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

Chapitre 1

Classical methods

Contents

1.1 Direct methods

1.1.1 Gauss method

Example

$$\underbrace{\begin{pmatrix} 1 & 3 & 1 \\ 1 & 1 & -1 \\ 3 & 11 & 6 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 9 \\ 1 \\ 36 \end{pmatrix}}_{b}$$

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 :

$$M_1[A\,|\,b] = \begin{pmatrix} 1 & 3 & 1 & 9 \\ 0 & -2 & -2 & -8 \\ 0 & 2 & 3 & 9 \end{pmatrix}.$$

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

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

$$M_2\,M_1[A\,|\,b] = \begin{pmatrix} 1 & 3 & 1 & 9 \\ 0 & -2 & -2 & -8 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

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

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

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

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

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 $Ly = b$ $\quad \mathcal{O}(n^2)$ 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

```matlab
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(:);
```

```matlab
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 :

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

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

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

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

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

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

```matlab
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(:);
```

```matlab
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,

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

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

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   Stationary iterative methods

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

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

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

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 eigenvaues of A.

1.2.2 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

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

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, $\rho_m(R) \ge \rho(R)$. 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

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

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 $i = 1, \ldots, n$,*

2. *$a_{ij} \leq 0$ for $i \neq j$, $i, j = 1, \ldots, n$,*

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 $\rho(\mathcal{L}_{\omega^*}) = \omega^* - 1$.
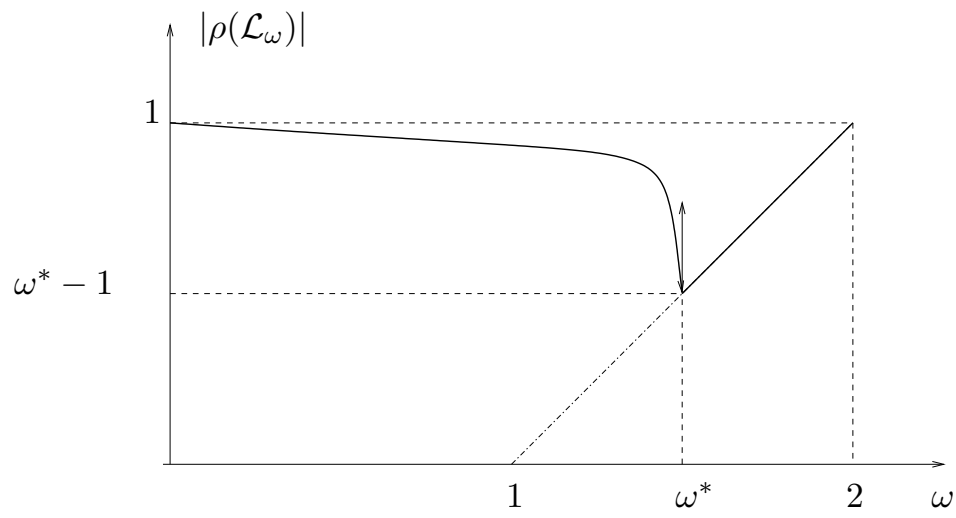
FIGURE 1.1 – Variations of $\rho(\mathcal{L}_\omega)$ as a fonction of ω

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.
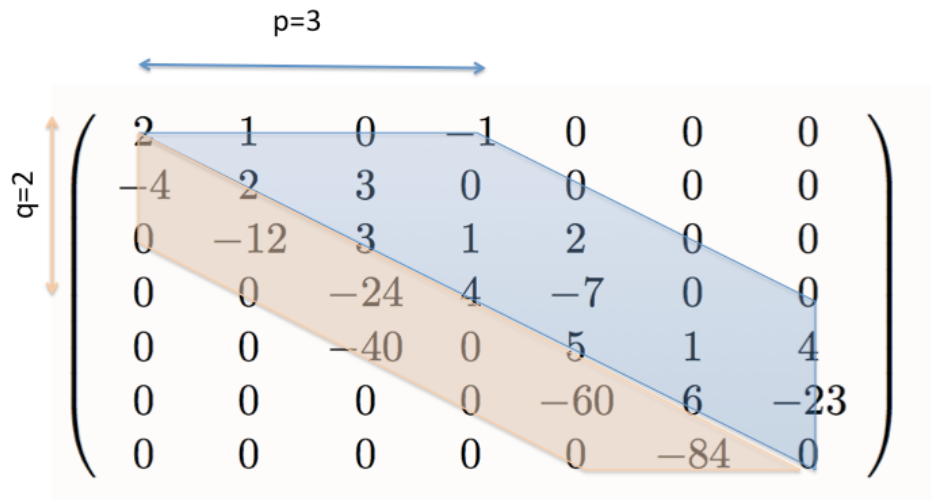

FIGURE 1.2 – A bandmatrix

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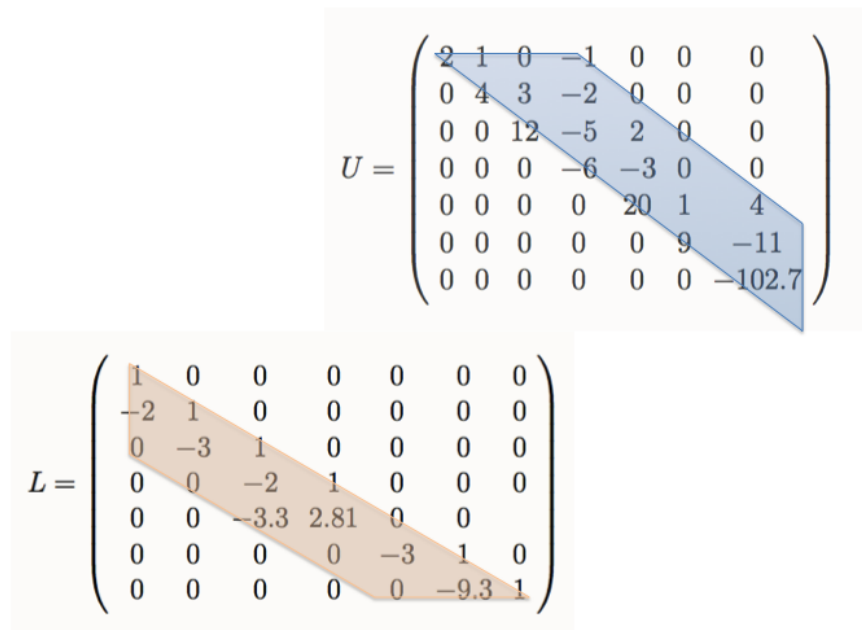

FIGURE 1.3 – LU decomposition

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 :
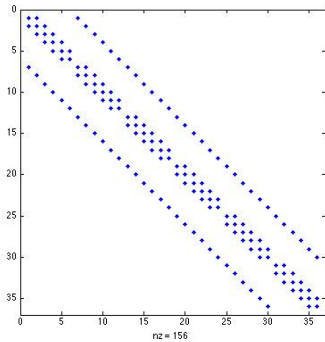
A bandmatrix sparse matrix
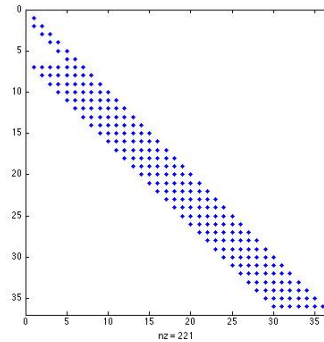
Corresponding Cholewski

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

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

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

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

```matlab
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.

```matlab
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

```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 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.

1.3.3   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(nxn), 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 (8x8 input matrix) into A2 :

$$
A1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \qquad A2 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}
$$

A1   A2

Notions of Graph

The graph G(A) corresponding to the matrix A we will have n nodes labelled i = 1,2, ... ,n. For each non-zero element aij, 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

Graph(A1)   Graph(A2)
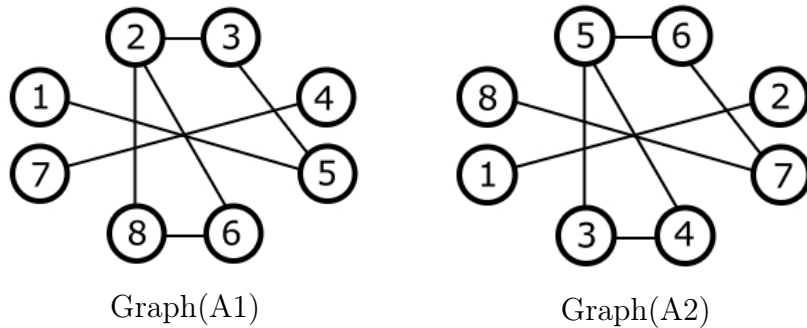
The two graph structures are identical, the only thing that is different is the node (vertex) labelling. In other words the bandwidth reduction problem can also be viewed as a graph labelling problem :

Find the node labelling that minimizes the bandwidth B of the adjacency matrix of the graph G(A) , where we can formally define : $B = \max |L_i - L_j|$, 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 swaped 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) + 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

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 : $\boxed{\|y\|_A^2 = (Ay, y).}$

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

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

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

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

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

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

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

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

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

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.

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

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

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

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

.

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

$$\textcolor{blue}{Au = b}$$

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 $(h \times (n+1) = 1)$.

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

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

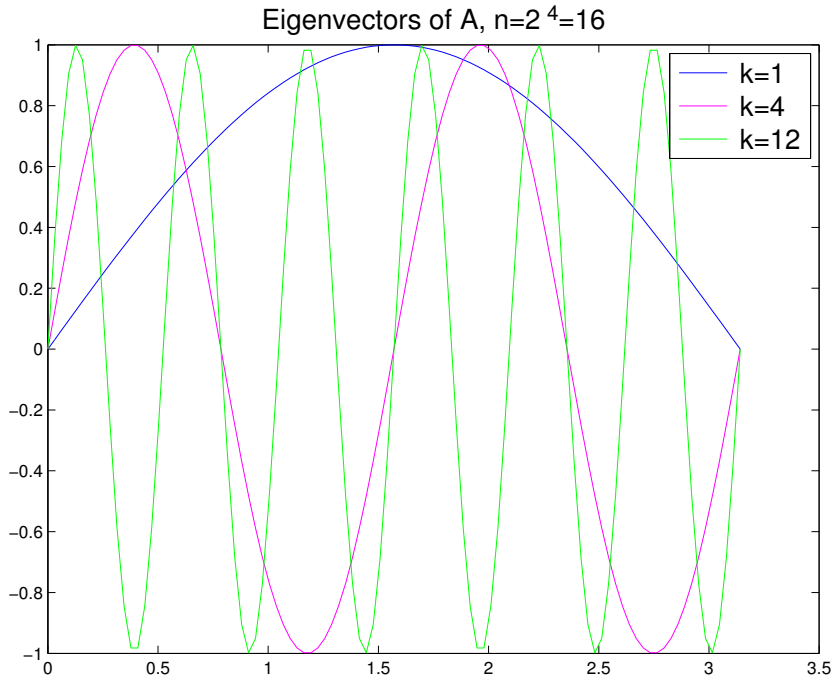For any iterative method, the eigenfunctions of the iteration matrix are equal to those of A.

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	$\dfrac{1 - \sin \pi h}{1 + \sin \pi h}$	0.26	0.74	0.9021
steepest descent	$\dfrac{K(A) - 1}{K(A) + 1} = \cos \pi h$	0.81	0.99	0.9987
conjugate gradient	$\dfrac{\sqrt{K(A)} - 1}{\sqrt{K(A)} + 1} = \dfrac{\cos \pi h - \sin \pi h}{\cos \pi h + \sin \pi h}$	0.51	0.86	0.9020

TABLE 2.2 – Convergence factor

Algorithm	convergence factor ρ_∞	convergence rate F
Jacobi	$1 - \dfrac{\varepsilon^2}{2}$	$\dfrac{\varepsilon^2}{2}$
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 $M \sim -\dfrac{\ln(10)}{F}$
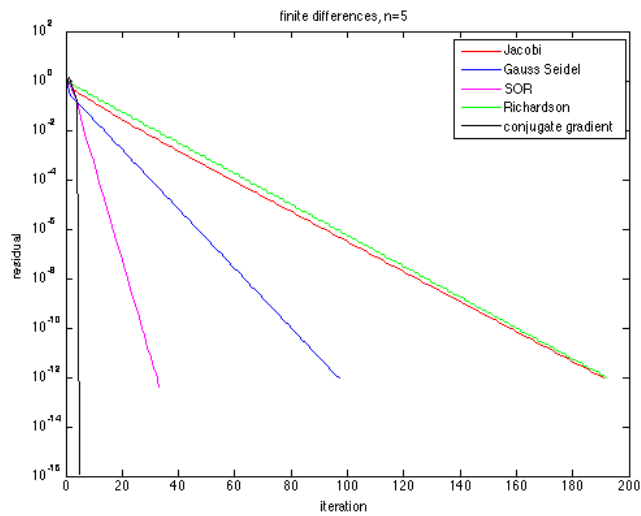
FIGURE 2.2 – Convergence history, $n = 5$

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
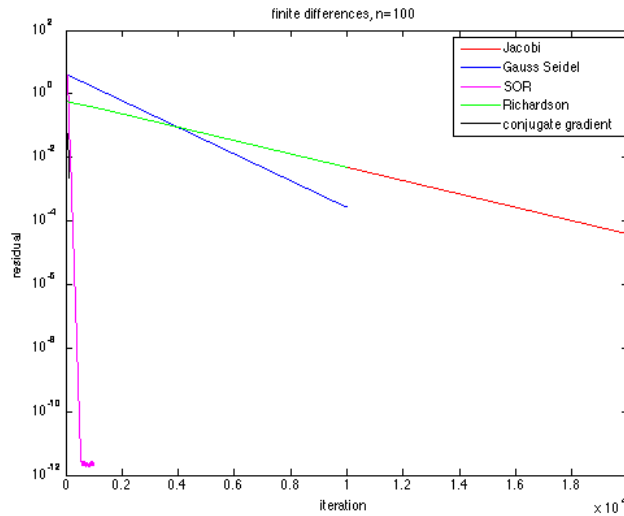

FIGURE 2.3 – Convergence history, $n = 100$

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

2.2 Krylov methods for non symmetric matrices, Arnoldi algorithm

2.2.1 Gram-Schmidt orthogonalization and QR decomposition

Starting with a free family $(v_1, \cdots, v_m, \cdots)$ in a vector space E with a scalar product $(\cdot, \cdot)$, 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 $(w_1, \cdots, w_{j-1})$ an orthonormal basis of $\mathcal{L}(v_1, \cdots, v_{j-1})$. 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 $(w_1, \cdots, w_j)$ 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 ;

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

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

$$K_{k,j} = \sum_{i=1}^{j} r_{i,j} Q_{k,i} \qquad K = QR \tag{2.2}$$

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 $\mathcal{K}_m = vect(r, Ar, \cdots, A^{m-1}r)$ 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 $q_1 = \dfrac{r}{\|r\|}$.

• 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 of $\mathcal{K}_2$. All this can be rewritten as

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

Then $\mathcal{K}_3 = Vec(q_1, q_2, A^2 r) = Vec(q_1, q_2, Aq_2)$. Therefore, instead of orthonormalizing with the new vector $A^2 r$, 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 2.5 *If the algorithm goes through m, then $(q_1, \ldots, q_m)$ 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

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

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

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

Proposition 2.1

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

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

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

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

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

∎

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

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

The small Hessenberg system

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

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=QHQ?, 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 $J(y) = \|r_0 - AV_m y\|_2$ 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

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

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
```

```matlab
 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
```

```matlab
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    QT = zeros(restart+1,restart+1); % orthogonal factor st QT*H = R
59    R = zeros(restart,restart);       % upper triangular factor st H = Q
          *R
60    f = zeros(restart,1);             % y = R\f => x = x0 + V*y
61    W = zeros(n,restart);             % W = V*inv(R)
62    j = 0;                            % inner iteration counter
63
64    vh = r;
65    h(1) = norm(vh);
66    V(:,1) = vh / h(1);
67    QT(1,1) = 1;
68    phibar = h(1);
69
70    for j = 1 : restart
71      j
72  %   MapU(x,sqrt(n),sqrt(n));
73
74      u = feval(A,V(:,j));            % matrix multiply
75      for k = 1 : j
76        h(k) = V(:,k)' * u;
77        u = u − h(k) * V(:,k);
78      end
79      h(j+1) = norm(u);
80      V(:,j+1) = u / h(j+1);
81      R(1:j,j) = QT(1:j,1:j) * h(1:j);
82      rt = R(j,j);
83
84  % find cos(theta) and sin(theta) of Givens rotation
85      if h(j+1) == 0
86        c = 1.0;                      % theta = 0
87        s = 0.0;
88      elseif abs(h(j+1)) > abs(rt)
89        temp = rt / h(j+1);
90        s = 1.0 / sqrt(1.0 + temp^2); % pi/4 < theta < 3pi/4
91        c = − temp * s;
92      else
93        temp = h(j+1) / rt;
94        c = 1.0 / sqrt(1.0 + temp^2); % −pi/4 <= theta < 0 < theta <=
              pi/4
```

```matlab
          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
```

```matlab
        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
```

39

```matlab
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                ...
                        ' solution with relative residual %0.2g'], ...
212                      restart,iter(1),iter(2),relres);
213
214      case 1,
215        os = sprintf(['GMRES(%d) stopped after the maximum %d
              iterations' ...
216                      ' without converging to the desired tolerance
                         %0.2g' ...
217                      '\n         The iterate returned (number %d(%d))'
                         ...
218                      ' has relative residual %0.2g'], ...
219                      restart,maxit,tol,iter(1),iter(2),relres);
220
221      case 2,
222        os = sprintf(['GMRES(%d) stopped at iteration %d(%d)' ...
223                      ' without converging to the desired tolerance
                         %0.2g' ...
224                      '\n         because the system involving the' ...
225                      ' preconditioner was ill conditioned.' ...
226                      '\n         The iterate returned (number %d(%d))'
                         ...
227                      ' has relative residual %0.2g'], ...
228                      restart,i,j,tol,iter(1),iter(2),relres);
229
230      case 3,
231        os = sprintf(['GMRES(%d) stopped at iteration %d(%d)' ...
232                      ' without converging to the\n         desired'
                         ...
233                      ' tolerance %0.2g because the method stagnated.'
                         ...
234                      '\n         The iterate returned (number %d(%d))'
                         ...
```

40

```matlab
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 $x_1, \cdots, x_M$.

If r_M is small enough, stop,

else restart with $x_0 = x_M$.

Chapitre 3

Preconditioning

Contents

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 $\tilde{A}$

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

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

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

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

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

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

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

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

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

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 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

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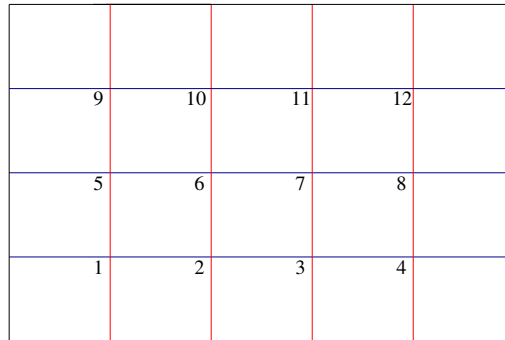
$$1 \le i \le M, 1 \le j \le M$$

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 :

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

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 :

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

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

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 $M = 15$.

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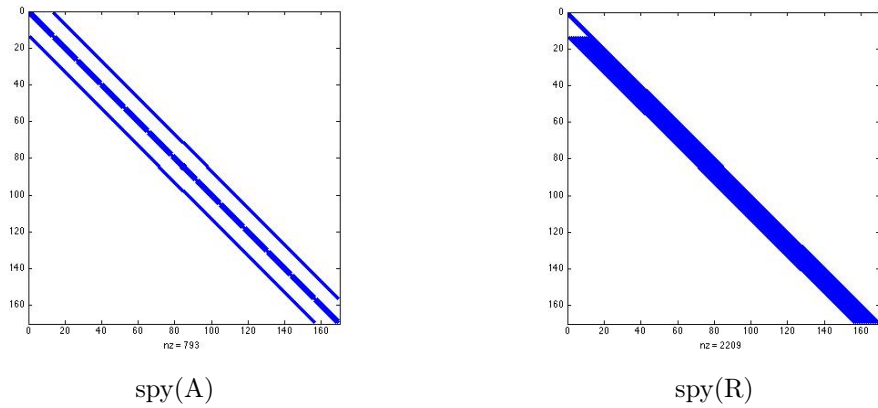
However, if we look closely to the values of R between the main diagonales where A was non zero, we see that where the coefficients of A are zero, the coefficients of R are small. Therefore the incomplete Cholewski preconditioning computes only the values of R where the coefficient of A is not zero,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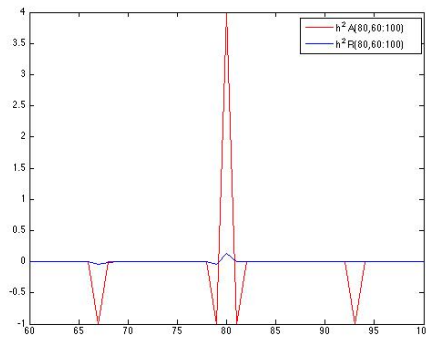

FIGURE 3.2 – Variation of the coefficients of Cholewski in the line 80 for $M = 15$

The matlab codes are as follows ([3])

Cholewski

```
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,
           k);
5      for j=k+1:nn
6          Ch(j:nn,j)=Ch(j:nn,j)-Ch(j:
               nn,k)*Ch(j,k);
7      end
8  end
```

46

Incomplete Cholewski

```
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
                    ,k);
7           end
8       end
9       for j=k+1:nn
10          for i=j:n
11              if ChI(i,j) ~= 0
12                  ChI(i,j)=ChI(i,j)—
                        ChI(i,k)*ChI(j,k
                        );
13              end
14          end
15      end
16  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.
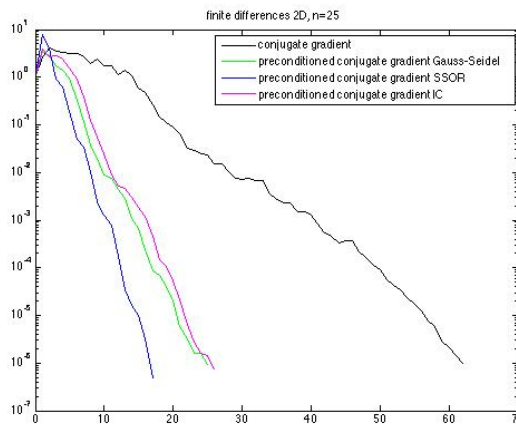
FIGURE 3.3 – Convergence history, influence of preconditioning

3.2   Deflation method for GMRES

Contents

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 garantee 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 $|\lambda_1| \leq |\lambda_2| \leq |\lambda_n|$.

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 $T = U^T AU$ 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 $\lambda_{r+1}, \lambda_{r+2}, ..., \lambda_r, |\lambda_n|$, 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 Hesssenberg matrix $H_k = V_k^T AV_k$, 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 $H_k = SRS^T$ 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 $D = diag(1, 2, ..., 100)$
Case 2 : $\beta = 0.9$ and $D = diag(1, 100, 200, ..., 10000)$

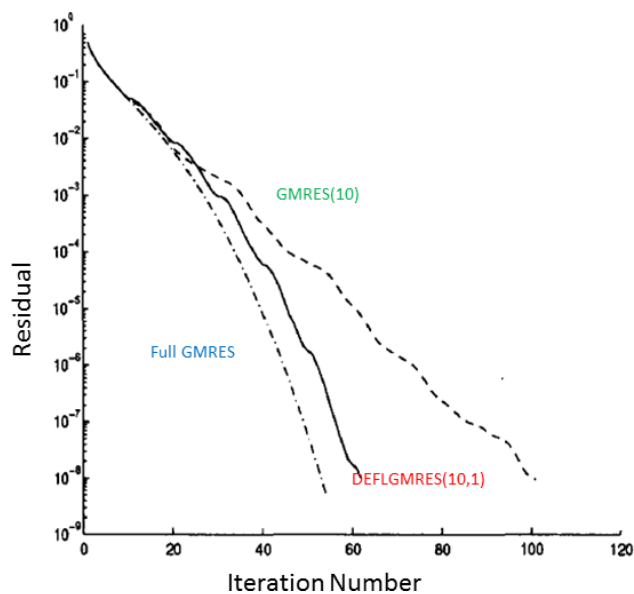DEFLGMRES(10,1) is compared with GMRES(10) and full GMRES . Tolerance is set to 10^{-8}
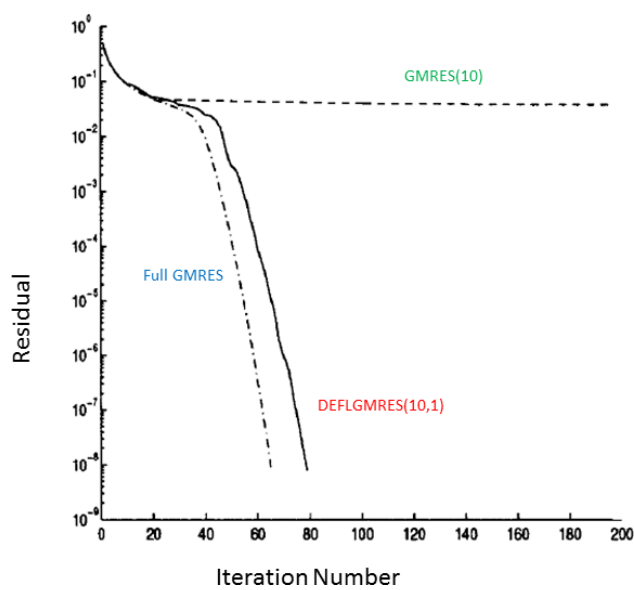
FIGURE 3.4 – Convergence history, Case 1

FIGURE 3.5 – Case 2

3.3 Fast methods using Fast Fourier Transform

Contents

3.3.1 Presentation of the method

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

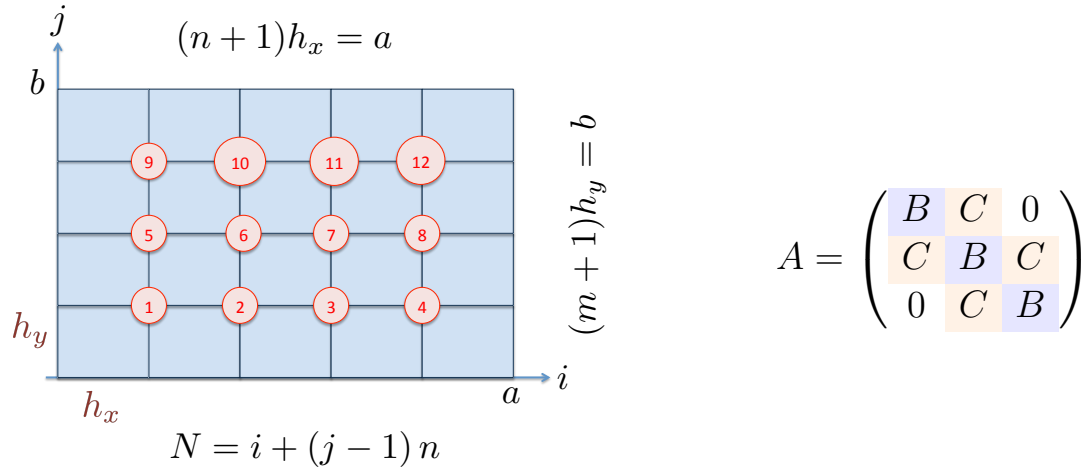$$A = \begin{pmatrix} B & C & 0 \\ C & B & C \\ 0 & C & B \end{pmatrix}$$

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

$$j$$

$$(n+1)h_x = a$$

$$b$$

$$(m+1)h_y = b$$

$$h_y$$

$$h_x \qquad a \qquad i$$

$$N = i + (j-1)\,n$$
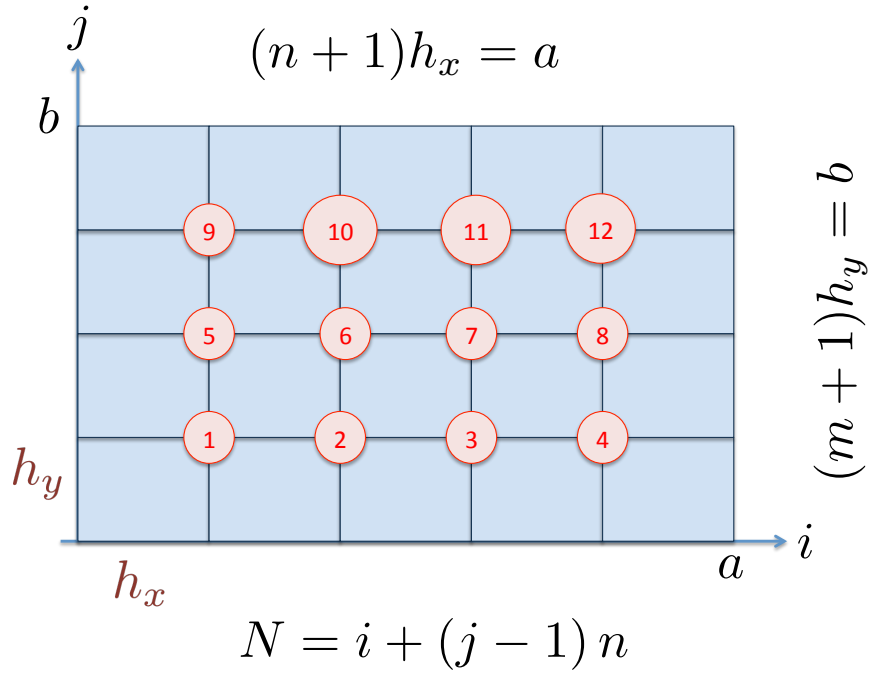
FIGURE 3.7 – Pavage de $[0,a] \times [0,b]$, $n = 4$ and $m = 3$

$$
\begin{pmatrix}
\frac{2}{h_x^2}+\frac{2}{h_y^2} & -\frac{1}{h_x^2} & 0 & 0 & -\frac{1}{h_y^2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-\frac{1}{h_x^2} & \frac{2}{h_x^2}+\frac{2}{h_y^2} & -\frac{1}{h_x^2} & 0 & 0 & -\frac{1}{h_y^2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -\frac{1}{h_x^2} & \frac{2}{h_x^2}+\frac{2}{h_y^2} & -\frac{1}{h_x^2} & 0 & 0 & -\frac{1}{h_y^2} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -\frac{1}{h_x^2} & \frac{2}{h_x^2}+\frac{2}{h_y^2} & 0 & 0 & 0 & -\frac{1}{h_y^2} & 0 & 0 & 0 & 0 \\
-\frac{1}{h_y^2} & 0 & 0 & 0 & \frac{2}{h_x^2}+\frac{2}{h_y^2} & -\frac{1}{h_x^2} & 0 & 0 & -\frac{1}{h_y^2} & 0 & 0 & 0 \\
0 & -\frac{1}{h_y^2} & 0 & 0 & -\frac{1}{h_x^2} & \frac{2}{h_x^2}+\frac{2}{h_y^2} & -\frac{1}{h_x^2} & 0 & 0 & -\frac{1}{h_y^2} & 0 & 0 \\
0 & 0 & -\frac{1}{h_y^2} & 0 & 0 & -\frac{1}{h_x^2} & \frac{2}{h_x^2}+\frac{2}{h_y^2} & -\frac{1}{h_x^2} & 0 & 0 & -\frac{1}{h_y^2} & 0 \\
0 & 0 & 0 & -\frac{1}{h_y^2} & 0 & 0 & -\frac{1}{h_x^2} & \frac{2}{h_x^2}+\frac{2}{h_y^2} & 0 & 0 & 0 & -\frac{1}{h_y^2} \\
0 & 0 & 0 & 0 & -\frac{1}{h_y^2} & 0 & 0 & 0 & \frac{2}{h_x^2}+\frac{2}{h_y^2} & -\frac{1}{h_x^2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -\frac{1}{h_y^2} & 0 & 0 & -\frac{1}{h_x^2} & \frac{2}{h_x^2}+\frac{2}{h_y^2} & -\frac{1}{h_x^2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{h_y^2} & 0 & 0 & -\frac{1}{h_x^2} & \frac{2}{h_x^2}+\frac{2}{h_y^2} & -\frac{1}{h_x^2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{h_y^2} & 0 & 0 & -\frac{1}{h_x^2} & \frac{2}{h_x^2}+\frac{2}{h_y^2}
\end{pmatrix}
$$

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

$$B = \begin{pmatrix} \dfrac{2}{h_x^2} + \dfrac{2}{h_y^2} & -\dfrac{1}{h_x^2} & 0 & 0 \\[2mm] -\dfrac{1}{h_x^2} & \dfrac{2}{h_x^2} + \dfrac{2}{h_y^2} & -\dfrac{1}{h_x^2} & 0 \\[2mm] 0 & -\dfrac{1}{h_x^2} & \dfrac{2}{h_x^2} + \dfrac{2}{h_y^2} & -\dfrac{1}{h_x^2} \\[2mm] 0 & 0 & -\dfrac{1}{h_x^2} & \dfrac{2}{h_x^2} + \dfrac{2}{h_y^2} \end{pmatrix} = A_1(h_x) + \dfrac{2}{h_y^2}\, I_n$$

$$C = -\begin{pmatrix} \dfrac{1}{h_y^2} & 0 & 0 & 0 \\[2mm] 0 & \dfrac{1}{h_y^2} & 0 & 0 \\[2mm] 0 & 0 & \dfrac{1}{h_y^2} & 0 \\[2mm] 0 & 0 & 0 & \dfrac{1}{h_y^2} \end{pmatrix} = -\dfrac{1}{h_y^2}\, I_n.$$

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

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

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

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

The system can be rewritten as

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

which is a system of m systems of dimension n :

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

Suppose B and C are symmetric, and diagonalise in the same orthonormal basis $(\boldsymbol{q}^1, \ldots, \boldsymbol{q}^n)$. This is the case for our previous example. Denote by Q the corresponding orthogonal matrix $Q = [\boldsymbol{q}^1, \ldots, \boldsymbol{q}^n]$. 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

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

and replace B and C :

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

Multiply by Q^T :

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

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

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

Then the problem takes the form

$$
\begin{aligned}
D^1\boldsymbol{y}^1 + D^2\boldsymbol{y}^2 \qquad\qquad\qquad\qquad &= \ \boldsymbol{c}^1 \\
\ddots \qquad\qquad\qquad\qquad\qquad& \\
D^2\boldsymbol{y}^{i-1} + D^1\boldsymbol{y}^i + D^2\boldsymbol{y}^{i+1} \qquad\qquad &= \ \boldsymbol{c}^i \\
\ddots \qquad\qquad& \\
D^2\boldsymbol{y}^{m-1} + D^1\boldsymbol{y}^m &= \ \boldsymbol{c}^m
\end{aligned}
$$

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

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

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, $1 \le j \le n$, 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 $\mathbb{R}^m$

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

We have now n tridiagonal systems of size m,

$$
T_j \boldsymbol{z}^j = \boldsymbol{d}^j, \quad 1 \le j \le n.
$$

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, $\boldsymbol{x}^j = Q\boldsymbol{y}^j$. The matrice C is $-\dfrac{1}{h_y^2} I_n$, 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)

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

Define the matrix Q as the matrix of eigenvectors

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

By

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

we obtain

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

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

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

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

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

55

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

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

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*

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

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,

$$\begin{cases} \text{for } z \neq 1, \quad \sum_{k=1}^{n'} z^k = 0, \\[2mm] \text{for } z = 1, \quad \sum_{k=1}^{n'} z^k = n'. \end{cases}$$

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 $j = 1, \cdots, 2p$,

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

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

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

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 $\{w_j\}_{1 \le j \le 2p}$ from $\{v_j\}_{1 \le j \le 2p}$, do

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

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

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

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

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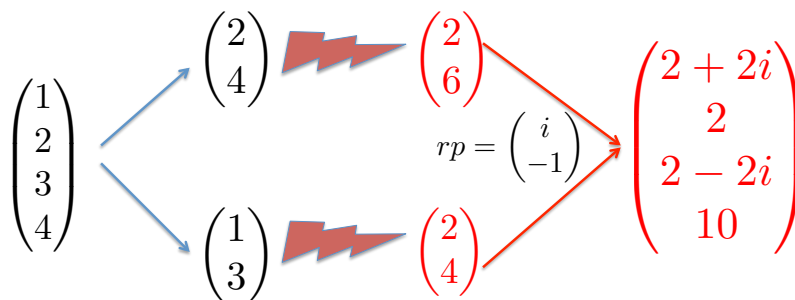
$$r = e^{2i\frac{\pi}{n'}}, w_j = \sum_{k=1}^{n'} v_k \, r^{-kj}, \quad j = 1, \cdots, n'.$$

$n' = 2$, $r = -1$, 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 tranform 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*.

58

3.3.3 The algorithm

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

$$\boldsymbol{v} \in \mathbb{R}^n, \ n' = n + 1 \ \text{EVEN}$$

$$Q\boldsymbol{v} = \sqrt{\frac{2}{n+1}} \ \boldsymbol{z} \in \mathbb{R}^n, \qquad z_j = \sum_{k=1}^{n} v_k \sin \frac{kj\pi}{n'} \quad 1 \le j \le n,$$

$$\tilde{\boldsymbol{v}} = [v; 0] \in \mathbb{R}^{n'},$$

$$DFT(\tilde{\boldsymbol{v}}) = \boldsymbol{w} \in \mathbb{R}^{n'}, \qquad w_j = \sum_{k=1}^{n'} \tilde{v}_k \, e^{-2i \frac{kj\pi}{n'}} \quad 1 \le j \le n'$$

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

$$z_{2\ell} = \sum_{k=1}^{n'} \tilde{v}_k \sin \frac{2\ell k\pi}{n'} = -\mathcal{I}\mathrm{m} w_\ell, \quad \ell = 1, \cdots, \frac{n-1}{2}.$$

Consider now the odd indices, $z_1, \cdots, z_n$

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

Resuming with matlab notations

QFFT

$$
\begin{aligned}
\boldsymbol{r_0} &= e^{i \frac{\pi}{n'}} \\
(Q\boldsymbol{v})_{2\ell} &= -\sqrt{\frac{2}{n+1}} \, \mathcal{I}\mathrm{m}(FFT(\tilde{\boldsymbol{v}}))_\ell, & \ell = 1, \cdots, \frac{n-1}{2} \\
(Q\boldsymbol{v})_{2\ell-1} &= -\sqrt{\frac{2}{n+1}} \, \mathcal{I}\mathrm{m}(FFT(\tilde{\boldsymbol{v}} \cdot * \boldsymbol{r_0}^{(1:n')'}))_\ell, & \ell = 1, \cdots, \frac{n+1}{2}
\end{aligned}
$$

(3.6)

Summarizing the solution of

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

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

Step 2 : Sort $\{\boldsymbol{c}^1, \cdots, \boldsymbol{c}^m\}$ The righthand side has been build by rows in the mesh :
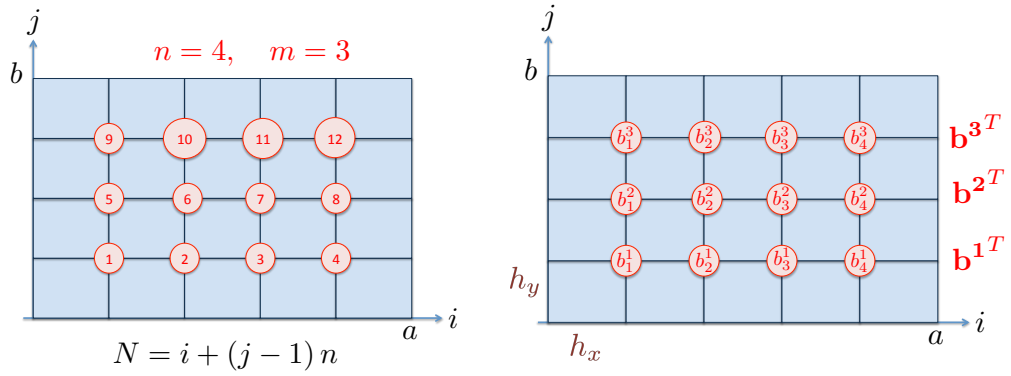$\boldsymbol{b}^j$ is the vector of the values of the forcing term on the line $y = j * h_y$.
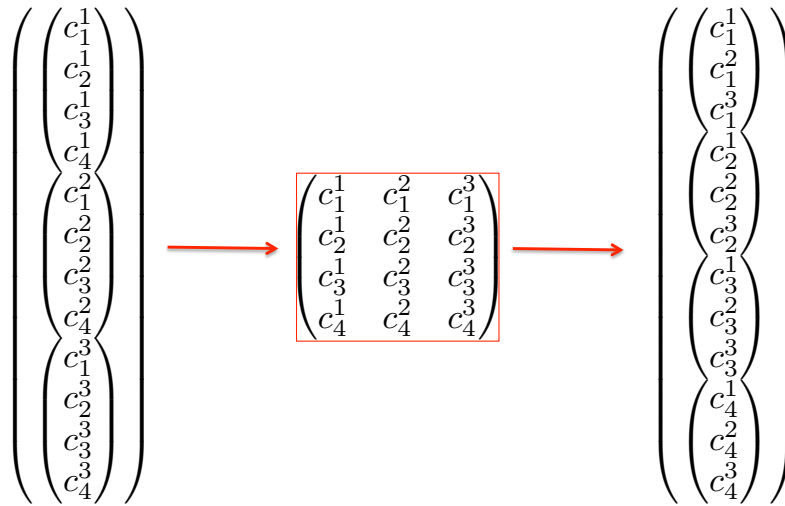
FIGURE 3.9 – Numbering

FIGURE 3.10 – Renumbering

The total vector $\boldsymbol{\sigma}$ is numbered from 1 to nm, with $N = i + (j-1)*n$. The matrix C is built as follows

$$\boldsymbol{\sigma}(1:n) \quad \rightarrow C(:,1)$$
$$\boldsymbol{\sigma}(n+1:2n) \quad \rightarrow C(:,2)$$
$$\vdots$$
$$\boldsymbol{\sigma}((m-1)n+1:mn) \quad \rightarrow C(:,m)$$

```
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,

$$T_j \boldsymbol{z}^j = \boldsymbol{d}^j, \quad 1 \le j \le n.$$

60

with $\boldsymbol{d}^j = C(j,:)$, 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 $\boldsymbol{z}^j$ into $\boldsymbol{y}^j$

Step 5 : Recovering $\boldsymbol{x}^j = Q\boldsymbol{y}^j$ by (3.6).

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

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

That is we write

$$\mathcal{Q}A\mathcal{Q}^T\mathcal{Q}\boldsymbol{u} = \mathcal{Q}\boldsymbol{b}.$$

Chapitre 4

Multigrid methods

Contents

Multigrid methods are a prime source of important advances in algorithmic efficiency , finding a rapidly increasing number of users. Unlike other known methods, multigrid offers the possibility of solving problems with N unknowns with O(N) work and storage, not just for special cases, but for large classes of problems. It relies on the use of several nested grids.

4.1 Geometric multigrid

For the modal presentation of the method, we refer to [7],[2], [5]. For the finite element part, we refer to [1].

4.1.1 The V- cycle process

One cycle of the multigrid method is given as follows. Suppose we want to solve $A^h \bar{U}^h = b^h$. We take an initial guess U^h, and define $MG(A^h, b, U^h)$ to be

Step 1 : smoothing N_1 iterations of the smoother, with initial guess U^h.

$$U^{h,1} = \mathcal{S}^h(A^h, b, U^h, N_1), \quad e^{h,1} = \bar{U}^h - U^{h,1}.$$

The residual is $r^{h,1} = b^h - A^h U^{h,1} = A^h e^{h,1}$.
It is projected on the coarse grid

$$r^{2h} = P_h^{2h} r^{h,1}$$

Step 2 : Coarse resolution The system $A^{2h}\tilde{U}^{2h} = r^{2h}$ is solved approximately by p iterations of the multigrid solver on the coarse grid

$$U^{2h,r} = MG(A^{2h}, r^{2h}, U^{2h,r-1}), \quad U^{2h,0} = 0, 1 \leq r \leq p.$$

It is projected on the fine grid

$$U^{h,2} = U^{h,1} + P_{2h}^h U^{2h,r}, \quad e^{h,2} = e^{h,1} - P_{2h}^h U^{2h,r}$$

Step 3 : Smoothing again N_2 iterations of the smoother

$$U^{h,3} = \mathcal{S}^h(A^h, b^h, U^{h,2}, N_2).$$

We will describe the process in the simple case where the coarse problem is solved exactly, *i.e.*

$$U^{h,2} = U^{h,1} - P_{2h}^h \tilde{U}^{2h}$$

Define $Df2(p)$ the $p \times p$ matrix of $1 - D$ finite differences on a grid of mesh 1 :

$$Df_2(p) = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & 0 \\ & \ddots & \ddots & \ddots & \\ 0 & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad (Df_2(p)U)_j = -U_{j-1} + 2U_j - U_{j+1}.$$

Then $A^h = \frac{1}{h^2} Df_2(n-1)$ and $A^{2h} = \frac{1}{4h^2} Df_2(2n-1)$.

The Smoother

If S is the iteration matrix of the smoother, the result of the smoothing is

$$e^{h,1} = S^{N_1} e^0, \quad r^{h,1} = A^h e^{h,1}. \tag{4.1}$$

Projection on the coarse grid

The fine grid is $(\frac{k}{2n})$ for $1 \leq k \leq 2n - 1$. The coarse grid is $(\frac{k}{n})$ for $1 \leq k \leq n - 1$. Define $h = 1/2n$.

$$P_h^{2h} : \mathbb{R}^{2n-1} \to R^{n-1}, \quad (P_h^{2h} U^h)_j = \frac{1}{4}(U_{2j-1}^h + 2U_{2j}^h + U_{2j+1}^h).$$

The matrix of P_h^{2h} is

$$P_h^{2h} = \begin{pmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & \ldots & \ldots \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & \ldots & \ldots \\ & & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & & \ldots & & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{pmatrix}$$

Define now

$$r^{2h} := P_h^{2h} r^h = P_h^{2h} A^h e^{h,1}.$$

Coarse resolution

Suppose the coarse grid problem is solved exactly.

$$A^{2h} \tilde{U}^{2h} = r^{2h}$$

.

Projection on the fine grid

We define the projection operator as :

$$P_{2h}^h : \mathbb{R}^{n-1} \to R^{2n-1}, \quad \begin{cases} (P_{2h}^h U^{2h})_{2j} = U_j^{2h} \\ (P_{2h}^h U^{2h})_{2j+1} = \frac{1}{2}(U_j^{2h} + U_{j+1}^{2h}) \end{cases}$$

The matrix is

$$P_{2h}^h = \begin{pmatrix} \frac{1}{2} & 0 & 0 & \ldots & 0 \\ 1 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \vdots & & & & \\ \vdots & & & & \\ 0 & 0 & \ldots & 0 & \frac{1}{2} \end{pmatrix}$$

Result of the coarse walk

$$e^{h,2} = (I - P_{2h}^h (A^{2h})^{-1} P_h^{2h} A^h) e^{h,1}$$

Lemma 4.1

$$\mathcal{K}er\, P_h^{2h} A^h = \{V \in \mathbb{R}^{2n-1}, V_{2j} = 0, j = 1 \cdots, n-1\}, \tag{4.2}$$
$$\mathcal{K}er\, P_h^{2h} A^h \oplus \mathcal{I}m P_{2h}^h = \mathbb{R}^{2n-1}, \tag{4.3}$$
$$\forall V \in \mathbb{R}^{2n-1}, \forall j, (A^h P_{2h}^h V)_{2j+1} = 0, \tag{4.4}$$
$$P_h^{2h} A^h P_{2h}^h = A^{2h}. \tag{4.5}$$

It is easy to compute

$$
\begin{aligned}
(P_h^{2h} A^h U)_j &= \tfrac{1}{4}\left((A^h U)_{2j-1} + 2(A^h U)_{2j} - (A^h U)_{2j+1}\right) \\
&= \tfrac{1}{4h^2}\left(-U_{2j-2} + 2U_{2j-1} - U_{2j} + 2(-U_{2j-1} + 2U_{2j} - U_{2j+1}) - U_{2j} + 2U_{2j+1} - U_{2j+2}\right) \\
&= \tfrac{1}{4h^2}\left(-U_{2j-2} + 2U_{2j} - U_{2j+2}\right) \\
&= A^{2h}\begin{pmatrix} U_2 \\ \vdots \\ U_{2n-2} \end{pmatrix}
\end{aligned}
$$

Denoting by U^e the vector of the even coordinates of U, we have proved that for any vector $U \in \mathbb{R}^{2n-1}$,

$$
P_h^{2h} A^h U = U^e.
$$

Therefore the kernel of $P_h^{2h} A^h$ is equal to the space of U such that $U^e = 0$, which proves (4.2).

Now by the rank theorem,

$$
\dim \mathcal{K}\mathrm{er} P_h^{2h} + \dim \mathcal{I}\mathrm{m} P_h^{2h} = 2n - 1.
$$

Since A^h is an isomorphism in $\mathbb{R}^{2n-1}$, $\dim \mathcal{K}\mathrm{er} P_h^{2h} = \dim \mathcal{K}\mathrm{er} P_h^{2h} A^h$. Then

$$
\dim \mathcal{K}\mathrm{er} P_h^{2h} A^h + \mathrm{rg} P_h^{2h} = 2n - 1.
$$

Since $P_h^{2h} = \tfrac{1}{2}(P_{2h}^h)^T$, they have the same rank, and therefore

$$
\dim \mathcal{K}\mathrm{er} P_h^{2h} A^h + \mathrm{rg} P_{2h}^h = 2n - 1.
$$

Furthermore, any U in $\mathcal{K}\mathrm{er} P_h^{2h} A^h \cap \mathcal{I}\mathrm{m} P_{2h}^h$ is equal to $P_{2h}^h w$, and $v_{2j} = 0$. Since $(P_{2h}^h w)_{2j} = w_j$, this proves that $w = 0$. Hence (4.3) is proved.

We now can prove in the same way, first that for V in $\mathbb{R}^{n-1}$,

$$
(A^h P_{2h}^h V)_{2j+1} = 0, \quad (A^h P_{2h}^h V)_{2j} = \frac{1}{2h^2}(-v_{j-1} + 2v_j - v_{j+1}) = 2(A^{2h} v)_j.
$$

Then

$$
(P_h^{2h} A^h P_{2h}^h V)_j = (A^{2h} v)_j.
$$

∎

Lemma 4.2
$$
e^{h,1} = d^h + P_{2h}^h e^{2h},
$$

with
$$
d_{2j}^h = 0, \quad d_{2j+1}^h = \frac{h^2}{2}(A^h e^{h,1})_{2j+1}, \quad e_j^{2h} = e^{h,1})_{2j}
$$

By (4.3), we can expand $e^{h,1}$ as

$$
e^{h,1} = d^h + P_{2h}^h e^{2h},
$$

with $d^h \in Ker P_h^{2h} A^h$. By (4.2), $d_{2j}^h = 0$, and

$$
e_{2j}^{h,1} = (P_{2h}^h e^{2h})_{2j} = e_j^{2h},
$$

which determines the components of e^{2h}. Compute now the odd components,

$$
e_{2j+1}^{h,1} = d_{2j+1}^h + (P_{2h}^h e^{2h})_{2j+1} = d_{2j+1}^h + \frac{1}{2}(e_j^{2h} + e_{j+1}^{2h}) = d_{2j+1}^h + \frac{1}{2}(e_{2j}^{h,1} + e_{2j+2}^{h,1})
$$

66

Therefore

$$d^h_{2j+1} = \frac{1}{2}(2e^{h,1}_{2j+1} - e^{h,1}_{2j} - e^h_{2j+2}) = \frac{h^2}{2}(A^h e^{h,1})_{2j+1}.$$

∎

Apply the lemma to compute $e^{h,2}$.

$$P^h_{2h}(A^{2h})^{-1}P^{2h}_h A^h e^{h,1} = P^h_{2h}(A^{2h})^{-1}P^{2h}_h A^h(d^h + P^h_{2h}e^{2h}) = P^h_{2h}(A^{2h})^{-1}\underbrace{P^{2h}_h A^h P^h_{2h}}_{A^{2h}} e^{2h} = P^h_{2h}e^{2h}.$$

Therefore

$$e^{h,2} = e^{h,1} - P^h_{2h}e^{2h} = d^h,$$

which implies the elegant formula

$$e^{h,2}_{2j} = 0, \quad e^{h,2}_{2j+1} = \frac{h^2}{2}(A^h e^{h,1})_{2j+1} = \frac{h^2}{2}r^{h,1}_{2j+1}.$$

the even components have disappeared.

Postsmoothing

$$e^{h,3} = S^{N_2}e^{h,2}.$$

$$\boxed{e^{h,3} = S^{N_2}\Pi_o\frac{h^2}{2}A^h S^{N_1}e^h}$$

Spectral analysis

The smoothing matrix S has eigenvalues λ_k, and eigenvectors $\Phi^{(k)}$. For relaxed Jacobi or the Gauss-Seidel algorithm, the eigenvalues are

$$\begin{aligned}
\lambda^J_k(\omega) &= 1 - 2\omega\sin^2\left(\frac{k\pi h}{2}\right) & \text{for } 1 \le k \le 2n-1, \\
\lambda^{GS}_k &= \cos^2 k\pi h & \text{for } 1 \le k \le 2n-1,
\end{aligned}$$

Figure 4.1 shows the eigenvalues as a function of k for $n = 16$.
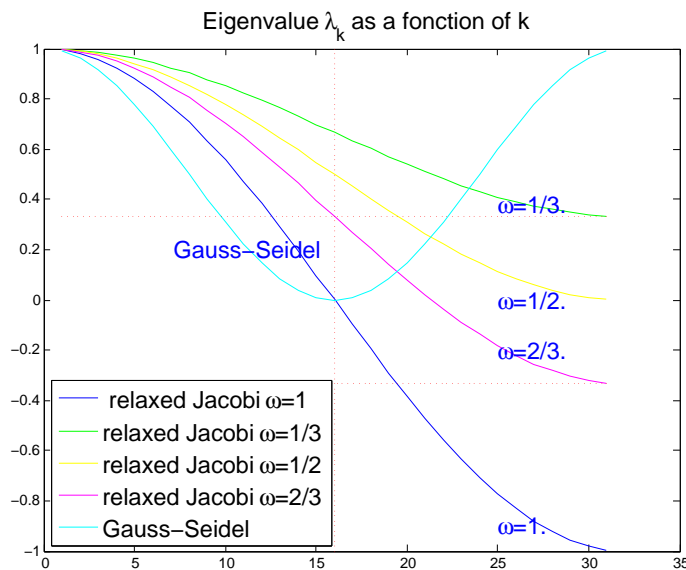
FIGURE 4.1 – Eigenvalues of the relaxed Jacobi iteration matrix as a function of k for several values of ω together with Gauss-Seidel

* For small k, $\lambda_k^J(\omega) \sim 1 - \omega\dfrac{k^2\pi^2 h^2}{2}$.
* For $\omega = 2/3$, $n \leq k \leq 2n-1 \Rightarrow |\lambda_k^J(\omega)| \leq \underbrace{\quad 1/3 \quad}_{\text{smoothing factor}}$

* For other modes. $|\lambda_k^J(\omega)| \in (1/3, 1 - \dfrac{4}{3}\sin^2(\dfrac{\pi h}{2}))$

When using Gauss-Seidel as a smoother, one can observe that the eigenvalues are small when $k \sim n : \lambda_k \leq 1/2$ for $n/2 \leq k \leq 3n/2.$.

For an initial error $e^h = \Phi^{(k)}$, the error and residual after N_1 iterations is

$$e^{h,1} = \lambda_k^{N_1}\Phi^{(k)}, \quad r^{h,1} = \mu_k\lambda_k^{N_1}\Phi^{(k)}.$$

From

$$e_{2j}^{h,2} = 0, \quad e_{2j+1}^{h,2} = \frac{h^2}{2}r_{2j+1}^{h,1}$$

we obtain

$$e^{h,2} = \frac{h^2}{2}\mu_k\lambda_k^{N_1}\Phi_{2j+1}^k.$$

If the same smoother is applied in postprocessing,

$$e^{h,3} = \lambda_k^{N_2}e^{h,2},$$

and finally,

$$e_{2j}^{h,3} = 0, \quad e_{2j+1}^{h,3} = \frac{h^2}{2}\mu_k\lambda_k^{N_1+N_2}\Phi_{2j+1}^k.$$

We can see now that even the low frequencies are damped. Choose relaxed Jacobi with $\omega = 2/3$. For $n \leq k \leq 2n-1$, we have, with $N = N_1 + N_2$,

$$|e_{2j+1}^{h,3}| \leq (\frac{2}{3})^N|\Phi_{2j+1}^k|,$$

and for $1 \leq k \leq n-1$,

$$|e_{2j+1}^{h,3}| \leq \sup_{x\in(0,1)}(x(1-\omega x)^N)|\Phi_{2j+1}^k| \leq \frac{1}{\omega(N+1)}\left(\frac{N}{N+1}\right)^N|\Phi_{2j+1}^k|$$

For three iterations of the smoother (N=3), the low frequencies have been damped by a factor 0.1582**, and the high frequencies by a factor 0.2963 !!** The figures below show the result of one cycle of the above described algorithm, compared to three iterations of relaxed Jacobi, or Gauss-Seidel, for several inital guesses. $n = 10$.
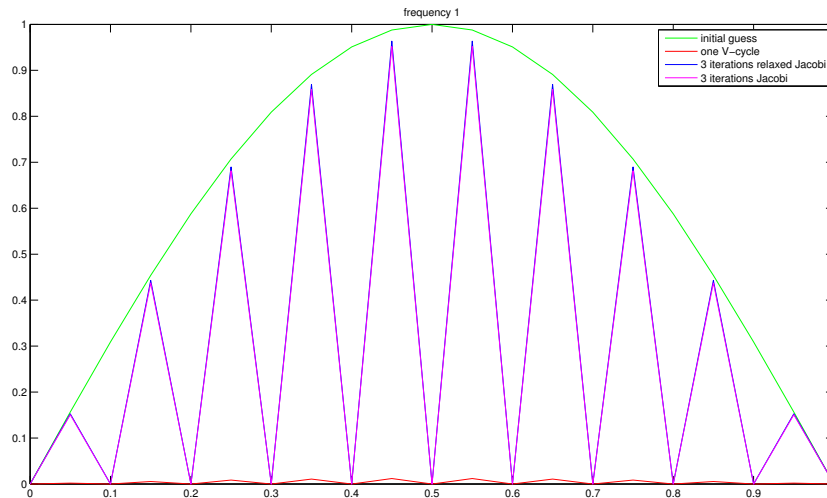

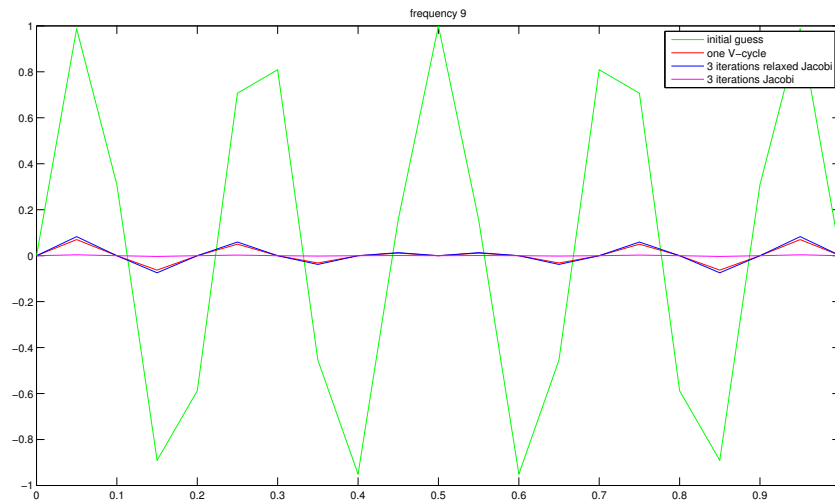FIGURE 4.2 – Comparison of the iterative methods. Initial guess $\sin \pi x$

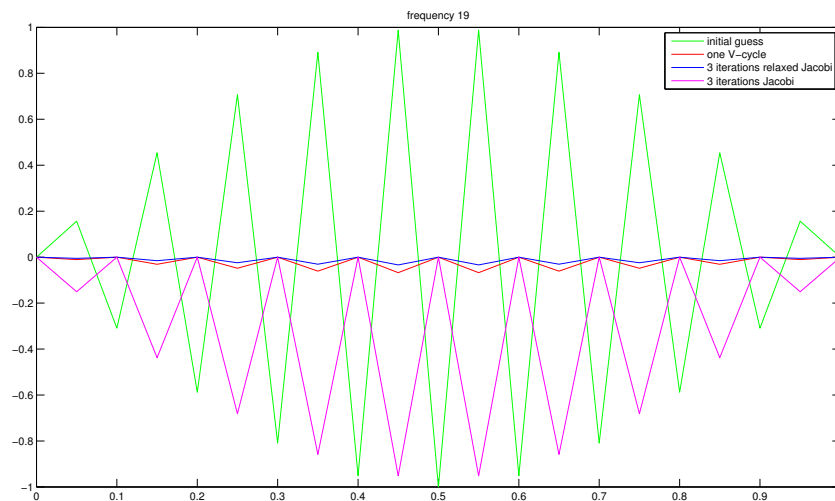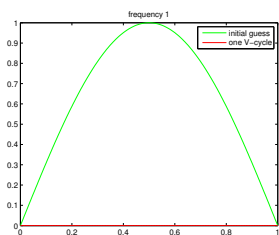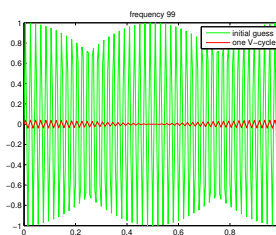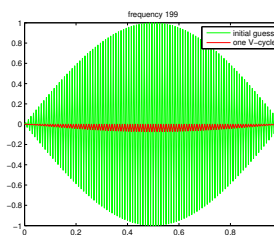FIGURE 4.3 – Comparison of the iterative methods. Initial guess $\sin(n-1)\pi x$.

FIGURE 4.4 – Comparison of the iterative methods. Initial guess $\sin(2n-1)\pi x$.

Initial guess $\sin \pi x$ Initial guess $\sin(n-1)\pi x$ Initial guess $\sin(2n-1$

TABLE 4.1 – The effect of one V-cycle on one single mode for $n = 100$.

The N_2 last smoothing steps helps to reduce the high frequencies by a factor $(\frac{2}{3})^{N_2}$.

Number of elementary operations

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

TABLE 4.2 – Asymptotic order of the number of elementary operations as a function of the number of grid points in one dimension for the Laplace equation (sparse matrix)

4.1.2 The finite elements multigrid algorithm

Details on finite elements can be found in [4][6] and.[1].

We consider here an elliptic problem in $V = H_0^1(\Omega)$, where Ω is a convex polygone. If $\alpha_1 \leq a_{ij} \leq \alpha_2$ *a.e.* in Ω, ,

$$a(u,v) = \sum_{i,j=1}^{2} \int_{\Omega} \left(a_{ij}(x)\nabla u(x)\nabla v(x) + a_0(x)u(x)v(x) \right) dx$$

is an elliptic bilinear form. It therefore defines a norm, which is equivalent to the H^1 norme, that we call the energy norm

$$\|v\|_E = \sqrt{a(v,v)}$$

The variational problem is, to find $u \in V$ such that

$$\forall v \in V, a(u,v) = (f,v) \tag{4.6}$$

We know that there is a unique solution in V which, furthermore, belongs to $H^2(\Omega)$.and $\|u\|_{H^2(\Omega)} \leq C\|f\|_{L^2(\Omega)}$.

Preliminaries

Let $\mathcal{T}_k$ be a sequence of triangulations of Ω. h_k is the longest measure of the side of the triangles in $\mathcal{T}_k$. $\mathcal{T}_k$ is obtaind from $\mathcal{T}_{k-1}$ by dividing each triangle into four triangles.

Let (N^T, N^E, N) be the number of triangles outside the boundary of Ω, edges and vertices respectively. There is a recursion relation :

$$N_{k+1}^T = 4N_k^T, \ N_{k+1} = N_k + N_k^E, \ N_{k+1}^E = 2N_k^E + 3N_k^T$$

which provides the total number of each, starting with the triangulation $\mathcal{T}_1$ in Figure 4.5 : $(N^T, N^E, N) = (N_1, N_1, N_1 + 1)$.

$$N_{k+1}^T = 2^{2k}N_1, \ N_{k+1} = 2^{k-1}(2^k - 1)N_1, \ N_{k+1}^E = 2^{k-1}(2^{k+2} - 1)N_1$$

We have asymptotically

$$N_k \sim 2^{2k-1}N_1 \tag{4.7}$$

FIGURE 4.5 – Recursive triangulation

For each k, the diameter of the triangulation h_k is the largest length of edge, therefore $h_{k+1} = h_k/2$. Then the triangulation is quasi-uniform (cf [1]), in the sense that there exists $\rho > 0$ such that

$$\inf_{T \in \mathcal{T}_k} \mathrm{diam} B_T \geq \rho h_k$$

where B_T is the largest ball contained in T. Its diameter is given by $\frac{4|T|}{length(T)}$ with $|T| := area(T) = \frac{1}{2}(AB)(AC)\sin(\widehat{BAC})$, and $length(T)$ is the perimeter of T.
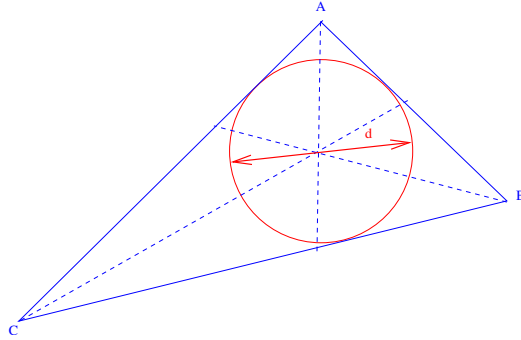
FIGURE 4.6 – triangle

It is easy to see that, after a refinement, the diameter is divided by 2, and so is h, therefore it suffices to define $\rho = \frac{1}{h_1} \inf_{T \in \mathcal{T}_1} \mathrm{diam} B_T$.

$$V_k = \{v \in V \cap \mathcal{C}^0(\bar{\Omega}), \forall T \in \mathcal{T}_k, v|_T \in \mathbb{P}_1\}$$

This defines a sequence of finite-dimensional spaces, of dimension N_k, with $V_k \subset V_{k+1}$. We define the variational problem in V_k, to find $u_k \in V_k$ such that

$$\forall v \in V_k, a(u_k, v) = (f, v) \tag{4.8}$$

Classical finite element results assert that this problem has a unique solution, and the following error estimate holds :

$$\|u - u_k\|_{H^1(\Omega)} \leq C h_k \, |u|_{H^2(\Omega)}$$

We denote by P_k the projection operator on V_k, defined for any w in V by

$$\forall v \in V_k, a(P_k w, v) = a(w, v) \tag{4.9}$$

For w in V, we introduce the solution z of problem (4.6), and z_k the solution of the discrete problem (4.8), both with data $w - P_k w$. Elementary algebra shows that

$$\|w - P_k w\|_{L^2(\Omega)}^2 = a(w - P_k w, z - z_k)$$

71

It follows that there exists a constant independent of h_k such that

$$\forall w \in V, \|w - P_k w\|_{L^2(\Omega)} \leq C h_k \|w - P_k w\|_{H^1(\Omega)} \qquad (4.10)$$

We obtain the estimate on the error in $L^2(\Omega)$ by using the same argument (duality argument), replacing $w - P_k w$ by $u - u_k$.

$$\|u - u_k\|_{L^2(\Omega)} \leq C h_k \|u - u_k\|_{H^1(\Omega)} \leq C h_k^2 |u|_{H^2(\Omega)}$$

We will need the

Theorem 4.1 (Inverse estimate)

$$\forall v \in V_k, \|v\|_{H^1} \leq \frac{C}{h_k} \|v\|_{L^2}$$

For a proof see [6], [1].

The goal of the multigrid method is to compute an approximate value U_k of u_k in $\mathcal{O}(N_k)$ operations, and such that

$$\|U_k - u_k\|_{L^2(\Omega)} \leq C h_k^2 |u|_{H^2(\Omega)}$$

Discrete norm

Note globally $S_1, \cdots, S_{N_k}$ the vertices. Define a scalar product on V_k by

$$(v, w)_k = h_k^2 \sum_{i=1}^{N_k} v(S_i) w(S_i) \qquad (4.11)$$

Theorem 4.2 *It is equivalent to the L^2 scalar product on V_k.*

Proof Use the exact integration formula in dimension 2 : denoting by M_α the mid-points of the edge in the triangle, we have for any $v \in \mathbb{P}_1$,

$$\|v\|_{L^2(T)}^2 = \frac{|T|}{3} \sum_{\alpha=1}^{3} v^2(M_\alpha)$$

Now since v is affine, the values at point M_α are the half-sum of values at points S_α.

$$\sum_{\alpha=1}^{3} v^2(M_\alpha) = \frac{1}{4} \sum_{\alpha=1}^{3} (v(M_\beta) + v(M_\gamma))^2$$

But

$$(x + y)^2 + (y + z)^2 + (z + x)^2 = x^2 + y^2 + z^2 + (x + y + z)^2$$

therefore

$$\sum_{\alpha=1}^{3} (v(M_\alpha))^2 \leq \sum_{\alpha=1}^{3} (v(M_\beta) + v(M_\gamma))^2 \leq 4 \sum_{\alpha=1}^{3} (v(M_\alpha))^2,$$

$$\frac{|T|}{12} \sum_{\alpha=1}^{3} v^2(M_\alpha) \leq \|v\|_{L^2(T)}^2 \leq \frac{|T|}{3} \sum_{\alpha=1}^{3} v^2(M_\alpha),$$

and the result follows by summing over all the triangles. ∎

We define the operator A_k by

$$\forall v, w \in V_k, (A_k v, w)_k = a(v, w) \tag{4.12}$$

and $f_k \in V_k$ by $(f_k, v) = (f, v)_k$ for all v in V_k. A_k is the operator whose matrix in the basis of hat functions φ_i is the stiffness matrix K.
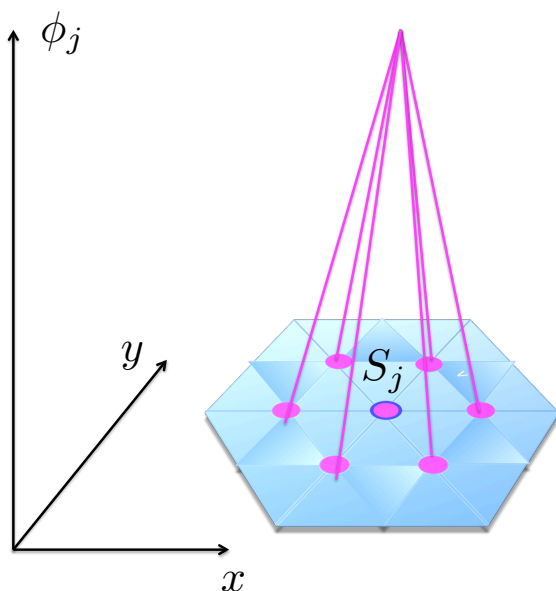

FIGURE 4.7 – Hat basis function φ_j associated to vertex S_j in two dimensions

Solve the discrete problem amounts to solving the N_k dimensional system of equations

$$A_k u_k = f_k$$

The operator A_k is obviously symmetric positive definite wih respect to $(\cdot, \cdot)_k$. We define mesh-dependent norms as

$$|||v|||_{s,k} = \sqrt{(A_k^s v, v)_k}$$

Theorem 4.2 asserts that $||| \cdot |||_{0,k}$ is equivalent to the L^2 norm in V_k. As to the norm for $s = 1$, it coincides with the energy norm thanks to (4.12). We now estimate the spectral radius of A_k :
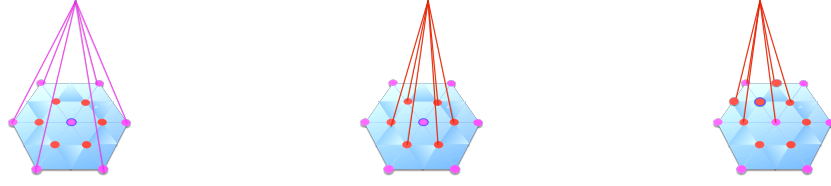
Lemma 4.3

$$\rho(A_k) \le \frac{C}{h_k^2}$$

Proof Let λ be a (positive) eigenvalue, with eigenvector v.

$$a(v, v) = \lambda |||v|||_{0,k}^2$$

$$\lambda \le b \frac{\|v\|_{H^1}^2}{\|v\|_{L^2}^2} \le \frac{C^2}{h_k^2}$$

by the inverse inequality in Theorem 4.1. ∎

(a) Φ_i^{k-1} for $S_i \in \mathcal{T}^{k-1}$ (b) Φ_i^k for $S_i \in \mathcal{T}^{k-1}$ (c) Φ_i^k for $S_i \in \mathcal{T}^k \setminus \mathcal{T}^{k-1}$

FIGURE 4.8 – Hat basis functions

Definition of the multigrid algorithm

In order to pass from one grid to the finer or coarser grid, we need to define transfer operators, which are mutually dual

$$\mathcal{I}_k : V_{k-1} \to V_k,$$
$$\forall v \in V_{k-1}, \mathcal{I}_k v := v; \tag{4.13}$$

$$\mathcal{R}_k : V_k \to V_{k-1},$$
$$\forall w \in V_{k-1}, (\mathcal{R}_k v, w)_{k-1} := (\mathcal{I}_k w, v)_k = (w, v)_k;$$

For any k and initial guess $z_0 \in V_k$, and right-hand side $g \in V_k$, the $k-$th level iteration is an approximate solution $MG(A^k, z_0, g)$ in V_k of

$$A_k z = g \tag{4.14}$$

defined as follows :

For k=1, there is only one grid to deal with, and $MG(A^1, z_0, g)$ is obtained by a direct method.

For $k > 1$, z is obtained in three steps

1 Presmoothing on the fine grid : m_1 steps of a gradient algorithm

$$z_{l+1} = z_l - \mu_k(A_k z_l - g), \ 0 \le l \le m_1 - 1$$

2 Error correction on the coarse grid The residual $g - A_k z_{m_1}$ is transferred on the grid $\mathcal{T}_{k-1}$,

$$G = \mathcal{R}_k(g - A_k z_{m_1}). \tag{4.15}$$

Now we compute an approximate solution of the residual equation

$$A_{k-1} q = G \tag{4.16}$$

by performing p steps of the multigrid algorithm on $\mathcal{T}_{k-1}$:

$$q_0 = 0, q_l = MG(A^{k-1}, q_{l-1}, G), \ 1 \le l \le p$$

Then we project on the fine grid again

$$z_{m_1+1} := z_{m_1} + \mathcal{I}_k q_p$$

3 Smoothing on the fine grid we perform again a few steps of the gradient algorithm

$$z_{l+1} = z_l - \mu_k(A_k z_l - g), \ m_1 + 1 \le l \le m_1 + m_2$$
$$MG(k, z_0, g) = z_{m_1+m_2}$$

74

m_1 and m_2 are positive integers, p=1 is a V-cycle, p=2 is a W-cycle. Usually one uses $m_1 = 3$ and $m_2 = 1$.

The full-multigrid algorithm to solve $A_k f = f_k$ is therefore

$$U_1 = A_1^{-1} f_1,$$
$$U_k : = MG(A_k, \mathcal{I}_k U_{k-1}, f_k)$$

Convergence property of the multigrid algorithm

We suppose here for simplicity that there is no postsmoothing, *i.e.* $m_2 = 0$, we note $m := m_1$, and we consider a W-cycle, *i.e.* $p = 2$.

Theorem 4.3 *If the relaxation coefficient μ_k satisfies*

$$\rho_k \leq \frac{1}{\mu_k} \leq \frac{C}{h_k^2} \tag{4.17}$$

the one-sided W-cycle is convergent, and the following estimate holds :

$$\|U_k - u_k\|_E \leq C h_k |u|_{H^2(\Omega)}$$

Proof The total error is

$$u_k - U_k = u_k - MG(A_k, \mathcal{I}_k U_{k-1}, f_k)$$

First, for z in V_k solution of (4.14), we must estimate $z - MG(A^k, z_0, g)$. It is equal to $z_{m_1} + \mathcal{I}_k q_2$. We rewrite the error as :

$$z - (z_{m_1} + \mathcal{I}_k q_2) = z - (z_{m_1} + \mathcal{I}_k q) + \mathcal{I}_k(q - q_2).$$

We start with the estimate of the first part :

Lemma 4.4 *Let $q \in V_{k-1}$ the solution of (4.16), then $q = P_{k-1}(z - z_m)$.*

Proof We should show that for any $v \in V_{k-1}$,

$$a(q, v) = a(z - z_m, v)$$

We have successively

$$
\begin{aligned}
a(q, v) &= (A_{k-1} q, v)_{k-1} \text{ by definition of } A_{k-1} \text{ in (4.12)} \\
&= (G, v)_{k-1} \text{ by definition of } q \text{ in (4.16)} \\
&= (\mathcal{R}_k(g - A_k z_m), v)_{k-1} \text{by definition of } G \text{ in (4.15)} \\
&= (g - A_k z_{m_1}, v)_k \text{ by definition of } \mathcal{R}_k \text{ in (4.13)} \\
&= (A_k(z - z_{m_1}), v)_k \text{ by definition of } z \text{ in (4.14)} \\
&= a(z - z_{m_1}, v) \text{ by definition of } A_k \text{ in (4.12)}
\end{aligned}
$$

$\blacksquare$

We can now write

$$z - (z_{m_1} + q) = z - z_{m_1} - P_{k-1}(z - z_{m_1}) = (I - P_{k-1})(z - z_{m_1}).$$

Since

$$z - z_m = (I - \mu_k A_k)^m (z - z_0)$$

so

$$z - (z_m + \mathcal{I}_k q) = (I - P_{k-1})(I - \mu_k A_k)^m (z - z_0). \tag{4.18}$$

We have to estimate the projection first :

Lemma 4.5
$$\forall v \in V_k, \|v - P_{k-1}v\|_E \leq Ch_k\|A_kv\|_k$$

$$
\begin{aligned}
\|v - P_{k-1}v\|_E^2 &= a(v - P_{k-1}v, v) \text{ by the definition of } P_{k-1}, \\
&= (v - \mathcal{I}_kP_{k-1}v, A_kv)_k, \text{ by definition of } A_k \text{ in } (4.12) \\
&\leq \|v - \mathcal{I}_kP_{k-1}v\|_k\|A_kv\|_k, \\
&\leq C\|v - P_{k-1}v\|_{L^2(\Omega)}\|A_kv\|_k \text{ by the equivalence of norms}, \\
&\leq C\|v - P_{k-1}v\|_{L^2(\Omega)}\|A_kv\|_k \\
&\leq Ch_k\|v - P_{k-1}v\|_{H^1(\Omega)}\|A_kv\|_k \text{ by } (4.10) \\
&\leq Ch_k\|v - P_{k-1}v\|_E\|A_kv\|_k \text{ by the equivalence of norms}.
\end{aligned}
$$

■

We now study the relaxation operator

$$\mathcal{S}_k = I - \mu_k A_k$$

Lemma 4.6 *For any v in V_k,*
$$\|\mathcal{S}_kv\|_E \leq \|v\|_E$$
Furthermore, there exists $C > 0$ such that, for any k, for any v in V_k,
$$\|A_k\mathcal{S}_k^mv\|_k \leq Ch_k^{-1}m^{-1/2}\|v\|_{H^1(\Omega)}.$$

we expand v on the orthonormal eigenfunctions (with respect to the scalar product $(\ )_k$) of the positive definite operator A_k, called $(\psi_1, \cdots, \psi_{N_k})$ associated to $(\lambda_1, \cdots, \lambda_{N_k})$, $v = \sum_{j=1}^{N_k} v_j\psi_j$.

$$a(v, v) = (A_kv, v)_k = (\sum_{j=1}^{N_k} \lambda_j v_j\psi_j, \sum_{j=1}^{N_k} v_j\psi_j)_k = \sum_{j=1}^{N_k} \lambda_j v_j^2$$

$$\mathcal{S}_kv = \sum_{j=1}^{N_k}(1 - \mu_k\lambda_j)v_j\psi_j$$

$$a(\mathcal{S}_kv, \mathcal{S}_kv) = \sum_{j=1}^{N_k}\lambda_j(1 - \mu_k\lambda_j)^2 v_j^2$$

by the assumptions on μ_k, we have $0 \leq \mu_k\lambda_j \leq 1$, and

$$a(\mathcal{S}_kv, \mathcal{S}_kv) \leq a(v, v)$$

$$
\begin{aligned}
\|A_k\mathcal{S}_k^mv\|_k &= \sum_{j=1}^{N_k}\lambda_j(1 - \mu_k\lambda_j)^{2m}v_j^2 \\
&\leq \frac{1}{\mu_k}\sup_{x \in (0,1)}(x(1-x)^{2m})\sum_{j=1}^{N_k}v_j^2 \\
&\leq \frac{1}{\mu_k}\frac{1}{2m}\|v\|_k^2 \\
&\leq C\frac{1}{mh_k^2}\|v\|_k^2 \leq C\frac{1}{mh_k^2}\|v\|_{L^2(\Omega)}^2 \leq C\frac{1}{mh_k^2}\|v\|_{H^1(\Omega)}^2.
\end{aligned}
$$

We return to the error in (4.18)

$$
\begin{aligned}
\|z - (z_m + \mathcal{I}_kq)\|_{H^1(\Omega)} &\leq Ch_k\|A_k\mathcal{S}_k^m(z - z_0)\|_k \\
&\leq \frac{C}{\sqrt{m}}\|z - z_0\|_{H^1(\Omega)}
\end{aligned}
$$

■

Lemma 4.7 *For any γ, $0 < \gamma < 1$, we can choose m such that*

$$\forall k \geq 1, \|z - MG(A^k, z_0, g)\|_E \leq \gamma \|z - z_0\|_E$$

The convergence rate in W-cycle is **independent of the mesh size** h_k

   Proof The proof goes by recursion.

   For $k = 1$, $z = MG(A^k, z_0, g)$.

   Suppose for any $j < k$, $\|z - MG(A^j, z_0, g)\|_E \leq \gamma \|z - z_0\|_E$ with $A^j z = g$. we now have $z - MG(A^k, z_0, g) = z - (z_m + \mathcal{I}_k q) + \mathcal{I}_k(q - q_2)$. By the recursion relation

$$\|q - q_2\|_E \leq \gamma^2 \|q\|_E \leq \gamma^2 \|z - z_m\|_E$$
$$\leq \gamma^2 \|\mathcal{S}^m(z - z_0)\|_E$$
$$\leq \gamma^2 \|z - z_0\|_E$$

and

$$\|z - MG(k, z_0, g)\|_E \leq (\frac{C}{\sqrt{m}} + \gamma^2)\|z - z_0\|_E$$

Choosing $m > (\frac{C}{\gamma - \gamma^2})^2$, we get the result. ∎

   We can now conclude the proof of the theorem :

$$\|u_k - U_k\|_E \leq \gamma \|u_k - U_{k-1}\|_E$$
$$\leq \gamma(\|u_k - u_{k-1}\|_E + \|u_{k-1} - U_{k-1}\|_E)$$
$$\leq \gamma(C(h_k + h_{k-1})|u|_{H^2(\Omega)} + \|u_{k-1} - U_{k-1}\|_E)$$
$$\leq \gamma(3Ch_k|u|_{H^2(\Omega)} + \|u_{k-1} - U_{k-1}\|_E)$$

Since the error at step 1 vanishes, we see by recursion that

$$\|u_k - U_k\|_E \leq 3C\gamma|u|_{H^2(\Omega)} \sum_{j=2}^{k-2} \gamma^j h_{k-j}$$

Now we can choose $\gamma < 1/2$, and we obtain

$$\|u_k - U_k\|_E \leq h_k \frac{3C\gamma}{1 - 2\gamma}|u|_{H^2(\Omega)}$$

which concludes the proof of the theorem. ∎

Proposition 4.1 *For a number of cycles $p < 4$, the work involved in the full multigrid method is $\mathcal{O}(N_k)$.*

   Proof

   ∗ We call d the maximum number of neighbours of a vertex ($d \sim 15$ for a general construction). Then the matrix A^k has at most d non zero elements in each line. The average number of elementary operations ($+, -, \times, :$) to make the product of A^k by a vector is $2d \times N_k$. The number of operations involved in one step of the gradient is $(2d + 3) \times N_k$. All smoothings therefore require

$$(2d + 3)(m_1 + m_2) \times N_k \text{ elementary operations.}$$

   ∗ As for the projection of the residual, G is defined by

$$G(S_i^{k-1}) = \frac{1}{4} \sum_{\text{neighbours of } S_i \text{ in } \mathcal{T}_k} r_m(S_j^k)$$

   where S_i^k are the vertices in $\mathcal{T}_k$. Therefore the number of operations in the projection step is also

$$d \times N_k \text{ elementary operations.}$$

Let us call n_k the number of operations needed to run one cycle of the multigrid algorithm. We have the recursion relation

$$n_k = (2d + 3) \times N_k + pn_{k-1}$$

and n_k can be estimated asymptotically

$$n_k \sim p^{k-1} n_1 + (2d + 3) N_k \sum_{j=1} k - 2 \left(\frac{p}{4}\right)^j$$

and if $p < 4$, we can write

$$n_k \sim (\frac{n_1}{N_1} + \frac{4\alpha}{3}) N_k$$

∗ For the full multigrid, the number of operations $\tilde{n}_k$ can also be estimated recursively by

$$\tilde{n}_k = n_k + \tilde{n}_{k-1}$$

which we solve as

$$\tilde{n}_k \sim \tilde{n}_1 + \sum_{j=2}^{k} n_j,$$

which altogether produces the result in the Proposition.

■

4.2 Algebraic Multigrid AMG

Two of the drawbacks of the geometric multigrid are 1) for complex geometries, it is not always easy to extract coarse levels, and 2) linear interpolations do not work well when confronted with nearly discontinuous coefficients (see [**?**]).

Algebraic multigrid (see [**?**]) is a method for solving linear systems based on multigrid principles, but requires no explicit knowledge of the problem geometry. AMG determines coarse grids, intergrid transfer operators, and coarse-grid equations based solely on the matrix entries. Since the method's introduction, researchers have developed numerous AMG algorithms with different robustness and efficiency properties that target a variety of problem classes.

The key points are the following :
— The smoother : in AMG, the smoother is generally fixed to be a simple pointwise method such as Gauss-Seidel. An error not eliminated by the smoother is called a smooth error, and must be handled by coarse-grid correction.
— The Coarse Grid : in AMG, the coarse grid is a subset of the fine grid. The algorithm chooses points such that the grid is coarsened in directions of strong matrix connections.
— Defining Interpolation :smooth error e is characterized by small residuals. To derive interpolation in AMG, it is taken to its extreme and it is assumed that $r_i = (Ae)_i = 0$. By rewritting this equation in terms of the coefficients of A, keeping only coarse coefficients and connected fine coefficients , an interpolation operator can be defined (for full details see [**?**])

Chapitre 5

Parallelism

5.1 Substructuring methods

Contents

Principle
— Split the domain into sub-domains,
— solve iteratively a "condensed interface problem" : at each iteration , solve independantly local problems in the subdomains (using a direct or an iterative method).
Advantages :
These methods are :
• More robust than classical iterative ones and cheaper than direct methods.
• Better adapted to distributed parallel computing with message passing programming :
→ one sub-domain per processor
→ interface data update by message passing .
• Use of sequential legacy codes for local problems, modular approach to parallelism.

5.1.1 The Schur Complement method

Consider the problem

$$\begin{cases} -\Delta u &= f \quad \text{dans } \Omega, \quad \eta \geq 0 \\ u &= 0 \quad \text{sur } \partial\Omega \end{cases}$$

We write a variational formulation in $V = H_0^1(\Omega)$:

$$\forall v \in V, \quad a(u,v) = (f,v)$$
$$\text{with } a(u,v) = \int_\Omega \nabla u \nabla v dx$$

We introduce a triangulation $\mathcal{T}_h = \cup K$ with vertices S_i, $1 \leq i \leq N$,

$$V_h = \{v \in V, \forall K \in \mathcal{T}_h, v_h|_K \in \mathbb{P}_1\}.$$

where $\mathbb{P}_n$ is the space of polynomials of degree lower than n in two variables. φ_i is the basis function associated to S_i, as described in Figure **??**. We write the linear system $KU = F$. The entries of the matrix K are the

$$a(\varphi_i, \varphi_j) = \int_\Omega \nabla\varphi_i \nabla\varphi_j \, dx.$$

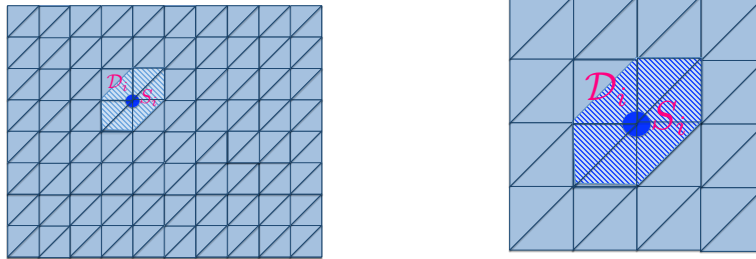The components of U are the degrees of freedom, $U_i = u_h(S_i)$, and $F_i = (f, \varphi_i)..$

FIGURE 5.1 – Mesh, $\mathcal{D}_i$, support of the basis function φ_i associated to vertex S_i.

$$a(\varphi_i, \varphi_j) = \int_{\mathcal{D}_i \cap \mathcal{D}_j} \nabla\varphi_i \nabla\varphi_j \, dx.$$

The domain Ω is split into two nonoverlapping subdomains Ω_1 and Ω_2, and Γ is the common boundary.
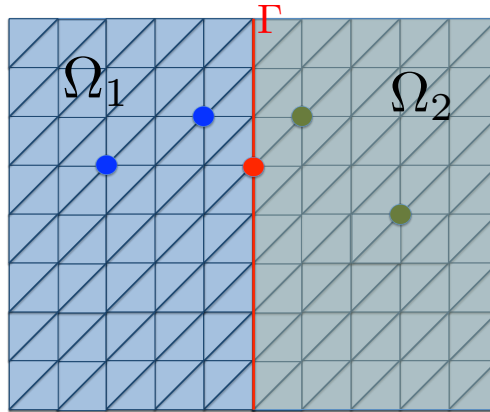
FIGURE 5.2 – Domain Decomposition

$$u_h = \sum_{S_j \in \Omega_1} u_h(S_j)\varphi_j + \sum_{S_j \in \Omega_2} u_h(S_j)\varphi_j + \sum_{S_j \in \Gamma} u_h(S_j)\varphi_j$$

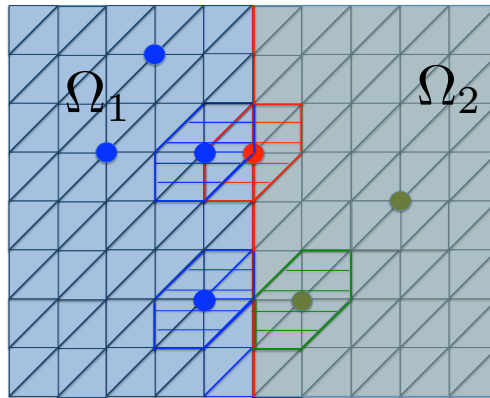$$a(\phi_i, \phi_j) = \int_{\mathcal{D}_i \cap \mathcal{D}_j} \nabla\phi_i(x) \cdot \nabla\phi_j(x)\, dx$$

FIGURE 5.3 – Supports

$$a(u_h, \varphi_l) = \sum_{S_j \in \Omega_1} u_h(S_j)a(\varphi_j, \varphi_l) + \sum_{S_j \in \Omega_2} u_h(S_j)a(\varphi_j, \varphi_l) + \sum_{S_j \in \Gamma} u_h(S_j)a(\varphi_j, \varphi_l)$$

$$S_l \in \Omega_1, \ S_j \in \Omega_2 \ \Rightarrow a(\varphi_j, \varphi_l) = 0 \Rightarrow \text{second sum vanishes}$$

$$S_l \in \Omega_2, \ S_j \in \Omega_1 \ \Rightarrow a(\varphi_j, \varphi_l) = 0 \Rightarrow \text{first sum vanishes}$$

For $S_l \in \Gamma$, all sums contribute, but for the last one, the support of S_l is split according to Figure 5.4.
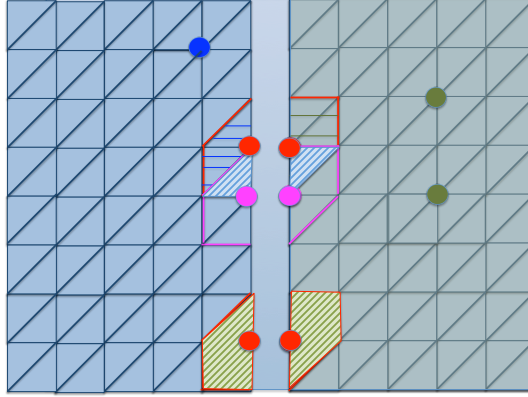
FIGURE 5.4 – Decomposition of the interface nodes

If $S_l \in \Gamma$ and $S_j \in \Gamma$ are neighbours,

$$\int_{\mathcal{D}_l \cap \mathcal{D}_j} \nabla \varphi_j \cdot \nabla \varphi_l \, dx = \int_{\mathcal{D}_l \cap \mathcal{D}_j \cap \Omega_1} \nabla \varphi_j \cdot \nabla \varphi_l \, dx + \int_{\mathcal{D}_l \cap \mathcal{D}_j \cap \Omega_2} \nabla \varphi_j \cdot \nabla \varphi_l \, dx$$

and the same for the computation of (f, φ_l). The unknown U is split into three blocks : U_1 is the block of the unknowns in the open domain Ω_1, U_2 is the block of the unknowns in the open domain Ω_2, U_3 is the block of the unknowns on the boundary Γ. The matrix K is split according to the previous formula. We shall write

$$\begin{bmatrix} K_{11} & 0 & K_{13} \\ 0 & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \end{bmatrix} \tag{5.1}$$

with $K_{33} = K_{33}^1 + K_{33}^2$ and $F_3 = F_3^1 + F_3^2$. We rewrite as a system of three systems.

$$\begin{cases} K_{11}U_1 & +K_{13}U_3 & = & F_1 \\ & K_{22}U_2 & +K_{23}U_3 & = & F_2 \\ K_{31}U_1 & +K_{32}U_2 & +K_{33}U_3 & = & F_3 \end{cases} \tag{5.2}$$

$$K_{11} = [a(\varphi_i, \varphi_j)]_{S_i, \S_j \in \Omega_1} :$$

K_{11} is the matrix of the Laplace problem in Ω_1 with homogeneous Dirichlet boundary conditions on $\partial\Omega_1$, and is therefore invertible. Solving the first equation in (5.2) amounts to solving the Laplace equation in Ω_1 with homogeneous Dirichlet boundary conditions on $\partial\Omega_1 \setminus \Gamma$, and Dirichlet data U_3 on Γ. Same for the second equation. The first two problems can be solved in U_1, U_2 knowing U_3 as

$$U_1 = (K_{11})^{-1}(F_1 - K_{13}U_3), \quad U_2 = (K_{22})^{-1}(F_2 - K_{23}U_3)$$

Carrying these values into the first equation gives

$$K_{31}(K_{11})^{-1}(F_1 - K_{13}U_3) + K_{32}(K_{22})^{-1}(F_2 - K_{23}U_3) + K_{33}U_3 = F_3.$$

$$K_{31}(K_{11})^{-1}(F_1 - K_{13}U_3) + K_{32}(K_{22})^{-1}(F_2 - K_{23}U_3) + K_{33}U_3 = F_3.$$

$$SU_3 = (K_{33} - K_{31}K_{11}^{-1}K_{13} - K_{32}K_{22}^{-1}K_{23})U_3 = G_3$$
$$\text{with } G_3 = F_3 - K_{31}K_{11}^{-1}F_1 - K_{32}K_{22}^{-1}F_2$$

Definition 5.1 *The matrix* $S = K_{33} - K_{31}K_{11}^{-1}K_{13} - K_{32}K_{22}^{-1}K_{23}$ *is the Schur Complement matrix.*

Theorem 5.1 *The matrix S est symmetric, positive, definite.*

It will be computed in parallel as
$$S = S^1 + S^2$$
with
$$S^i \quad = \quad K_{33}^i - K_{3i}K_{ii}^{-1}K_{i3}$$

Then the interface problem will be solved with direct or parallel methods.

The first two equations in (5.2) is the resolution of Laplace equations. But what is the third one ?

$$K_{31}U_1 + K_{32}U_2 + K_{33}U_3 - F_3 = 0 \tag{5.3}$$

Suppose w is a "regular" solution of $-\Delta w = f$ in Ω_1. By the Green formula we have for any v in $H^{1/2}(\Gamma)$,

$$< \frac{\partial w}{\partial n_1}, v >_{\partial\Omega} = (\nabla w, \nabla v) + (\Delta w, v) = a_1(w, v) - (f, v)_1$$

We apply this to $w = (u_1)_h$, and $v = \varphi_i$, with $S_i \in \Gamma$, and obtain

$$
\begin{aligned}
< \tfrac{\partial (u_1)_h}{\partial n_1}, \varphi_i >_\Gamma &= a_1((u_1)_h, \varphi_i) - (f, \varphi_i)_1 \\
&= \sum_{S_j \in \Omega_1} (u_1)_h(S_j) a_1(\varphi_j, \varphi_i) + \sum_{S_j \in \Gamma} (u_1)_h(S_j) a_1(\varphi_j, \varphi_i) - (f, \varphi_i)_1
\end{aligned}
$$

$$[< \frac{\partial (u_1)_h}{\partial n_1}, \varphi_i >_\Gamma] = K_{31}U_1 + K_{33}^1 U_3 - F_3^1 = S^1 U_3 - F_3^1, \quad \text{with } K_{11}U_1 + K_{13}U_3 = F_1.$$

Now we have in (5.3)

$$K_{11}U_1 + K_{13}U_3 = F_1$$

$$K_{22}U_2 + K_{23}U_3 = F_2$$

$$SU_3 - F_3 = K_{31}U_1 + K_{32}U_2 + K_{33}U_3 - F_3 = [< \frac{\partial (u_1)_h}{\partial n_1} + \frac{\partial (u_2)_h}{\partial n_2}, \varphi_i >_\Gamma] = 0$$

The full substructuring method can now be understood as the finite element discretization of : find g defined on the interface Γ such that, defining u_1 and u_2 as the solutions of

$$-\Delta u_j = f \text{ in } \Omega_j,$$

$$u_j = 0 \text{ on } \partial\Omega_j - \Gamma,$$

$$u_j = g \text{ on } \Gamma$$

then

$$\frac{\partial u_1}{\partial n_1} + \frac{\partial u_2}{\partial n_2} = 0 \text{ on } \Gamma.$$

The resolution of the interface problem can be solved either by a direct method, or by a Krylov method.

5.1.2 Direct method for the resolution of the interface problem

We work on system (5.1), and write a block-LU decomposition of K as follows

$$\left(\begin{array}{c|c|c} K_{11} & 0 & K_{13} \\ \hline 0 & K_{22} & K_{23} \\ \hline K_{31} & K_{32} & K_{33} \end{array}\right) = \left(\begin{array}{c|c|c} L_{11} & 0 & 0 \\ \hline 0 & L_{22} & 0 \\ \hline L_{31} & L_{32} & L_{33} \end{array}\right) \left(\begin{array}{c|c|c} U_{11} & 0 & U_{13} \\ \hline 0 & U_{22} & U_{23} \\ \hline 0 & 0 & U_{33} \end{array}\right) \quad (5.4)$$

We identify

$$K_{11} = L_{11}U_{11}; \quad K_{13} = L_{11}U_{13},$$

$$K_{22} = L_{22}U_{22}; \quad K_{23} = L_{22}U_{23},$$

$$K_{31} = L_{31}U_{11}; \quad K_{32} = L_{32}U_{22}; \quad K_{33} = L_{31}U_{13} + L_{32}U_{23} + L_{33}U_{33}$$

Notice that $L_{3i}U_{i3} = K_{3i}K_{ii}^{-1}K_{i3}$, therefore $K_{33} - L_{31}U_{13} - L_{32}U_{23} = S$, and $S = L_{33}U_{33}$. The computations are made in parallel on the processors :

PROCESSOR (i)

Computation and storage of K_{ii}, K_{i3},
Computation of F^i and F_3^i

Decomposition $L_{ii}U_{ii}$ de K_{ii},
Computation of U_{i3}, L_{3i},
Computation of $S^i = K_{33}^i - L_{3i}U_{i3}$

ASSEMBLING

Computation of $S = S^1 + S^2$ and $F_3 = F_3^1 + F_3^2$,
Decomposition $L_{33}U_{33}$ of S.

We then solve the triangular problems

$$\left(\begin{array}{c|c|c} L_{11} & 0 & 0 \\ \hline 0 & L_{22} & 0 \\ \hline L_{31} & L_{32} & L_{33} \end{array}\right) \left(\begin{array}{c} Z_1 \\ Z_2 \\ Z_3 \end{array}\right) = \left(\begin{array}{c} F_1 \\ F_2 \\ F_3 \end{array}\right), \quad \left(\begin{array}{c|c|c} U_{11} & 0 & U_{13} \\ \hline 0 & U_{22} & U_{23} \\ \hline 0 & 0 & U_{33} \end{array}\right) \left(\begin{array}{c} X_1 \\ X_2 \\ X_3 \end{array}\right) = \left(\begin{array}{c} Z_1 \\ Z_2 \\ Z_3 \end{array}\right)$$

PROCESSEUR (i)
$L_{ii}z_i = F_i$, $G_3^i = F_3^i - L_{3i}Z_i$
ASSEMBLING
$L_{33}Z_3 = G_3^1 + G_3^2$
$U_{33}X_3 = Z_3$
PROCESSOR (i)
$U_{ii}X_i = Z_i - U_{i3}X_3$

5.1.3 The conjugate gradient algorithm

$$SU_3 := K_{33}U_3 - K_{31}K_{11}^{-1}K_{13}U_3 - K_{32}K_{22}^{-1}K_{23}U_3 = G_3$$

$$\text{with } G_3 = F_3 - K_{31}K_{11}^{-1}F_1 - K_{32}K_{22}^{-1}F_2$$

S is a symmetric positive definite matrix. The conjugate gradient algorithm reduces to a descent method,defined by the initial guess U_3^0,the initial descent direction $d^0 = r^0 = SU_3^0 - G_3$. Let r^k be the residual a step k. The next step will be

$$v^k = Sd^k$$
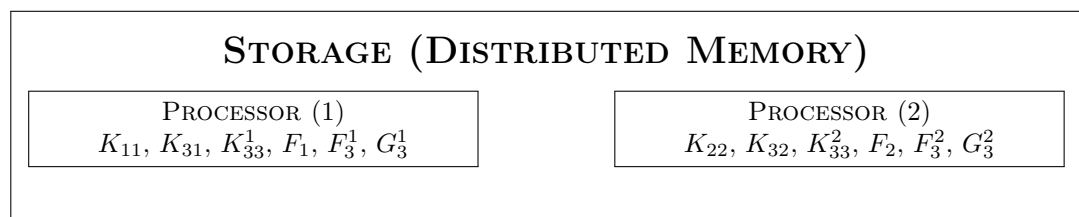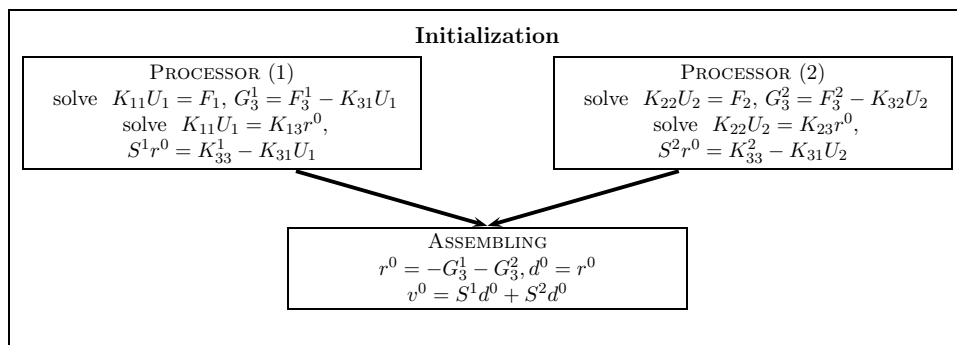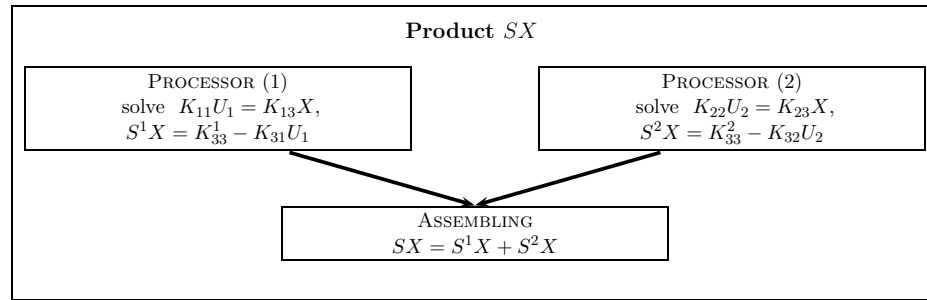$$\rho^k = \frac{\|r^k\|^2}{(v^k, d^k)}$$
$$U_3^{k+1} = U_3^k - \rho^k d^k$$
$$r^{k+1} = r^k - \rho^k v^k$$
$$d^{k+1} = r^{k+1} + \frac{\|r^{k+1}\|^2}{\|r^k\|^2}d^k$$

All the products have to be made in parallel. Let us go into details.
For the initialization choose $U_3^0 = 0$, thus $r^0 = -G_3 = -F_3 + K_{31}K_{11}^{-1}F_1 + K_{32}K_{22}^{-1}F_2$.

We define a special box for the product SX :

Product SX

PROCESSOR (1)
solve $K_{11}U_1 = K_{13}X$,
$S^1X = K_{33}^1 - K_{31}U_1$

PROCESSOR (2)
solve $K_{22}U_2 = K_{23}X$,
$S^2X = K_{33}^2 - K_{32}U_2$

ASSEMBLING
$SX = S^1X + S^2X$

Initialization

PROCESSOR (1)
solve $K_{11}U_1 = F_1$, $G_3^1 = F_3^1 - K_{31}U_1$
solve $K_{11}U_1 = K_{13}r^0$,
$S^1r^0 = K_{33}^1 - K_{31}U_1$

PROCESSOR (2)
solve $K_{22}U_2 = F_2$, $G_3^2 = F_3^2 - K_{32}U_2$
solve $K_{22}U_2 = K_{23}r^0$,
$S^2r^0 = K_{33}^2 - K_{31}U_2$

ASSEMBLING
$r^0 = -G_3^1 - G_3^2, d^0 = r^0$
$v^0 = S^1d^0 + S^2d^0$

STORAGE (DISTRIBUTED MEMORY)

PROCESSOR (1)
$K_{11}, K_{31}, K_{33}^1, F_1, F_3^1, G_3^1$

PROCESSOR (2)
$K_{22}, K_{32}, K_{33}^2, F_2, F_3^2, G_3^2$

$$
\boxed{
\begin{array}{c}
\textbf{ITERATION} \\[6pt]
v^k = S d^k \\[6pt]
\rho^k = \dfrac{\|r^k\|^2}{(v^k, d^k)} \\[6pt]
U_3^{k+1} = U_3^k - \rho^k d^k \\[4pt]
r^{k+1} = r^k - \rho^k v^k \\[6pt]
d^{k+1} = r^{k+1} + \dfrac{\|r^{k+1}\|^2}{\|r^k\|^2} d^k
\end{array}
}
$$

Note that the scalar products can also be done partly in parallel.

5.1.4   Interest of subtructuring

- The interface problem has n unknowns when the full problem has n^2 unknowns.
- It can be proved that the interface problem is much better conditioned than the full problem.
- Therefore the conjugate gradient algorithm converges rapidly.
- Futhermore most part of computation part can be made in parallel.

5.1.5 The Dirichlet Neumann algorithm

The purpose of the algorithm is to solve the coupling problem

$$\mathcal{L}u = f \text{ on } \Omega,$$
$$u = 0 \text{ on } \partial\Omega$$

by splitting Ω into two subdomains with interface Γ, and solving iteratively with an initial guess g_0,

Presentation of the algorithm

$$\begin{cases} \mathcal{L}u_1^n = f \text{ in } \Omega_1, \\ \\ u_1^n = 0 \text{ on } \partial\Omega \cup \overline{\Omega_1}, \quad u_1^n = g^n \text{ on } \Gamma. \end{cases}$$

$$\begin{cases} \mathcal{L}u_2^n = f \text{ in } \Omega_2, \\ \\ u_2^n = 0 \text{ on } \partial\Omega \cup \overline{\Omega_2}, \quad \dfrac{\partial u_2^n}{\partial\nu} = \dfrac{\partial u_1^n}{\partial\nu} \text{ on } \Gamma. \end{cases}$$

where $\dfrac{\partial}{\partial\nu}$ in Ω_2 is the normal derivative, with ν the exterior normal to Ω_2.

$$g^{n+1} = \theta u_2^n + (1-\theta)g^n.$$

The choice of the parameter is crucial and unfortunately depends on the position of the interface. If the subdomains and the problems are symmetric, the choice $\theta = \frac{1}{2}$ is optimal.

Convergence analysis in one dimension

Let $\mathcal{L} = \eta - d_x^2$, $\Omega = (a, b)$. Take c in (a, b). Then we have $\dfrac{\partial}{\partial\nu} = -\dfrac{d}{dx}$ on the interface at point c.

Define the error in the subdomain, $e_j^n = u_j^n - u$, and $h^n = g^n - u(c)$. The algorithm for the error is

$$\begin{cases} \mathcal{L}e_1^n = 0 \text{ in } \Omega_1, \\ \\ e_1^n = 0 \text{ on } \partial\Omega \cup \overline{\Omega_1}, \quad e_1^n = h^n \text{ on } \Gamma. \end{cases}$$

$$\begin{cases} \mathcal{L}e_2^n = 0 \text{ in } \Omega_2, \\ \\ e_2^n = 0 \text{ on } \partial\Omega \cup \overline{\Omega_2}, \quad \dfrac{\partial e_2^n}{\partial\nu} = \dfrac{\partial e_1^n}{\partial\nu} \text{ on } \Gamma. \end{cases}$$

$$h^{n+1} = \theta e_2^n(c) + (1-\theta)h^n.$$

This can be solved as

$$e_1^n = h^n \frac{\text{sh}(\sqrt{\eta}(x-a))}{\text{sh}(\sqrt{\eta}(c-a))}, \quad e_2^n = \beta^n \text{sh}(\sqrt{\eta}(b-x)).$$

The coefficient β^n is determined by the transmission condition $d_x e_2^n(c) = d_x e_1^n(c)$, that gives

$$-\beta^n \text{ch}(\sqrt{\eta}(b-c) = h^n \frac{\text{ch}(\sqrt{\eta}(c-a))}{\text{sh}(\sqrt{\eta}(c-a))}$$

$$h^{n+1} = (\underbrace{-\theta \frac{\text{sh}(\sqrt{\eta}(b-c))\text{ch}(\sqrt{\eta}(c-a))}{\text{sh}(\sqrt{\eta}(c-a))\text{ch}(\sqrt{\eta}(b-c))} + (1-\theta)}_{\text{Convergence factor } \rho}) \, h^n.$$
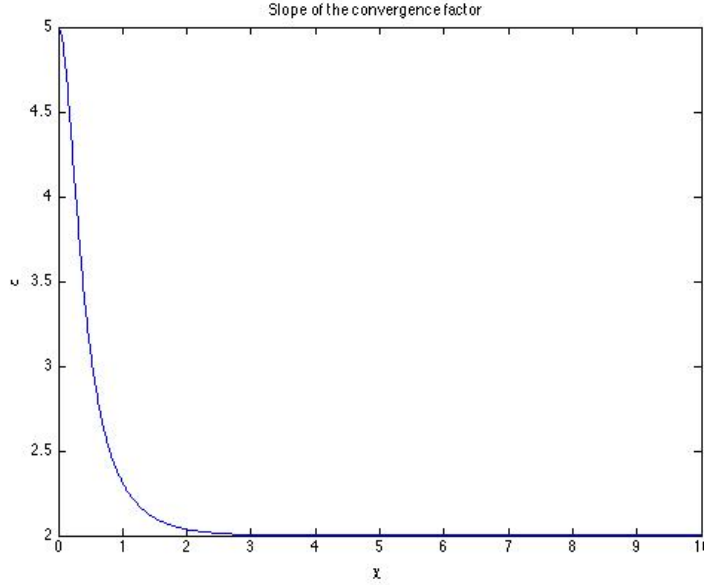
88

If the geometry is symmetric, that is if $b - c = c - a$, then the convergence factor reduces to

$$\rho = 1 - 2\theta,$$

that is smaller than 1 for $\theta \in (0, 1)$, and vanishes for $\theta = 1/2$. Suppose now that $(c - a) = (b - a)/5$. Then defining $\chi = \sqrt{\bar{\eta}}/5$, then

$$\rho = 1 - \theta\left(\frac{\tanh(4\chi)}{\tanh(\chi)} + 1\right).$$

It is a linear function of θ, with a slope $\alpha = -\left(\frac{\tanh(4\chi)}{\tanh(\chi)} + 1\right) \in (-5, -2)$.

Slope of the convergence factor

Therefore ρ is an decreasing function of θ, and it is equal to 1 for $\theta = \theta_0$, with

$$\theta_0 = \frac{2}{\frac{\tanh(4\chi)}{\tanh(\chi)} + 1} \in \left(\frac{2}{5}, 1\right).$$

Then the algorithm is convergent if and only if $\theta \leq \theta_0$.

5.1.6 Appendix : `matlab` scripts in 1-D

```matlab
function u=SolveDD(f,eta,a,b,ga,gb)
% SOLVEDD solves eta-Delta in 1d using finite differences
%    u=SolveDD(f,eta,a,b,ga,gb,n) solves the one dimensional equation
%    (eta-Delta)u=f on the domain Omega=(a,b) with Dirichlet boundary
%    conditions u=ga at x=a and u=gb at x=b using a finite
%    difference approximation with length(f) interior grid points

J=length(f);
h=(b-a)/(J+1);
% construct 1d finite difference operator
e=ones(J,1);
A=spdiags([-e/h^2 (eta+2/h^2)*e -e/h^2],[-1 0 1],J,J);
f(1)=f(1)+ga/h^2;          % add boundary conditions into rhs
f(end)=f(end)+gb/h^2;
u=A\f;
u=[ga;u;gb];               % add boundary values to solution
```

```matlab
function u=SolveND(f,eta,a,b,ga,gb)
% SOLVEND solves eta-Delta in 1d using finite differences
%    u=SolveND(f,eta,a,b,ga,gb) solves the one dimensional equation
%    (eta-Delta)u=f on the domain Omega=(a,b) with Neumann boundary
%    condition u'=ga at x=a and Dirichlet boundary
%    condition u=gb at x=b using a finite
%    difference approximation.
%    note the second order appproximation of the derivative

J=length(f);
h=(b-a)/J;
% construct 1d finite difference operator
e=ones(J,1);
A=spdiags([-e/h^2 (eta+2/h^2)*e -e/h^2],[-1 0 1],J,J);
A(1,2)=2*A(1,2);  %% Neumann boundary condition
% construct 1d finite difference operator
f(1)=f(1)-2*ga/h;          % add boundary conditions into rhs
f(end)=f(end)+gb/h^2;
u=A\f;
u=[u;gb];                  % add boundary value to solution on the right
```

```matlab
function [g,u1,u2]=algoDN(f,eta,a,b,step,ga,gb,g1,Nc,Imax,t)
% algoDN solves the Laplace equation by the Dirichlet—Neumann algorithm
%[g,u1,u2]=algoDN(f,eta,a,b,step,ga,gb,g,Nc,Imax,t)
%solves the Laplace equation eta u —Delta u = f in (a,b)
% by the Dirichlet—Neumann algorithm on (a+Nc*step) and (Nc*step,c)
% note the second order reconstruction of   u_1'(c)
 g=zeros(1,Imax);
g(1)=g1;
c=a+Nc*step;
x=(a:step:b);x1=(a:step:c); x2=(c:step:b);
y= SolveDD(f',eta,a,b,ga,gb);
for j=1:Imax—1
    % Dirichlet on (a,c)
    f1=f(1:Nc—1);
    u1=SolveDD((f1)',eta,a,c,ga,g(j));
    %extraction de u_1'(c) : second order
    up1= (—u1(end—1)+(1+eta*step^2/2)*u1(end))/step—step*f(Nc)/2;
     % Neumann on (c,b) with u_2'(c)=u_1'(c)
    f2=f(Nc:end);
    u2=SolveND((f2)',eta,c,b,up1,gb);
    g(j+1)=(1—t)*g(j)+t*u2(1);
    h=figure
    plot(x1,u1,'b',x2,u2,'m',x,y,'r',c,linspace(u1(end),u2(1),100),'k');
    legend('u_1','u_2','solution discrete')
    title({['Algorithme de Dirichlet—Neumann',' c=',num2str(c),'\theta=',
        num2str(t)];...
        ['Iteration number ',int2str(j)]})
        filename = ['figDNpos' int2str(Nc) 'relax' num2str(t) 'iter' int2str
            (j) '.eps']
        print(h,'—depsc',filename)

    pause%(1)
end
```

```matlab
function u=algoSchur(f,eta,a,b,h,ga,gb,Nc)
% algoSchur solves the Laplace equation by the Schur method
%[g,u1,u2]=algoSchur(f,eta,a,b,step,ga,gb,Nc)
%solves the Laplace equation eta u —Delta u = f in (a,b)
% by the Schur method m on (a+Nc*h) and (Nc*h,c)
J=length(f);
e=ones(J,1);
A=spdiags([—e/h^2 (eta+2/h^2)*e —e/h^2],[—1 0 1],J,J);
% decomposition of A
A11=A(1:Nc—1,1:Nc—1);
A22=A(Nc+1:end,Nc+1:end);
A1g=A(1:Nc—1,Nc);
Ag1=A(Nc,1:Nc—1);
A2g=A(Nc+1:end,Nc);
Ag2=A(Nc,Nc+1:end);
Agg=A(Nc,Nc);
%decomposition of f
f1=f(1:Nc—1);
f2=f(Nc+1:end);
fg=f(Nc);
% Construction of the Schur problem
funS=@(x) Agg*x—Ag1*(A11\(A1g*x))—Ag2*(A22\(A2g*x));
fS=fg—Ag1*(A11\f1)—Ag2*(A22\f2);
ug=pcg(funS,fS)
%reconstruct u1 and u2
u1=A11\(f1—A1g*ug)
u2=A22\(f2—A2g*ug)
%reconstruct u
u=[ga; u1 ; ug ; u2 ; gb];
```

```matlab
clear all;close all;
% Validation of the Dirichlet and Neumann codes
a=0;
b=1;
Step=(b-a)*0.1./10.^(0:2);
for j=1:length(Step)
    step=Step(j);
    x=(a:step:b);
    y=sin(pi*x);
    eta=1;
    f=(eta+pi^2)*y(2:end-1);
    ga=0;gb=0;
    sol=SolveDD(f',eta,a,b,ga,gb);
    X=a:step/100:b;
    Y=sin(pi*X);
    figure(1)
    plot(x,sol,'b',X,Y,'r');
    hold on

    e1d(j)=max(abs(sol-y'));
    f=(eta+pi^2)*y(1:end-1);
    ga=pi;
    sol1=SolveND(f',eta,a,b,ga,gb);
    plot(x,sol1,'b',X,Y,'r');

    e1n(j)=max(abs(sol1-y'));
    figure(2)
    plot(x,sol1-y');
        pause
end

figure(3)
loglog(Step,e1d,'m*-')
hold on
loglog(Step,e1n,'bo-')
hold on
loglog(Step,Step.^2,'r')
legend('Dirichlet','Neumann','slope 2')


% Algorithme de Dirichlet Neumann sur (a,c), (c,b)
clear all; close all;
a=0;
b=1;
J=9;
h=(b-a)/(J+1);
x=(a:h:b);
% eta=1;
% y=x.^3;
% f=-6*x(2:end-1)+eta*y(2:end-1);
% ga=0;gb=1;
eta=1;
y=sin(pi*x);
f=(eta+pi^2)*y(2:end-1);
ga=0;gb=0;
```

```
56  sol=SolveDD(f',eta,a,b,ga,gb);
57  % position de l interface
58  Nc=floor(length(x)/2);
59  Nc=2;
60  c=a+Nc*h;
61  % nombre d'iterations
62  Imax=10;
63  %parametre de relaxation
64  t=0.5;
65  % initialisation avec la valeur exacte
66  g1=y(Nc+1);
67  % ou initialisation avec 0
68  g1=0;
69  [g,u1,u2]=algoDN(f,eta,a,b,h,ga,gb,g1,Nc,Imax,t)
70  % algorithme
71  figure(99)
72  plot(g)
73  title('Interface value')
74  xlabel('Iteration number')
75
76  % Methode de Schur
77  u=algoSchur(f',eta,a,b,h,ga,gb,Nc);
78  %plot(x,y,'r',x,yd,'g',x,u,'b')
79  figure(55)
80  plot(x,sol,'g',x,u,'b')
81
82
83  %%
84  N=10;
85  chi=linspace(0,N,N*100)
86  Y=tanh(4*chi)./tanh(chi)+1;
87  plot(chi,Y,'b')
88  xlabel('\chi')
89  ylabel('\alpha')
90  title('Slope of the convergence factor')
```

5.2 Schwarz Algorithms

5.2.1 Introduction and a brief historical review

Schwarz method was brought about by H.A.Schwarz around 1870 to prove the existence of harmonic functions in open sets that were not disks or rectangles (as those cases had been dealt with analytically), or obtained from those above by conformal transformation. Schwarz's typical problem is the following (in the following, $\mathcal{L}$ can be any elliptic operator , bus mostly our favorite $-\Delta$).

$$
\begin{aligned}
\mathcal{L}(u) &= 0, & x \in \Omega \\
u(x) &= g(x), & x \in \partial\Omega
\end{aligned}
\tag{5.5}
$$

in domain Ω defined in figure 5.5.

The classical Schwarz iteration consists in solving one after the other the problem in each sub-domain Ω_1 and Ω_2 defined in figure 5.6
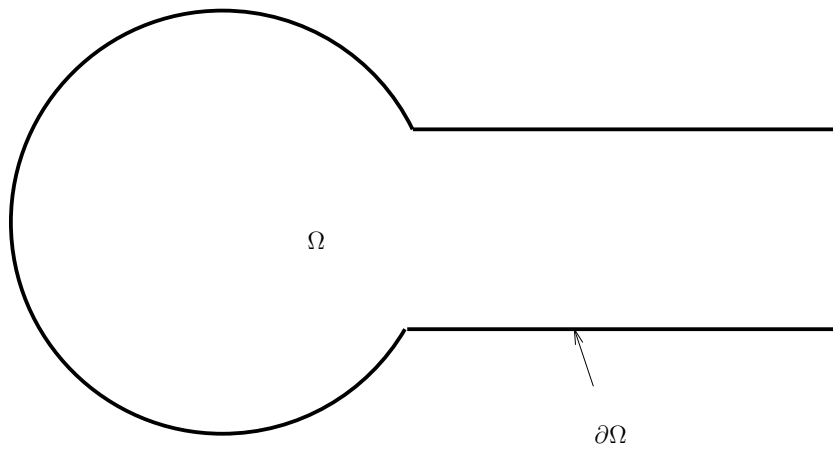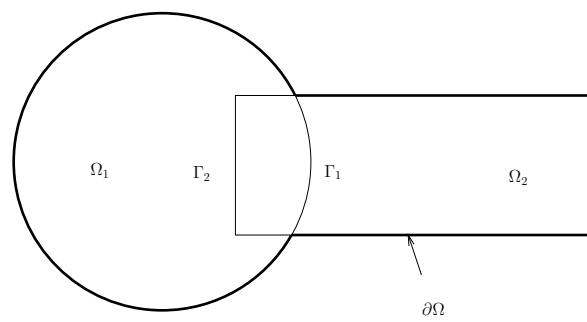
FIGURE 5.5 – The initial Schwarz domain

FIGURE 5.6 – Schwarz Decomposition

$$\begin{cases} \mathcal{L}(v^{n+1}) & = & f(x), & x \in \Omega_1 \\ v^{n+1}(x) & = & g(x), & x \in \partial\Omega \cap \overline{\Omega_1} \\ v^{n+1}(x) & = & w^n(x), & x \in \Gamma_1 \end{cases} \tag{5.6}$$

$$\begin{cases} \mathcal{L}(w^{n+1}) & = & f(x), & x \in \Omega_2 \\ w^{n+1}(x) & = & g(x), & x \in \partial\Omega \cap \overline{\Omega_2} \\ w^{n+1}(x) & = & v^{n+1}(x), & x \in \Gamma_2 \end{cases} \tag{5.7}$$

Schwarz showed that the sequence (v^n, w^n) converged using the maximum principle. The converged value is thus a solution u of (5.5). In 1988, this method was modified by P.L. Lions in a series of papers presented at Domain Decomposition Conferences in order to make it parallel in the following way :

$$\begin{cases} \mathcal{L}(v^{n+1}) & = & f(x), & x \in \Omega_1 \\ v^{n+1}(x) & = & g(x), & x \in \partial\Omega \cap \overline{\Omega_1} \\ v^{n+1}(x) & = & w^n(x), & x \in \Gamma_1 \end{cases} \tag{5.8}$$

$$\begin{cases} \mathcal{L}(w^{n+1}) & = & f(x), & x \in \Omega_2 \\ w^{n+1}(x) & = & g(x), & x \in \partial\Omega \cap \overline{\Omega_2} \\ w^{n+1}(x) & = & v^n(x), & x \in \Gamma_2 \end{cases} \tag{5.9}$$

He also put it in a hilbertian frame more adapted to using numerical methods. It was then extended to general geometrical configurations , with an arbitrary number of sub-domains, and to more general equations. He also suggested modifying transmission conditions between sub-domains, using Robin type or even boundary elements operators.

The principle of this approach is thus :
— an iterative method,
— N sub-domains,
— non zero overlapping between sub-domains,
— well posed problems in each sub-domain,
— convergence of the algorithm (as a function of the overlapping),
— easy to implement

Today, for most problems, solution existence and unicity in known in the initial domain. If a Domain Decomposition method is used, it is mainly for reasons of data storage and use of fast local solvers.

In the sequel, we will refer to (5.6,5.7) as alternate Schwarz, and (5.8,5.9) as parallel Schwarz.

5.2.2   A very simple 1D example

Lets us consider the following problem

$$\begin{cases} u - \dfrac{d^2u}{dx^2} = f, & x \in \mathbb{R}, \\ u \to 0 \text{ as } x \to \pm\infty \end{cases} \tag{5.10}$$

where f is a "nice" function. For example if f belongs to $L^2(\mathbb{R})$, it is well known that there exists a unique variational solution in $H^1(\mathbb{R})$ which furthermore belongs to $H^2(\mathbb{R})$.

Let us look at the Schwarz parallel algorithm with two sub-domains (see figure 5.7) :)
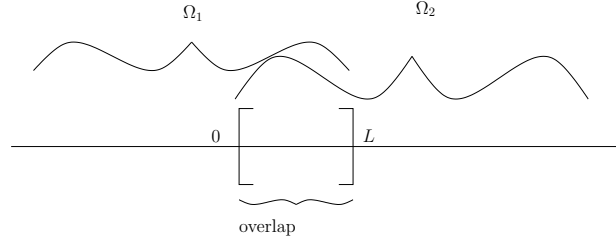
FIGURE 5.7 – 1D decomposition in 2 sub-domains with overlap

To initialize the iterative process, (v^0, w^0) is given, then at iteration n is solved

$$\begin{cases} v^n - \dfrac{d^2 v^n}{dx^2} = f, & x \in]-\infty, L[\\ v^n \to 0 \text{ as } x \to -\infty \\ v^n(L) = w^{n-1}(L), \end{cases} \tag{5.11}$$

$$\begin{cases} w^n - \dfrac{d^2 w^n}{dx^2} = f, & x \in]L, +\infty[\\ w^n \to 0 \text{ as } x \to +\infty \\ w^n(0) = v^{n-1}(0), \end{cases} \tag{5.12}$$

Theorem 5.2 *The algorithm is well defined in $H^1(-\infty, L) \times H^1(0, +\infty)$.*

Exercice 5.1 *Proof is left as an exercise.*

The error in each sub-domain is now defined $V^n = v^n - u \in H^1(-\infty, L)$, $W^n = w^n - u \in H^1(0, +\infty)$. V^n (resp. W^n) is solution to an homogeneous problem, that is for $f = 0$. It is solution to an homogeneous differential equation and goes to 0 at $-\infty$ (resp. at $+\infty$). It can thus be easily expressed up to a multiplicative constant :

$$V^n = a_n e^x, \quad W^n = b_n e^{-x} \tag{5.13}$$

with the recurrence relation on the coefficients

$$a_n e^L = b_{n-1} e^{-L}, \quad b_n = a_{n-1} \tag{5.14}$$

thus

$$\begin{cases} a_{2k} = (e^{-2L})^k a_0 \\ b_{2k} = (e^{-2L})^k b_0 \end{cases} \begin{cases} a_{2k+1} = (e^{-2L})^{k+1} b_0 \\ b_{2k+1} = (e^{-2L})^k a_0 \end{cases} \tag{5.15}$$

The L^2 norm of the error can be computed. For example

$$||v^{2k} - u||_{L^2(]-\infty, L[)} = (e^{-L})^{2k} ||v^0 - u||_{L^2(]-\infty, L[)} \tag{5.16}$$

Theorem 5.3 *The Domain Decomposition algorithm with Dirichlet transmission conditions converges linearly with a linear coefficient equal to e^{-L}.*

97

Alas this means that **the smaller the overlap, the slower the convergence will be.**

If *Dirichlet* conditions are replaced by *Neumann* conditions or a combination of both, the same analysis is made.

How can one make convergence independant of the overlap?

Let us try a *Robin* condition:

$$\begin{cases} (\dfrac{d}{dx} + \alpha)v^n & = & (\dfrac{d}{dx} + \alpha)w^{n-1}, & x = L \\ (\dfrac{d}{dx} - \beta)w^n & = & (\dfrac{d}{dx} - \beta)v^{n-1}, & x = 0. \end{cases} \tag{5.17}$$

Let us start the analysis again.

Theorem 5.4 *For any α and β strictly positive, **for** $L \geq 0$, the algorithm associated to the (5.17) transmission conditions is well defined in $H^1(-\infty, L) \times H^1(0, +\infty)$.*

Exercice 5.2 *Prove the theorem by writing the variational formulation.*

Back to (5.13). We now have

$$a_n = \frac{\alpha - 1}{\alpha + 1} \, e^{-2L} b_{n-1}, \quad b_n = \frac{\beta - 1}{\beta + 1} \, a_{n-1} \tag{5.18}$$

Theorem 5.5 *For any α and β strictly positive, for any overlap $L \geq 0$, the algorithm associated to the (5.17) transmission conditions converges linearly. It converges in two iterations if and only if $\alpha = \beta = 1$.*

Exercice 5.3 *Prove the theorem using the proof of 5.3.*

These transmission conditions are said to be exact. This is because the solution v to the left hand problem satisfies the transmission condition which is imposed on the right hand side.

This approach will now be generalized to higher dimensions.

5.2.3 A 2D, 3D tool : the Fourier transform. Optimal transmission condition

The following notes are 2 dimensional but are easily extendable to 3D and more. Let u be a function of two variables x and y. Variable y lives on the whole real axis. The partial Fourier transform with respect to y is defined as:

$$\hat{u}(x, k) = \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} u(x, y) e^{-iky} dy$$

If u is in L^2, then $\hat{u}$ is in L^2, the inverse formula and the Plancherel theorem can be used. Derivation formulae are obtained:

$$\widehat{\frac{\partial^p u}{\partial y^p}}(x, k) = (ik)^p \hat{u}(x, k)$$

Let us consider the 2D operator $-\Delta + I$. For f in $L^2(\mathbb{R}^2)$, the solution u of

$$u - \Delta u = f, \quad x \in \mathbb{R}^2, \tag{5.19}$$

and let the Domain Decomposition be given by figure 5.8.

Let us once again look at the parallel Schwarz algorithm with Dirichlet boundary conditions. A y Fourier transform is carried out. The errors are denoted V^n in Ω_1 and
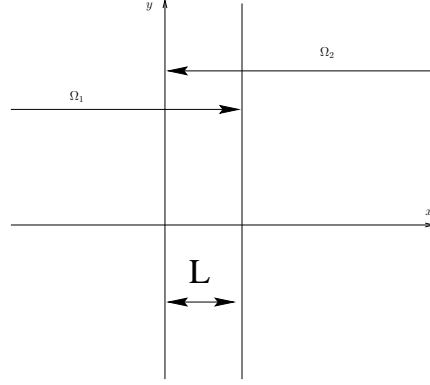
FIGURE 5.8 – 2D decomposition in 2 sub-domains with overlap

W^n in Ω_2. Their Fourier transform are solution to the ordinary differential equation with variable x and k appears as a parameter.

$$-\frac{\partial^2 \hat{U}}{\partial x^2} + (k^2 + 1)\hat{U} = 0 \tag{5.20}$$

leading to

$$V^n = a_n(k)e^{\sqrt{k^2+1}x}, \quad W^n = b_n(k)e^{-\sqrt{k^2+1}x} \tag{5.21}$$

with the recurrence relation on the coefficients

$$a_{n+1}(k) = b_n(k)e^{-2\sqrt{k^2+1}L}, \quad b_{n+1}(k) = a_n(k) \tag{5.22}$$

thus

$$\begin{cases} a_{2p}(k) = (e^{-2\sqrt{k^2+1}L})^p a_0 \\ b_{2p}(k) = (e^{-2\sqrt{k^2+1}L})^p b_0 \end{cases} \begin{cases} a_{2p+1}(k) = (e^{-2\sqrt{k^2+1}L})^{p+1} b_0 \\ b_{2p+1}(k) = (e^{-2\sqrt{k^2+1}L})^p a_0 \end{cases} \tag{5.23}$$

Exercice 5.4 *Show that the parallel Schwarz algorithm with overlap converges linearly.*

By analogy with the one dimensional case , it is to be noticed that V^n and W^n satisfy the following equations

$$\begin{cases} \dfrac{\partial \hat{V}^n}{\partial x} - \sqrt{k^2+1}\,\hat{V}^n = 0 \text{ in } (-\infty, L) \\[4mm] \dfrac{\partial \hat{W}^n}{\partial x} + \sqrt{k^2+1}\,\hat{W}^n = 0 \text{ in } (0, +\infty) \end{cases} \tag{5.24}$$

and the following theorem

Theorem 5.6 *The 2D Domain Decomposition algorithm converges in two iterations if transmission conditions in Fourier variables can be written as*

$$\begin{cases} \dfrac{\partial \hat{V}^{n+1}}{\partial x} + \sqrt{k^2+1}\,\hat{V}^{n+1} = \dfrac{\partial \hat{W}^n}{\partial x} + \sqrt{k^2+1}\,\hat{W}^n, \quad x = L \\[4mm] \dfrac{\partial \hat{W}^{n+1}}{\partial x} - \sqrt{k^2+1}\,\hat{W}^{n+1} = \dfrac{\partial \hat{V}^n}{\partial x} - \sqrt{k^2+1}\,\hat{V}^n, \quad x = 0 \end{cases} \tag{5.25}$$

It remains to find a meaning to these transmission conditions which are expressed as square roots of operators, which are not easily dealt with. An approximation will be performed.

5.2.4 Approximation

Let us go back to the two sub-domain case with overlap in 5.10. The exact transmission conditions are given by (5.25). These conditions will be "approximated" in such a way to fit a differential operator. Let us consider the Schwarz algorithm with the following general transmission conditions

$$\begin{cases} \dfrac{\partial \hat{w}^{n+1}}{\partial x} - \varphi(k)\,\hat{w}^{n+1} = \dfrac{\partial \hat{v}^n}{\partial x} - \varphi(k)\,\hat{v}^n, & x = L \\[4mm] \dfrac{\partial \hat{v}^{n+1}}{\partial x} + \psi(k)\,\hat{v}^{n+1} = \dfrac{\partial \hat{w}^n}{\partial x} + \psi(k)\,\hat{w}^n, & x = 0 \end{cases} \tag{5.26}$$

Let us continue the Fourier analysis from (5.13).

$$a_{n+1}(k) = \frac{\psi(k) - \sqrt{k^2+1}}{\psi(k) + \sqrt{k^2+1}}\, e^{-2L\sqrt{k^2+1}}b_n(k), \quad b_{n+1}(k) = \frac{\varphi(k) - \sqrt{k^2+1}}{\varphi(k) + \sqrt{k^2+1}}\,a_n(k) \tag{5.27}$$

The convergence factor over a double swap is defined as :

$$\rho(k, L) = \frac{\psi(k) - \sqrt{k^2+1}}{\psi(k) + \sqrt{k^2+1}}\,\frac{\varphi(k) - \sqrt{k^2+1}}{\varphi(k) + \sqrt{k^2+1}}\, e^{-2L\sqrt{k^2+1}}. \tag{5.28}$$

and we have

$$a_{n+1}(k) = \rho(k, L)a_{n-1}(k), \quad b_{n+1}(k) = \rho(k, L)b_{n-1}(k) \tag{5.29}$$

The convergence factor vanishes for the exact transmission conditions. The convergence rate expression induces the following remark : for k large, convergence rate is high, while for small values of k, overlap has little influence. Thus only for small values of k, is it necessary to approximate $\sqrt{k^2+1}$. Thus the following transmission conditions

$$\begin{cases} \dfrac{\partial w^{n+1}}{\partial x} - w^{n+1} = \dfrac{\partial v^n}{\partial x} - v^n, & x = L \\[4mm] \dfrac{\partial v^{n+1}}{\partial x} + v^{n+1} = \dfrac{\partial w^n}{\partial x} + w^n, & x = 0 \end{cases} \tag{5.30}$$

For these transmission conditions, the convergence factor is

$$\rho(k, L) = \left(\frac{1 - \sqrt{k^2+1}}{1 + \sqrt{k^2+1}}\, e^{-L\sqrt{k^2+1}}\right)^2. \tag{5.31}$$

Theorem 5.7 *The algorithm associated to transmission conditions* (5.30) *is well defined in* $H^1(\Omega_1) \times H^1(\Omega_2)$.

Exercice 5.5 *Prove the theorem by writing a variational formulation in each* $H^1(\Omega_i)$.

Another idea is to perform a 2nd order Taylor development of $\sqrt{k^2+1}$:

$$\sqrt{k^2+1} \approx 1 + \frac{k^2}{2}$$

Effective transmission conditions are obtained by inverse Fourier transform. For example

$$(1 + \frac{k^2}{2})\hat{v} = \widehat{v - \frac{1}{2}\frac{\partial^2 v}{\partial y^2}}.$$

Transmission conditions are written with respect to the physical variables :

$$\begin{cases} \dfrac{\partial w^{n+1}}{\partial x} - w^{n+1} + \dfrac{1}{2}\dfrac{\partial^2 w^{n+1}}{\partial y^2} = \dfrac{\partial v^n}{\partial x} - v^n + \dfrac{1}{2}\dfrac{\partial^2 v^n}{\partial y^2}, \quad x = L \\[4mm] \dfrac{\partial v^{n+1}}{\partial x} + v^{n+1} - \dfrac{1}{2}\dfrac{\partial^2 v^{n+1}}{\partial y^2} = \dfrac{\partial w^n}{\partial x} + w^n - \dfrac{1}{2}\dfrac{\partial^2 w^n}{\partial y^2}, \quad x = 0. \end{cases} \tag{5.32}$$

The convergence factor is obtained by using $\varphi(k) = \psi(k) = 1 + k^2/2$:

$$\rho(k, L) = \Big(\frac{1 + k^2/2 - \sqrt{k^2 + 1}}{1 + k^2/2 + \sqrt{k^2 + 1}} \, e^{-L\sqrt{k^2+1}} \Big)^2. \tag{5.33}$$

Theorem 5.8 *The algorithm associated to transmission conditions* (5.30) *is well defined in* $H_1^1(\Omega_1) \times H_1^1(\Omega_2)$ *where*

$$H_1^1(\Omega_i) = \{v \in H^1(\Omega_i), v \in H^1(\Gamma_i)\}$$

Exercice 5.6 *Prove the theorem by writing a variational formulation in each* $H^1(\Omega_i)$.

Theorem 5.9 *For any* $L > 0$, *for any initial guess* (u^0, v^0) *belonging to* $H^1(\Omega_1) \times H^1(\Omega_2)$ *(resp.* $H_1^1(\Omega_1) \times H_1^1(\Omega_2)$*) , the algorithm associated to the transmission conditions*(5.30) *(resp.* (5.32)*) converges in* $H^1(\Omega_1) \times H^1(\Omega_2)$ *(resp.* $H_1^1(\Omega_1) \times H_1^1(\Omega_2)$*). If* $L > 0$, *the convergence is linear.*

Proof We estimate the Fourier transforms

$$V^{2n} = (\rho(k, L))^n \, V^0$$

For any $L \geq 0$, the sequence V^{2n} converges *a.e.* in Ω_1, and is bounded by V^0 in $L^2(\Omega_1)$. By the Lebesgue theorem, it converges in $L^2(\Omega_1)$. The same holds for the gradients. If $L > 0$, we have

$$||V^{n+1}||_{L^2(\Omega_1)} \leq \sup_{k \in \mathbb{R}} |\rho(k, L)|^2 \, ||V^{n-1}||_{L^2(\Omega_1)}$$

which shows the linear convergence, since

$$\sup_{k \in \mathbb{R}} |\rho(k, L)|^2 \leq e^{-2L} < 1.$$

∎

Therefore these methods converge at least as fast as the original parallel Schwarz method. For general domains, the convergence proof will not be valid anymore, and we shall use an energy approach.

Another fruitful approach is to approximate $\sqrt{1 + k^2}$ for a large range of frequencies by an 2nd order even polynomial in k with coefficient optimization.

5.2.5   A convergence proof for $L = 0$

Let us go back to the algorithm associated to conditions (5.30), and denote by Γ the common interface. The error (V^n, W^n) satisfies

$$\begin{cases} -\Delta V^{n+1} + V^{n+1} = 0 \\ \dfrac{\partial V^{n+1}}{\partial x} + V^{n+1} = \dfrac{\partial W^n}{\partial x} + W^n \text{ on } \Gamma \end{cases} \qquad \begin{cases} -\Delta W^{n+1} + W^{n+1} = 0 \\ \dfrac{\partial W^{n+1}}{\partial x} - W^{n+1} = \dfrac{\partial V^n}{\partial x} - V^n \text{ on } \Gamma \end{cases}$$

$$\tag{5.34}$$

Let us multiply equations on the inner points by V^{n+1} and W^{n+1}, integrate over the domain and use Green's formula.

$$||V^{n+1}||^2_{H^1(\Omega_1)} - \int_\Gamma V^{n+1}\frac{\partial V^{n+1}}{\partial x} = 0$$
$$||W^{n+1}||^2_{H^1(\Omega_2)} - \int_\Gamma W^{n+1}\frac{\partial W^{n+1}}{\partial x} = 0$$

$\mathcal{B}_1$ denotes operator $\frac{\partial}{\partial x} + 1$ and $\mathcal{B}_2$ operator $\frac{\partial}{\partial x} - 1$. We now have

$$4||V^{n+1}||^2_{H^1(\Omega_1)} + \int_\Gamma [\mathcal{B}_2(V^{n+1})]^2 = \int_\Gamma [\mathcal{B}_1(V^{n+1})]^2$$
$$4||W^{n+1}||^2_{H^1(\Omega_2)} + \int_\Gamma [\mathcal{B}_1(W^{n+1})]^2 = \int_\Gamma [\mathcal{B}_2(W^{n+1})]^2$$

Let us use the boundary conditions

$$4||V^{n+1}||^2_{H^1(\Omega_1)} + \int_\Gamma [\mathcal{B}_2(V^{n+1})]^2 = \int_\Gamma [\mathcal{B}_1(W^n)]^2$$
$$4||W^{n+1}||^2_{H^1(\Omega_2)} + \int_\Gamma [\mathcal{B}_1(W^{n+1})]^2 = \int_\Gamma [\mathcal{B}_2(V^n)]^2$$

Add these two equations

$$4[||V^{n+1}||^2_{H^1(\Omega_1)} + ||W^{n+1}||^2_{H^1(\Omega_2)}] + \int_\Gamma \left([\mathcal{B}_2(V^{n+1})]^2 + [\mathcal{B}_1(W^{n+1})]^2\right) =$$
$$\int_\Gamma \left([\mathcal{B}_2(V^n)]^2 + [\mathcal{B}_1(W^n)]^2\right)$$

This implies that the series with general term $||V^n||^2_{H^1(\Omega_1)} + ||W^n||^2_{H^1(\Omega_2)}$ is convergent thus its general term goes to 0 when n goes to infinity. The error goes to 0 in H^1 norm in each of the sub-domains. Notice that the limit (v, w) satisfies , the meaning of which is to be specified, the transmission conditions

$$\begin{cases} \dfrac{\partial v}{\partial x} + v &=& \dfrac{\partial w}{\partial x} + w \text{ on } \Gamma \\ \dfrac{\partial v}{\partial x} - v &=& \dfrac{\partial w}{\partial x} - w \text{ on } \Gamma \end{cases} \tag{5.35}$$

Thus

$$\begin{cases} \dfrac{\partial v}{\partial x} &=& \dfrac{\partial w}{\partial x} \text{ on } \Gamma \\ v &=& w \text{ on } \Gamma \end{cases} \tag{5.36}$$

which are transmission conditions for u. The algorithm limit is therefore u.

5.2.6 Notions on transmission conditions

Let us consider the problem in Ω described in figure 5.9.

This problem (see Analysis course) has a unique solution in $H^1(\Omega)$. Furthermore, if Ω is sufficiently regular, u belongs to $H^2(\Omega)$. Consider a partition of Ω , $\bar{\Omega} = \bar{\Omega}_1 \cup \bar{\Omega}_2$ and $\Gamma = \bar{\Omega}_1 \cap \bar{\Omega}_2$. On Γ , there are two unit normals at each point : $\bar{n}_1$ is the outgoing normal to Ω_1 and $\bar{n}_2$ is the outgoing normal to Ω_2 with the relation $\bar{n}_1 + \bar{n}_2 = 0$. Let us denote $\bar{n} = \bar{n}_1$ thus $\bar{n}_2 = -\bar{n}$. The problem is now equivalent to the coupling problem defined in figure 5.10, where u_1 is the restriction of u to Ω_1 and u_2 is the restriction of u to Ω_2

Conditions

$$u_1 = u_2 \text{ on } \Gamma$$
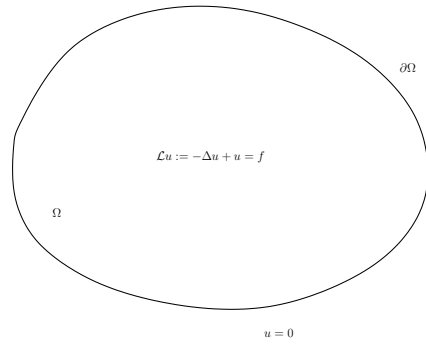$$\frac{\partial u_1}{\partial n} = \frac{\partial u_2}{\partial n} \text{ on } \Gamma \tag{5.37}$$
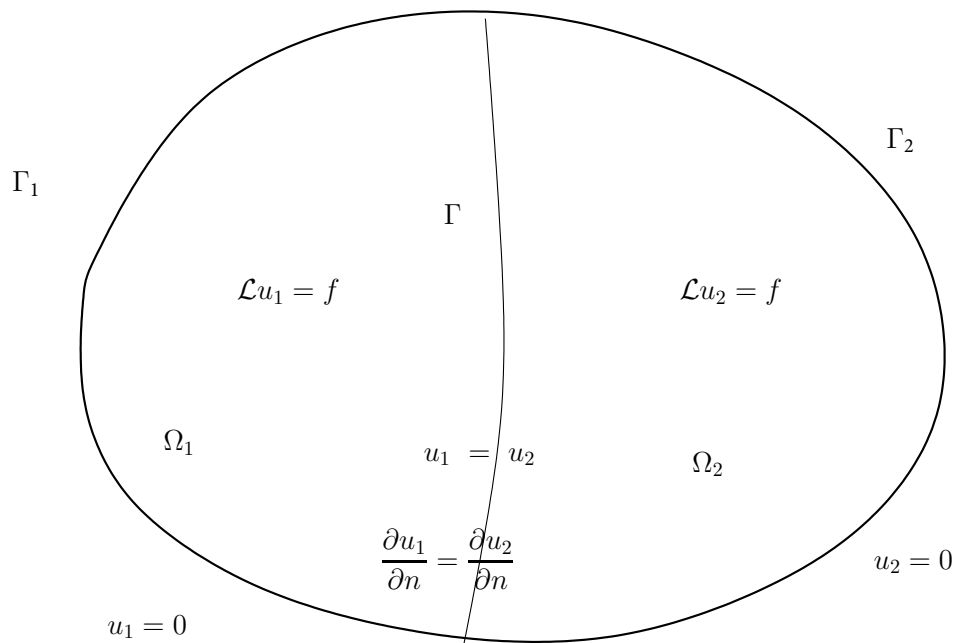
FIGURE 5.9 – Original problem

FIGURE 5.10 – Two sub-domain decomposition with no overlap.

are transmission conditions for u.

Let us now identify the transparent condition in Ω_2. Let g in $H^{\frac{1}{2}}(\Gamma)$. The following problem is now considered :

$$
\begin{cases}
\mathcal{L}u = 0 \text{ in } \Omega_2 \\
u = 0 \text{ on } \Gamma_2 \\
u = g \text{ on } \Gamma
\end{cases}
\tag{5.38}
$$

This problem has a unique solution. The trace of the normal derivative can now be defined on Γ, $\dfrac{\partial u}{\partial n_2}$. We now define $\mathcal{K}_2$ by

$$
\mathcal{K}_2 g = \frac{\partial u}{\partial n_2}
\tag{5.39}
$$

such that any solution to the boundary value problem in Ω_2

$$
\begin{cases}
\mathcal{L}u = 0 \text{ in } \Omega_2 \\
u = 0 \text{ on } \Gamma_2
\end{cases}
\tag{5.40}
$$

satisfies identically the equation on Γ

$$
\frac{\partial u}{\partial n_2} - \mathcal{K}_2 u = 0
\tag{5.41}
$$

Remark 5.1 *Let us define for any (u, v) in $H^1(\Omega_2)$,*

$$
a_2(u, v) = (u, v)_{H^1(\Omega_2)} = \int_{\Omega_2} [\nabla u \cdot \nabla v + uv] dx\, dy
$$

Using the variational formulation for any couple (g, h) in $H^{\frac{1}{2}}(\Gamma)$,

$$
< \mathcal{K}_2 g, h >= a_2(u, v)
$$

where u is the solution to 5.38 and v is the result of any lifting operator on g in $H^1(\Omega_2)$. As a consequence , $\mathcal{K}_2$ is a self-adjoint coercive operator on $H^{\frac{1}{2}}(\Gamma)$.

In the same manner, operator $\mathcal{K}_1$ is introduced : Let g be in $H^{\frac{1}{2}}(\Gamma)$. Let us consider the following problem

$$
\begin{cases}
\mathcal{L}u = 0 \text{ in } \Omega_2 \\
u = 0 \text{ on } \Gamma_1 \\
u = g \text{ on } \Gamma
\end{cases}
\tag{5.42}
$$

This problem has a unique solution. The trace of the normal derivative can now be defined on Γ, $\dfrac{\partial u}{\partial n_1}$. $\mathcal{K}_1$ is defined by

$$
\mathcal{K}_1 g = \frac{\partial u}{\partial n_1}
\tag{5.43}
$$

such that any solution to the boundary value problem in Ω_1

$$
\begin{cases}
\mathcal{L}u = 0 \text{ in } \Omega_1 \\
u = 0 \text{ on } \Gamma_1
\end{cases}
\tag{5.44}
$$

satisfies identically the equation on Γ

$$\frac{\partial u}{\partial n_1} - \mathcal{K}_1 u = 0 \tag{5.45}$$

For any (u, v) in $H^1(\Omega_1)$, let us define

$$a_1(u, v) = (u, v)_{H^1(\Omega_1)} = \int_{\Omega_1} [\nabla u \cdot \nabla v + uv] dx\, dy$$

Using the variational formulation, for any couple (g, h) in $H^{\frac{1}{2}}(\Gamma$,

$$< \mathcal{K}_1 g, h > = a_1(u, v)$$

where u is solution to (5.42) and v is the result of any lifting operator on g in $H^1(\Omega_1)$. $\mathcal{K}_1$ is thus a self-adjoint coercive operator on $H^{\frac{1}{2}}(\Gamma)$.

Equations

$$\begin{aligned}
\frac{\partial u_1^{n+1}}{\partial n_2} - \mathcal{K}_2 u_1^{n+1} &= \frac{\partial u_2^n}{\partial n_2} - \mathcal{K}_2 u_2^n \\
\frac{\partial u_2^{n+1}}{\partial n_1} - \mathcal{K}_1 u_2^{n+1} &= \frac{\partial u_1^n}{\partial n_1} - \mathcal{K}_1 u_1^n
\end{aligned} \tag{5.46}$$

constitute the **exact transmission conditions**.

Operators

$$\begin{aligned}
\mathcal{B}_1^T &= \frac{\partial}{\partial n_2} - \mathcal{K}_2 \\
\mathcal{B}_2^T &= \frac{\partial}{\partial n_1} - \mathcal{K}_1
\end{aligned} \tag{5.47}$$

are **exact transmission operators**, or **transparent operators**.

Exercice 5.7 *Show that the Domain Decomposition algorithm with conditions (5.46) converges in two iterations. Generalize to N sub-domains with no overlap.*

Exercice 5.8 *Recover exact transmission conditions (5.24).*

Going through the Schwarz algorithm step by step, we shall now see how it can be be seen as a Jacobi algorithm on a problem set on the interface. This will allow to apply more performant solvers.

5.2.7 Identification of the interface problem

Problem with no overlap

The Domain Decomposition algorithm with no overlap is now studied in a general form.

$$\begin{cases}
\mathcal{L} u_1^{n+1} = f, & \text{in } \Omega_1 \\
\dfrac{\partial u_1^{n+1}}{\partial n} = 0 \text{ on } \Gamma_1 \\
\mathcal{B}_1 u_1^{n+1} = \mathcal{B}_1 u_2^n \text{ on } \Gamma
\end{cases} \tag{5.48}$$

$$\begin{cases}
\mathcal{L} u_2^{n+1} = f, & \text{in } \Omega_2 \\
\dfrac{\partial u_2^{n+1}}{\partial n} = 0 \text{ on } \Gamma_2 \\
\mathcal{B}_2 u_2^{n+1} = \mathcal{B}_2 u_1^n \text{ on } \Gamma
\end{cases} \tag{5.49}$$

Let us now define an algorithm on the interface in the following way.
For any λ defined on Γ, and f defined on Ω, M_1 and M_2 are defined as

$$M_1 : (\lambda, f) \mapsto u_1 \text{ solution of } \begin{cases} \mathcal{L}u_1 = f, & \in \Omega_1 \\ \dfrac{\partial u_1}{\partial n} = 0 \text{ on } \Gamma_1 \\ \mathcal{B}_1 u_1 = \lambda \text{ on } \Gamma \end{cases} \tag{5.50}$$

$$M_2 : (\lambda, f) \mapsto u_2 \text{ solution of } \begin{cases} \mathcal{L}u_2 = f, & \in \Omega_2 \\ \dfrac{\partial u_2}{\partial n} = 0 \text{ on } \Gamma_2 \\ \mathcal{B}_2 u_2 = \lambda \text{ on } \Gamma \end{cases} \tag{5.51}$$

Notice that $M_j(\lambda, f) = M_j(0, f) + M_j(\lambda, 0)$.
Suppose that $(\lambda_1^n, \lambda_2^n)$ are defined by par

$$\lambda_j^n = \mathcal{B}_j u_j^n$$

But in Ω_j,

$$\begin{cases} \mathcal{L}u_j^n = f, & \in \Omega_j \\ \dfrac{\partial u_j^n}{\partial n} = 0 \text{ on } \Gamma_j \\ \mathcal{B}_j u_j^n = \mathcal{B}_j u_j^n \text{ on } \Gamma \end{cases}$$

thus

$$u_j^n = M_j(\lambda_j^n, f) \tag{5.52}$$

We now have the following equalities

$$\lambda_1^{n+1} = \mathcal{B}_1 u_1^{n+1} = \mathcal{B}_1 M_2(\lambda_2^n, f) = \mathcal{B}_1[M_2(\lambda_2^n, 0) + M_2(0, f)]$$
$$\lambda_2^{n+1} = \mathcal{B}_2 u_2^{n+1} = \mathcal{B}_2 M_1(\lambda_1^n, f) = \mathcal{B}_2[M_1(\lambda_1^n, 0) + M_1(0, f)]$$

Thus the interface system

$$\begin{aligned} \lambda_1^{n+1} &= \mathcal{B}_1 M_2(\lambda_2^n, 0) + \mathcal{B}_1 M_2(0, f) \\ \lambda_2^{n+1} &= \mathcal{B}_2 M_1(\lambda_1^n, 0) + \mathcal{B}_2 M_1(0, f) \end{aligned} \tag{5.53}$$

Or , setting

$$J(\lambda_1, \lambda_2) = \begin{pmatrix} 0 & \mathcal{B}_1 M_2(., 0) \\ \mathcal{B}_2 M_1(., 0) & 0 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix}$$

This can be written as

$$\Lambda^{n+1} = J\Lambda^n + \begin{pmatrix} \mathcal{B}_1 M_2(0, f) \\ \mathcal{B}_2 M_1(0, f) \end{pmatrix}$$

which corresponds to the Jacobi algorithm applied to matrix

$$A = \begin{pmatrix} I & -\mathcal{B}_1 M_2(., 0) \\ -\mathcal{B}_2 M_1(., 0) & I \end{pmatrix}$$

and the system

$$A\Lambda = b = \begin{pmatrix} \mathcal{B}_1 M_2(0, f) \\ \mathcal{B}_2 M_1(0, f) \end{pmatrix}$$

1D example with overlap

Let us consider the example of a parallel Schwarz method with overlap and Neumann transmission conditions i.e. $\mathcal{B}_1 \equiv \mathcal{B}_2 \equiv \dfrac{d}{dx}$ on $\Gamma = \{x = 0\}$.

$$\begin{cases} -\dfrac{d^2}{dx^2} u_1^{n+1} + u_1^{n+1} = f \text{ in } \Omega_1 = (-\infty, L) \\ \dfrac{d}{dx} u_1^{n+1} = \dfrac{d}{dx} u_2^n \text{ for } x = L, \end{cases} \tag{5.54}$$

$$\begin{cases} -\dfrac{d^2}{dx^2} u_2^{n+1} + u_2^{n+1} = f \text{ in } \Omega_2 = (0, +\infty) \\ \dfrac{d}{dx} u_2^{n+1} = \dfrac{d}{dx} u_1^n \text{ for } x = 0. \end{cases} \tag{5.55}$$

For $\lambda \in \mathbb{R}$, $M_1(\lambda, 0)$ is defined by

$$\begin{cases} -\dfrac{d^2}{dx^2} u_1 + u_1 = 0, \quad x \leq L \\ \dfrac{d}{dx} u_1(L) = \lambda \end{cases} \tag{5.56}$$

which can be solved as

$$M_1(\lambda, 0) = \lambda e^{x-L}$$

et

$$\mathcal{B}_2 M_1(\lambda, 0) = \lambda e^{-L}$$

Similarly we have

$$\begin{cases} -\dfrac{d^2}{dx^2} u_2 + u_2 = 0, \quad x \geq 0 \\ \dfrac{d}{dx} u_2 = \lambda \text{ en } x = 0 \end{cases} \tag{5.57}$$

which can be written as

$$M_2(\lambda, 0) = -\lambda e^{-x}$$

et

$$\mathcal{B}_1 M_2(\lambda, 0) = \lambda e^{-L}$$

Matrix J is

$$J = e^{-L} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Thus the algorithm

$$\Lambda^{n+1} = J\Lambda^n + \begin{pmatrix} \mathcal{B}_1 M_2(0, f) \\ \mathcal{B}_2 M_1(0, f) \end{pmatrix}$$

which again can be written as

$$\begin{cases} \lambda_1^{n+1} = e^{-L} \lambda_2^n + \alpha \\ \lambda_2^{n+1} = e^{-L} \lambda_1^n + \beta \end{cases}$$

The spectral radius of J is strictly less than 1 for any $L > 0$. The Jacobi algorithm converges thus towards the solution (λ_1, λ_2) to system

$$\begin{pmatrix} 1 & -e^{-L} \\ -e^{-L} & 1 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

With no overlap, the algorithm diverges.

Remark 5.2 *For any $L > 0$, the matrix of the system is symmetric positive definite . For $L = 0$, it is not inversible.*

107

5.2.8 Substructuring method revisited

Let us go back to the example in chapter 5.1 with a Schwarz algorithm. The approximate transmission conditions are of Robin type on a boundary Γ (no overlap case). Using chapter 5.1 notations

$$\mathcal{B}_1 = \frac{\partial}{\partial n_1} + p, \quad \mathcal{B}_2 = \frac{\partial}{\partial n_2} + p.$$

Here p is a strictly positive real number.

$$\begin{cases} -\Delta u_1^n = f \\ \dfrac{\partial u_1^n}{\partial n_1} + p u_1^n = \dfrac{\partial u_2^{n-1}}{\partial n_1} + p u_2^{n-1} \text{ on } \Gamma \\ u_1^n = 0 \text{ on } \Gamma_1 \end{cases} \tag{5.58}$$

$$\begin{cases} -\Delta u_2^n = f \\ \dfrac{\partial u_2^n}{\partial n_2} + p u_2^n = \dfrac{\partial u_2^{n-1}}{\partial n_2} + p u_2^{n-1} \text{ on } \Gamma \\ u_2^n = 0 \text{ on } \Gamma_2 \end{cases} \tag{5.59}$$

To compute M_1 , let us use its definition and write the variational formulation

$$\begin{cases} -\Delta u_1 = 0 \\ \dfrac{\partial u_1}{\partial n_1} + p u_1 = \lambda_1 \\ u_1 = 0 \text{ on } \Gamma_1 \end{cases} \tag{5.60}$$

We define on $V_1 = \{v \in H^1(\Omega_1), v = 0 \text{ on } \Gamma_1\}$ the bilinear form b_1 by

$$b_1(v, \varphi) = a_1(v, \varphi) + p \int_\Gamma v\varphi dy, \ a_1(v, \varphi) = \int_{\Omega_1} \nabla v \nabla \varphi dx dy. \tag{5.61}$$

The bilinear form a_1 is the scalar product of V_1. It corresponds to a homogeneous Dirichlet problem. The additional term corresponds to the boundary condition on Γ.

$M_1(\lambda_1, 0)$ is thus solution to the variational problem in V_1

$$\forall \varphi \in V_1, b_1(M_1(\lambda_1, 0), \varphi) = \int_\Gamma \lambda_1 \varphi dy \tag{5.62}$$

Moreover $M_1(0, f)$ is solution to the variational problem in V_1

$$\forall \varphi \in V_1, b_1(M_1(0, f), \varphi) = \int_\Omega f\varphi dx dy \tag{5.63}$$

Let us consider applying a finite element technique, $\mathbb{P}_1$ for example to this problem. Basis functions are noted $\{\varphi_j\}_{1 \leq j \leq N_S^2}$. Nodes are numbered as inner nodes $\{S_j\}_{1 \leq j \leq N_{int}^2}$, and boundary nodes as $\{S_j\}_{N_{int}^2 + 1 \leq j \leq N_S^2}$. Same for the basis functions. If (S_i, S_j) are such that that one of them does not belong to Γ, $b_1(\varphi_i, \varphi_j) = a_1(\varphi_i, \varphi_j)$. The system matrix becomes

$$B_1 = \begin{pmatrix} K_{11} & K_{13} \\ K_{31} & K_{33}^1 + pA_\Gamma \end{pmatrix} \tag{5.64}$$

The right hand side $\{\int_\Gamma \lambda_1 \varphi_j dy\}$ can be expressed as $(0, A_\Gamma \Lambda_1)$, and the system can be written by decomposing vector $(u_h^1(S_j))$ into (U_1, U_3) :

$$\begin{pmatrix} \underbrace{K_{11}}_{\text{inner points}} & \underbrace{K_{13}}_{\text{coupling}} \\ K_{31} & \underbrace{K_{33}^1 + pA_\Gamma}_{\text{boundary}} \end{pmatrix} \begin{pmatrix} U_1 \\ U_3 \end{pmatrix} = \begin{pmatrix} 0 \\ A_\Gamma \Lambda_1 \end{pmatrix} \tag{5.65}$$

which be written as the following system

$$\begin{cases} K_{11}U_1 + K_{13}U_3 & = & 0 \\ K_{31}U_1 + (K_{33}^1 + pA_\Gamma)U_3 & = & A_\Gamma \Lambda_1 \end{cases} \tag{5.66}$$

Matrix K_{11} is invertible as it is the matrix associated to a homogeneous Dirichlet problem. U_1 can be expressed as a function of U_3 and carrying it in the second equation as in chapter 5.1 :

$$U_1 = -(K_{11})^{-1}K_{13}U_3,$$
$$(K_{33}^1 - K_{31}(K_{11})^{-1}K_{13} + pA_\Gamma)U_3 = A_\Gamma\Lambda_1 \tag{5.67}$$

The new Schur Complement matrix in Ω_1, S_{Robin}^1 is defined from the old one by $S_{Robin}^1 = S^1 + pA_\Gamma$. It is also symmetric positive definite.

For $M_1(0, f)$, with similar notations , the following system is obtained

$$\begin{cases} K_{11}U_1 + K_{13}U_3 & = & F_1 \\ K_{31}U_1 + (K_{33}^1 + pA_\Gamma)U_3 & = & F_3^1 \end{cases} \tag{5.68}$$

and thus U_3 is solution to system

$$S_{Robin}^1 U_3 = F_3^1 - K_{31}(K_{11})^{-1}F_1 \tag{5.69}$$

It now remains to compute $\mathcal{B}_2 M_1(\Lambda_1, 0)$ and $\mathcal{B}_2 M_1(0, f)$. To do this, let us note that thanks to the boundary conditions, we have

$$\mathcal{B}_2 M_1(\lambda_1, 0) = \frac{\partial u_1}{\partial n_2} + pu_1 = -\frac{\partial u_1}{\partial n_1} + pu_1 = -\lambda_1 + 2pM_1(\lambda_1, 0)$$
$$\mathcal{B}_2 M_1(0, f) = -\frac{\partial u_1}{\partial n_1} + pu_1 = 2pM_1(0, f)$$

Thus

$$\mathcal{B}_2 M_1(\lambda_1, 0) = 2p(S_{Robin}^1)^{-1}A_\Gamma\Lambda_1 - \Lambda_1$$
$$\mathcal{B}_2 M_1(0, f) = 2(S_{Robin}^1)^{-1}(F_3^1 - K_{31}(K_{11})^{-1}F_1). \tag{5.70}$$

Similarly we have

$$\mathcal{B}_1 M_2(\lambda_2, 0) = 2p(S_{Robin}^2)^{-1}A_\Gamma\Lambda_2 - \Lambda_2$$
$$\mathcal{B}_1 M_2(0, f) = 2(S_{Robin}^2)^{-1}(F_3^2 - K_{32}(K_{22})^{-1}F_2). \tag{5.71}$$

where S_2 the Schur Complement matrix in Ω_2.

Remark 5.3 *With notation U_3, we have implicitly supposed that discretizations coincide on the common boundary. This is not necessary.*

The matrix algorithm associated to the Schwarz can now be written as

$$\begin{pmatrix} \Lambda_1^{n+1} \\ \Lambda_2^{n+1} \end{pmatrix} = \begin{pmatrix} 0 & -I + 2p(S_{Robin}^2)^{-1}A_\Gamma \\ -I + 2p(S_{Robin}^1)^{-1}A_\Gamma & 0 \end{pmatrix} \begin{pmatrix} \Lambda_1^n \\ \Lambda_2^n \end{pmatrix}$$
$$+ \begin{pmatrix} 2(S_{Robin}^1)^{-1}(F_3^1 - K_{31}(K_{11})^{-1}F_1) \\ 2(S_{Robin}^2)^{-1}(F_3^2 - K_{32}(K_{22})^{-1}F_2) \end{pmatrix} \tag{5.72}$$

which is the Jacobi method to solve a system whose matrix is

$$\begin{pmatrix} I & (I - 2p(S_{Robin}^2)^{-1}A_\Gamma) \\ (I - 2p(S_{Robin}^1)^{-1}A_\Gamma) & I \end{pmatrix} \tag{5.73}$$

The alternate Schwarz algorithm mentioned in (5.7) corresponds to a Gauss-Seidel method. Convergence can be accelerated by applying a Krylov method as in chapter 5.1.

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.