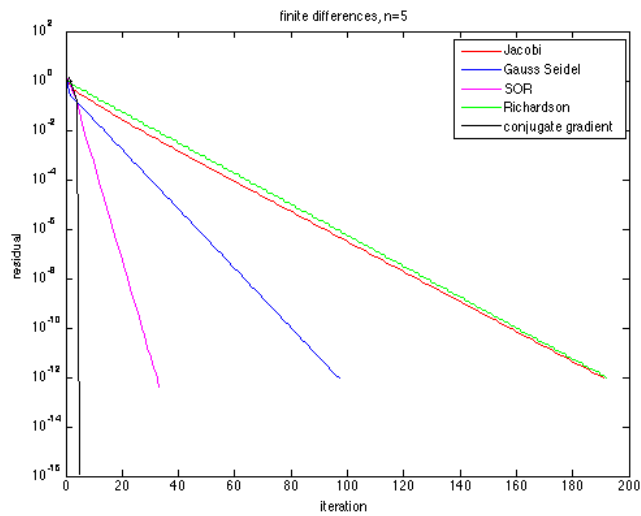


FIGURE 2.2 – Convergence history, $n = 5$

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

TABLE 2.5 – Asymptotic order of the number of elementary operations needed to solve the 1 – D problem as a function of the number of grid points

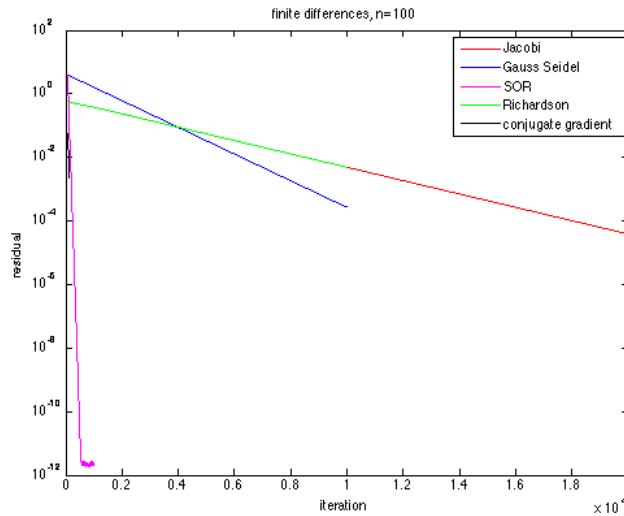


FIGURE 2.3 – Convergence history, $n = 100$

Not only the conjugate gradient is better, but the convergence rate being $\mathcal{O}(h^{1/2})$, the number of iterations necessary to increase 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.

2.2 Krylov methods for non symmetric matrices, Arnoldi algorithm

2.2.1 Gram-Schmidt orthogonalization and QR decomposition

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

- Define $w_1 = \frac{v_1}{\|v_1\|}$.
- 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 $r_{2,2} = \|u_2\|$, $w_2 = \frac{u_2}{r_{2,2}}$.
- Suppose we have built (w_1, \dots, w_{j-1}) an orthonormal basis of $\mathcal{L}(v_1, \dots, v_{j-1})$. Take v_j

and define $r_{i,j} = (v_j, w_i)$ for $1 \leq i \leq j-1$, and

$$u_j = v_j - \sum_{i=1}^{j-1} r_{i,j} w_i, \quad r_{j,j} = \|u_j\|, \quad w_j = \frac{u_j}{r_{j,j}}.$$

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

$$v_j = r_{j,j} w_j + \sum_{i=1}^{j-1} r_{i,j} w_i,$$

which gives for each j ;

$$v_j = \sum_{i=1}^j r_{i,j} w_i. \quad (2.1)$$

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 (2.1) takes the matrix form

$$K_{k,j} = \sum_{i=1}^j r_{i,j} Q_{k,i} \quad K = QR \quad (2.2)$$

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 $m \times m$.

2.2.2 Arnoldi algorithm

Let A a $N \times N$ matrix. The purpose is to build recursively a orthonormal basis of the Krylov space $\mathcal{K}_m = \text{vect}(r, Ar, \dots, A^{m-1}r)$ for $r \in \mathbb{R}^N$. 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 $q_2 \in \text{Vect}(q_1, Aq_1) = \text{Vect}(r, Ar) = \mathcal{K}_2$ and (q_1, q_2) is an orthonormal basis of \mathcal{K}_2 . All this can be rewritten as

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

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

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

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

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

Theorem 2.5 *If the algorithm goes through m , then (q_1, \dots, q_m) is a basis of \mathcal{K}_m .*

Below is the matlab script


```

1 for j=1:m do
2     h(i,j)=(A*v(j,:),v(i,:)) , i=1:i
3     w(j,:)=A*v(j,:)-sum(h(i,j)v(i,:))
4     h(j+1,j)=norm(w(j,:),2)
5     If h(j+1,j) == 0 stop
6     v(j+1,:)= w(j,:)/h(j+1,j)

```

The definition of the q_j above can be rewritten as

$$Aq_j = \sum_{i=1}^{j+1} h_{i,j} q_i, \quad (2.3)$$

$$[Aq_1, \dots, Aq_m] = [q_1, \dots, q_m, q_{m+1}] \underbrace{\begin{bmatrix} h_{1,1} & \cdots & \cdots & h_{1,m} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,m} \\ 0 & h_{3,2} & \ddots & \vdots \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & 0 & h_{m,m-1} & h_{m,m} \\ 0 & 0 & 0 & 0 & h_{m+1,m} \end{bmatrix}}_{\text{Hessenberg matrix } \tilde{H}_m}$$

Define $V_m = [q_1, \dots, q_m] \in \mathcal{M}_{N,m}(\mathbb{R})$. H_m is the $m \times m$ matrix obtained from the $(m+1) \times m$ matrix \tilde{H}_m by deleting the last row.

Proposition 2.1

$$AV_m = V_{m+1} \tilde{H}_m = V_m H_m + h_{m+1,m} q_{m+1} e_m^T, \quad V_m^T AV_m = H_m. \quad (2.4)$$

Proof The first identity is just rewriting (2.3). As for the second one, rewrite the first one in blocks as

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

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 $V_m^T q_{m+1} = [(q_1, q_{m+1}), \dots, (q_m, q_{m+1})]^T = 0$, we obtain

$$V_m^T AV_m = H_m. \quad \blacksquare$$

2.2.3 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 (2.4) as

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

The small Hessenberg system

$$H_m y = \|r_0\| e_1 \quad (2.5)$$

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
```

```
1 u(1,:)=h(1,:);
2 for i=1:m-1
3 l(i)=h(i+1,i)/u(i,i);
4     for j=i+1:n
5         u(i+1,j)=h(i+1,j)-l(i)*u(i,j)
6     end
7 end
```

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

Theorem 2.6 *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} . ■

```
1 function [X,R,H,Q]=FOM(A,b,x0);
2 % FOM full orthogonalization method
3 % [X,R,H,Q]=FOM(A,b,x0) computes the decomposition A=QH?, Q
   orthogonal
4 % and H upper Hessenberg, Q(:,1)=r/norm(r), using Arnoldi in order to
5 % solve the system Ax=b with the full orthogonalization method. X
   contains
6 % the iterates and R the residuals
7 n=length(A); X=x0;
8 r=b-A*x0; R=r; r0norm=norm(r);
9 Q(:,1)=r/r0norm;
10 for k=1:n
11     v =A*Q(:,k);
12     for j=1:k
13         H(j,k)=Q(:,j)'\*v; v=v-H(j,k)*Q(:,j);
14     end
```

```

15 e0=zeros(k,1); e0(1)=r0norm; % solve system
16 y=H\e0; x= x0+Q*y;
17 X=[X x];
18 R=[R b-A*x];
19 if k<n
20     H(k+1,k)=norm(v); Q(:,k+1)=v/H(k+1,k);
21 end
22 end

```

2.2.4 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 $J(y) = \|r_0 - AV_my\|_2$ for y in \mathbb{R}^m . Use the Proposition to write

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

Since V_{m+1} is orthogonal, then

$$\|r_0 - AV_my\| = \|\|r_0\|e_1 - \tilde{H}_m y\|.$$

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 $\inf \|\|r_0\|e_1 - \tilde{H}_m y\|$. This small minimization problem has a special form with an upper Hessenberg form, and is best solved by the Givens reflection method. Let us consider the case of $m = 3$ ($\sigma_0 = \|r_0\|$).

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

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

$$Q_3 Q_2 Q_1 z = \begin{pmatrix} \tilde{h}_{1,1} & \tilde{h}_{1,2} & \tilde{h}_{1,3} \\ 0 & \tilde{h}_{2,2} & \tilde{h}_{2,3} \\ 0 & 0 & \tilde{h}_{3,3} \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} - \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix}$$

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.

```

1 function [x,iter,resvec] = GMRES(A,b,restart,tol,maxit)
2 %GMRES Generalized Minimum Residual Method for Schwarz methods
3 % [x,iter]=GMRES(A,b,RESTART,TOL,MAXIT) uses gmres to solve a
   system

```

```

4 % Ax=b where A is defined as the procedure 'A'.
5 % This is an adapted copy of Matlabs GMRES.
6
7 n = length(b);
8
9 n2b = norm(b); % Norm of rhs vector, b
10
11 % x0=rand(n,1);
12 % x0 = ones(n,1);
13 f=1; % all frequencies to initialize
14 x0 = sin((1:n/2)'/(n/2+1)*pi*f); x0=[x0; x0];
15 for f=2:n/2,
16     x0 = x0+[sin((1:n/2)'/(n/2+1)*pi*f); sin((1:n/2)'/(n/2+1)*pi*f)];
17 end;
18
19 x = x0;
20
21 % Set up for the method
22 flag = 1;
23 xmin = x; % Iterate which has minimal residual
    so far
24 imin = 0; % Outer iteration at which xmin was
    computed
25 jmin = 0; % Inner iteration at which xmin was
    computed
26 tolb = tol * n2b; % Relative tolerance
27 if tolb==0,
28     tolb=tol; % use absolute error to find zero
    solution
29 end;
30 r = b - feval(A,x); % Zero-th residual
31 normr = norm(r); % Norm of residual
32
33 if normr <= tolb % Initial guess is a good enough
    solution
34     flag = 0;
35     relres = normr / n2b;
36     iter = 0;
37     resvec = normr;
38     os = sprintf(['The initial guess has relative residual %0.2g' ...
39                 ' which is within\nthe desired tolerance %0.2g' ...
40                 ' so GMRES returned it without iterating.'], ...
41                 relres,tol);
42     disp(os);
43     return;
44 end
45
46
47 resvec = zeros(restart*maxit+1,1); % Preallocate vector for norm of
    residuals

```

```

48 resvec(1) = normr; % resvec(1) = norm(b-A*x0)
49 normrmin = normr; % Norm of residual from xmin
50 rho = 1;
51 stag = 0; % stagnation of the method
52
53 % loop over maxit outer iterations (unless convergence or failure)
54
55 for i = 1 : maxit
56     V = zeros(n,restart+1); % Arnoldi vectors
57     h = zeros(restart+1,1); % upper Hessenberg st A*V = V*H
58     ...
59     QT = zeros(restart+1,restart+1); % orthogonal factor st QT*H = R
60     R = zeros(restart,restart); % upper triangular factor st H = Q
61     *R
62     f = zeros(restart,1); % y = R\f => x = x0 + V*y
63     W = zeros(n,restart); % W = V*inv(R)
64     j = 0; % inner iteration counter
65
66     vh = r;
67     h(1) = norm(vh);
68     V(:,1) = vh / h(1);
69     QT(1,1) = 1;
70     phibar = h(1);
71
72     for j = 1 : restart
73         j
74         % MapU(x,sqrt(n),sqrt(n));
75
76         u = feval(A,V(:,j)); % matrix multiply
77         for k = 1 : j
78             h(k) = V(:,k)' * u;
79             u = u - h(k) * V(:,k);
80         end
81         h(j+1) = norm(u);
82         V(:,j+1) = u / h(j+1);
83         R(1:j,j) = QT(1:j,1:j) * h(1:j);
84         rt = R(j,j);
85
86         % find cos(theta) and sin(theta) of Givens rotation
87         if h(j+1) == 0
88             c = 1.0; % theta = 0
89             s = 0.0;
90         elseif abs(h(j+1)) > abs(rt)
91             temp = rt / h(j+1);
92             s = 1.0 / sqrt(1.0 + temp^2); % pi/4 < theta < 3pi/4
93             c = - temp * s;
94         else
95             temp = h(j+1) / rt;
96             c = 1.0 / sqrt(1.0 + temp^2); % -pi/4 <= theta < 0 < theta <=
97             pi/4

```

```

95     s = - temp * c;
96     end
97
98     R(j,j) = c * rt - s * h(j+1);
99 % zero = s * rt + c * h(j+1);
100
101     q = QT(j,1:j);
102     QT(j,1:j) = c * q;
103     QT(j+1,1:j) = s * q;
104     QT(j,j+1) = -s;
105     QT(j+1,j+1) = c;
106     f(j) = c * phibar;
107     phibar = s * phibar;
108
109     if j < restart
110         if f(j) == 0 % stagnation of the method
111             stag = 1;
112         end
113         W(:,j) = (V(:,j) - W(:,1:j-1) * R(1:j-1,j))/ R(j,j);
114 % Check for stagnation of the method
115         if stag == 0
116             stagtest = zeros(n,1);
117             ind = (x ~= 0);
118             if ~(i==1 & j==1)
119                 stagtest(ind) = W(ind,j) ./ x(ind);
120                 stagtest(~ind & W(:,j) ~= 0) = Inf;
121                 if abs(f(j))*norm(stagtest,inf) < eps
122                     stag = 1;
123                 end
124             end
125         end
126         x = x + f(j) * W(:,j); % form the new inner iterate
127     else % j == restart
128         vrf = V(:,1:j)*(R(1:j,1:j)\f(1:j));
129 % Check for stagnation of the method
130         if stag == 0
131             stagtest = zeros(n,1);
132             ind = (x0 ~= 0);
133             stagtest(ind) = vrf(ind) ./ x0(ind);
134             stagtest(~ind & vrf ~= 0) = Inf;
135             if norm(stagtest,inf) < eps
136                 stag = 1;
137             end
138         end
139         x = x0 + vrf; % form the new outer iterate
140     end
141     normr = norm(b-feval(A,x));
142     resvec((i-1)*restart+j+1) = normr;
143
144     if normr <= tolb % check for convergence

```

```

145     if j < restart
146         y = R(1:j,1:j) \ f(1:j);
147         x = x0 + V(:,1:j) * y;      % more accurate computation of xj
148         r = b - feval(A,x);
149         normr = norm(r);
150         resvec((i-1)*restart+j+1) = normr;
151     end
152     if normr <= tolb                % check using more accurate xj
153         flag = 0;
154         iter = [i j];
155         break;
156     end
157 end
158
159 if stag == 1
160     flag = 3;
161     break;
162 end
163
164 if normr < normrmin                % update minimal norm quantities
165     normrmin = normr;
166     xmin = x;
167     imin = i;
168     jmin = j;
169 end
170 end                                % for j = 1 : restart
171
172 if flag == 1
173     x0 = x;                          % save for the next outer
174     iteration
175     r = b - feval(A,x0);
176 else
177     break;
178 end
179 end                                % for i = 1 : maxit
180
181 % returned solution is that with minimum residual
182
183 if n2b==0,
184     n2b=1;                          % here we solved for the zero solution and thus show
185 end;                                % the absolute residual !
186
187 if flag == 0
188     relres = normr / n2b;
189 else
190     x = xmin;
191     iter = [imin jmin];
192     relres = normrmin / n2b;
193 end

```

```

194
195 % truncate the zeros from resvec
196 if flag <= 1 | flag == 3
197     resvec = resvec(1:(i-1)*restart+j+1);
198 else
199     if j == 0
200         resvec = resvec(1:(i-1)*restart+1);
201     else
202         resvec = resvec(1:(i-1)*restart+j);
203     end
204 end
205
206
207 % only display a message if the output flag is not used
208 switch(flag)
209     case 0,
210         os = sprintf(['GMRES(%d) converged at iteration %d(%d) to a'
211             ...
212             ' solution with relative residual %0.2g'], ...
213             restart,iter(1),iter(2),relres);
214     case 1,
215         os = sprintf(['GMRES(%d) stopped after the maximum %d
216             iterations' ...
217             ' without converging to the desired tolerance
218             %0.2g' ...
219             '\n          The iterate returned (number %d(%d))'
220             ...
221             ' has relative residual %0.2g'], ...
222             restart,maxit,tol,iter(1),iter(2),relres);
223     case 2,
224         os = sprintf(['GMRES(%d) stopped at iteration %d(%d)' ...
225             ' without converging to the desired tolerance
226             %0.2g' ...
227             '\n          because the system involving the' ...
228             ' preconditioner was ill conditioned.' ...
229             '\n          The iterate returned (number %d(%d))'
230             ...
231             ' has relative residual %0.2g'], ...
232             restart,i,j,tol,iter(1),iter(2),relres);
233     case 3,
234         os = sprintf(['GMRES(%d) stopped at iteration %d(%d)' ...
235             ' without converging to the\n          desired'
236             ...
237             ' tolerance %0.2g because the method stagnated.'
238             ...
239             '\n          The iterate returned (number %d(%d))'
240             ...

```



```

235         ' has relative residual %0.2g'], ...
236         restart,i,j,tol,iter(1),iter(2),relres);
237
238     end                                     % switch(flag)
239     if flag == 0
240         disp(os);
241     else
242         warning(os);
243     end
244
245     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, \dots, x_M .

If r_M is small enough, stop,

else restart with $x_0 = x_M$.

Chapitre 3

Preconditioning

Contents

3.1 Introduction	43
3.2 Deflation method for GMRES	47
3.2.1 Building the preconditioner	48
3.2.2 Computing the invariant subspace	48
3.2.3 Numerical results	49
3.3 Fast methods using Fast Fourier Transform	50
3.3.1 Presentation of the method	51
3.3.2 Discrete and Fast Fourier Transform	55
3.3.3 The algorithm	59

3.1 Introduction

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}

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

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 algorithm 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 $\boxed{\beta_{m+1} = \frac{(z^{m+1}, r^{m+1})}{(z^m, r^m)}}$.

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

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

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

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

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

The algorithm for A

Define $C = M^2$.

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

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

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

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 Cholewsky 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

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

$$1 \leq i \leq M, 1 \leq j \leq M$$

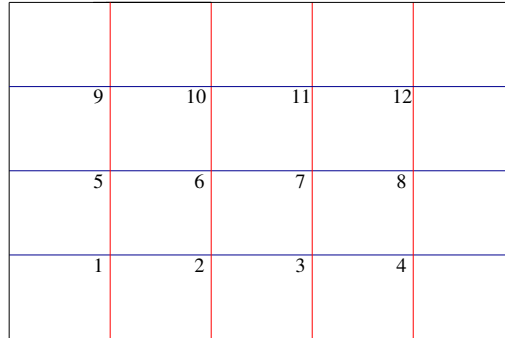


FIGURE 3.1 – Numbering by line

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

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

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 :

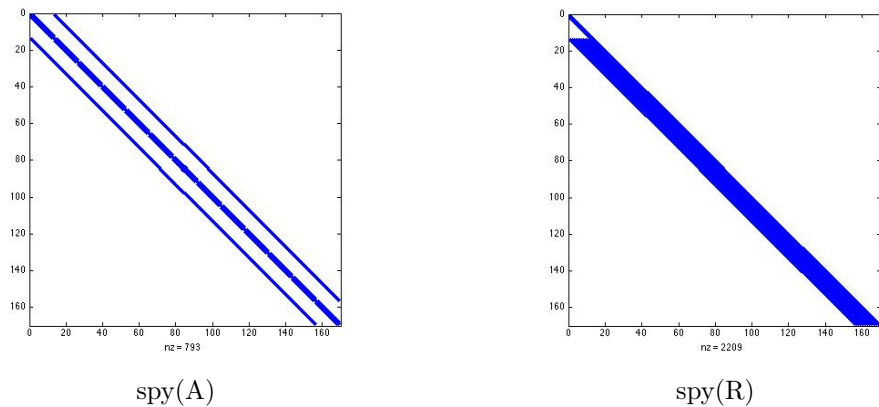
$$A = \frac{1}{h^2} \begin{pmatrix} B & -C & & 0_M \\ -C & B & -C & \\ & \ddots & \ddots & \ddots \\ & & 0_M & -C & B & -C \\ & & & -C & B \end{pmatrix} \quad (3.1)$$

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

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 $M = 15$.



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.

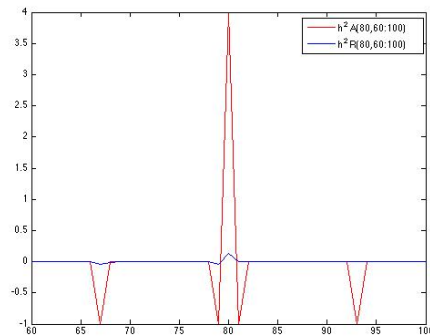


FIGURE 3.2 – Variation of the coefficients of Cholewski in the line 80 for $M = 15$

The matlab codes are as follows ([3])

```

1 Ch=tril(A);
2 for k=1:nn
3     Ch(k,k)=sqrt(Ch(k,k));
4     Ch(k+1:nn,k)=Ch(k+1:nn,k)/Ch(k,
5         k);
Cholewski   for j=k+1:nn
6         Ch(j:nn,j)=Ch(j:nn,j)-Ch(j:
7             nn,k)*Ch(j,k);
8     end
end

```

```

1 ChI=tril(A);
2 for k=1:nn
3     ChI(k,k)=sqrt(ChI(k,k));
4     for j=k+1:nn
5         if ChI(j,k) ~= 0
6             ChI(j,k)=ChI(j,k)/ChI(k
7                 ,k);
8         end
9     end
10    for j=k+1:nn
11        for i=j:n
12            if ChI(i,j) ~= 0
13                ChI(i,j)=ChI(i,j)-
14                    ChI(i,k)*ChI(j,k
15                        );
16            end
17        end
18    end
19 end

```

Incomplete Cholewski

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.

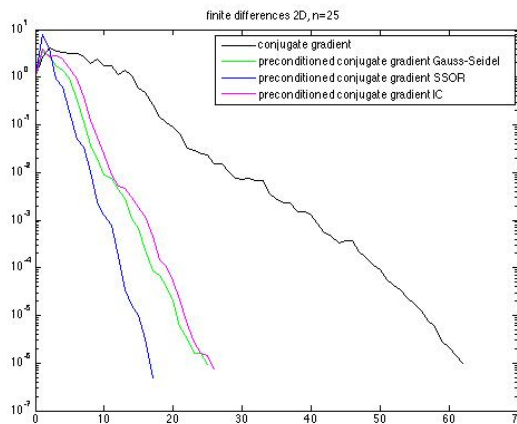


FIGURE 3.3 – Convergence history, influence of preconditioning

3.2 Deflation method for GMRES

Contents

3.1	Introduction	43
3.2	Deflation method for GMRES	47
3.2.1	Building the preconditioner	48
3.2.2	Computing the invariant subspace	48
3.2.3	Numerical results	49

3.3	Fast methods using Fast Fourier Transform	50
3.3.1	Presentation of the method	51
3.3.2	Discrete and Fast Fourier Transform	55
3.3.3	The algorithm	59

Recall the restarted GMRES algorithm to solve $Ax = b$:

Algorithm GMRES(m)

Choose x_0 ;

1. $r_0 = b - Ax_0$, $\beta = \|r_0\|$, $v_1 := r_0/\beta$;
2. Generate the Arnoldi basis applied to A and the associated Hessenberg matrix \tilde{H}_m starting with v_1 ;
3. Compute y_m which minimises $\|\beta e_1 - \tilde{H}_m y\|$ and $x_m = x_0 + V_m y_m$;
4. If convergence Stop, else set $x_0 = x_m$ and Go To 1 ;

Here we choose a right preconditioning M in order to guarantee a non increasing residual. This would not be true with a left preconditioner since the residual is multiplied by M^{-1}

This preconditioner can change at each restart. The algorithm becomes

Algorithm PRECGMRES(m)

Choose x_0 ;

Choose M ;

1. $r_0 = b - Ax_0$, $\beta = \|r_0\|$, $v_1 := r_0/\beta$;
2. Generate the Arnoldi basis applied to AM^{-1} and the associated Hessenberg matrix \tilde{H}_m starting with v_1 ;
3. Compute y_m which minimises $\|\beta e_1 - \tilde{H}_m y\|$ and $x_m = x_0 + M^{-1}V_m y_m$;
4. If convergence Stop, else set $x_0 = x_m$ update M and Go To 1 ;

The objective of deflation is to remove the smallest eigenvalues of A which slow down the GMRES convergence. With a restarted GMRES, information on these eigenvalues is lost which explains why restarted GMRES can be quite slow and even fail to converge. Deflation aims to replace them by real positive eigenvalues equal to the largest modulus of the eigenvalues.

3.2.1 Building the preconditioner

In the following we assume that A is diagonalizable in \mathbb{C} with eigenvalues $|\lambda_1| \leq |\lambda_2| \leq \dots \leq |\lambda_n|$.

Let P be an invariant subspace of dimension r corresponding to the r smallest eigenvalues of A and U an orthonormal basis of P . The deflating preconditioner is based on the idea that the linear system is solved exactly in space P .

Theorem 3.1 *if $T = U^T A U$ and $M = I_n + U(1/|\lambda_n|T - I_r)U^T$ then M is non singular and $M^{-1} = I_n + U(|\lambda_n|T^{-1} - I_r)U^T$ and the eigenvalues of AM^{-1} are $\lambda_{r+1}, \lambda_{r+2}, \dots, \lambda_r, |\lambda_n|$, and $|\lambda_n|$ is an eigenvalue of multiplicity at least r .*

Note : If only a close approximation \tilde{P} is known , an improved convergence rate is still to be obtained.

3.2.2 Computing the invariant subspace

The GMRES algorithm provides the Hessenberg matrix $H_k = V_k^T A V_k$, which is the restriction of A onto the Krylov subspace $K(k, A, r_0)$. The eigenvalues of H_k are called Ritz values. Let $H_k = S R S^T$ be the Schur canonical form of H_k with the eigenvalues ordered by increasing values. Then vectors $U = V_k S$ approximate the Schur vectors of

A. The largest Ritz value approximates the largest eigenvalue of A thus providing a first approximation of M .

After each restart new Ritz values can be estimated approximating eigenvalues of AM^{-1} also approximating remaining eigenvalues of A. By increasing the invariant subspace at each restart , a more powerful preconditioner is thus built. To avoid loss of orthogonality , these vectors are orthogonalized against the previous basis U .

Note : In some sense this algorithm recovers the superlinear convergence of the full GMRES version which behaves as if the smallest eigenvalues were removed. The preconditioner keeps the information on the smallest Ritz values which would be lost by restarting.

Algorithm DEFLGMRES(m)

Choose x_0 ;

$M = I_n$;

$U =$;

1. $r_0 = b - Ax_0$, $\beta = \|r_0\|$, $v_1 := r_0/\beta$;

2. Generate the Arnoldi basis applied to AM^{-1} and the associated Hessenberg matrix \tilde{H}_m starting with v_1 ;

3. Compute y_m which minimises $\|\beta e_1 - \tilde{H}_m y\|$ and $x_m = x_0 + M^{-1}V_m y_m$;

4. If convergence Stop, else set ;

$x_0 = x_m$;

Compute l Schur vectors of H_m noted S_l ;

Compute the approximation of $|\lambda_n|$;

Orthogonalize $V_m S_l$ against U ;

Increase U with $V_m S_l$;

$T = U^T A U$;

$M^{-1} = I_n + U(|\lambda_n|T^{-1} - I_r)U^T$;

Go To 1 ;

3.2.3 Numerical results

Results on two matrices of dimension 100 are given . A has the form $A = SDS^{-1}$ with $S = (1, \beta)$ an upper bidiagonal matrix.

Case 1 : $\beta = 0.9$ and $D = \text{diag}(1, 2, \dots, 100)$

Case 2 : $\beta = 0.9$ and $D = \text{diag}(1, 100, 200, \dots, 10000)$

DEFLGMRES(10,1) is compared with GMRES(10) and full GMRES . Tolerance is set to 10^{-8}

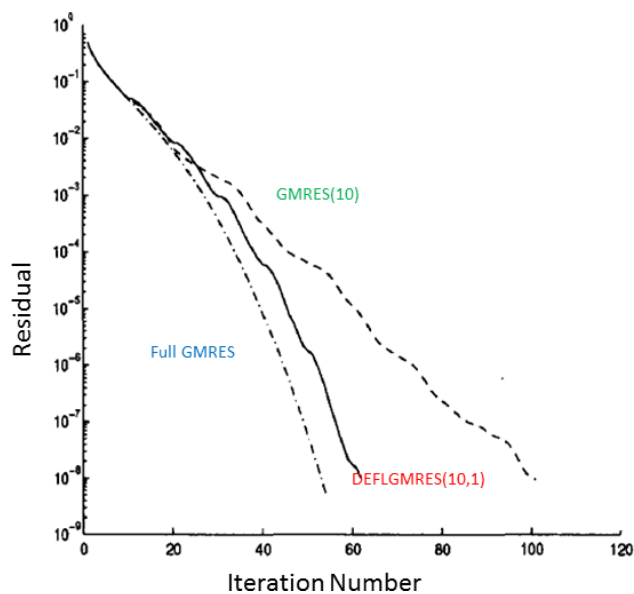


FIGURE 3.4 – Convergence history, Case 1

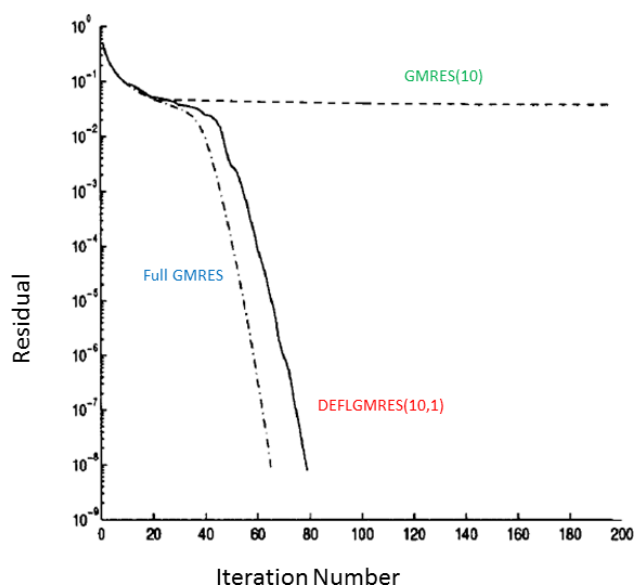


FIGURE 3.5 – Case 2

3.3 Fast methods using Fast Fourier Transform

Contents

3.1 Introduction	43
3.2 Deflation method for GMRES	47
3.2.1 Building the preconditioner	48
3.2.2 Computing the invariant subspace	48
3.2.3 Numerical results	49
3.3 Fast methods using Fast Fourier Transform	50

3.3.1	Presentation of the method	51
3.3.2	Discrete and Fast Fourier Transform	55
3.3.3	The algorithm	59

3.3.1 Presentation of the method

We'll work with the finite difference approximation of the Laplace equation in dimension 2.

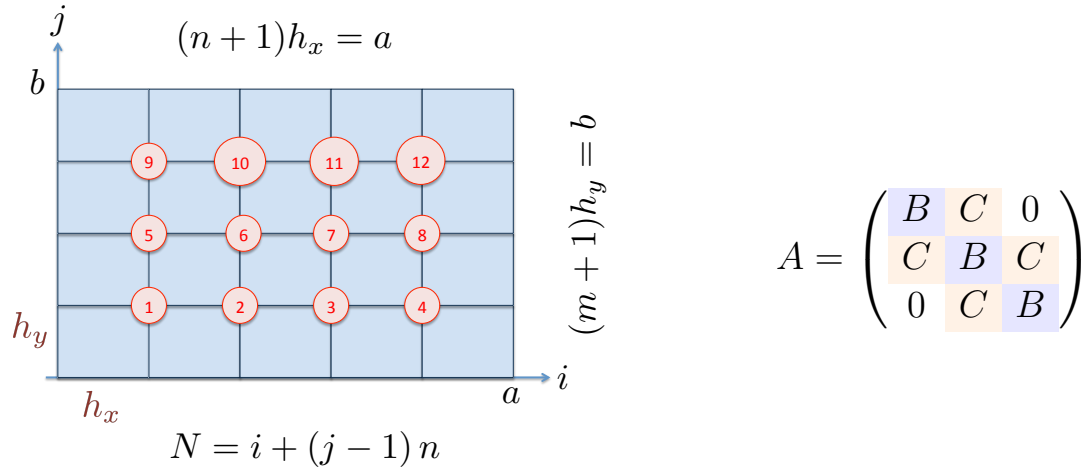


FIGURE 3.6 – Pavage de $[0, a] \times [0, b]$, $n = 4$ and $m = 3$

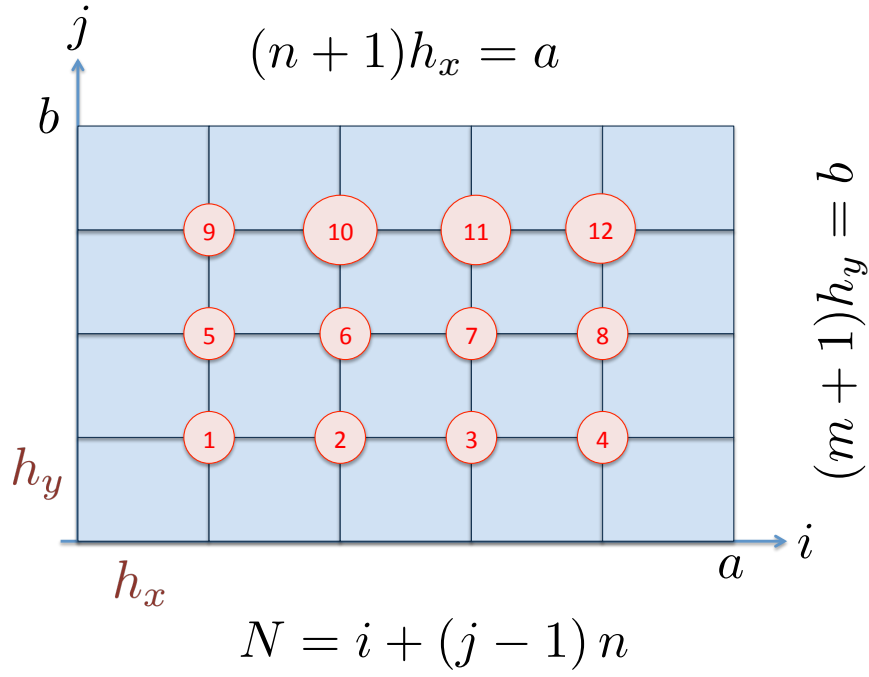


FIGURE 3.7 – Pavage de $[0, a] \times [0, b]$, $n = 4$ and $m = 3$

$\frac{2}{h_x^2} + \frac{2}{h_y^2}$	$-\frac{1}{h_x^2}$	0	0	$-\frac{1}{h_y^2}$	0	0	0	0	0	0	0	0	0
$-\frac{1}{h_x^2}$	$\frac{2}{h_x^2} + \frac{2}{h_y^2}$	$-\frac{1}{h_x^2}$	0	0	$-\frac{1}{h_y^2}$	0	0	0	0	0	0	0	0
0	$-\frac{1}{h_x^2}$	$\frac{2}{h_x^2} + \frac{2}{h_y^2}$	$-\frac{1}{h_x^2}$	0	0	$-\frac{1}{h_y^2}$	0	0	0	0	0	0	0
0	0	$-\frac{1}{h_x^2}$	$\frac{2}{h_x^2} + \frac{2}{h_y^2}$	0	0	0	$-\frac{1}{h_y^2}$	0	0	0	0	0	0
$-\frac{1}{h_y^2}$	0	0	0	$\frac{2}{h_x^2} + \frac{2}{h_y^2}$	$-\frac{1}{h_x^2}$	0	0	$-\frac{1}{h_x^2}$	0	0	0	0	0
0	$-\frac{1}{h_y^2}$	0	0	$-\frac{1}{h_x^2}$	$\frac{2}{h_x^2} + \frac{2}{h_y^2}$	$-\frac{1}{h_x^2}$	0	0	$-\frac{1}{h_x^2}$	0	0	0	0
0	0	$-\frac{1}{h_y^2}$	0	0	$-\frac{1}{h_x^2}$	$\frac{2}{h_x^2} + \frac{2}{h_y^2}$	$-\frac{1}{h_x^2}$	0	0	$-\frac{1}{h_x^2}$	0	0	$-\frac{1}{h_y^2}$
0	0	0	$-\frac{1}{h_y^2}$	0	0	$-\frac{1}{h_x^2}$	$\frac{2}{h_x^2} + \frac{2}{h_y^2}$	0	0	0	0	0	0
0	0	0	0	$-\frac{1}{h_y^2}$	0	0	0	$\frac{2}{h_x^2} + \frac{2}{h_y^2}$	$-\frac{1}{h_x^2}$	0	0	0	0
0	0	0	0	0	$-\frac{1}{h_y^2}$	0	0	$-\frac{1}{h_x^2}$	$\frac{2}{h_x^2} + \frac{2}{h_y^2}$	$-\frac{1}{h_x^2}$	0	0	0
0	0	0	0	0	0	$-\frac{1}{h_y^2}$	0	0	$-\frac{1}{h_x^2}$	$\frac{2}{h_x^2} + \frac{2}{h_y^2}$	$-\frac{1}{h_x^2}$	0	0
0	0	0	0	0	0	0	$-\frac{1}{h_y^2}$	0	0	$-\frac{1}{h_x^2}$	$\frac{2}{h_x^2} + \frac{2}{h_y^2}$	$-\frac{1}{h_x^2}$	0

$$A = \begin{pmatrix} B & C & 0 \\ C & B & C \\ 0 & C & B \end{pmatrix}$$

$$B = \begin{pmatrix} \frac{2}{h_x^2} + \frac{2}{h_y^2} & -\frac{1}{h_x^2} & 0 & 0 \\ -\frac{1}{h_x^2} & \frac{2}{h_x^2} + \frac{2}{h_y^2} & -\frac{1}{h_x^2} & 0 \\ 0 & -\frac{1}{h_x^2} & \frac{2}{h_x^2} + \frac{2}{h_y^2} & -\frac{1}{h_x^2} \\ 0 & 0 & -\frac{1}{h_x^2} & \frac{2}{h_x^2} + \frac{2}{h_y^2} \end{pmatrix} = A_1(h_x) + \frac{2}{h_y^2} I_n$$

$$C = - \begin{pmatrix} \frac{1}{h_y^2} & 0 & 0 & 0 \\ 0 & \frac{1}{h_y^2} & 0 & 0 \\ 0 & 0 & \frac{1}{h_y^2} & 0 \\ 0 & 0 & 0 & \frac{1}{h_y^2} \end{pmatrix} = -\frac{1}{h_y^2} I_n.$$

Consider now the general problem $Ax = b$, where A is a $nm \times nm$ symmetric matrix A , block tridiagonal in the form

$$A = A(B, C) = \begin{pmatrix} B & C & & 0 \\ C & B & C & \\ & \ddots & \ddots & \ddots \\ & & C & B & C \\ 0 & & & C & B \end{pmatrix}. \quad (3.2)$$

Each block is a $n \times n$ matrix. The vectors \mathbf{b} and \mathbf{x} can be split by block of size n as well, x^j is the sought solution on the ligne j .

$$\mathbf{b} = \begin{pmatrix} \mathbf{b}^1 \\ \vdots \\ \mathbf{b}^m \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} \mathbf{x}^1 \\ \vdots \\ \mathbf{x}^m \end{pmatrix}$$

The system can be rewritten as

$$\begin{pmatrix} B & C & & 0 \\ C & B & C & \\ & \ddots & \ddots & \ddots \\ & & C & B & C \\ 0 & & & C & B \end{pmatrix} \begin{pmatrix} \mathbf{x}^1 \\ \mathbf{x}^2 \\ \vdots \\ \mathbf{x}^{m-1} \\ \mathbf{x}^m \end{pmatrix} = \begin{pmatrix} \mathbf{b}^1 \\ \mathbf{b}^2 \\ \vdots \\ \mathbf{b}^{m-1} \\ \mathbf{b}^m \end{pmatrix}$$

which is a system of m systems of dimension n :

$$\begin{aligned} B\mathbf{x}^1 + C\mathbf{x}^2 &= \mathbf{b}^1 \\ \vdots & \\ C\mathbf{x}^{i-1} + B\mathbf{x}^i + C\mathbf{x}^{i+1} &= \mathbf{b}^i \\ \vdots & \\ C\mathbf{x}^{m-1} + B\mathbf{x}^m &= \mathbf{b}^m \end{aligned}$$

Suppose B and C are symmetric, and **diagonalise in the same orthonormal basis** $(\mathbf{q}^1, \dots, \mathbf{q}^n)$. This is the case for our previous example. Denote by Q the corresponding orthogonal matrix $Q = [\mathbf{q}^1, \dots, \mathbf{q}^n]$. There exist two diagonal matrices D^1 and D^2 such that

$$B = QD^1Q^T, \quad C = QD^2Q^T.$$

Take for example the first equation

$$B\mathbf{x}^1 + C\mathbf{x}^2 = \mathbf{b}^1$$

and replace B and C :

$$QD^1Q^T\mathbf{x}^1 + QD^2Q^T\mathbf{x}^2 = \mathbf{b}^1$$

Multiply by Q^T :

$$D^1Q^T\mathbf{x}^1 + D^2Q^T\mathbf{x}^2 = Q^T\mathbf{b}^1$$

Denote by $(\mathbf{c}^i, \mathbf{y}^i)$ the coordinates of $(\mathbf{b}^i, \mathbf{x}^i)$ in the new basis :

$$Q^T\mathbf{b}^i = \mathbf{c}^i, \quad Q^T\mathbf{x}^i = \mathbf{y}^i, \quad 1 \leq i \leq m.$$

Then the problem takes the form

$$\begin{aligned} D^1\mathbf{y}^1 + D^2\mathbf{y}^2 &= \mathbf{c}^1 \\ \vdots & \\ D^2\mathbf{y}^{i-1} + D^1\mathbf{y}^i + D^2\mathbf{y}^{i+1} &= \mathbf{c}^i \\ \vdots & \\ D^2\mathbf{y}^{m-1} + D^1\mathbf{y}^m &= \mathbf{c}^m \end{aligned}$$

These are all diagonal systems. Take the component number j in each block of the previous system, for $1 \leq j \leq n$:

$$\begin{aligned} D_j^1 y_j^1 + D_j^2 y_j^2 &= c_j^1 \\ \vdots &= \\ D_j^2 y_j^{i-1} + D_j^1 y_j^i + D_j^2 y_j^{i+1} &= c_j^i \\ \vdots & \\ D_j^2 y_j^{m-1} + D_j^1 y_j^m &= c_j^m \end{aligned}$$

which is written in matrix form as

$$\begin{pmatrix} D_j^1 & D_j^2 & & 0 \\ D_j^2 & D_j^1 & D_j^2 & \\ & \ddots & \ddots & \ddots \\ & & D_j^2 & D_j^1 & D_j^2 \\ 0 & & & D_j^2 & D_j^1 \end{pmatrix} \begin{pmatrix} y_j^1 \\ y_j^2 \\ \vdots \\ y_j^{m-1} \\ y_j^m \end{pmatrix} = \begin{pmatrix} c_j^1 \\ c_j^2 \\ \vdots \\ c_j^{m-1} \\ c_j^m \end{pmatrix}$$

For each j , $1 \leq j \leq n$, define the tridiagonal $m \times m$ matrix

$$T_j = \begin{pmatrix} D_j^1 & D_j^2 & & 0 \\ D_j^2 & D_j^1 & D_j^2 & \\ & \ddots & \ddots & \ddots \\ & & D_j^2 & D_j^1 & D_j^2 \\ & 0 & & D_j^2 & D_j^1 \end{pmatrix}$$

and 2 vectors in \mathbb{R}^m

$$\mathbf{d}^j = \begin{pmatrix} c_j^1 \\ \vdots \\ c_j^m \end{pmatrix}, \quad \mathbf{z}^j = \begin{pmatrix} y_j^1 \\ \vdots \\ y_j^m \end{pmatrix}$$

We have now n tridiagonal systems of size m ,

$$T_j \mathbf{z}^j = \mathbf{d}^j, \quad 1 \leq j \leq n.$$

which can be solved in parallel with a LU decomposition for instance. For the 2D Laplace equation with equidistant grid, the computation of the c^j and the reconstruction of x can be done by Fast Fourier transform.

We have to compute for each j , $\mathbf{x}^j = Q\mathbf{y}^j$. The matrice C is $-\frac{1}{h_y^2}I_n$, the matrix B is $A_1(h_x) + \frac{2}{h_y^2}I_n$. The eigenvalues of B are those of $A_1 + \frac{2}{h_y^2}$, which are $\frac{2}{h_y^2} + \frac{4}{h_x^2} \sin^2 \frac{k\pi h_x}{2}$, the eigenvectors of B and C are those of A_1 , given by (after orthonormalisation)

$$\Phi_j^{(k)} = \sqrt{\frac{2}{n+1}} \sin \frac{jk\pi}{n+1}, \quad 1 \leq j \leq n,$$

Define the matrix Q as the matrix of eigenvectors

$$Q = [\Phi^{(1)}, \dots, \Phi^{(n)}].$$

By

$$Q\mathbf{v} = \sum_{k=1}^n v_k \Phi^{(k)},$$

we obtain

$$(Q\mathbf{v})_j = (Q^T \mathbf{v})_j = \sqrt{\frac{2}{n+1}} \sum_{k=1}^n v_k \sin \frac{kj\pi}{n+1}.$$

Note that the sum can be extended to $k = 0$ and $k = n + 1$ since the sinus vanishes.

$$(Q\mathbf{v})_j = (Q^T \mathbf{v})_j = \sqrt{\frac{2}{n+1}} \sum_{k=1}^{n+1} v_k \sin \frac{kj\pi}{n+1}. \quad (3.3)$$

The next section is occupied with the FFT, we'll come back to the algorithm later.

3.3.2 Discrete and Fast Fourier Transform

Let $n' = n + 1$. The Discrete Fourier Transform of length n' is defined by

$$w_j = \sum_{k=1}^{n'} v_k e^{-2i \frac{kj\pi}{n'}}, \quad j = 1, \dots, n'.$$

Define $r = e^{2i\frac{\pi}{n'}}$ the basic root of unity, then we rewrite the formula above as

$$w_j = \sum_{k=1}^{n'} v_k r^{-kj}, \quad j = 1, \dots, n'. \quad (3.4)$$

Lemma 3.1 (Inverse DFT) *If $w = (w_j)_{1 \leq j \leq n'}$ is the discrete Fourier transform of $v = (v_j)_{1 \leq j \leq n'}$ from (3.4), then the inverse discrete Fourier transform is given by*

$$v_j = \frac{1}{n'} \sum_{k=1}^{n'} w_k r^{kj}, \quad j = 1, \dots, n'. \quad (3.5)$$

Proof Just replace

$$\sum_{k=1}^{n'} \left(\frac{1}{n'} \sum_{p=1}^{n'} w_p r^{kp} \right) r^{-kj} = \frac{1}{n'} \sum_{p=1}^{n'} w_p \sum_{k=1}^{n'} r^{k(p-j)}$$

Since $z = r^{p-j}$ is a n' -root of unity,

$$\begin{cases} \text{for } z \neq 1, & \sum_{k=1}^{n'} z^k = 0, \\ \text{for } z = 1, & \sum_{k=1}^{n'} z^k = n'. \end{cases}$$

Therefore

$$\frac{1}{n'} \sum_{p=1}^{n'} w_p \sum_{k=1}^{n'} r^{k(p-j)} = w_j$$

and the lemma is proven. ■

We now suppose that $n' = 2p$. We need to specify more r , that we call $r_{n'}$. Note for further use that $r_{n'}^{n'} = 1$ and $r_{n'}^p = -1$. Split the sum above into even ($k = 2\ell, \ell = 1 : p$) and odd terms ($k = 2\ell - 1, \ell = 1 : p$). For $j = 1, \dots, 2p$,

$$\begin{aligned} w_j &= \sum_{k=1}^{n'} v_k r_{n'}^{-kj} \\ w_j &= \sum_{\ell=1}^p v_{2\ell} r_{n'}^{-2\ell j} + \sum_{\ell=1}^p v_{2\ell-1} r_{n'}^{-(2\ell-1)j} \\ &= \sum_{\ell=1}^p v_{2\ell} r_{n'}^{-2\ell j} + r^j \sum_{\ell=1}^p v_{2\ell-1} r_{n'}^{-2\ell j}. \end{aligned}$$

Defining for $j = 1, \dots, 2p$,

$$u_j = \sum_{\ell=1}^p v_{2\ell} r_{n'}^{-2\ell j}, \quad t_j = \sum_{\ell=1}^p v_{2\ell-1} r_{n'}^{-2\ell j}.$$

Then

$$w_j = u_j + r_{n'}^j t_j.$$

We verify that for each j , $u_{j+p} = u_j$ and $t_{j+p} = t_j$:

$$u_{j+p} = \sum_{\ell=1}^p v_{2\ell} r_{n'}^{-2\ell(j+p)} = r_{n'}^{-2\ell p} u_j = u_j.$$

This implies that we only need to compute (u_j, t_j) for $1 \leq j \leq p$. Furthermore

$$w_{j+p} = u_{j+p} + r_{n'}^{j+p} t_{j+p} = u_j + r_{n'}^j r_{n'}^p t_j = u_j - r_{n'}^j t_j.$$

To compute u_j and t_j note that

$$\sum_{\ell=1}^p v_{2\ell} r_{n'}^{-2\ell j} = \sum_{\ell=1}^p v_{2\ell} (r_{n'}^2)^{-\ell j}$$

But $r_{n'}^2 = (e^{-\frac{2i\pi}{2p}})^2 = e^{-\frac{2i\pi}{p}} : r_{n'}^2 = r_p$. Therefore

$$u_j = \sum_{\ell=1}^p v_{2\ell} r_p^{-\ell j}, \quad t_j = \sum_{\ell=1}^p v_{2\ell-1} r_p^{-\ell j}.$$

The sums above are similar sums as that defining w_j , but with $p = n'/2$. This is the starting point for a dyadic computation of the w_j : the Fast Fourier Transform.

To obtain $\{w_j\}_{1 \leq j \leq 2p}$ from $\{v_j\}_{1 \leq j \leq 2p}$, do

$$\text{Compute } r_{n'}^j, \quad j = 1, \dots, p$$

$$\text{Compute } u_j = \sum_{\ell=1}^p v_{2\ell} r_p^{-\ell j}, \quad t_j = \sum_{\ell=1}^p v_{2\ell-1} r_p^{-\ell j} \quad j = 1, \dots, p$$

$$\text{Compute } w_j = u_j + r_{n'}^j t_j, \quad w_{j+p} = u_j - r_{n'}^j t_j \quad j = 1, \dots, p.$$

$$r = e^{2i\frac{\pi}{n'}}, w_j = \sum_{k=1}^{n'} v_k r^{-kj}, \quad j = 1, \dots, n'.$$

$n' = 2, r = -1$, initialization $w_1 = -v_1 + v_2, \quad w_2 = v_1 + v_2$.

```

1 function w=myFFT(v)
2 % MYFFT fast Fourier transform
3 % w=myFFT(v); computes recursively the Fourier transform of
4 % the vector v whose length must be a power of 2.
5 n=length(v);
6 if n==2,
7     w=[-v(1)+v(2);v(1)+v(2)];
8 else
9     rp=exp(2i*pi/n*(1:n/2)');
10    t=myFFT(v(1:2:n-1));
11    u=myFFT(v(2:2:n));
12    w=[u+rp.*t; u-rp.*t];
13 end;
```

$$r = e^{2i\frac{\pi}{n'}}, w_j = \sum_{k=1}^{n'} v_k r^{-kj}, \quad j = 1, \dots, n'.$$

$n' = 2, r = -1$, initialization $w_1 = -v_1 + v_2, \quad w_2 = v_1 + v_2$.

```

1 function w=myFFT(v)
2 % MYFFT fast Fourier transform
3 % w=myFFT(v); computes recursively the Fourier transform of
4 % the vector v whose length must be a power of 2.
5 n=length(v);
6 if n==2,
7     w=[-v(1)+v(2);v(1)+v(2)];
8 else
9     rp=exp(2i*pi/n*(1:n/2)');
10    t=myFFT(v(1:2:n-1));
11    u=myFFT(v(2:2:n));
12    w=[u+rp.*t; u-rp.*t];
13 end;
```

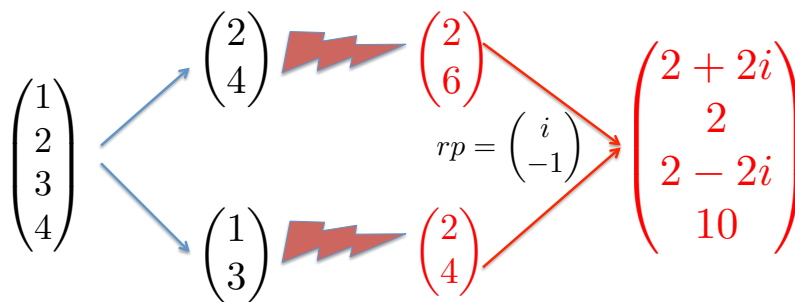


FIGURE 3.8 – FFT for $n' = 4$

It is easy to count the number of operations in the algorithm to be $\mathcal{O}(n \log_2(n))$, which is much better than *blockLU*.

3.3.3 The algorithm

We now show how to obtain the computation of Qv in (3.3) with FFT.

$$\begin{aligned} v &\in \mathbb{R}^n, \quad n' = n + 1 \text{ EVEN} \\ Qv &= \sqrt{\frac{2}{n+1}} \quad z \in \mathbb{R}^n, \quad z_j = \sum_{k=1}^n v_k \sin \frac{kj\pi}{n'} \quad 1 \leq j \leq n, \\ \tilde{v} &= [v; 0] \in \mathbb{R}^{n'}, \\ DFT(\tilde{v}) &= w \in \mathbb{R}^{n'}, \quad w_j = \sum_{k=1}^{n'} \tilde{v}_k e^{-2i \frac{kj\pi}{n'}} \quad 1 \leq j \leq n' \end{aligned}$$

Note first that $z_j = \sum_{k=1}^{n'} \tilde{v}_k \sin \frac{kj\pi}{n'}$ as well. Consider first the even indices z_2, \dots, z_{n-1} :

$$z_{2\ell} = \sum_{k=1}^{n'} \tilde{v}_k \sin \frac{2\ell k\pi}{n'} = -\text{Im} w_\ell, \quad \ell = 1, \dots, \frac{n-1}{2}.$$

Consider now the odd indices, z_1, \dots, z_n

$$\begin{aligned} z_{2\ell-1} &= -\text{Im} \sum_{k=1}^{n'} \tilde{v}_k e^{-i \frac{k(2\ell-1)\pi}{n'}} = -\text{Im} \sum_{k=1}^{n'} (\tilde{v}_k e^{i \frac{k\pi}{n'}}) e^{-2i \frac{k\ell\pi}{n'}} \\ &= -\text{Im}(DFT(\{\tilde{v}_k e^{i \frac{k\pi}{n'}}\}_k))_\ell, \quad \ell = 1, \dots, \frac{n+1}{2}. \end{aligned}$$

Resuming with matlab notations

QFFT

$$\begin{aligned} r_0 &= e^{i \frac{\pi}{n'}} \\ (Qv)_{2\ell} &= -\sqrt{\frac{2}{n+1}} \text{Im}(FFT(\tilde{v}))_\ell, \quad \ell = 1, \dots, \frac{n-1}{2} \\ (Qv)_{2\ell-1} &= -\sqrt{\frac{2}{n+1}} \text{Im}(FFT(\tilde{v} \cdot * r_0^{(1:n')'}))_\ell, \quad \ell = 1, \dots, \frac{n+1}{2} \end{aligned} \quad (3.6)$$

Summarizing the solution of

$$\begin{pmatrix} B & C & & 0 \\ C & B & C & \\ & \ddots & \ddots & \ddots \\ & & C & B & C \\ 0 & & & C & B \end{pmatrix} \begin{pmatrix} x^1 \\ x^2 \\ \vdots \\ x^{m-1} \\ x^m \end{pmatrix} = \begin{pmatrix} b^1 \\ b^2 \\ \vdots \\ b^{m-1} \\ b^m \end{pmatrix}$$

Step 1 : FFT Compute $c^j = Q^T b^j$ by (3.6) for $1 \leq j \leq m$.

Step 2 : Sort $\{c^1, \dots, c^m\}$ The righthand side has been build by rows in the mesh : b^j is the vector of the values of the forcing term on the line $y = j * h_y$.

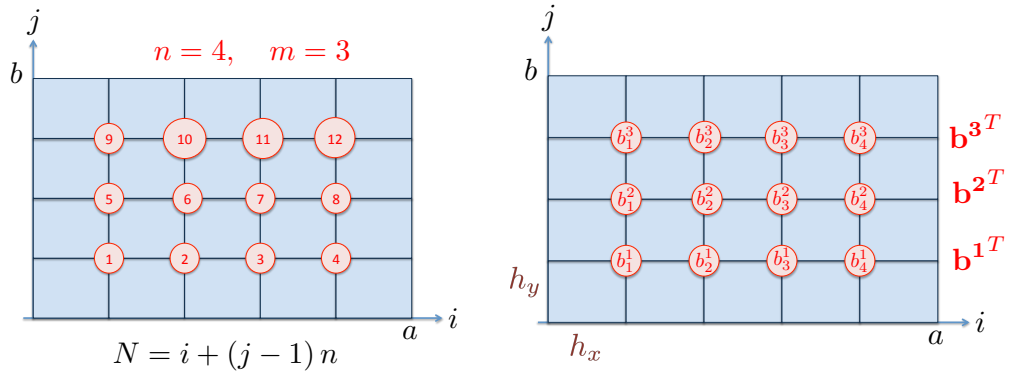


FIGURE 3.9 – Numbering

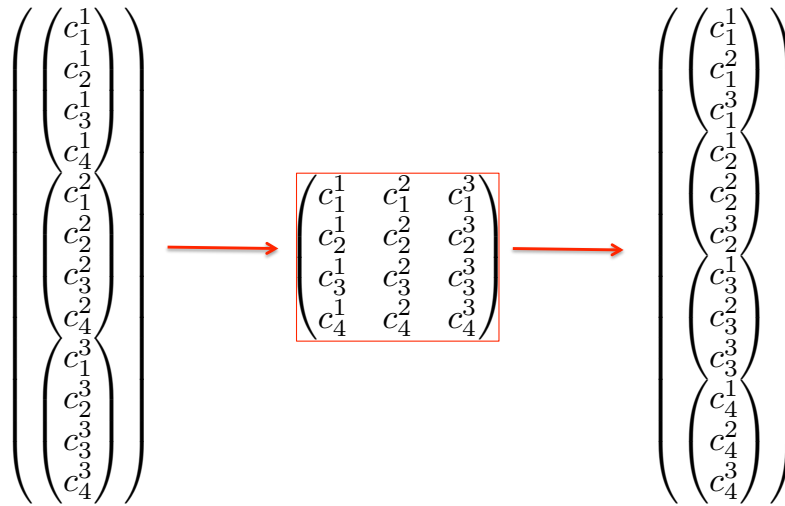


FIGURE 3.10 – Renumbering

The total vector σ is numbered from 1 to nm , with $N = i + (j - 1)n$. The matrix C is built as follows

$$\begin{aligned}
 \sigma(1 : n) &\rightarrow C(:, 1) \\
 \sigma(n + 1 : 2n) &\rightarrow C(:, 2) \\
 &\vdots \\
 \sigma((m - 1)n + 1 : mn) &\rightarrow C(:, m)
 \end{aligned}$$

```

1 for j=1:m
2   C(:,j)=sig((j-1)*n+1:j*n )
3 end

```

and then instead of reading the columns, we read the rows.

Step 3 : Solving the n tridiagonal systems of size m ,

$$T_j z^j = d^j, \quad 1 \leq j \leq n.$$

with $\mathbf{d}^j = C(j, :)$, and

$$T_j = \begin{pmatrix} D_j^1 & D_j^2 & & 0 \\ D_j^2 & D_j^1 & D_j^2 & \\ & \ddots & \ddots & \ddots \\ & & D_j^2 & D_j^1 & D_j^2 \\ & 0 & & D_j^2 & D_j^1 \end{pmatrix},$$

$$D_j^2 = -\frac{1}{h_y^2}, \quad D_j^1 = \frac{2}{h_y^2} + \frac{4}{h_x^2} \sin^2 \frac{j\pi h}{2(n+1)}.$$

Step 4 : Reordering the z^j into y^j

Step 5 : Recovering $x^j = Qy^j$ by (3.6).

For this method, we talk about FFT preconditioning, since the system $A\mathbf{u} = \mathbf{b}$ is premultiplied by the block-diagonal matrix

$$Q = \begin{pmatrix} Q^T & & & \\ & Q^T & 0 & \\ & & \ddots & \\ & 0 & & Q^T \end{pmatrix}$$

That is we write

$$QAQ^T Q\mathbf{u} = Q\mathbf{b}.$$

Bibliographie

- [1] Suzanne.C Brenner and Ridgway Scott. *The mathematical theory of finite element methods*. Springer-Verlag, 1994.
- [2] William L. Briggs and Van Emden Henson. A multigrid tutorial. www.llnl.gov/CASC/people/henson/mgtut/ps/mgtut.pdf.
- [3] Charles F Loan Gene H Golub. *Matrix computations*. Johns Hopkins University Press, 1996.
- [4] Wolfgang Hackbusch. *Multigrid methods and applications*. Springer-Verlag, 1985.
- [5] Hans Petter Langtangen and Aslak Tveitog. What is multigrid? folk.uio.no/infima/doc/mg-underscore-nmfpd.pdf.
- [6] Pierre-Arnaud Raviart and Jean-Marie Thomas. *Introduction à l'analyse numérique des équations aux dérivées partielles*. Masson, 1988.
- [7] P. Wesseling. *An introduction to multigrid methods*. Wiley -Interscience, 1992.