Centre Universitaire de Mila
Département Mathématiques et Informatique

Chapitre 1: Résolution des systèmes linéaires

Dr. Aissa Boulmerka
aissa.boulmerka@gmail.com

1-MÉTHODES DIRECTES

Substitution arrière ou avant

Considérons un système inversible 3×3 triangulaire inferieur :

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

La matrice étant inversible, ses termes diagonaux $\boldsymbol{l_{ii}, i = 1, 2, 3}$ sont non nuls. On peut donc déterminer successivement les valeurs inconnues $\boldsymbol{x_i}$ pour $\boldsymbol{i = 1, 2, 3}$:

$$\begin{aligned} x_1 &= b_1/l_{11}, \\ x_2 &= (b_2 - l_{21}x_1)/l_{22}, \\ x_3 &= (b_3 - l_{31}x_1 - l_{32}x_2)/l_{33}. \end{aligned}$$

Substitution avant (directe)

Cet algorithme peut être étendu aux systèmes $\boldsymbol{n} \times \boldsymbol{n}$. On l'appelle **substitution avant (directe)**. Dans le cas d'un système $\boldsymbol{Lx} = \boldsymbol{b}$, où $\boldsymbol{L}$ est une matrice inversible triangulaire inférieure d'ordre $\boldsymbol{n}$ ($\boldsymbol{n} \geq \boldsymbol{2}$), la méthode s'écrit

$$x_1 = \frac{b_1}{l_{11}},$$
$$x_i = \frac{1}{l_{ii}} \left(b_i - \sum_{j=1}^{i-1} l_{ij} x_j \right), \quad i = 2, \ldots, n.$$

On appelle ces relations formules de "descente".
L'algorithme effectue $\boldsymbol{n}(\boldsymbol{n}+\boldsymbol{1})/\boldsymbol{2}$ multiplications et divisions et $\boldsymbol{n}(\boldsymbol{n}-\boldsymbol{1})/\boldsymbol{2}$ additions et soustractions.

Substitution directe: version orientée ligne

```
function [x]=forwardrow(L,b)
% FORWARDROW substitution directe: version orientée ligne.
% X=FORWARDROW(L,B) résout le système triangulaire inférieur L*X=B
% avec la méthode de substitution directe dans sa version
% orientée ligne
[n,m]=size(L);
if n ~= m, error('Seulement des systèmes carrés'); end
if min(abs(diag(L))) == 0, error('Le système est singulier'); end
x(1,1) = b(1)/L(1,1);
for i = 2:n
x (i,1) = (b(i)-L(i,1:i-1)*x(1:i-1,1))/L(i,i);
end
return
```

Exemple: Substitution directe (ligne)

```
clear, clc;
L = [5, 0, 0; 1, 2, 0; -1, 3, 2]
b = [15; 7; 5]
[x] = forwardrow(L,b)
```

```
L =
    5    0    0
    1    2    0
   -1    3    2
b =
   15
    7
    5
x =
    3
    2
    1
```

Substitution directe: version orientée colonne

```
function [b]=forwardcol(L,b)
% FORWARDCOL substitution directe: version orientée colonne.
% X=FORWARDCOL(L,B) résout le système triangulaire inférieur L*X=B
% avec la méthode de substitution directe dans sa version
% orientée colonne
[n,m]=size(L);
if n ~= m, error('Seulement des systèmes carrés'); end
if min(abs(diag(L))) == 0, error('Le système est singulier'); end
for j=1:n-1
b(j)= b(j)/L(j,j); b(j+1:n)=b(j+1:n)-b(j)*L(j+1:n,j);
end
  b(n) = b(n)/L(n,n);
return
```

Exemple: Substitution directe (colonne)

```
clear, clc;
L = [5, 0, 0; 1, 2, 0; -1, 3, 2]
b = [15; 7; 5]
[x] = forwardcol(L,b)
```

```
L =

    5    0    0
    1    2    0
   -1    3    2
b =
   15
    7
    5
x =
    3
    2
    1
```

Substitution arrière

On traite de manière analogue un système linéaire $\boldsymbol{Ux = b}$, où $\boldsymbol{U}$ est une matrice inversible triangulaire supérieure d'ordre $\boldsymbol{n}$ $(\boldsymbol{n \geq 2})$. Dans ce cas l'algorithme s'appelle **substitution arrière (rétrograde)** et s'écrit dans le cas général

$$x_n = \frac{b_n}{u_{nn}},$$
$$x_i = \frac{1}{u_{ii}} \left(b_i - \sum_{j=i+1}^{n} u_{ij} x_j \right), \quad i = n-1, \ldots, 1$$

Substitution arrière (orientée colonne)

```
function [b]=backwardcol(U,b)
% BACKWARDCOL substitution rétrograde: version orientée colonne.
% X=BACKWARDCOL(U,B) résout le système triangulaire supérieur
% U*X=B avec la méthode de substitution rétrograde dans sa
% version orientée colonne.
[n,m]=size(U);
if n ~= m, error('Seulement des systèmes carrés'); end
if min(abs(diag(U))) == 0, error('Le système est singulier'); end
for j = n:-1:2
  b(j)=b(j)/U(j,j); b(1:j-1)=b(1:j-1)-b(j)*U(1:j-1,j);
end
b(1) = b(1)/U(1,1);
return
```

Exemple: Substitution arrière (colonne)

```
clear, clc;
U = [1, -4, 3; 0, 5, -3; 0, 0, -2]
b = [-2; 7; -2]
[x] = backwardcol(U,b)
```

```
U =
     1    -4     3
     0     5    -3
     0     0    -2
b =
    -2
     7
    -2
x =
     3
     2
     1
```

Factorisation LU

La matrice $\boldsymbol{A}$ est factorisée comme

$$\boldsymbol{A} = \boldsymbol{LU}$$

où $\boldsymbol{L}$ est triangulaire inferieure (lower) et U triangulaire supérieure (upper). (On parle aussi de factorisation $\boldsymbol{LR}$, avec $\boldsymbol{L}$ triangulaire à gauche (left) et $\boldsymbol{R}$ triangulaire à droite (right).)

Cette factorisation n'est pas unique, L'équation se traduit alors par

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ l_{n,1} & \cdots & l_{n,n-1} & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ 0 & u_{2,2} & & u_{2,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{n,n} \end{bmatrix}.$$

Factorisation LU

Après factorisation $\boldsymbol{LU}$, le système à résoudre est

$$\boldsymbol{LUx} = \boldsymbol{b}$$

La solution est évaluée en deux étapes. On évalue d'abord $\boldsymbol{y}$ solution de

$$\boldsymbol{Ly} = \boldsymbol{b}$$

Puisque $\boldsymbol{L}$ est triangulaire inferieure, ceci est mené à bien par substitution avant, chaque équation fournissant la solution pour une nouvelle inconnue. Comme les éléments diagonaux de $\boldsymbol{L}$ sont tous égaux à un, ceci est particulièrement simple.

On évalue ensuite $\boldsymbol{x}$ solution de

$$\boldsymbol{Ux} = \boldsymbol{y}$$

Comme $\boldsymbol{U}$ est triangulaire supérieure, ceci est mené à bien par **substitution arrière**, chaque équation fournissant ici aussi la solution pour une nouvelle inconnue.

Factorisation LU de la matrice A

```
function [A]=lukji(A)
% LUKJI Factorisation LU de la matrice A dans la version kji
% Y=LUKJI(A): U est stocké dans la partie triangulaire supérieure
% de Y et L est stocké dans la partie triangulaire inferieure
% stricte de Y.
[n,m]=size(A);
if n ~= m, error('Seulement les systèmes carrés'); end
for k=1:n-1
  if A(k,k)==0; error('Pivot nul'); end
  A(k+1:n,k)=A(k+1:n,k)/A(k,k);
  for j=k+1:n
    i=[k+1:n]; A(i,j)=A(i,j)-A(i,k)*A(k,j);
  end
end
return
```

Exemple: Factorisation LU

```
clear, clc;
A = [5, 0, 0; 1, 2, 0; -1, 3, 2]
[Y] = lukji(A)

L = tril(Y, -1) + diag(ones(size(A,1),1))
U = triu(Y)

L*U
```

```
A=5    0    0
   1    2    0
  -1    3    2
Y=5.0000         0         0
   0.2000    2.0000         0
  -0.2000    1.5000    2.0000
L=1.0000         0         0
   0.2000    1.0000         0
  -0.2000    1.5000    1.0000
U=5    0    0
   0    2    0
   0    0    2
L*U =
   5    0    0
   1    2    0
  -1    3    2
```

Factorisation de Cholesky

Soit la matrice qui admet la factorisation $\boldsymbol{LDL^T}$ suivante

$$\mathrm{H}_3 = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{3} & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{12} & 0 \\ 0 & 0 & \frac{1}{180} \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

Dans le cas où $\boldsymbol{A}$ est aussi **définie positive**, les termes diagonaux de $\boldsymbol{D}$ dans la factorisation $\boldsymbol{LDL^T}$ sont strictement positifs. On a de plus le résultat suivant :

Factorisation de Cholesky

Théorème *Soit* $\boldsymbol{A} \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{n}}$ *une matrice symétrique définie positive. Alors, il existe une unique matrice triangulaire supérieure* $\boldsymbol{H}$ *dont les termes diagonaux sont strictement positifs telle que*

$$\boldsymbol{A} = \boldsymbol{H}^{\boldsymbol{T}}\boldsymbol{H}.$$

Cette factorisation est appelée **factorisation de Cholesky** et les coefficients $\boldsymbol{h_{ij}}$ de $\boldsymbol{H^T}$ peuvent être calculés comme suit : $\boldsymbol{h_{11}} = \sqrt{\boldsymbol{a_{11}}}$ et, pour $\boldsymbol{i} = \boldsymbol{2},\ldots,\boldsymbol{n}$,

$$h_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} h_{ik}h_{jk} \right) / h_{jj}, \quad j = 1, \ldots, i-1,$$

$$h_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} h_{ik}^2 \right)^{1/2}.$$

Factorisation de Cholesky

```
function [A]=chol2(A)
% CHOL2 Factorisation de Cholesky d'une matrice A symétrique définie positive.
% R=CHOL2(A) renvoie une matrice triangulaire supérieure R telle que R'*R=A.
[n,m]=size(A);
if n ~= m, error('Seulement les systèmes carrés'); end
for k=1:n-1
if A(k,k) <= 0, error('Pivot nul ou n´négatif'); end
A(k,k)=sqrt(A(k,k)); A(k+1:n,k)=A(k+1:n,k)/A(k,k);
for j=k+1:n, A(j:n,j)=A(j:n,j)-A(j:n,k)*A(j,k); end
end
A(n,n)=sqrt(A(n,n));
A = tril(A); A=A';
return
```

Note: La matrice $\boldsymbol{L}$ étant triangulaire inferieure avec des $\mathbf{1}$ sur la diagonale et $\boldsymbol{U}$ étant triangulaire supérieure.

Exemple: Factorisation de Cholesky

```
clear, clc;

A = [1, 1/2, 1/3; 1/2, 1/3, 1/4; 1/3, 1/4, 1/5]
[R] = chol2(A)

R'*R
```

```
A =
   1.0000    0.5000    0.3333
   0.5000    0.3333    0.2500
   0.3333    0.2500    0.2000

R =
   1.0000    0.5000    0.3333
        0    0.2887    0.2887
        0         0    0.0745

R'*R  =
   1.0000    0.5000    0.3333
   0.5000    0.3333    0.2500
   0.3333    0.2500    0.2000
```

2-MÉTHODES ITÉRATIVES

Méthodes itératives linéaires

Une technique générale pour définir une méthode itérative linéaire consistante est basée sur la décomposition, ou *splitting*, de la matrice A sous la forme A=P−N, où P est une matrice inversible. Pour des raisons qui s'éclairciront dans les prochaines sections, P est appelée *matrice de préconditionnement* ou *préconditionneur*.

On se donne $\mathbf{x}^{(0)}$, et on calcule $\mathbf{x}^{(k)}$ pour $k \geq 1$, en résolvant le système

$$\mathrm{P}\mathbf{x}^{(k+1)} = \mathrm{N}\mathbf{x}^{(k)} + \mathbf{b}, \quad k \geq 0. \tag{4.6}$$

La matrice d'itération de la méthode (4.6) est $\mathrm{B} = \mathrm{P}^{-1}\mathrm{N}$, et $\mathbf{f} = \mathrm{P}^{-1}\mathbf{b}$. On peut aussi écrire (4.6) sous la forme

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathrm{P}^{-1}\mathbf{r}^{(k)}, \tag{4.7}$$

Méthodes itératives linéaires

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathrm{P}^{-1}\mathbf{r}^{(k)}, \tag{4.7}$$

où

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathrm{A}\mathbf{x}^{(k)} \tag{4.8}$$

désigne le *résidu* à l'itération k. La relation (4.7) montre qu'on doit résoudre un système linéaire de matrice P à chaque itération. En plus d'être inversible, P doit donc être "facile à inverser" afin de minimiser le coût du calcul. Remarquer que si P est égale à A et N=0, la méthode (4.7) converge en une itération, mais avec le même coût qu'une méthode directe.

La méthode de Jacobi

Dans cette section, nous considérons quelques méthodes itératives linéaires classiques.

Si les coefficients diagonaux de A sont non nuls, on peut isoler l'inconnue x_i dans la i-ème équation, et obtenir ainsi le système linéaire équivalent

$$x_i = \frac{1}{a_{ii}} \left[b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j \right], \qquad i = 1, \ldots, n. \tag{4.9}$$

Dans la *méthode de Jacobi*, pour une donnée initiale arbitraire $\mathbf{x}^0$, on calcule $\mathbf{x}^{(k+1)}$ selon la formule

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k)} \right], \quad i = 1, \ldots, n. \tag{4.10}$$

La méthode de Jacobi

Cela revient à effectuer la décomposition suivante de la matrice A :

$$\mathrm{P} = \mathrm{D}, \quad \mathrm{N} = \mathrm{D} - \mathrm{A} = \mathrm{E} + \mathrm{F},$$

où D est la matrice diagonale composée des coefficients diagonaux de A, E est la matrice triangulaire inférieure de coefficients $e_{ij} = -a_{ij}$ si $i > j$, $e_{ij} = 0$ si $i \leq j$, et F est la matrice triangulaire supérieure de coefficients $f_{ij} = -a_{ij}$ si $j > i$, $f_{ij} = 0$ si $j \leq i$. Ainsi, $\mathrm{A} = \mathrm{D} - (\mathrm{E} + \mathrm{F})$.

La matrice d'itération de la méthode de Jacobi est donc donnée par

$$\mathrm{B}_J = \mathrm{D}^{-1}(\mathrm{E} + \mathrm{F}) = \mathrm{I} - \mathrm{D}^{-1}\mathrm{A}. \tag{4.11}$$

La méthode de sur-relaxation

Une généralisation de la méthode de Jacobi est la *méthode de sur-relaxation* (ou JOR, pour *Jacobi over relaxation*), dans laquelle on se donne un paramètre de relaxation ω et on remplace (4.10) par

$$x_i^{(k+1)} = \frac{\omega}{a_{ii}} \left[b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k)} \right] + (1-\omega) x_i^{(k)}, \qquad i = 1, \ldots, n.$$

La matrice d'itération correspondante est

$$\mathrm{B}_{J_\omega} = \omega \mathrm{B}_J + (1-\omega)\mathrm{I}. \tag{4.12}$$

Sous la forme (4.7), la méthode JOR correspond à

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega \mathrm{D}^{-1} \mathbf{r}^{(k)}.$$

Cette méthode est consistante pour tout $\omega \neq 0$. Pour $\omega = 1$, elle coïncide avec la méthode de Jacobi.

Méthode JOR

```
function [x,iter]=jor(A,b,x0,nmax,tol,omega)
%Méthode JOR tente de résoudre le système A*X=B avec la méthode JOR.
% TOL est la tolérance de la méthode.
% NMAX est le nombre maximum d'itérations.
% X0 est la donnée initiale.
% OMEGA est le paramètre de relaxation.
% ITER est l'itération à laquelle la solution X a été calculée.
[n,m]=size(A);
if n ~= m, error('Seulement les systèmes carrés'); end
iter=0;
r = b-A*x0; r0=norm(r); err=norm(r); x=x0;
```

```
while err > tol && iter < nmax
iter = iter + 1;
for i=1:n
s = 0;
for j = 1:i-1, s=s+A(i,j)*x(j); end
for j = i+1:n, s=s+A(i,j)*x(j); end
xnew(i,1)=omega*(b(i)-s)/A(i,i)+(1-omega)
*x(i);
end
x=xnew; r=b-A*x; err=norm(r)/r0;
end
return
```

Exemple : Méthode JOR

```
clear, clc;


e = ones(5,1);
A = full(spdiags([-e 2*e -e],-1:1,5,5))
b = [1:5]'


x0 = b.*0+1;
nmax = 1000;
tol = 1e-3;
omega = 1;
[x,iter]=jor(A, b, x0, nmax, tol, omega)
```

```
A=2   -1    0    0    0
   -1    2   -1    0    0
    0   -1    2   -1    0
    0    0   -1    2   -1
    0    0    0   -1    2
b=1
    2
    3
    4
    5
x=5.8270
  10.6556
  13.4873
  13.3223
   9.1603
iter =  48
```

La méthode de Gauss-Seidel

La *méthode de Gauss-Seidel* diffère de la méthode de Jacobi par le fait qu'à la $k+1$-ième étape les valeurs $x_i^{(k+1)}$ déjà calculées sont utilisées pour mettre à jour la solution. Ainsi, au lieu de (4.10), on a

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right], \quad i = 1, \ldots, n. \qquad (4.13)$$

Cette méthode revient à effectuer la décomposition suivante de la matrice A :

$$\mathrm{P} = \mathrm{D} - \mathrm{E}, \quad \mathrm{N} = \mathrm{F},$$

et la matrice d'itération associée est

$$\mathrm{B}_{GS} = (\mathrm{D} - \mathrm{E})^{-1}\mathrm{F}. \qquad (4.14)$$

Méthode SOR

```
function [x,iter]=sor(A,b,x0,nmax,tol,omega)
%SOR Méthode SOR
% [X,ITER]=SOR(A,B,X0,NMAX,TOL,
OMEGA) tente de résoudre le système
% A*X=B avec la méthode SOR.
% TOL est la tolérance de la méthode.
% NMAX est le nombre maximum
d'itérations.
% X0 est la donnée initiale.
% OMEGA est le paramètre de relaxation.
% ITER est l'itération `a laquelle la solution X
a été calculée.
[n,m]=size(A);
if n ~= m, error('Seulement les systèmes
carrés'); end
```

```
iter=0; r=b-A*x0; r0=norm(r); err=norm(r);
xold=x0;
while err > tol & iter < nmax
iter = iter + 1;
for i=1:n
s=0;
for j = 1:i-1, s=s+A(i,j)*x(j); end
for j = i+1:n, s=s+A(i,j)*xold(j); end
x(i,1)=omega*(b(i)-s)/A(i,i)+(1-
omega)*xold(i);
end
xold=x; r=b-A*x; err=norm(r)/r0;
end
return
```

Exemple: Méthode SOR

```
clear, clc;


e = ones(5,1);
A = full(spdiags([-e 2*e -e],-1:1,5,5))
b = [1:5]'


x0 = b.*0+1;
nmax = 1000;
tol = 1e-3;
omega = 1;
 [x,iter]=sor(A, b, x0, nmax, tol, omega)
```

```
A=2   -1    0    0    0
   -1    2   -1    0    0
    0   -1    2   -1    0
    0    0   -1    2   -1
    0    0    0   -1    2
b=1
    2
    3
    4
    5
x= 5.8259
  10.6555
  13.4888
  13.3249
   9.1625
iter =  25
```