Fès Paris 13

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.

Octobre-Novembre 2018

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 :

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

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

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

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

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

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

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

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

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

1.1.3 Theoretical results

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

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

$$PA = LU$$

1.1.4 Symmetric definite matrices : Cholewski decomposition

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

1.1.5 Elimination with Givens rotations

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

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

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

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

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

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

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

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

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

1.1.6 QR Decomposition

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

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

Example for $n = 5$,

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

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

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

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

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 $G_{21},\dots,G_{n1}$ 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

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

Q is an orthogonal matrix,

$$Q^*Q=G_N\dots G_1G_1^*\dots G_N^*=I.$$

then

$$A=QR,$$

we have reached the QR decomposition of the matrix A.

1.2 Stationary iterative methods

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

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

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

Again, M is a preconditioner.

Definition 1.1

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

Then the sequence of the errors satisfies

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

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

1.2.1 Classical methods

Use $A = D - E - F$.

Jacobi	$M = D$	$R := J = I - D^{-1}A$
Relaxed Jacobi	$M = \frac{1}{\omega}D$	$R = I - \omega D^{-1}A$
Gauss-Seidel	$M = D - E$	$R := \mathcal{L}_1 = I - (D-E)^{-1}A$
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

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

Definition 1.2

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

Theorem 1.4

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

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

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

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

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

Diagonally dominant matrices

Theorem 1.5

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

M- matrices

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

1. *$a_{ii} > 0$ for $i = 1, \ldots, n$,*
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 $\rho(\mathcal{L}_{\omega^*}) = \omega^* - 1$.*

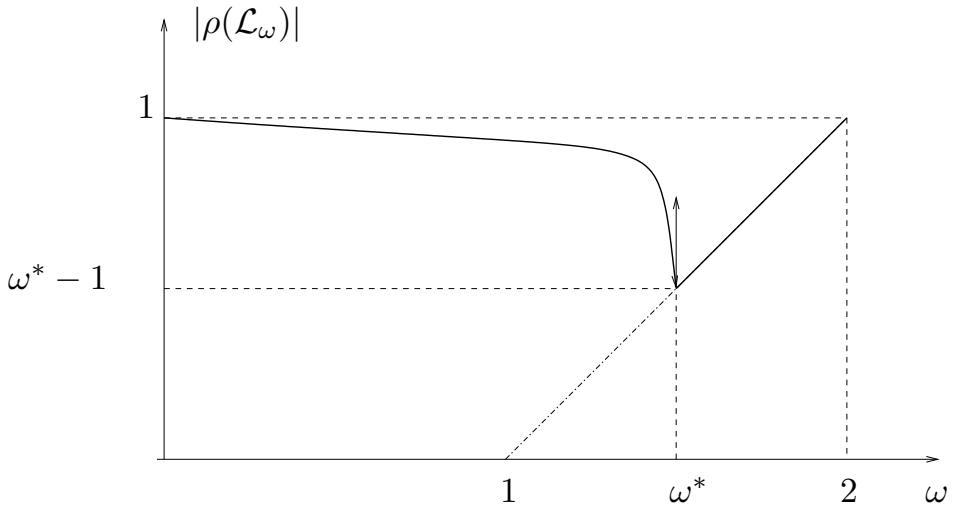


FIGURE 1.1 – Variations of $\rho(\mathcal{L}_\omega)$ as a fonction of ω

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.

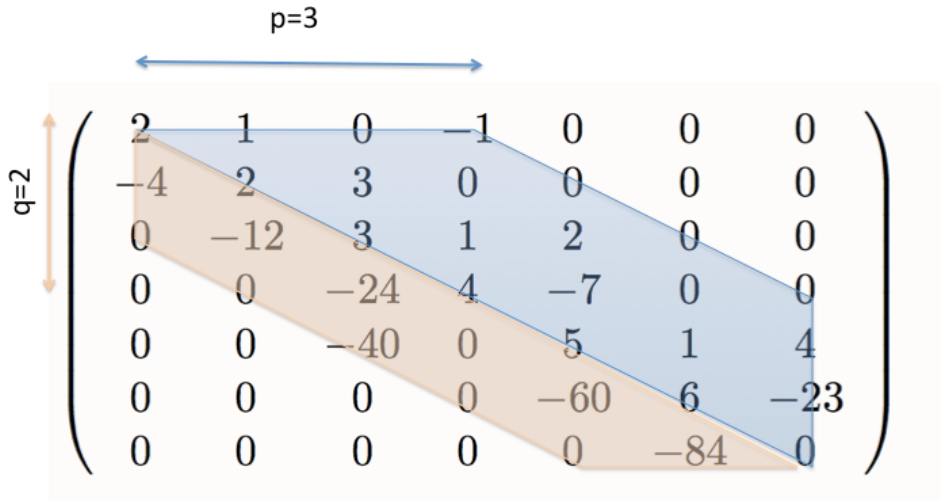


FIGURE 1.2 – A bandmatrix

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

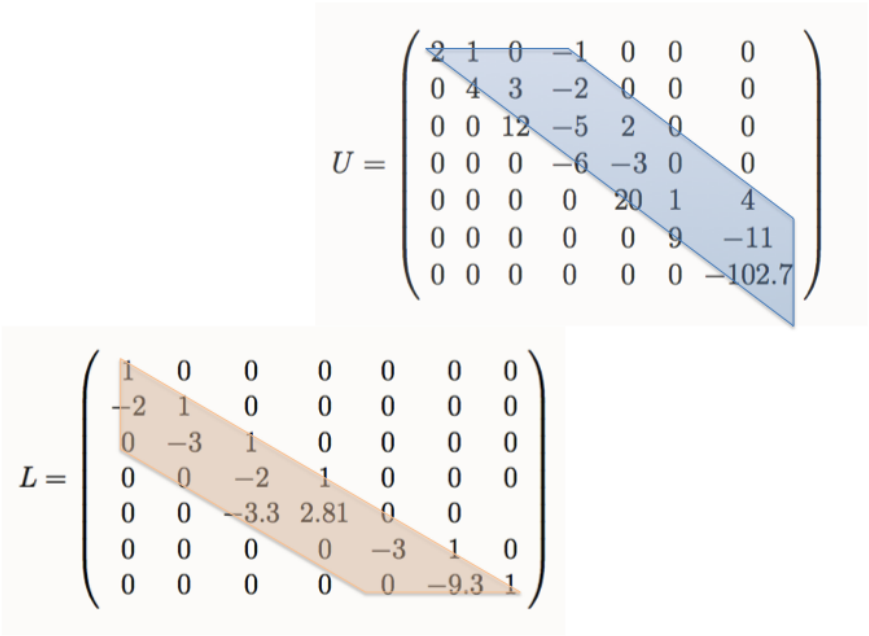


FIGURE 1.3 – LU decomposition

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

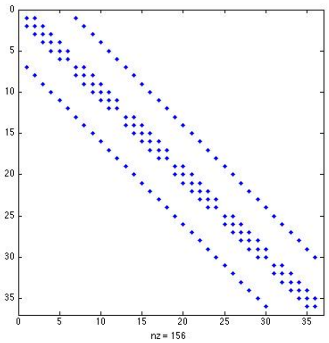
$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.6 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -0.5 & -0.17 & -0.05 & -0.21 & 0.025 & 0.0027 & 1 \end{pmatrix}$$

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

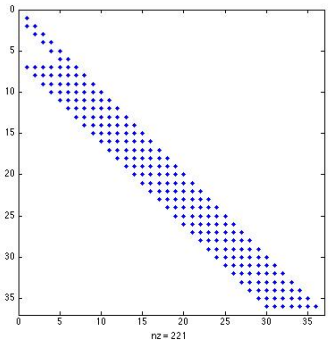
Here the permutation matrix is

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

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

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

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

Givens : verify that the process inserts an extra updiagonal.

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

Creation and manipulation of sparse matrices in matlab

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

 S =
```

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

>>S=speye(2,3)

S =

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

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

     1
     1
     1
     1

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

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

>>full(A)
ans =

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


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

```
 S =

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

>>S=speye(2,3)

S =

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

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

     1
     1
     1
     1

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

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

>>full(A)
ans =

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

The direct methods first transform the original system into a triangular

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

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 :

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

A1

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

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.

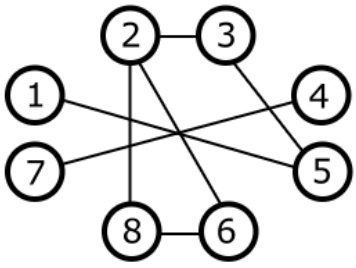
Two nodes of G(A) are said to be adjacent if they are connected by an edge.

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.

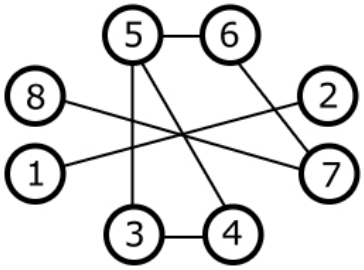
A component of a graph is a connected subgraph which is not contained in a larger connected subgraph.

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|Li-Lj|, i,j=1..n and Li is the label of node i, Lj 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 : $\|y\|_A^2 = (Ay, y)$.

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. $\text{vec}(r^0, \ldots, r^m) = \text{vec}(r^0, Ar^0, \ldots, A^m r^0)$.
3. $\text{vec}(p^0, \ldots, p^m) = \text{vec}(r^0, Ar^0, \ldots, A^m r^0)$.
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

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

end.

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 $u =^t (u_1, \ldots, u_n)$.

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

The matrix A is symmetric definite positive.

The discrete problem to be solved is

$$Au = b$$

2.1.3 Condition number and error

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

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

Theorem 2.4

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

and there is a b such that it is equal.

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

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

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

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

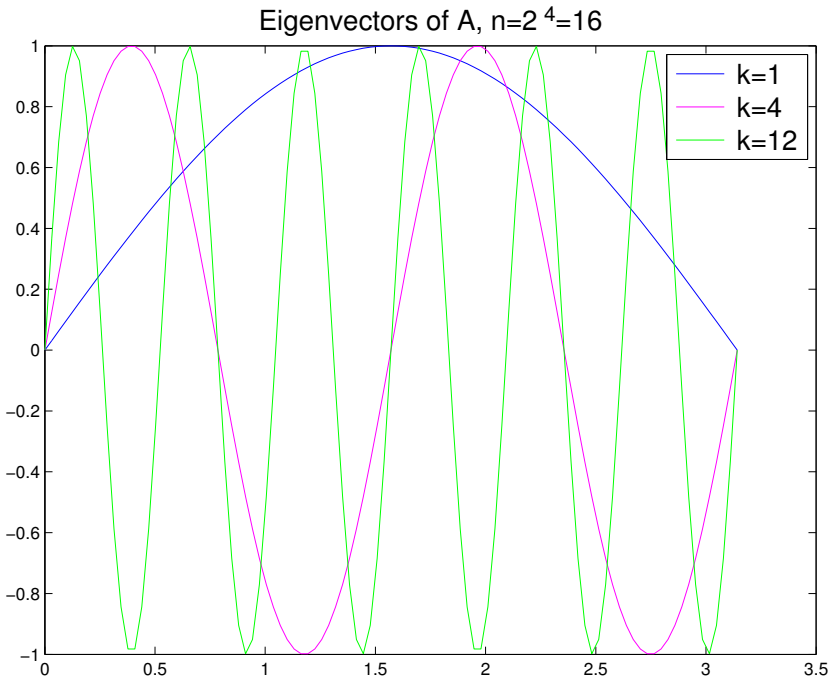


FIGURE 2.1 – Eigenvectors of A

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

TABLE 2.1 – Eigenvalues of the iteration matrix

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

TABLE 2.2 – Convergence factor

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

TABLE 2.3 – Asymptotic behavior in function of $\varepsilon = \pi h$

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

TABLE 2.4 – Reduction factor for one digit $M \sim -\dfrac{\ln(10)}{F}$

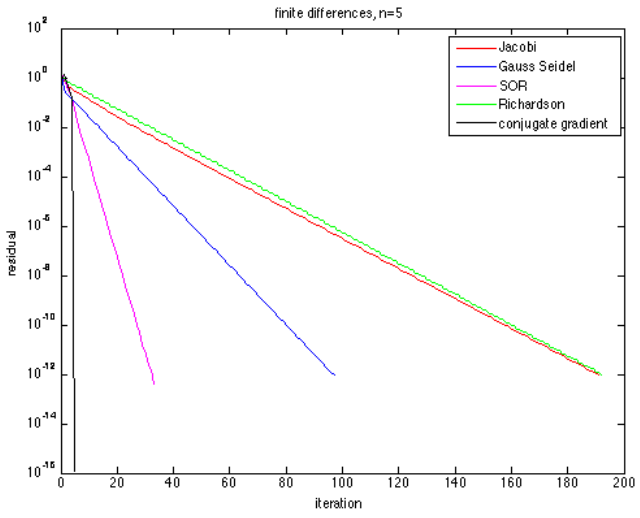


FIGURE 2.2 – Convergence history, $n = 5$

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

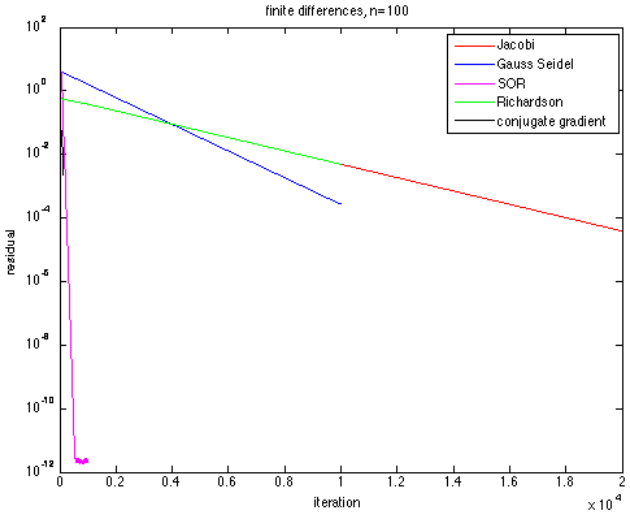


FIGURE 2.3 – Convergence history, $n = 100$

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

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

The definition of the q_j above can be rewritten as

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

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

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

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

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

$$H_m y = \|r_0\| e_1 \tag{2.5}$$

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
```

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}.

■

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

```
    if k<n
        H(k+1,k)=norm(v); Q(:,k+1)=v/H(k+1,k);
    end
end
```

2.2.4 GMRES algorithm

Here we minimize at each step the residual $r_m = r_0 - AV_m y$ 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 an orthogonal matrix, 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 matrix, and is best solved by the Givens reflection method. Let us consider the case of $m = 3$ ($\sigma_0 = \|r_0\|$).

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

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

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

to make the system triangular, and even better

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

Therefore

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

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

$$Ry = c^I,$$

which is now an upper triangular system.

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

```

```
n = length(b);

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

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

x = x0;

% Set up for the method
flag = 1;
xmin = x;                       % Iterate which has minimal residual
    so far
imin = 0;                       % Outer iteration at which xmin was
    computed
jmin = 0;                       % Inner iteration at which xmin was
    computed
tolb = tol * n2b;               % Relative tolerance
if tolb==0,
  tolb=tol;                     % use absolute error to find zero
      solution
end;
r = b - feval(A,x);             % Zero-th residual
normr = norm(r);                % Norm of residual

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


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

```
stag = 0;                           % stagnation of the method

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

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

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

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

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

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

```

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

```
148         r = b - feval(A,x);
149         normr = norm(r);
150         resvec((i-1)*restart+j+1) = normr;
151       end
152       if normr <= tolb            % check using more accurate xj
153         flag = 0;
154         iter = [i j];
155         break;
156       end
157     end
158
159     if stag == 1
160       flag = 3;
161       break;
162     end
163
164     if normr < normrmin           % update minimal norm quantities
165       normrmin = normr;
166       xmin = x;
167       imin = i;
168       jmin = j;
169     end
170   end                             % for j = 1 : restart
171
172   if flag == 1
173     x0 = x;                       % save for the next outer
          iteration
174     r = b - feval(A,x0);
175   else
176     break;
177   end
178
179 end                               % for i = 1 : maxit
180
181 % returned solution is that with minimum residual
182
183 if n2b==0,
184   n2b=1;         % here we solved for the zero solution and thus show
185 end;             % the absolute residual !
186
187 if flag == 0
188   relres = normr / n2b;
189 else
190   x = xmin;
191   iter = [imin jmin];
192   relres = normrmin / n2b;
193 end
194
195 % truncate the zeros from resvec
196 if flag <= 1 | flag == 3
```

```
  resvec = resvec(1:(i-1)*restart+j+1);
else
  if j == 0
    resvec = resvec(1:(i-1)*restart+1);
  else
    resvec = resvec(1:(i-1)*restart+j);
  end
end


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

    case 1,
      os = sprintf(['GMRES(%d) stopped after the maximum %d iterations' ...
                    ' without converging to the desired tolerance %0.2g' ...
                    '\n         The iterate returned (number %d(%d))' ...
                    ' has relative residual %0.2g'], ...
                   restart,maxit,tol,iter(1),iter(2),relres);

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

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

```

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

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

Remark If A is symmetric, H_m is tridiagonale.

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

Compute $x_1, \cdots, x_M$.

If r_M is small enough, stop,

else restart with $x_0 = x_M$.

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

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

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

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

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

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

The algorithm for A

Define $C = M^2$.

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

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

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

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

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

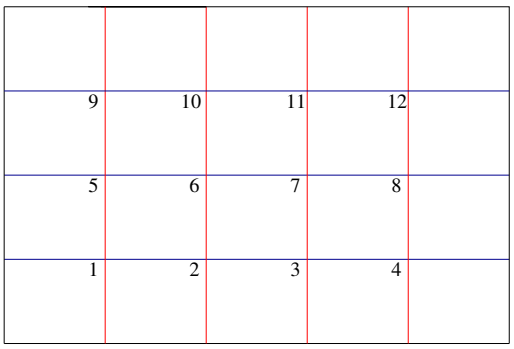


FIGURE 3.1 – Numbering by line

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

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

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 :

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

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

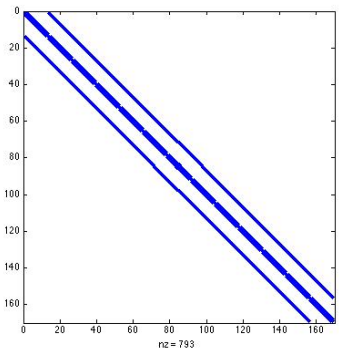
The righthand side is $b_{i+(j-1)*M} = f_{i,j}$, and the system takes the form $AZ = b$.

```
function A=lap1d(n)
% lap1d one dimensional finite difference approximation
%   A=lap1d(n) computes a sparse finite difference
%   approximation of the one dimensional operator -Delta on the
%   domain Omega=(0,1) using n interior points

h=1/(n+1);
e=ones(n,1);
A=spdiags([-e/h^2 2/h^2*e -e/h^2],[-1 0 1],n,n);
```

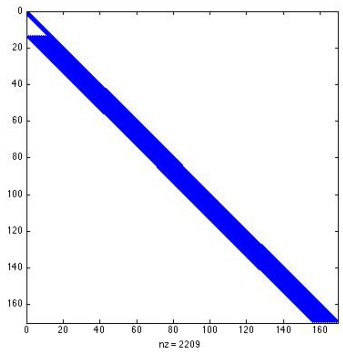
```
A=lap2d(nx,ny);
% A=lap2d(nx,ny) matrix of -delta in 2d on a grid
% of  nx  internal points in x and ny  internal points in y
% numbered by row. Uses the function kron of matlab
Dxx=lap1d(nx);
Dyy=lap1d(ny);

A=kron(speye(size(Dyy)),Dxx)+kron(Dyy,speye(size(Dxx)));
```

Cholewski decomposition of A

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



spy(A)



spy(R)

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

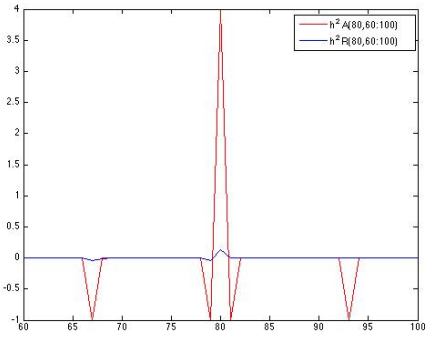


FIGURE 3.2 – Variation of the coefficients of Cholewski in the line 80 for $M = 15$

The matlab codes are as follows ([5])

Cholewski

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

Incomplete Cholewski

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

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

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

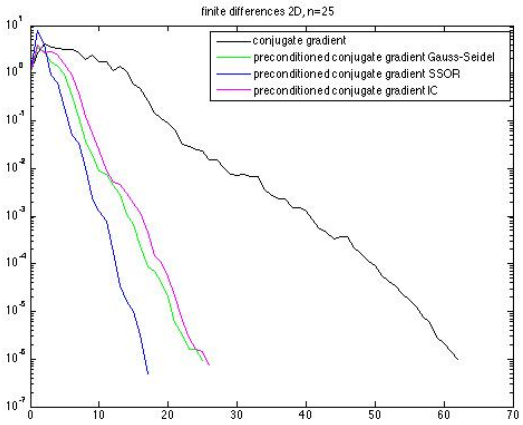


FIGURE 3.3 – Convergence history, influence of preconditioning

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 $H_k = {V_k}^T A V_k$, which is the restriction of A onto the Krylov subspace $K(k, A, r_0)$. The eigenvalues of H_k are called Ritz values. Let $H_k = SRS^T$ 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}

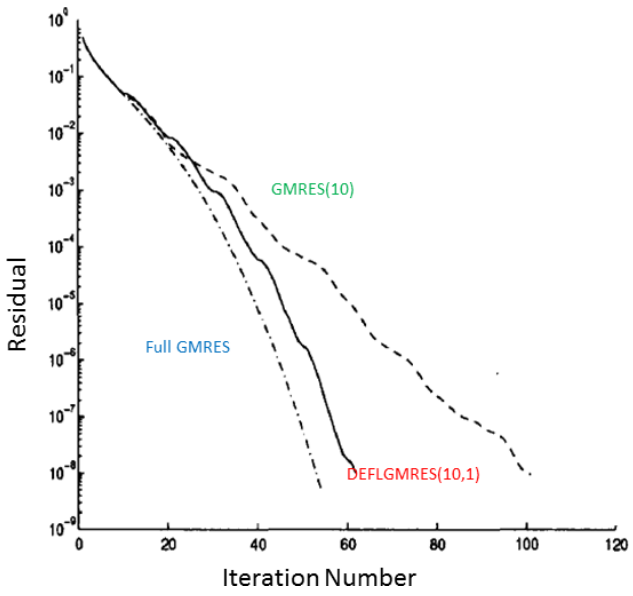


FIGURE 3.4 – Convergence history, Case 1

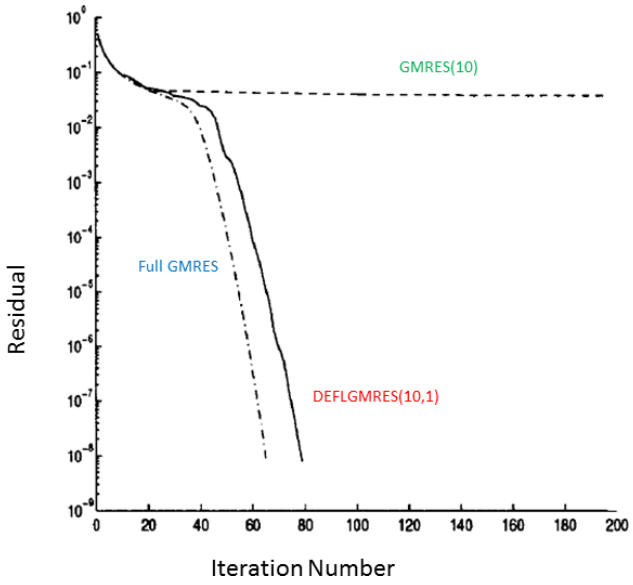


FIGURE 3.5 – Case 2

3.3 Fast methods using Fast Fourier Transform

3.3.1 Presentation of the method

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

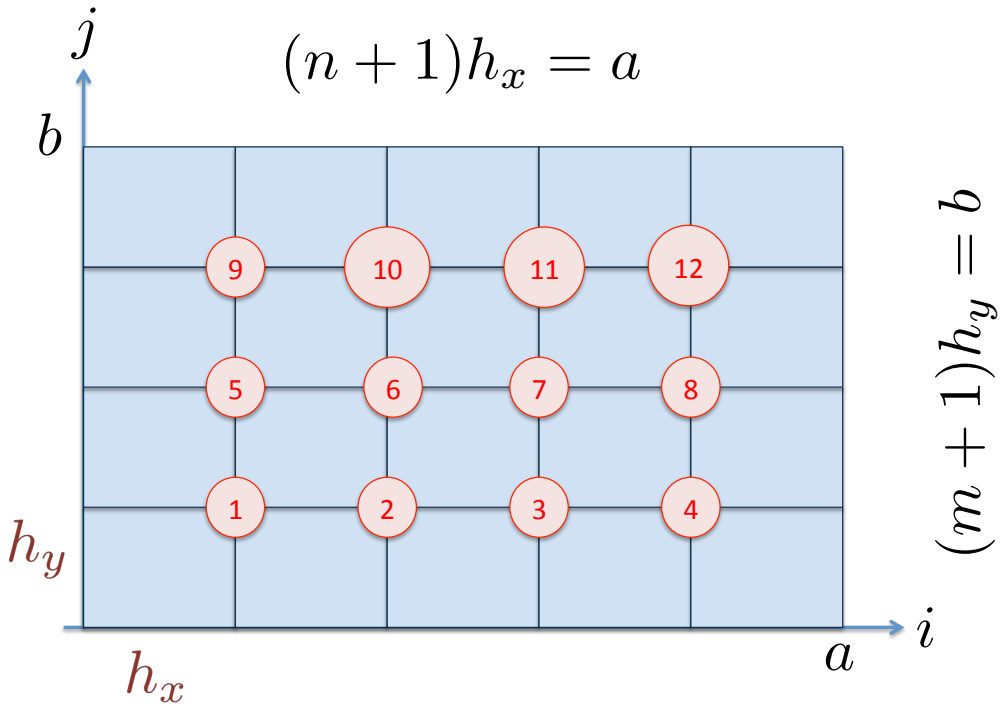


FIGURE 3.6 – Pavage de $[0, a] \times [0, b]$, $n = 4$ and $m = 3$

$$A = \begin{pmatrix} B & C & 0 \\ C & B & C \\ 0 & C & B \end{pmatrix} \qquad \text{size(blocks)=n}$$

$$-\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h_x^2} - \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h_y^2} = f_{i,j}$$

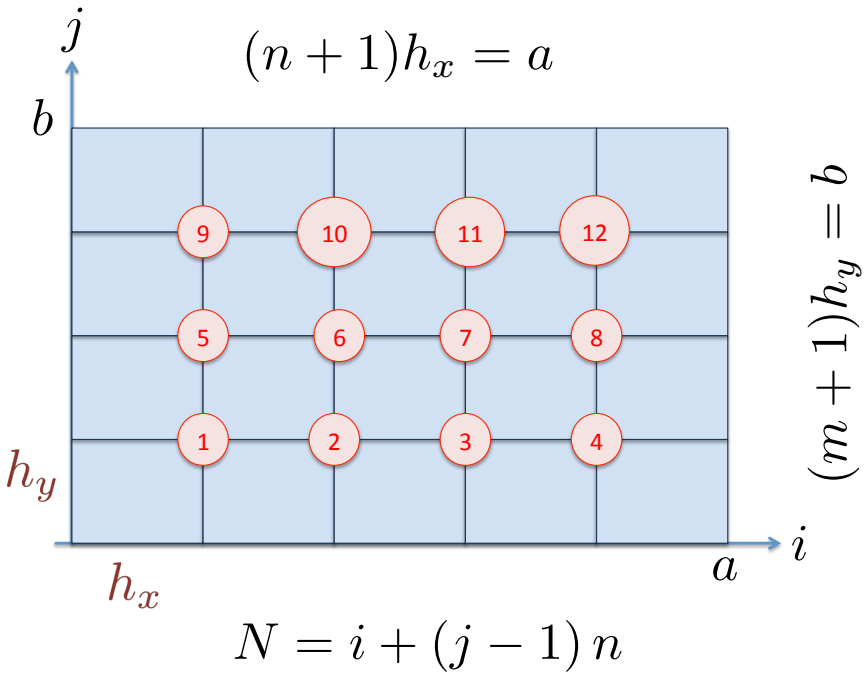


FIGURE 3.7 – Pavage de $[0,a] \times [0,b]$, $n = 4$ and $m = 3$

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

$$A = \begin{pmatrix} B & C & 0 \\ C & B & C \\ 0 & C & B \end{pmatrix} = I_3 \otimes A_1(h_x) + A_1(h_y) \otimes I_4.$$

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

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

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

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

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, $\boldsymbol{x}^j \in \mathbb{R}^n$ 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 :

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

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 :

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

Then the problem takes the form

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

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

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

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.
The matrix of the $\boldsymbol{z}^j$ is

$$Z = (\boldsymbol{z}^1, \dots, \boldsymbol{z}^j, \dots, \boldsymbol{z}^n) = \begin{pmatrix} y_1^1, \dots y_2^1 \dots, y_n^1 \\ \vdots \\ y_1^i \dots y_2^i \dots y_n^i \\ \vdots \\ y_1^m \dots y_2^m, \dots, y_n^m \end{pmatrix} = \begin{pmatrix} \boldsymbol{y}^1 \\ \vdots \\ \boldsymbol{y}^i \\ \vdots \\ \boldsymbol{y}^m \end{pmatrix}$$

We finally have to compute for each j, $\boldsymbol{x}^j = Q\boldsymbol{y}^j$ where Q is the orthogonal matrix which diagonalizes $B = A_1(h_x) + \frac{2}{h_y^2} I_n$ and C is $-\dfrac{1}{h_y^2} I_n$, the matrix . The eigenvectors of B and C are those of $A_1(h_x)$, given by (after orthonormalisation)

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

the eigenvalues of B are those of $A_1(h_x) + \frac{2}{h_y^2}$, which are $\frac{2}{h_y^2} + \frac{4}{h_x^2} \sin^2 \frac{k\pi h_x}{2}$. Define the matrix Q as the matrix of eigenvectors

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

Note that Q is symmetric. We want to compute efficiently $Q\boldsymbol{v}$ for any vector $\boldsymbol{v}$. By

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

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 $k = n+1$ 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

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

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

$$\text{DFT}: \qquad w_j = \sum_{k=1}^{n'} v_k \, e^{-2i\frac{kj\pi}{n'}} = \sum_{k=1}^{n'} v_k \, r^{-kj}, \quad j = 1, \cdots, n'. \tag{3.4}$$

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*

$$v_k = \frac{1}{n'} \sum_{p=1}^{n'} w_p \, r^{kp}, \quad p = 1, \cdots, n'. \tag{3.5}$$

Proof Just replace in (3.4),

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

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

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

The last case corresponds to $p = j$. 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 describe the FFT algorithm, and we must suppose that n' is even, that is $n' = 2p$. We need to specify more $r = e^{2i\frac{\pi}{n'}}$, that we call $r_{n'}$. Note for further use that $(r_{n'})^{n'} = 1$ and $(r_{n'})^p = -1$. Split the sum in (3.4) into even ($k = 2\ell, \ell = 1 : p$) and odd terms ($k = 2\ell - 1, \ell = 1 : p$). For $j = 1, \cdots, 2p$,

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

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

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

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

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

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

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

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

To compute u_j and t_j note that

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

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

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

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

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

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

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

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

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

Initialization : $n' = 2$, $r_{n'} = -1$, $w_1 = -v_1 + v_2$, $\quad w_2 = v_1 + v_2$.

```
function w=myFFT(v)
% MYFFT fast Fourier transform
% w=myFFT(v); computes recursively the Fourier tranform of
% the vector v whose length must be a power of 2.
n=length(v);
if n==2,
        w=[-v(1)+v(2);v(1)+v(2)];
else
        rv=exp(2i*pi/n*(1:n/2)');
        t=myFFT(v(1:2:n-1));
        u=myFFT(v(2:2:n));
        w=[u+rv.*t; u-rv.*t];
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 $w_1 = -v_1 + v_2, \quad w_2 = v_1 + v_2$.

```
function w=myFFT(v)
% MYFFT fast Fourier transform
% w=myFFT(v); computes recursively the Fourier tranform of
% the vector v whose length must be a power of 2.
n=length(v);
if n==2,
        w=[-v(1)+v(2);v(1)+v(2)];
else
        rp=exp(2i*pi/n*(1:n/2)');
        t=myFFT(v(1:2:n-1));
        u=myFFT(v(2:2:n));
        w=[u+rp.*t; u-rp.*t];
end;
```

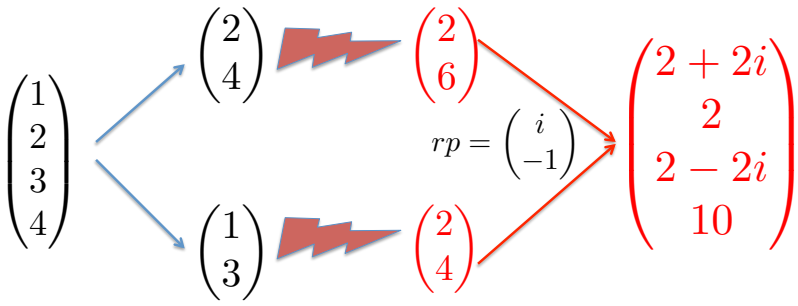


FIGURE 3.8 – FFT for $n' = 4$

It is easy to count the number of operations in the algorithm to be $\mathcal{O}(n \log_2(n))$.

3.3.3 The algorithm

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

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

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, $z_1, \cdots, z_n$

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

Resuming with matlab notations

QFFT

$$
\begin{aligned}
\boldsymbol{r_0} &= e^{i\frac{\pi}{n'}} \\
(Q\boldsymbol{v})_{2\ell} &= -\sqrt{\tfrac{2}{n+1}}\ \mathcal{I}\mathrm{m}(FFT(\tilde{\boldsymbol{v}}))_\ell, \qquad \ell = 1, \cdots, \tfrac{n-1}{2} \\
(Q\boldsymbol{v})_{2\ell-1} &= -\sqrt{\tfrac{2}{n+1}}\ \mathcal{I}\mathrm{m}(FFT(\tilde{\boldsymbol{v}} \cdot * \boldsymbol{r_0}^{(1:n')'}))_\ell, \quad \ell = 1, \cdots, \tfrac{n+1}{2}
\end{aligned}
\tag{3.6}
$$

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 $\boldsymbol{c}^j = Q\boldsymbol{b}^j$ 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 $y = j * h_y$.

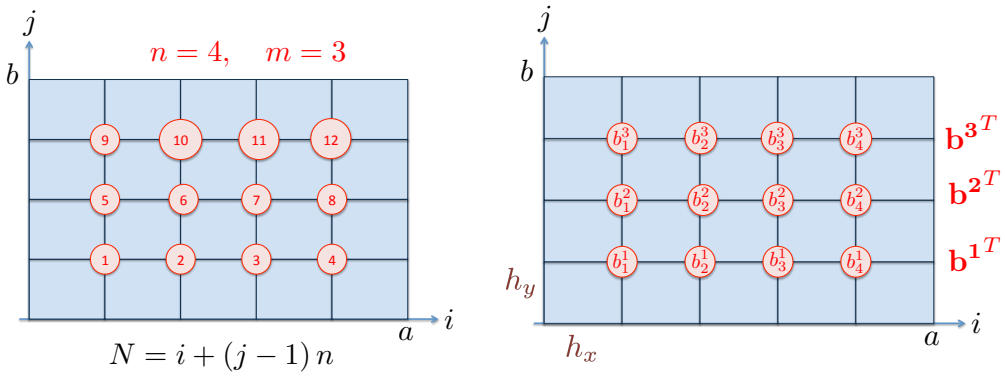


FIGURE 3.9 – Numbering

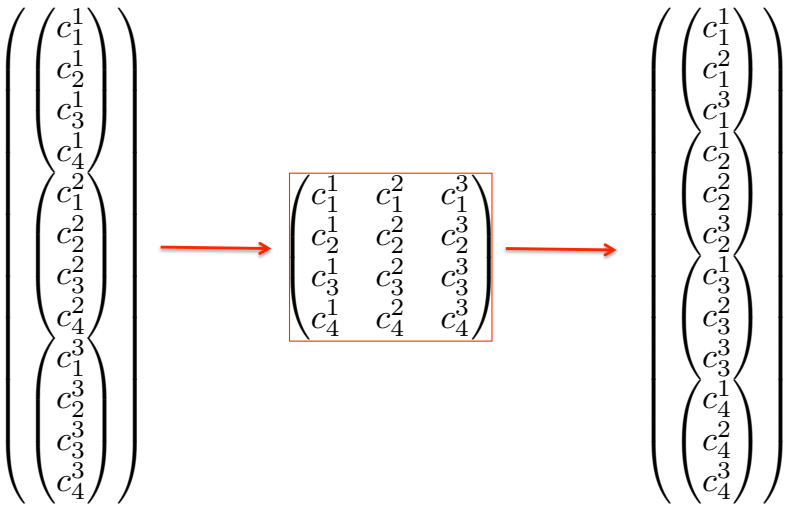


FIGURE 3.10 – Renumbering

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

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

```
for j=1:m
        C(:,j)=sig((j−1)*n+1:j*n )
end
```

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

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

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

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

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

That is we write

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

The total cost is

2 FFT : $2n\log_2(n)$,

n **resolutions in parallel of tridiagonal systems of size** m : m

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],[3], [6]. For the finite element part, we refer to [2].

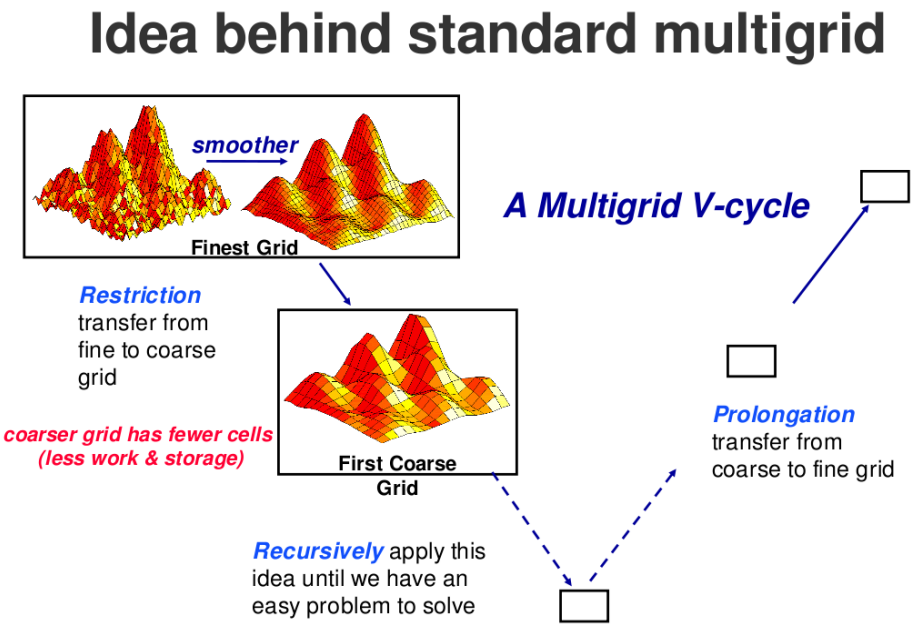


FIGURE 4.1 – scheme for a V-cycle, courtesy of David Keyes, Columbia University

4.1.1 The V- cycle process

One cycle of the multigrid method is given as follows. Suppose we have a grid of size h. We want to solve $A^h\bar{U}^h = b^h$. We take an initial guess U^h, $e^{h,0} = \bar{U}^h - U^h$, and define $MG(A^h, b^h, U^h)$

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

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

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

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

If the coarse grid is "sufficiently coarse", the coarse problem is solved exactly by a direct method.

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

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

$$h = \frac{1}{n+1}, \quad n = 2^\ell - 1, \quad A^h \bar{U}^h = f^h, \quad A^h \in \mathcal{M}_n(\mathbb{R})$$

$$2h = \frac{1}{\frac{n+1}{2}}, \quad n' = \frac{n-1}{2}$$

Example :

$$h = 2^{-3},\ n = 7, \quad 2h = 2^{-2},\ n' = 3.$$

$$A^h = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix}, \quad \bar{U}^h = \begin{pmatrix} \bar{U}_1^h \\ \bar{U}_2^h \\ \bar{U}_3^h \\ \bar{U}_4^h \\ \bar{U}_5^h \\ \bar{U}_6^h \\ \bar{U}_7^h \end{pmatrix}$$

$$A^{2h} = \frac{1}{4h^2}\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}, \quad U^{2h} = \begin{pmatrix} U_1^{2h} \\ U_2^{2h} \\ U_3^{2h} \end{pmatrix}$$

The Smoother

We will use one of the stationary methods , relaxed Jacobi or Gauss-Seidel. The matrix of the iteration is

$$S = I - \frac{1}{\omega} D^{-1} A \quad \text{or } I - (D-E)^{-1} A.$$

See chapter 1.

$$e^{h,1} = S^{N_1} e^{h,0}, \quad r^{h,1} = b^h - A^h U^{h,1} = A^h(\bar{U}^h - U^{h,1}) = A^h e^{h,1}. \tag{4.1}$$

Projection on the coarse grid

The fine grid is $(kh) = (\frac{k}{n+1})$ for $1 \le k \le n$. The coarse grid is $(k2h) = (\frac{2k}{n+1})$ for $1 \le k \le (n-1)/2$.

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

The matrix of P_h^{2h} is

$$P_h^{2h} = \begin{pmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{pmatrix} \quad \mathbb{R}^7 \to \mathbb{R}^3$$

$$P_h^{2h}V = \begin{pmatrix} \frac{1}{4}V_1 + \frac{1}{2}V_2 + \frac{1}{4}V_3 \\ \frac{1}{4}V_3 + \frac{1}{2}V_4 + \frac{1}{4}V_5 \\ \frac{1}{4}V_5 + \frac{1}{2}V_6 + \frac{1}{4}V_7 \end{pmatrix} \quad \mathbb{R}^7 \to \mathbb{R}^3$$

Define now

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

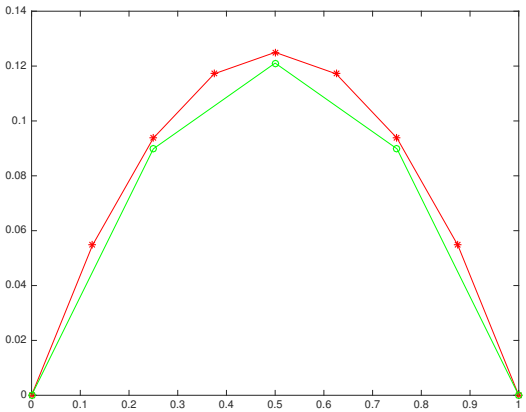


FIGURE 4.2 – Projection from fine to coarse grid

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 :

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

The matrix is

$$P_{2h}^h = \begin{pmatrix} \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \\ 0 & 0 & \frac{1}{2} \end{pmatrix} = 2(P_h^{2h})^T \quad \mathbb{R}^3 \to \mathbb{R}^7 \quad P_{2h}^h \begin{pmatrix} U_1 \\ U_2 \\ U_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}U_1 \\ U_1 \\ \frac{1}{2}(U_1 + U_2) \\ U_2 \\ \frac{1}{2}(U_2 + U_3) \\ U_3 \\ \frac{1}{2}U_3 \end{pmatrix}$$

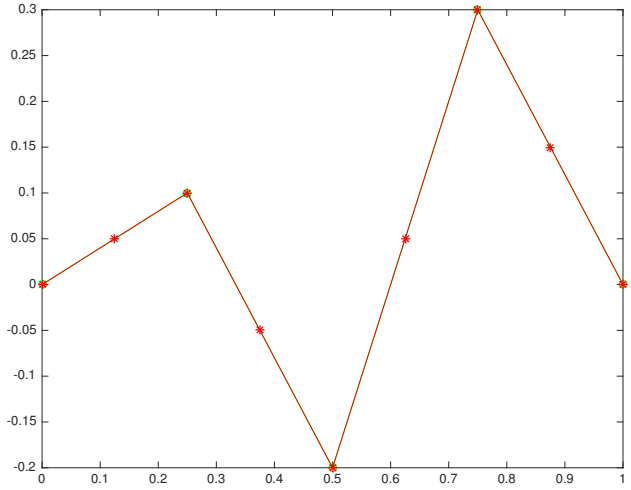


FIGURE 4.3 – Projection from coarse to fine grid

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

$$\mathcal{K}er\, P_h^{2h} A^h = \{V \in \mathbb{R}^n,\, V_{2j} = 0,\, j = 1 \cdots, (n-1)/2\}, \tag{4.2}$$

$$\mathcal{K}er\, P_h^{2h} A^h \oplus \mathcal{I}m P_{2h}^h = \mathbb{R}^n, \tag{4.3}$$

$$\forall V \in \mathbb{R}^{(n-1)/2}, \forall j, (A^h P_{2h}^h V)_{2j+1} = 0, \tag{4.4}$$

$$P_h^{2h} A^h P_{2h}^h = A^{2h}. \tag{4.5}$$

Proof It is easy to compute for $n = 7$,

$$P_h^{2h} A^h U = \frac{1}{h^2} \begin{pmatrix} \frac{1}{2}U_2 - \frac{1}{4}U_4 \\ -\frac{1}{4}U_2 + \frac{1}{2}U_4 - \frac{1}{4}U_6 \\ -\frac{1}{4}4_4 + \frac{1}{2}U_6 \end{pmatrix} = \frac{1}{4h^2} \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} U_2 \\ U_4 \\ U_6 \end{pmatrix} = A^{2h} \begin{pmatrix} U_2 \\ U_4 \\ U_6 \end{pmatrix}$$

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

$$P_h^{2h} A^h U = A^{2h} U^e. \tag{4.6}$$

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,

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

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

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

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

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

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 $U_{2j} = 0$. 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 $\mathbb{R}^{(n-1)/2}$ (coarse),

$$(A^h P^h_{2h} V)_{2j+1} = 0, \quad (A^h P^h_{2h} V)_{2j} = \frac{1}{2h^2}(-V_{j-1} + 2V_j - V_{j+1}) = 2(A^{2h} v)_j.$$

Then using (4.6), V in $\mathbb{R}^{(n-1)/2}$ (coarse),

$$P^{2h}_h A^h P^h_{2h} V = A^{2h}(P^h_{2h} V)^e = A^{2h} V$$

which finally gives (4.5). ■

Lemma 4.2

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

with

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

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

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

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

$$e^{h,1}_{2j} = (P^h_{2h} e^{2h})_{2j} = e^{2h}_j,$$

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

$$e^{h,1}_{2j+1} = d^h_{2j+1} + (P^h_{2h} e^{2h})_{2j+1} = d^h_{2j+1} + \frac{1}{2}(e^{2h}_j + e^{2h}_{j+1}) = d^h_{2j+1} + \frac{1}{2}(e^{h,1}_{2j} + e^{h,1}_{2j+2})$$

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

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

$$e^{h,3} = S^{N_2} \Pi_o \frac{h^2}{2} A^h S^{N_1} e^{h,0}$$

Spectral analysis

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

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

For any iterative method, the eigenfunctions of the iteration matrix are equal to those of A. Therefore The smoothing matrix S has eigenvalues λ_k, and eigenvectors $\Phi^{(k)}$. For relaxed Jacobi or the Gauss-Seidel algorithm, the eigenvalues are

$$\begin{aligned} \lambda_k^J(\omega) &= 1 - 2\omega \sin^2 (\frac{k\pi h}{2}) \quad \text{for } 1 \leq k \leq n, \\ \lambda_k^{GS} &= \cos^2 k\pi h \quad \text{for } 1 \leq k \leq n, \end{aligned}$$

Figure 4.4 shows the eigenvalues as a function of k for $n = 2^5 - 1$.

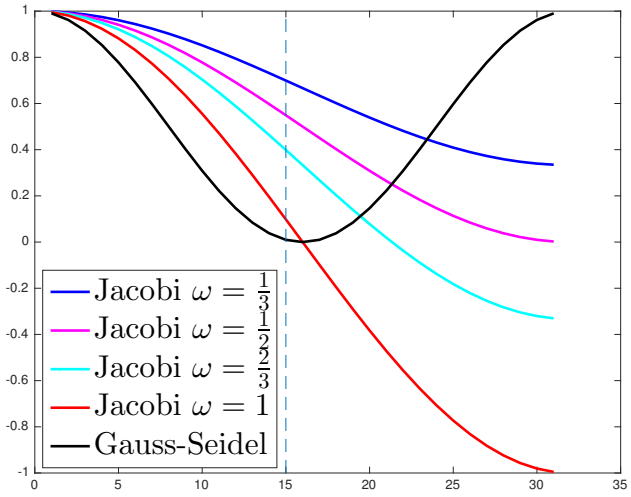


FIGURE 4.4 – Eigenvalues (39) of the relaxed Jacobi iteration matrix as a function of k for several values of ω together with Gauss-Seidel

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

$$-1/3 < 1 - 4/3 \sin^2(\frac{k\pi}{2(n+1)}) \leq 1 - 4/3 \sin^2(\frac{(n+1)\pi}{4(n+1)}) = 1/3$$

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

When using Gauss-Seidel as a smoother, one can observe that the eigenvalues are small in the neighbourhood of $k \sim (n-1)/2$.

For an initial error $e^{h,0} = \Phi^{(k)}$, $\#k$ eigenmode of A associated to eigenvalue μ_k, the error and residual after N_1 iterations are

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

From

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

we obtain

$$e^{h,2} = \frac{h^2}{2} \mu_k \lambda_k^{N_1} \Phi_{odd}^{(k)}.$$

It is easy to see that

$$\Phi_{2j}^{(n+1-k)} = -\Phi_{2j}^{(k)}, \quad \Phi_{2j+1}^{(n+1-k)} = \Phi_{2j}^{(k)}, \quad \Phi_{odd}^{(k)} = \frac{1}{2}(\Phi^{(k)} + \Phi^{(n+1-k)})$$

$$e^{h,2} = \frac{h^2}{4}\mu_k \lambda_k^{N_1}(\Phi^{(k)} + \Phi^{(n+1-k)})$$

If the same smoother is applied in postprocessing,

$$e^{h,3} = \frac{h^2}{4}\mu_k \lambda_k^{N_1}(\lambda_k^{N_2}\Phi^{(k)} + \lambda_{n+1-k}^{N_2}\Phi^{(n+1-k)})$$

For Gauss-Seidel $\lambda_{n+1-k}^{GS} = \cos^2\frac{(n+1-k)\pi}{n+1} = \lambda_k^{GS}$, and

$$\begin{aligned} e^{h,3} &= \tfrac{h^2}{4}\mu_k \lambda_k^{N_1+N_2}(\Phi^{(k)} + \Phi^{(n+1-k)}) \\ &= 2\sin^2\tfrac{k\pi h}{2}\cos^{2(N_1+N_2)}(k\pi h)\Phi_{odd}^{(k)} \\ &= 2\sin^2\tfrac{k\pi h}{2}\cos^{2(N_1+N_2)}(k\pi h)\Phi_{odd}^{(k)} \end{aligned}$$

The convergence factor over one round is therefore

$$(1-\cos(k\pi h))\cos^{2(N_1+N_2)}(k\pi h)$$

For relaxed Jacobi, $\lambda_{n+1-k}^J(\omega) + \lambda_k^J(\omega) = 2(1-\omega)$. and

$$e^{h,3} = \frac{h^2}{4}\mu_k \lambda_k^{N_1}(\lambda_k^{N_2}\Phi^{(k)} + (2(1-\omega) - \lambda_k)^{N_2}\Phi^{(n+1-k)}$$

$$e_{2j}^{h,3} = \frac{h^2}{4}\mu_k \lambda_k^{N_1}(\lambda_k^{N_2} - (2(1-\omega) - \lambda_k)^{N_2})\Phi_{2j}^{(k)}$$

$$e_{2j+1}^{h,3} = \frac{h^2}{4}\mu_k \lambda_k^{N_1}(\lambda_k^{N_2} + (2(1-\omega) - \lambda_k)^{N_2})\Phi_{2j+1}^{(k)}$$

Choose relaxed Jacobi with $\omega = 2/3$. For $(n+1)/2 \le k \le n$, $|\lambda_k^J| \le 1/3$, and we have

$$e_j^{h,3} = 2(1/3)^{N_1+N_2}|\Phi_j^{(k)}|$$

and for $1 \le k \le (n-1)/2$,

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

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. $n = 10$.

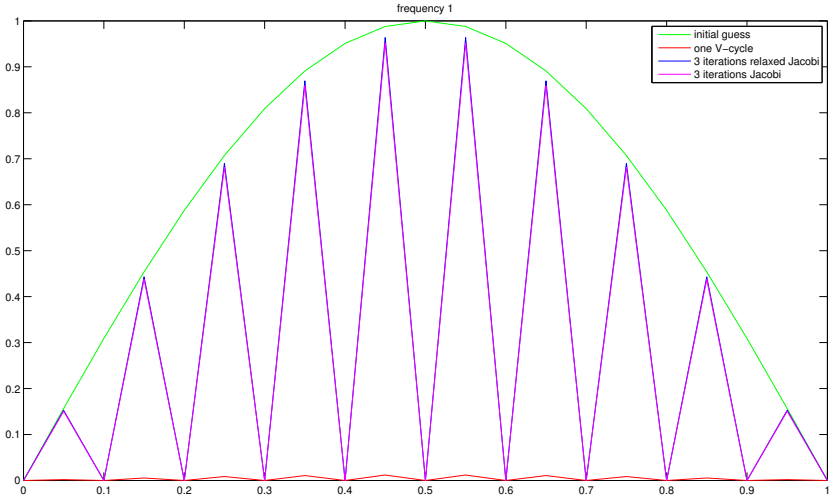


FIGURE 4.5 – Comparison of the iterative methods. Initial guess $\sin \pi x$

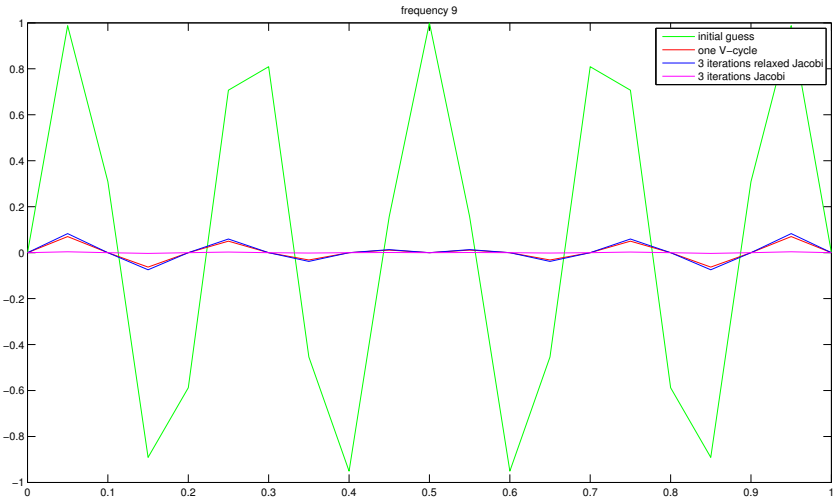


FIGURE 4.6 – Comparison of the iterative methods. Initial guess $\sin(n-1)/2\pi x$.

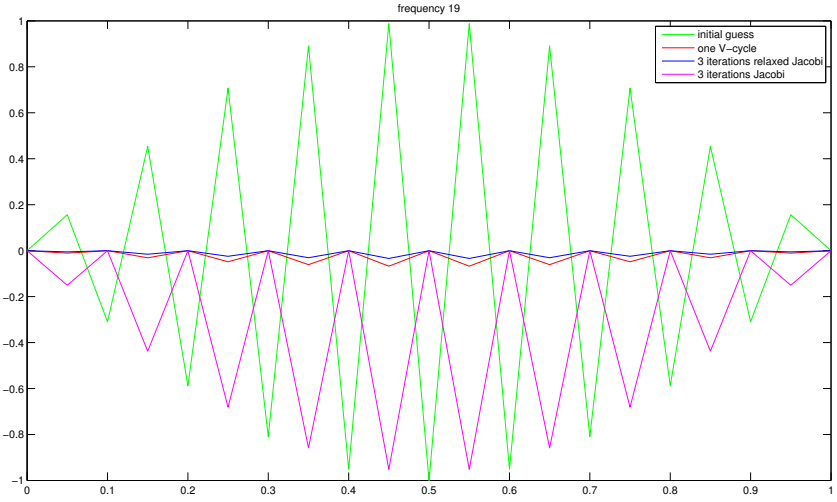
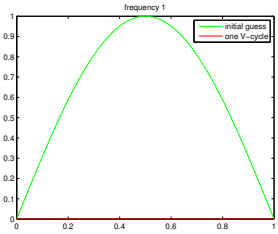
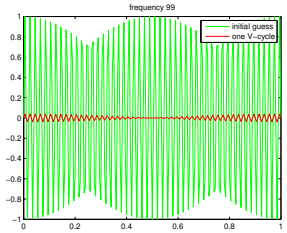


FIGURE 4.7 – Comparison of the iterative methods. Initial guess $\sin(n-1)\pi x$.

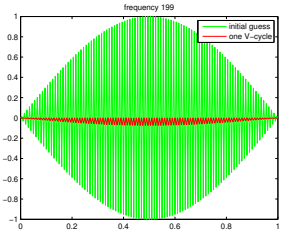
THE EFFECT OF ONE V-CYCLE ON ONE SINGLE MODE FOR $n = 201$.



Initial guess $\sin \pi x$



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



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

4.1.2 L^∞ *estimates*

Suppose the computation on the coarse grid can be done exactly, and use relaxed Jacobi with $\omega = 2/3$. Then $S = I - h^2 A^h/3 = 1/3B$ where

$$B = \begin{pmatrix} 1 & 1 & & & \\ 1 & 1 & 1 & & 0 \\ & \ddots & \ddots & \ddots & \\ 0 & & 1 & 1 & 1 \\ & & & 1 & 1 \end{pmatrix}.$$

Compute

$$(SU)_j = \frac{1}{3}(U_{j-1} + U_j + U_{j+1}), \Rightarrow \|SU\|_\infty \leq \|U\|_\infty.$$

Furthermore

$$h^2(SA^h e^h)_{2j+1} = \frac{1}{6}(-e^h_{2j-2} + e^h_{2j-1} + e^h_{2j+1} - e^h_{2j+2}),$$

from which we deduce that

$$max_j |h^2(SA^h e^h)_{2j+1}| \leq \frac{2}{3}\|e^h\|_\infty.$$

Therefore we obtain

$$\|e^{h,2}\|_\infty = \sup_j |\frac{h^2}{2} A^h r^{h,1}_{2j+1}|$$

from formula (**??**), we deduce for $\omega = 2/3$:

$$e^{h,2}_{2j+1} = -\frac{1}{6}e_{2j-1} + \frac{1}{6}e_{2j} + \frac{1}{6}e_{2j+2} - \frac{1}{6}e_{2j+3}.$$

and therefore we have the error estimate

$$e^{h,2}_{2j} = 0, \quad |e^{h,2}_{2j+1}| = \frac{2}{3}\|e_h\|_\infty.$$

Since the smoothing produces for any $U \in \mathbb{R}^{2n-1}$

$$(SU)_j = \frac{1}{3}(U_{j-1} + U_j + U_{j+1}),$$

we have

$$\|SU\|_\infty \leq \|U\|_\infty,$$

and

$$\|e^{h,3}\|_\infty \leq \frac{2}{3}\|e_h\|_\infty.$$

THE CONVERGENCE IS INDEPENDENT OF THE SIZE OF THE MATRIX

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 4.1 – 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)

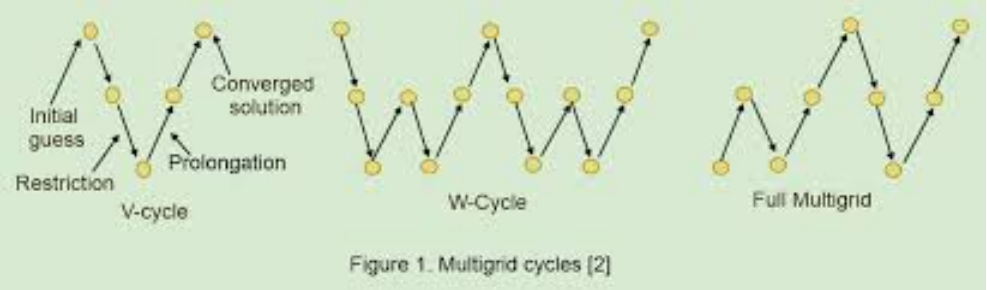


FIGURE 4.8 – full multigrid

```
function A=A1d(eta,a,b,n)
% A1D one dimensional finite difference approximation
%   A=A1d(a,b,J) computes a sparse finite difference
%   approximation of the one dimensional operator -Delta on the
%   domain Omega=(a,b) using n interior points

h=(b-a)/(n+1);
e=ones(n,1);
A=spdiags([-e/h^2 (eta+2/h^2)*e -e/h^2],[-1 0 1],n,n);
```

```
% resolution of AU=b by 1 V cycle, starting with a sin function
% size of the matrix A
clear all;close all;
%I= 1 smoother Jacobi, I=2 smoother Gauss-Seidel
I=1
N=2^5;
n=2*N-1;
% Smoother de depart
N1=2;
% smoother arrivee
N2=1;

h=1/(n+1);
x=0:h:1;
A=A1d(0,0,1,n);
b=ones(n,1);
X=linspace(0,1,1000);
Uex=X.*(1-X)/2;
plot(X,Uex,'y','Linewidth',3)
hold on
pause
 udex=A\b; Udex=[0;udex;0]
plot(x,Udex,'m','Linewidth',2)
pause
hold on
k=N;
%initial guess
U=sin(pi*x')+sin(N*pi*x')+sin(n*pi*x');
%U=sin(k*pi*x');
uo=U(2:end-1);
plot(x,U,'b*-')
pause
hold on

% N1 iterations of the smoother
% I=1 Jacobi with parameter 2/3
if I==1
    a=2/3;
elseif I==2
    % I=2 Gauss-Seidel
Lo=tril(A,-1);
Up=triu(A,1);
D=diag(diag(A));
end
u=uo;
```

```
pause

for i=1:N1
    if I==1 % relaxed Jacobi
        u=u+a*(b-A*u)./diag(A);
    elseif I==2  % Gauss-Seidel

        u=(D+Lo)\(-Up*u+b);
    end
    U=[0;u;0];
    plot(x,U,'*-m')
    pause
    hold on



end;

%pause
r=b-A*u;
n2=(n+1)/2-1;
% projection on the coarse grid
figure(2)
for j=1:n2
    v(j)=(r(2*j-1)+2*r(2*j)+r(2*j+1))/4;
end
xf=x(1:2:end);
V=[0;v';0]
plot(xf,V,'--og','Linewidth',2)
hold on
pause
% exact resolution on the coarse grid
A1=A1d(0,0,1,n2);
w=A1\v';
V=[0;w;0]
plot(xf,V,'--og','Linewidth',2)
% projection on the fine grid
w(n2+1)=0;
wf(1)=w(1)/2;
for j=1:n2
    wf(2*j)=w(j);
    wf(2*j+1)=(w(j)+w(j+1))/2;
end
V=[0;wf';0]
plot(x,V,'--ob','Linewidth',2)
hold off
pause
% ajouter a u
figure(1)
u=u+wf';
U=[0;u;0];
plot(x,U,'c')
pause

% N2 iterations of the smoother
for i=1:N2,
```

```
    if I==1 % relaxed Jacobi
        u=u+a*(b-A*u)./diag(A);
    elseif I==2  % Gauss-Seidel

        u=(D+Lo)\(-Up*u+b);
    end
    % relaxed Jacobi
end;
umg=u;
U=[0;umg;0]
plot(x,U,'r','Linewidth',1)
errmg=  norm(udex-umg)/norm(udex);
figure(2)
%Jacobi or Gauss-Seidel without coarse grid
a=1
Nt=N1+N2;
u=uo;
for i=1:Nt
    if I==1 % relaxed Jacobi
        u=u+a*(b-A*u)./diag(A);
    elseif I==2  % Gauss-Seidel

        u=(D+Lo)\(-Up*u+b);
    end
    U=[0;u;0];
    plot(x,U,'k')
    pause
    hold on
end;
hold off

errd= sqrt(h)*norm(udex-u);
uo=umg;

% 2^4   0.0015 5.1839e-05   1.5898e-06
% 2^6   9.7793e-05  3.6067e-06  1.3291e-07
% 2^9   1.5314e-06  5.6711e-08   2.1003e-09
```

4.2 Algebraic Multigrid AMG

4.2.1 Introduction

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 [4]).

Algebraic multigrid (see [1]) 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 [3])

In the following, we will assume that the problem to be solved $Au = f$ where A is a real $n \times *n$ symmetric definite positive matrix , u and f vectors in $\mathbb{R}^n$.
Coarse-grid correction involves operators that transfer information between fine and coarse "grids", which are denoted in linear algebra terms simply as the vector space $\mathbb{R}^n$ and the lower-dimensional (coarse) vector space $\mathbb{R}^{nc}$. Interpolation (prolongation) maps the coarse grid to the fine grid and is just the $n \times nc$ matrix $P : \mathbb{R}^{nc} \rightarrow \mathbb{R}^n$. Restriction maps the fine grid to the coarse grid and is the transpose of interpolation (P^T). in this work.

The 2 grid method to solve our problem is then defined as before as follows

Do ν_1 smoothing steps on Au = f ;
Compute residual $r = f - Au = Ae$;
Solve $A_c E_c = P^T r$;
Correct $u \leftarrow u + Pe_c$;
Do ν_2 smoothing steps on $Au = f$

Because AMG is based only on the matrix A, there are few options for defining the coarse systemA_c The most common approach is to use the Galerkin operator, $A_c = P^T AP$, which has the nice property that it minimizes the error after correction (in the energy norm).

The adjacency graph of the matrix plays an important role in AMG. The graph has a directed edge from vertex i to vertex j for every nonzero entry a_{ij} in the matrix A (see Figure 4.9). The grid in AMG is simply the set of vertices in the graph, i.e., grid point i is just vertex i. If the linear system comes from the discretization of a PDE, then we can draw the grid points in their actual geometric locations along with the associated graph. We illustrate this in Figure 4.9 for a simple 2D Laplacian problem.

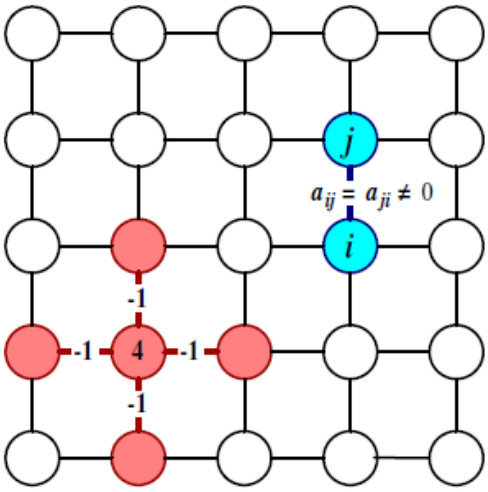
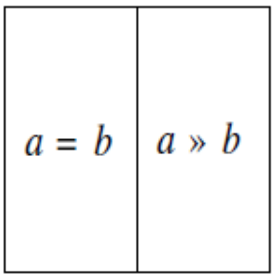


FIGURE 4.9 – 2D Laplacian adjacency graph

Algebraic smoothness
In AMG, the smoother is generally fixed to be a simple pointwise method such as Gauss-

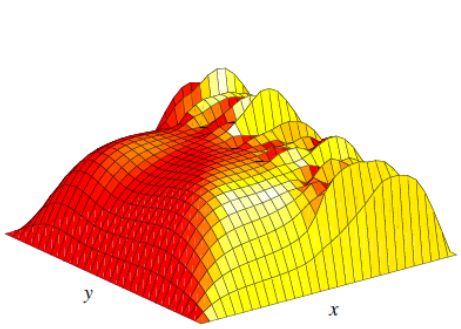
Seidel. Error not eliminated by the smoother is called smooth error, and must be handled by coarse-grid correction. In the classical geometric multigrid setting, smooth error is smooth in the usual geometric sense. In the AMG setting, however, smooth error may actually be geometrically oscillatory. We often use the term algebraically smooth to be clear about the distinction. To see this, consider the following simple 2D example discretized by finite elements on a uniform mesh :

$$\begin{aligned} -au_{xx} - bu_{yy} &= f \quad \text{on } \Omega \\ u &= g \quad \text{on } \Gamma \end{aligned}$$

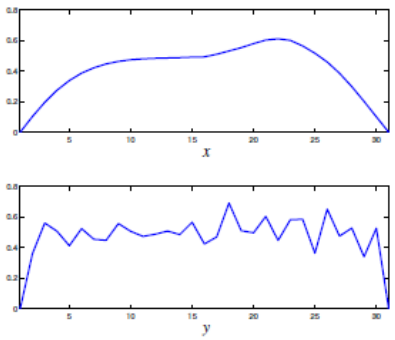


Problem 1

Figure 4.10a shows the error after 7 Gauss-Seidel iterations. We see that the error is geometrically smooth in both the x and y directions in the left-half plane where the problem is isotropic, but it is geometrically oscillatory in the y direction in the right-half plane where the problem is anisotropic (Figure 4.10b).



(a) Smooth Error after 7 Gauss Seidel iterations



(b) Error along x and y

FIGURE 4.10 – Smoothing error

As illustrated in figure 4.11, AMG coarsens in directions of geometric smoothness. That is, in the left-half plane, the grid is coarsened in both directions (so-called full coarsening), but in the right-half plane, the grid is coarsened only in the x direction (so-called semi-coarsening). This ability of AMG to "follow the physics" during coarse-grid correction is another advantage it has over geometric approaches.

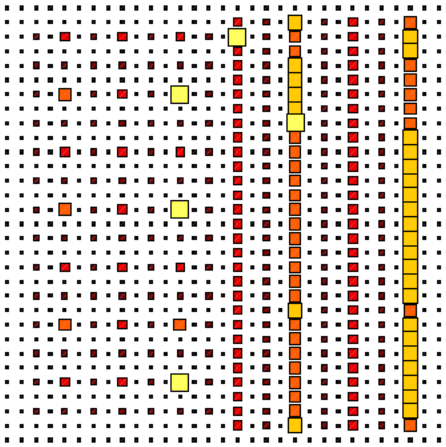


FIGURE 4.11 – Coarse grid

Larger squares correspond to coarser grids.

The key to designing an effective AMG algorithm is to have a good characterization of smooth error. In general, smooth error corresponds to eigenvectors of A with small associated eigenvalues (we call these small eigenmodes for short). In other words, smoothing damps large eigenmodes, leaving coarse-grid correction to eliminate the remaining small eigenmodes of A. The smaller the eigenmode, the more effective must be coarse-grid correction. This makes the smallest of the eigenmodes, called the near null space or near kernel of A, particularly important in the design of AMG algorithms.

As an example, we again consider the laplace problem above. Any linear function u is in the kernel of the differential operator since both u_{xx} and u_{yy} are zero. The same is true for the discrete operator A (away from boundaries). That is, the near null space of A for this problem consists of any vector that is almost linear when plotted on the grid. Hence, it makes perfect sense for AMG to coarsen in directions of geometric smoothness, as shown in Figure 2. The example underscores the distinction between smooth error (error not eliminated by the smoother) and the near null space (the smallest eigenmodes of A). In the example, smooth error consists of functions that are geometrically both smooth and oscillatory, while the near null space contains only geometrically smooth functions. For applications where the near null space contains geometrically oscillatory functions (such as electromagnetics), the approach of coarsening in directions of geometric smoothness is not sufficient.

4.2.2 AMG

This method is the classical AMG presented by Brandt, McCormick, Ruge, and StÃ¼ben in *Algebraic multigrid (AMG) for sparse matrix equations* in In D. J. Evans, editor, Sparsity and Its Applications. Cambridge University Press, Cambridge, 1984.

Choosing the coarse grid

The coarse grid is a subset of the fine grid. Points are chosen such that the grid is coarsened in directions of strong connections of the matrix. The procedure for doing this is actually quite simple.

Let us first define the notion of strengh connection.

Strength of Connection : Given a threshold θ, we say that variable u_i strongly depends on variable u_j if

$$a_{ij} \geq \theta \; max_{k \neq i}(-a_{ik})$$

1) Define a strength matrix, As, by deleting weak connections in A ;
2) First pass : Choose an independent set of fine grid points based on the graph of As ;
3) Second pass : Choose additional points if needed to satisfy interpolation requirements ;

The coarsening procedure partitions the grid into C-points (points on the coarse grid) and F-points (points not on the coarse grid).

Figure 4.12 illustrates the first pass of the algorithm for a 2D Laplacian problem discretized with finite elements on a uniform mesh. The discretization stencil is given by

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Since all of the off diagonal coefficients are -1, the connections in the matrix are all strong connections, regardless of the parameter choice θ. Thus A_s and A are the same.

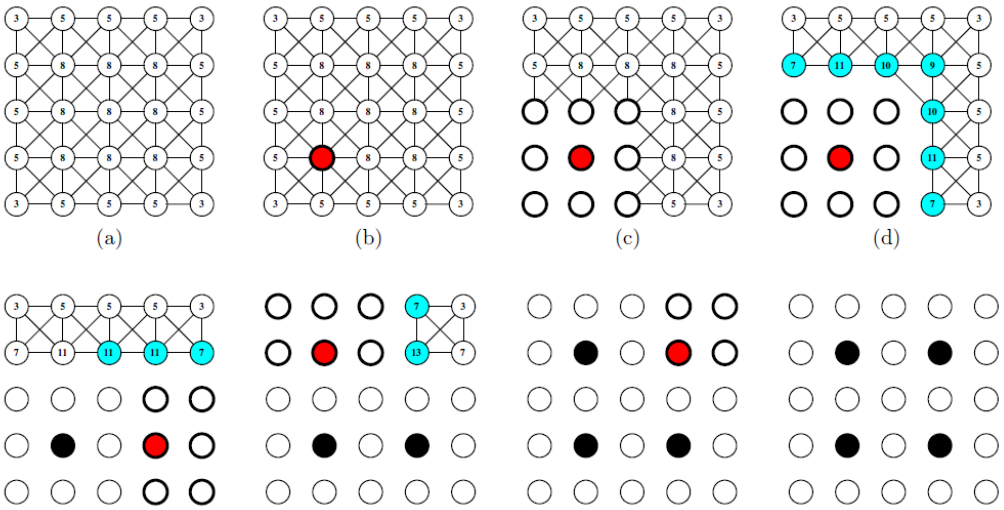


FIGURE 4.12 – Coarsening process

Illustration of the first pass of the AMG coarsening algorithm for a 9-point discretization stencil.
(a) The nodes of the graph of the strength matrix are assigned a weight equal to the number of off-diagonal connections.
(b) A point with maximal weight is chosen as a C-point.
(c) The neighbors of the new C-point are set to be F-points.
(d) For each new F-point, the weights of its neighbors are increased by one to make them more likely to be chosen next. The algorithm continues in this way until all points are either C-points or F-points.

The original AMG interpolation scheme (described below) requires each pair of strongly connected F-points to be strongly connected to a common C-point. The second pass of the coarsening algorithm searches for F-point pairs that do not satisfy this requirement, and changes one of them to a C-point. Researchers later found that the second pass leads to high computational costs, and they have largely abandoned it in favor of other approaches for defining interpolation.

As we saw in the example of Figure 4.11, the AMG coarsening algorithm is able to produce standard fully coarsened and semi-coarsened grids, and combinations thereof. The strength matrix is the key to making this happen, but as we see later, it can sometimes be sensitive to the choice of strength parameter θ. Some researchers today are exploring more reliable definitions of strength, while others are exploring completely different coarsening approaches based on so-called compatible relaxation that avoid defining strength altogether. Another area of active research is parallel coarsening algorithms. It is easy to see that the algorithm in Figure 4.12 is inherently sequential. Unfortunately, most parallel coarsening algorithms lead to increased computational costs and often degrade convergence.

Defining interpolation

We again use the fact that smooth error e is characterized by small eigenmodes

Since the residual r = Ae is such that $||r||$ small. We take this to the extreme and assume that

$$r_i = (Ae)_i = 0$$

If we rewrite this equation at an F-point i in terms of the coefficients of A, some regrouping leads to

$$a_{ii}e_i = -\sum_{j \in C_i} a_{ij}e_j - \sum_{j \in F_i^s} a_{ij}e_j - \sum_{j \in N_i^w} a_{ij}e_j \tag{4.7}$$

where :
C_i : C-points strongly connected to i
F_i^s : F-points strongly connected to i
N_i^w : all points weakly connected to i

The set C_i is the set of interpolatory points. That is, these are the points that F-point i will interpolate from.

The trick to deriving interpolation is to rewrite e_j in the last two terms of (4.7) in terms of either the interpolatory points in C_i or the F-point i. This produces an equation that involves only the F point and its interpolatory points, which we can use directly to define interpolation. This process is sometimes referred to as "collapsing the stencil", and it is illustrated in the two figures below for two finite element stencils.

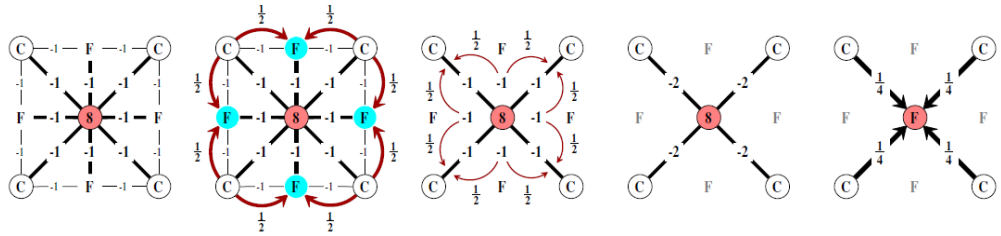


FIGURE 4.13 – Derivation of AMG interpolation for the standard 9-point finite element stencil. In the second image, we assume that strongly-connected F-points are interpolated from neighboring interpolatory points. The weights (all 1/2) are chosen based on the underlying matrix entries such that the constant function is interpolated exactly. In the third image, we "redistribute" the strong F connections according to the interpolation weights in the previous step. This produces the "collapsed stencil" in the fourth image, which leads directly to the interpolation rule in the last image.

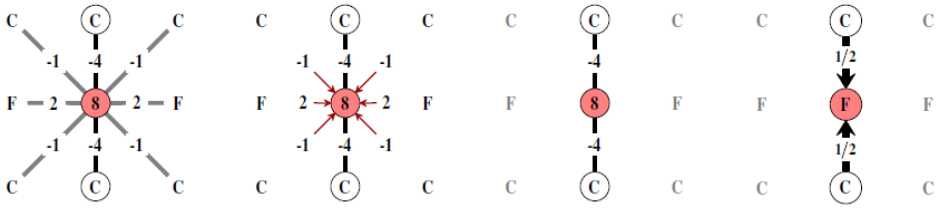


FIGURE 4.14 – Derivation of AMG interpolation for an anisotropic 9-point finite element stencil. In the second image, weak coefficients are added to the diagonal to produce the "collapsed stencil" in the third image, which leads directly to the interpolation rule in the last image

The stencil in Figure 4.14 helps to illustrate one of the potential problems with the strength of connection definition used in AMG. The stencil comes from a quadrilateral finite element discretization of the Laplacian on a mesh that is highly stretched in the x direction. The resulting problem is strongly anisotropic in the y direction, yet this strong anisotropy is not reflected in the size of the off-diagonal entries. In fact, any value of the strength threshold θ that is less than or equal to 0.25 will turn the corner couplings into strong connections. The resulting interpolation has 6 interpolatory points instead of 2, and degrades the convergence of AMG.

Fine Grid	Iterations	Convergence factor	Coarse grids	Grid complexity	Operator complexity	Setup time	Solve time
31×31	9	0.19	5	1.6	1.7	-	-
61×61	10	0.23	6	1.6	1.6	0.01	0.02
121×121	9	0.23	8	1.6	1.7	0.05	0.07
241×241	9	0.23	9	1.6	1.7	0.25	0.32
481×481	9	0.23	12	1.7	1.7	1.02	1.27
961×961	11	0.29	13	1.7	1.7	4.42	6.28

Table 1

Table 1 : AMG results for Problem (1) for different grid sizes with strength threshold $\theta = 0.4$, and with $\nu_1 = \nu_2 = 1$ smoothing steps of C-F Gauss-Seidel. Iterations were done until the relative residual was reduced below 10^{-9}. Grid complexity is the total number of grid points on all grids divided by the number of grid points on the fine grid. Operator complexity is the total number of nonzeroes in the linear operators on all grids divided by the number of nonzeroes in the fine grid operator. Setup time is the time required to choose coarse grids and build interpolation, restriction, and coarse-grid operators.

To conclude on AMG, interest in AMG methods is high, and probably still rising, because of the increasing importance of terascale simulations on unstructured grids. AMG has been shown to be a robust, efficient solver on a wide variety of problems of real-world interest. Much research is underway to find effective ways of parallelizing AMG, which is essential to large scale computing.

Chapitre 5

Parallelism

5.1 Substructuring methods

Contents

Principle
— Split the domain into sub-domains,
— solve a "condensed interface problem" : uses solving 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 Finite element method

Consider the Laplace equation with Dirichlet data

$$\left\{ \begin{array}{rcll} -\Delta u + u & = & f & \text{dans } \Omega, \\ u & = & g & \text{sur } \partial\Omega \end{array} \right.$$

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

$$u \in H^1(\Omega),\ u = g \text{ on } \partial\Omega$$
$$\forall v \in V, \quad a(u,v) = (f,v) \quad \text{with } a(u,v) = \int_\Omega \nabla u(x)\nabla v(x)dx + \int_\Omega u(x)v(x)dx$$

We introduce a triangulation $\mathcal{T}_h = \cup K$ with N_Ω vertices S_i, $i \in I_\Omega$ inside the open set Ω, and $N_{\partial\Omega}$ vertices S_i , $i \in I_{\partial\Omega}$ on the boundary.

$$V_h = \{v \in \mathcal{C}^0(\Omega) \cap H_0^1(\Omega), \forall K \in \mathcal{T}_h, v_h|_K \in \mathbb{P}_1\}.$$

where $\mathbb{P}_1$ is the space of polynomials of degree lower than 1 in two variables $(ax + by + c)$. The discrete formulation in V_h now is to find u_h such that

$$\forall v_h \in V_h, \quad a(u_h, v_h) = (f, v_h).$$

A basis of V_h is given by the functions φ_i, basis function associated to S_i by $\varphi_i(S_j) = \delta_{ij}$, as described in Figures 5.1, 5.2. The expansion of u_h on the basis

$$u_h = \sum_{j \in I_\Omega} u_h(S_j)\varphi_j + \sum_{j \in I_{\partial\Omega}} u_h(S_j)\varphi_j.$$

If the Dirichlet data are homogeneous ($g = 0$), then the second sum does not exist. Choose in the discrete formulation $v_h = \varphi_i$ for $i \in I_\Omega$.

$$\forall i \in I_\Omega, \quad \sum_{j \in I_\Omega} u_h(S_j)a(\varphi_i, \varphi_j) + \sum_{j \in I_{\partial\Omega}} u_h(S_j)a(\varphi_i, \varphi_j) = (f, \varphi_i).$$

We call K^I the square matrix of size $N_\Omega \times N_\Omega$ corresponding to interior basis functions,

$$K^I_{ij} = a(\varphi_i, \varphi_j), \quad (i,j) \in I_\Omega,$$

K^{IB} the matrix of size $N_\Omega \times N_{\partial\Omega}$

$$K^{IB}_{ij} = a(\varphi_i, \varphi_j), \quad i \in I_\Omega,\ j \in I_{\partial\Omega}.$$

$$\forall i \in I_\Omega, \quad \sum_{j \in I_\Omega} K^I_{ij} u_h(S_j) + \sum_{j \in I_{\partial\Omega}} K^{IB}_{ij} u_h(S_j) = (f, \varphi_i).$$

Define the vector of unknowns $U^I = \{u_h(S_j)\}_{j \in I_\Omega}$ and the boundary vector $U^B = \{u_h(S_j)\}_{j \in I_{\partial\Omega}}$. Define the vector of interior data $F = \{(f, \varphi_i)\}_{j \in I_\Omega}$ and the boundary data $G = \{(g, \varphi_i)\}_{j \in I_{\partial\Omega}}$.

$$K^I U^I + K^{IB} U^B = F, \quad U^B = G.$$

K is the matrix of the laplacian with homogeneous Dirichlet data in V_h, therefore it is symmetric definite positive.

FIGURE 5.1 – Basis hat functions

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

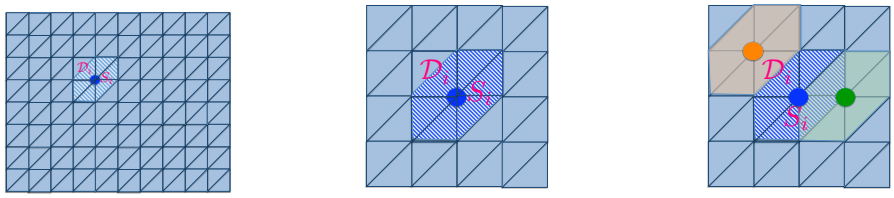


FIGURE 5.2 – Mesh, $\mathcal{D}_i$ support of the basis function φ_i associated to vertex S_i.

5.1.2 Finite element method and the Neumann problem

Consider back to one domain, and the problem

$$\left\{\begin{array}{rcll} -\Delta u+u & = & f & \text{dans } \Omega,\\ \frac{\partial u}{\partial n} & = & g & \text{sur } \partial\Omega \end{array}\right.$$

We write a variational formulation in $V=H^1(\Omega)$,

$$\begin{array}{l} u\in H^1(\Omega)\\ \forall v\in V,\quad a(u,v)=(f,v)+\int_{\partial\Omega}g(s)v(s)ds \end{array}$$

Add to the notations in 5.1.1, K^{BB} the matrix of size $N_{\partial\Omega}\times N_{\partial\Omega}$

$$K^{BB}_{ij}=a(\varphi_i,\varphi_j),\quad (i,j)\in I_{\partial\Omega}.$$

$$\forall i\in I_\Omega,\quad \sum_{j\in I_\Omega}K^I_{ij}u_h(S_j)+\sum_{j\in I_{\partial\Omega}}K^{IB}_{ij}u_h(S_j)=(f,\varphi_i).$$

$$\forall i\in I_{\partial\Omega},\quad \sum_{j\in I_\Omega}K^{IB}_{ij}u_h(S_j)+\sum_{j\in I_{\partial\Omega}}K^{BB}_{ij}u_h(S_j)=(f,\varphi_i)+(g,\varphi_i).$$

which gives in matrix form

$$\begin{array}{l} K^IU^I+K^{IB}U^B=F^I\\ (K^{IB})^TU^I+K^{BB}U^B=F^B+G. \end{array}$$

This is the matrix form finite elements discretisation of the Neumann boundary condition. The first line is the internal discretisation, while the second line involves the Neumann condition.

5.1.3 The Schur Complement method

The domain Ω is split into two nonoverlapping subdomains Ω_1 and Ω_2, and Γ is the common boundary.

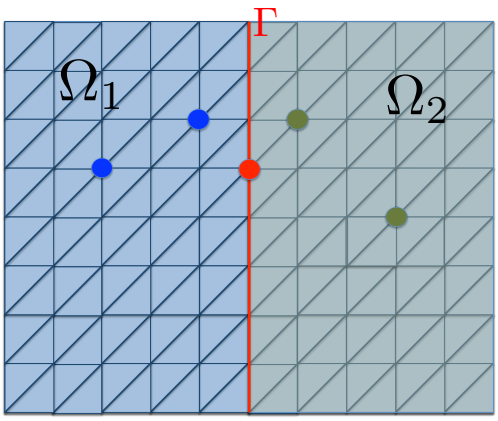


FIGURE 5.3 – Domain Decomposition

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

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

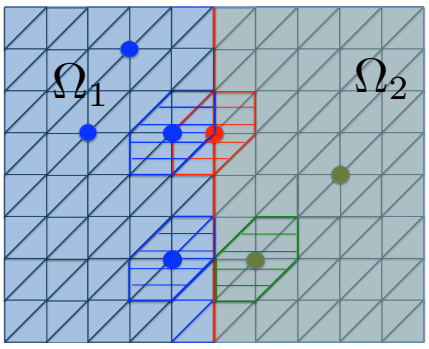


FIGURE 5.4 – Supports

$$a(u_h, \varphi_i) = \sum u_h(S_j)a(\varphi_i, \varphi_j)$$

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

$S_i \in \Omega_1,\ S_j \in \Omega_2 \ \Rightarrow \mathcal{D}_i \cap \mathcal{D}_j = \emptyset \ \Rightarrow a(\varphi_i, \varphi_j) = 0 \Rightarrow$ second sum vanishes

$S_i \in \Omega_2,\ S_j \in \Omega_1 \ \Rightarrow \mathcal{D}_i \cap \mathcal{D}_j = \emptyset \ \Rightarrow a(\varphi_i, \varphi_j) = 0 \Rightarrow$ first sum vanishes

For $S_i \in \Gamma$, all sums contribute. For the last one, the support of S_i is split according to Figure 5.5.

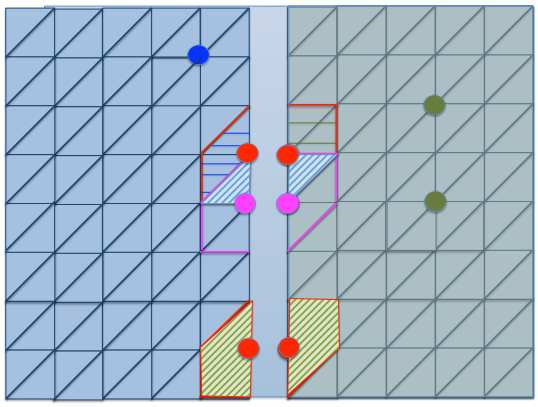


FIGURE 5.5 – Decomposition of the interface nodes

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

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

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_{jj} is the matrix of the Laplace problem in Ω_j with homogeneous Dirichlet boundary conditions on $\partial\Omega_j$, 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 for 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.*

Proof Compute (SU_3, U_3) by defining $U_1 = -K_{11}^{-1}K_{13}U_3$, $U_2 = -K_{22}^{-1}K_{23}U_3$, so that

$$SU_3 = K_{33}U_3 + K_{31}U_1 + K_{32}U_2.$$

Define $U = (U_1, U_2, U_3)$. Then

$$(KU, U) = (U_1, K_{11}U_1 + K_{13}U_3) + (U_2, K_{21}U_1 + K_{23}U_3) + (U_3, K_{31}U_1 + K_{32}U_2 + K_{33}U_3) = (SU_3, U_3).$$

■

The computations will be made in parallel as

$$S = S^1 + S^2$$

with

$$S^i = K_{33}^i - K_{3i}K_{ii}^{-1}K_{i3}$$

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

The first two equations in (5.2) is the resolution of Laplace equations. But what is the third one ? A Neumann condition $\partial_{n_1}u_1 = g$ in Ω_1 would be written

$$K^{31}U^1 + K_{33}^1U_3 = F_3^1 + G.$$

A Neumann condition $\partial_{n_2}u_2 = -g$ in Ω_2 would be written

$$K^{32}U^2 + K_{33}^2U_3 = F_3^2 - G.$$

Adding these two equations yields

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

which therefore is the discrete version of

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

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.

5.1.4 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

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

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)
$U_{ii}X_i = Z_i - U_{i3}X_3$

5.1.5 The conjugate gradient algorithm

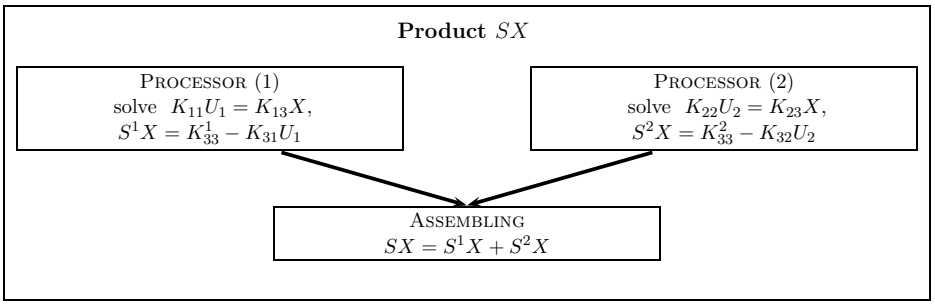
$$SU_3 := K_{33}U_3 - K_{31}K_{11}^{-1}K_{13}U_3 - K_{32}K_{22}^{-1}K_{23}U_3 = G_3$$
$$\text{with } G_3 = F_3 - K_{31}K_{11}^{-1}F_1 - K_{32}K_{22}^{-1}F_2$$

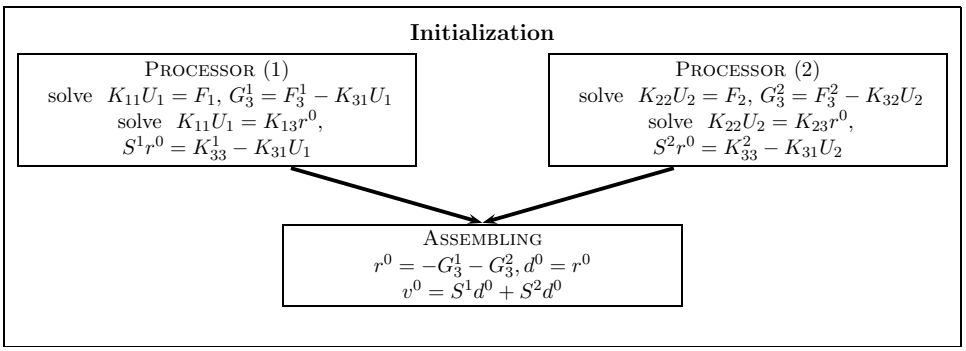
S is a symmetric positive definite matrix. The conjugate gradient algorithm reduces to a descent method,defined by the initial guess U_3^0,the initial descent direction $d^0 = r^0 = SU_3^0 - G_3$. Let r^k be the residual a step k. The next step will be

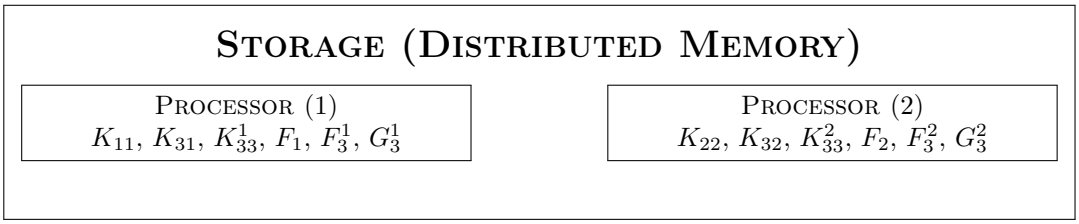
$$\begin{aligned} v^k &= Sd^k \\ \rho^k &= \frac{\|r^k\|^2}{(v^k, d^k)} \\ U_3^{k+1} &= U_3^k - \rho^k d^k \\ r^{k+1} &= r^k - \rho^k v^k \\ d^{k+1} &= r^{k+1} + \frac{\|r^{k+1}\|^2}{\|r^k\|^2} d^k \end{aligned}$$

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 :







ITERATION

$$v^k = Sd^k$$

$$\rho^k = \frac{\|r^k\|^2}{(v^k, d^k)}$$

$$U_3^{k+1} = U_3^k - \rho^k d^k$$

$$r^{k+1} = r^k - \rho^k v^k$$

$$d^{k+1} = r^{k+1} + \frac{\|r^{k+1}\|^2}{\|r^k\|^2} d^k$$

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

5.1.6 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.

5.1.7 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

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

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

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

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

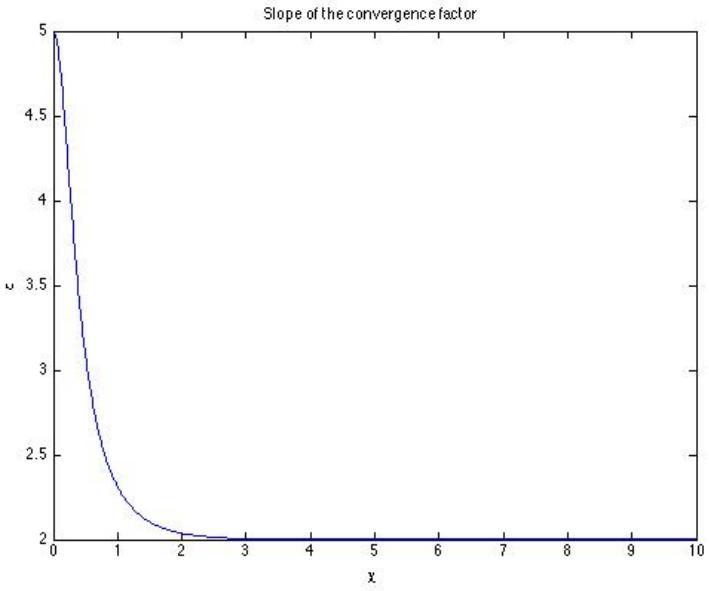
If the geometry is symmetric, that is if $b - c = c - a$, then the convergence factor reduces to

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

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 $\chi = \sqrt{\eta}/5$, then

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

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



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

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

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

5.1.8 Appendix : matlab scripts in 1-D

```
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
```

```
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
```

```
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
```

```
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];
```

```
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;
```

```
sol=SolveDD(f',eta,a,b,ga,gb);
% position de l interface
Nc=floor(length(x)/2);
Nc=2;
c=a+Nc*h;
% nombre d'iterations
Imax=10;
%parametre de relaxation
t=0.5;
% initialisation avec la valeur exacte
g1=y(Nc+1);
% ou initialisation avec 0
g1=0;
[g,u1,u2]=algoDN(f,eta,a,b,h,ga,gb,g1,Nc,Imax,t)
% algorithme
figure(99)
plot(g)
title('Interface value')
xlabel('Iteration number')

% Methode de Schur
u=algoSchur(f',eta,a,b,h,ga,gb,Nc);
%plot(x,y,'r',x,yd,'g',x,u,'b')
figure(55)
plot(x,sol,'g',x,u,'b')


%%
N=10;
chi=linspace(0,N,N*100)
Y=tanh(4*chi)./tanh(chi)+1;
plot(chi,Y,'b')
xlabel('\chi')
ylabel('\alpha')
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 $-\Delta$).

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

in domain Ω defined in figure 5.6.

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 5.7

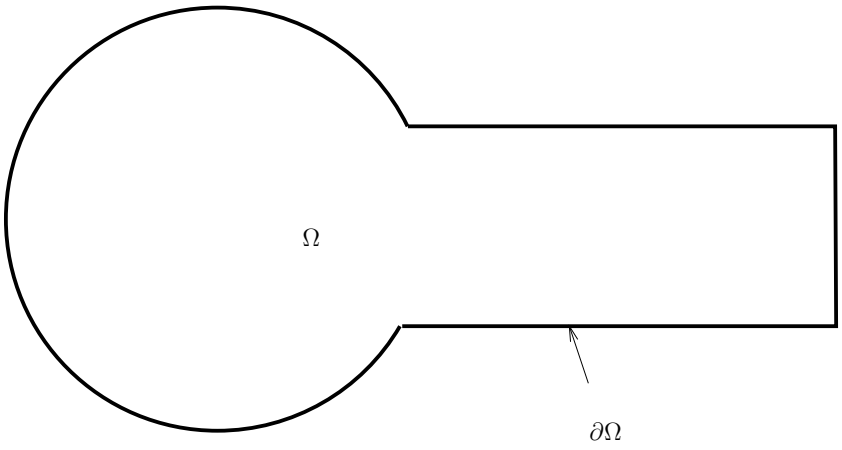


FIGURE 5.6 – The initial Schwarz domain

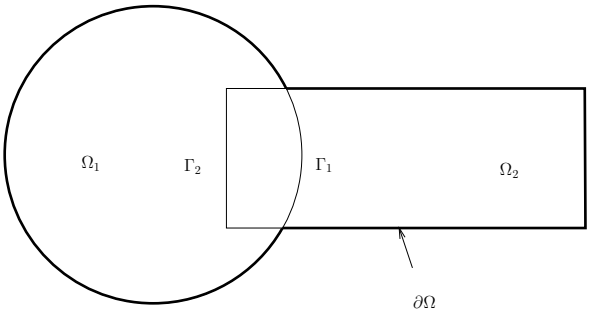


FIGURE 5.7 – Schwarz Decomposition

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

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

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 :

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

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

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

$$\left\{\begin{array}{l} u-\dfrac{d^2u}{dx^2}=f, \quad x\in\mathbb{R},\\ u\to 0 \text{ as } x\to\pm\infty\end{array}\right. \tag{5.10}$$

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 5.8) :)

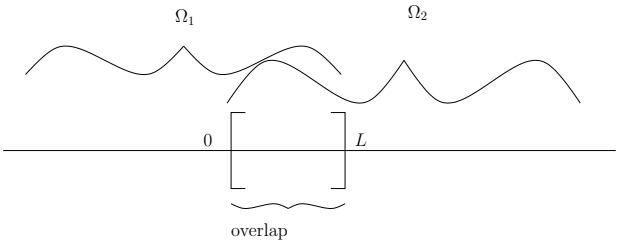


FIGURE 5.8 – 1D decomposition in 2 sub-domains with overlap

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

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

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

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

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

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}.

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 :

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

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 5.9.

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

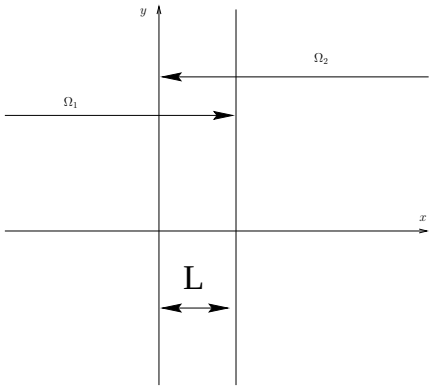


FIGURE 5.9 – 2D decomposition in 2 sub-domains with overlap

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

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

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

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

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*

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

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 5.11. 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

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

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

$$\begin{cases} \dfrac{\partial w^{n+1}}{\partial x} - w^{n+1} = \dfrac{\partial v^{n}}{\partial x} - v^{n}, \quad x = L \\ \\ \dfrac{\partial v^{n+1}}{\partial x} + v^{n+1} = \dfrac{\partial w^{n}}{\partial x} + w^{n}, \quad x = 0 \end{cases} \tag{5.30}$$

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 :

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

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

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

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

$$\begin{cases} -\Delta V^{n+1} + V^{n+1} = 0 \\ \dfrac{\partial V^{n+1}}{\partial x} + V^{n+1} = \dfrac{\partial W^n}{\partial x} + W^n \text{ on } \Gamma \end{cases} \quad \begin{cases} -\Delta W^{n+1} + W^{n+1} = 0 \\ \dfrac{\partial W^{n+1}}{\partial x} - W^{n+1} = \dfrac{\partial V^n}{\partial x} - V^n \text{ on } \Gamma \end{cases} \tag{5.34}$$

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

$$\left\{\begin{array}{rcl} \frac{\partial v}{\partial x} + v & = & \frac{\partial w}{\partial x} + w \text{ on } \Gamma \\ \frac{\partial v}{\partial x} - v & = & \frac{\partial w}{\partial x} - w \text{ on } \Gamma \end{array}\right. \tag{5.35}$$

Thus

$$\left\{\begin{array}{rcl} \frac{\partial v}{\partial x} & = & \frac{\partial w}{\partial x} \text{ on } \Gamma \\ v & = & w \text{ on } \Gamma \end{array}\right. \tag{5.36}$$

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 5.10.

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 5.11, where u_1 is the restriction of u to Ω_1 and u_2 is the restriction of u to Ω_2

Conditions

$$\begin{array}{l} u_1 = u_2 \text{ on } \Gamma \\ \frac{\partial u_1}{\partial n} = \frac{\partial u_2}{\partial n} \text{ on } \Gamma \end{array} \tag{5.37}$$

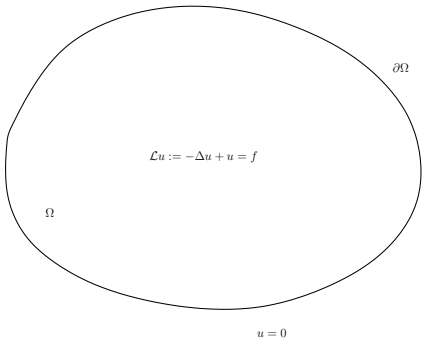


FIGURE 5.10 – Original problem

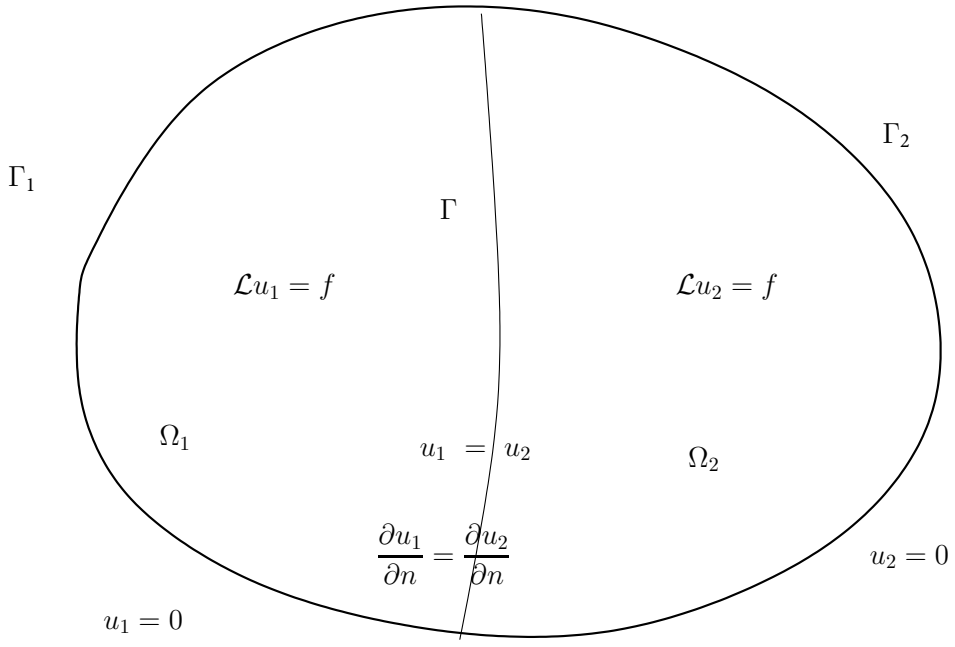


FIGURE 5.11 – Two sub-domain decomposition with no overlap.

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.

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

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

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

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

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

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,

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

thus

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

We now have the following equalities

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

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 $\Gamma = \{x = 0\}$.

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

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

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

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

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

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

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

$$\left\{ \begin{array}{l} \lambda_1^{n+1} = e^{-L} \lambda_2^n + \alpha \\ \lambda_2^{n+1} = e^{-L} \lambda_1^n + \beta \end{array} \right.$$

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.*

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 $\{\varphi_j\}_{1 \le j \le N_S^2}$. Nodes are numbered as inner nodes $\{S_j\}_{1 \le j \le N_{int}^2}$, and boundary nodes as $\{S_j\}_{N_{int}^2 + 1 \le j \le N_S^2}$. 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

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

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 :

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

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

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

Thus

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

Similarly we have

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

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

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

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

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

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] A. Brandt, S.F. McCormick, and J.W. Ruge. *Algebraic Multigrid (AMG) for Sparse Matrix Equations.* Sparsity and Its Applications, D.J. Evans, ed., Cambridge Univ. Press, 1984.

[2] Suzanne.C Brenner and Ridgway Scott. *The mathematical theory of finite element methods.* Springer-Verlag, 1994.

[3] William L. Briggs and Van Emden Henson. A multigrid tutorial. www.llnl.gov/CASC/people/henson/mgtut/ps/mgtut.pdf.

[4] Robert D. Falgout. *An Introduction to Algebraic Multigrid.* Computing in Science and Engineering, 2006.

[5] Charles F Loan Gene H Golub. *Matrix computations.* Johns Hopkins University Press, 1996.

[6] Hans Petter Langtangen and Aslak Tveitog. What is multigrid ? folk.uio.no/infima/doc/mg-underscore-nmfpd.pdf.

[7] P. Wesseling. *An introduction to multigrid methods.* Wiley -Interscience, 1992.