Chapitre 3

Analyse numérique matricielle

Ibrahim ALAME

ESTP

01/03/2023

Système triangulaire

Un système triangulaire supérieur est un système dont la matrice est triangulaire supérieure.

$$\begin{cases} a_{11}x_1 + a_{11}x_2 + \cdots + a_{1n}x_n = b_1 \\ \qquad\qquad a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ \qquad\qquad\qquad\qquad\qquad\quad a_{nn}x_n = b_n \end{cases}$$

Si a_{ii} non nuls, le système admet une solution unique.

```python
import random as r
import numpy as np

def triangSup(n):
    M=np.zeros((n,n),dtype=int)
    for i in range(n):
        for j in range(i,n):
            M[i,j]=r.randint(1,10)
    return M
```

Système triangulaire

La résolution se fait par substitution arrière :

$$\begin{cases} x_n = \dfrac{b_n}{a_{nn}} \\ x_k = \dfrac{1}{a_{kk}} \left(b_k - \displaystyle\sum_{j=k+1}^{n} a_{kj} x_j \right) \end{cases} \qquad \forall k \in [\![n-1, \ldots, 1]\!]$$

Système triangulaire

Algorithm 1: Système triangulaire supérieur

for k **from** $n-1$ **to** 1 **do**
    $x_k = b_k$
    for j **from** $k+1$ **to** n **do**
        $x_k = x_k - a_{kj}x_j$
    $x_k = \dfrac{x_k}{a_{kk}}$

Le calcul de x_k nécessite $n-k$ flops [1] et une division. Le coût de l'algorithme est donc $1+2+\cdots+n-1 = \frac{n(n-1)}{2}$ flops et n divisions.

1. *FLOPS* est l'acronyme anglais de *FLoating point Operations Per Second* signifiant *opérations à virgule flottante par seconde*. Il s'agit de la vitesse de calcul d'une opération élémentaire (addition, multiplication) en informatique.

```
n=5
A=triangSup(n)
x=np.arange(n)
b=np.dot(A,x)

#  Resolution du système traiangulaire supérieur Ax=b
def sysTriangSup(A,b):
    n=len(A)
    x=np.zeros(n,dtype=float)
    for k in range(n-1,-1,-1):
        x[k]=(b[k]-np.dot(A[k,k+1:],x[k+1:]))/A[k,k]
    return x

print(sysTriangSup(A,b))
```

Système triangulaire inférieur

Dans le cas d'un système triangulaire inférieur, l'algorithme est analogue et de même coût :

$$\begin{cases} x_1 = \dfrac{b_1}{a_{11}} \\ x_k = \dfrac{1}{a_{kk}} \left(b_k - \displaystyle\sum_{j=1}^{k-1} a_{kj} x_j \right) \end{cases} \qquad \forall k \in [\![2, n]\!]$$

Algorithm 2: Système triangulaire inférieur

for k **from** 1 **to** n **do**
    $x_k = b_k$
    for j **from** 1 **to** $k-1$ **do**
        $x_k = x_k - a_{kj} x_j$
    $x_k = \frac{x_k}{a_{kk}}$

```python
def triangInf(n):
    M=np.zeros((n,n),dtype=float)
    for i in range(n):
        for j in range(i+1):
            M[i,j]=r.randint(1,10)
    return M
#  Resolution du système traiangulaire inférieur Ax=b
def sysTriangInf(A,b):
    n=len(A)
    x=np.zeros(n,dtype=float)
    for k in range(n):
        x[k]=(b[k]-np.dot(A[k,:k],x[:k]))/A[k,k]
    return x


n=5
A=triangInf(n)
xp=np.arange(1,n+1)
b=np.dot(A,xp)
print(sysTriangInf(A,b))
```

Méthode de Gauss

On pose $A^{(1)} = A$, $b^{(1)} = b$ et on note $a_{ij}^{(1)} = a_{ij}$ et $b_{ij}^{(1)} = b_{ij}$ les coefficients de $A^{(1)}$ et $B^{(1)}$.
On suppose que pour un k fixé on a $A^{(k)} =$

$$\begin{bmatrix}
a_{1\,1}^{(k)} & a_{1\,2}^{(k)} & \cdots & & & \cdots & a_{1\,n}^{(k)} \\
0 & a_{2\,2}^{(k)} & & & & & a_{2\,n}^{(k)} \\
\vdots & 0 & \ddots & & & & \vdots \\
 & \vdots & & a_{k-1\,k-1}^{(k)} & & & \\
 & & & 0 & a_{k\,k}^{(k)} & & \\
 & & \cdots & \vdots & a_{k+1\,k}^{(k)} & a_{k+1\,k+1}^{(k)} & \\
 & & & & \vdots & \vdots & \ddots \\
0 & 0 & & 0 & a_{n\,k}^{(k)} & a_{n\,k+1}^{(k)} & a_{n\,n}^{(k)}
\end{bmatrix}$$

Les coefficients de la matrice $A^{(k+1)}$ sont déterminées par récurrence :

- Les coefficients dans les lignes L_i, $1 \leqslant i \leqslant k$ ne sont pas modifiés.
- Pour $i \geqslant k+1$, si $a_{kk}^{(k)} \neq 0$:

$$\begin{cases} a_{ij}^{(k+1)} = a_{ij}^{(k)} - \dfrac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)} & \forall i \in [\![k+1, n]\!] \\ b_i^{(k+1)} = b_i^{(k)} - \dfrac{a_{ik}^{(k)}}{a_{kk}^{(k)}} b_k^{(k)} \end{cases}$$

On obtient alors un système équivalent, triangulaire $A^{(n)} \cdot x = b^{(n)}$.

Algorithm 3: Méthode du pivot de Gauss

for k **from** 1 **to** $n-1$ **do**
  for i **from** $k+1$ **to** n **do**
    $c = \frac{a_{ik}}{a_{kk}}$
    $b_i = b_i - c b_k$
    $a_{ik} = 0$
    for j **from** $k+1$ **to** n **do**
      $a_{ij} = a_{ij} - c a_{kk}$

Le coût de la méthode de Gauss est de l'ordre de $\frac{2}{3}n^3$.
À présent, le système est triangulaire supérieur. Nous appliquons l'algorithme 1 pour achever la résolution.

```
def ReductionGauss(A,B):
    n=A.shape[0]
    U=np.concatenate((A,B),axis=1)
    #U.dtype=np.float64
    for j in range(n):
        for i in range(j+1,n):
            r=U[i,j]/U[j,j]
            U[i]=U[i]-r*U[j]
    return U
```

Factorisation LU

L'opération algébrique $L_i \leftarrow L_i + \lambda L_j$ se traduit matriciellement par la multiplication à gauche par la matrice $P = \mathrm{I} + \lambda E_{ij}$:

$$P = \begin{pmatrix} 1 & 0 & \cdots & & \cdots & 0 \\ 0 & \ddots & 0 & & & \vdots \\ \vdots & & & & & \\ & & \lambda & & & \\ & & & & \ddots & 0 \\ 0 & \cdots & & & 0 & 1 \end{pmatrix}$$

A l'étape k de la méthode de Gauss, l'annulation des coefficients sous la diagonale de la colonne k est équivalent à la multiplication par la matrice $P_k = \prod_{i=k+1}^{n}(I_d + \lambda_{ik}) = I_d + \sum_{i=k+1}^{n} \lambda_{ik} E_{ik}$. Cette matrice est inversible d'inverse :

$$P_k^{-1} = I_d - \sum_{i=k+1}^{n} \lambda_{ik} E_{ik}$$

Factorisation LU

La multiplication $P \cdot A$ ajoute à la ligne i λ fois la ligne j. Cette matrice est inversible et le produit à droite par $P^{-1} = \mathrm{I} - \lambda \cdot E_{ij}$ ajoute à la $j$ème colonne $-\lambda$ fois la colonne i. Les opérations linéaires simples appliquées à A afin d'obtenir une matrice triangulaire supérieure se traduisent par la relation :

$$P_{n-1} \cdots P_2 \cdot P_1 \cdot A = U$$

On a :

$$A = (P_1^{-1} P_2^{-1} \cdots P_{n-1}^{-1}) \cdot U$$

Le produit $L = P_1^{-1} \cdot P_2^{-1} \cdots P_l^{-1}$ est une matrice triangulaire inférieure dont les coefficients diagonaux sont tous égaux à 1.

Factorisation LU

$$
\begin{aligned}
L &= \prod_{k=1}^{n-1} \left(I_d - \sum_{i=k+1}^{n} \lambda_{ik} E_{ik} \right)^{-1} \\
&= I_d + \sum_{k=1}^{n-1} \sum_{i>k} \lambda_{ik} E_{ik} \\
&= \begin{pmatrix}
1 & & & & \\
\lambda_{21} & 1 & & & \\
\vdots & \lambda_{32} & \ddots & & \\
 & \vdots & \ddots & \ddots & \\
\lambda_{n1} & \lambda_{n2} & \cdots & \lambda_{n\,n-1} & 1
\end{pmatrix}
\end{aligned}
$$

Factorisation LU

En pratique, les termes non nuls de U et les termes en-dessous de la diagonale de L sont mémorisés dans la matrice A.

Algorithm 4: Factorisation LU

for k **from** 1 **to** $n-1$ **do**
  for i **from** $k+1$ **to** n **do**
    $a_{ik} = \dfrac{a_{ik}}{a_{kk}}$
    for j **from** $k+1$ **to** n **do**
      $a_{ij} = a_{ij} - a_{ik}a_{kj}$

La résolution du système linéaire consiste simplement à résoudre successivement deux systèmes linéaires triangulaires :

$$A \cdot x = b \Longleftrightarrow L \cdot U \cdot x = b \Longleftrightarrow \begin{cases} L \cdot y = b \\ U \cdot x = y \end{cases}$$

Les éléments non nuls de la matrice L, après factorisation, ne sont que les coefficients de A sous la diagonale $l_{ij} = a_{ij}$ $(i \leqslant j)$.

Algorithm 5: Résolution après factorisation LU(Première étape)

for i **from** 1 **to** n **do**
    $y_i = b_i$
    for j **from** $k+1$ **to** $i-1$ **do**
        $y_i = y_i - a_{ij} y_j$

Les éléments non nuls de la matrice U sont stockés dans la matrice A à la même position $u_{ij} = a_{ij}$ $(i \leqslant j)$.

Algorithm 6: Résolution après factorisation LU(2ème étape)

for i **from** $n-1$ **to** 1 **do**
    $x_i = y_i$
    for j **from** 1 **to** $i-1$ **do**
        $x_i = x_i - a_{ij} x_j$
    $x_i = \dfrac{x_i}{a_{ii}}$

```python
def ReductionGauss(M):
    n=M.shape[0]
    U=M.copy()
    U.dtype=np.float64
    L=np.eye(n,dtype=np.float64)
    for j in range(n):
        for i in range(j+1,n):
            r=U[i,j]/U[j,j]
            L[i,j]=r
            U[i]=U[i]-r*U[j]
    return (L,U)
```

Méthode de Cholesky

Il s'agit d'une méthode directe adaptée au cas d'une matrice $A \in \mathscr{M}_n(\mathbb{R})$ symétrique définie positive $(Ax, x) > 0; \forall x \in \mathbb{R}^n$.
Soit la décomposition $L \cdot U$ de A :

$$A = \widetilde{L} \cdot \widetilde{U}$$

Avec :

$$\widetilde{L} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ & \ddots & & \vdots \\ & & \ddots & 0 \\ & & & 1 \end{pmatrix}$$

Et :

$$\widetilde{U} = \begin{pmatrix} u_{11} & & & \\ 0 & u_{22} & & \\ \vdots & & \ddots & \\ 0 & \cdots & 0 & u_{nn} \end{pmatrix}$$

Soit D une matrice diagonale telle que :

$$\widetilde{U} = \underbrace{\begin{pmatrix} u_{11} & 0 & \cdots & 0 \\ 0 & u_{22} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & u_{nn} \end{pmatrix}}_{D} \underbrace{\begin{pmatrix} 1 & \star & \cdots & \star \\ 0 & 1 & & \vdots \\ \vdots & & \ddots & \star \\ 0 & \cdots & 0 & 1 \end{pmatrix}}_{\widetilde{L}^t}$$

On a :

$$\begin{aligned} A &= \widetilde{L} \cdot D \cdot \widetilde{L}^t \\ &= \left(\widetilde{L} \cdot \sqrt{D}\right) \cdot \left(\sqrt{D} \cdot \widetilde{L}^t\right) \\ A &= L \cdot L^t \end{aligned}$$

Example

Soit $A = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$ avec $a > 0$ et $ac - b^2 > 0$.

$$A = L \cdot U$$

$$= \begin{pmatrix} 1 & 0 \\ \frac{b}{a} & 1 \end{pmatrix} \begin{pmatrix} a & b \\ 0 & c - \frac{b^2}{a} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ \frac{b}{a} & 1 \end{pmatrix} \begin{pmatrix} a & 0 \\ 0 & c - \frac{b^2}{a} \end{pmatrix} \begin{pmatrix} 1 & \frac{b}{a} \\ 0 & 1 \end{pmatrix}$$

Or $A = L \cdot L^t$ avec $L = \begin{pmatrix} 1 & 0 \\ \frac{b}{a} & 1 \end{pmatrix} \begin{pmatrix} \sqrt{a} & 0 \\ 0 & \frac{\sqrt{ac - b^2}}{\sqrt{a}} \end{pmatrix}$.

Donc :

$$L = \begin{pmatrix} \sqrt{a} & 0 \\ \frac{b}{\sqrt{a}} & \frac{\sqrt{ac - b^2}}{\sqrt{a}} \end{pmatrix}$$

Theorem

Soit $A \in \mathscr{M}_n(\mathbb{R})$ symétrique définie positive. Alors il existe une unique matrice $L \in \mathscr{M}_n(\mathbb{R})$ triangulaire inférieure dont les termes diagonaux sont tous positifs telle que $A = L \cdot L^t$.

Par récurrence sur n.

- Pour $n = 1$, $A = (a_{11})$ avec $a_{11} > 0$. Donc $L = (l_{11})$ avec $l_{11} = \sqrt{a_{11}}$.
- Soit $A \in \mathscr{M}_{n+1}(\mathbb{R})$. On écrit A sous la forme :

$$A = \left(\begin{array}{c:c} B & a \\ \hdashline a^t & \alpha \end{array}\right)$$

B est symétrique et définie positive car $0 < \left(A \cdot \begin{pmatrix} y \\ \hdashline 0 \end{pmatrix}, \begin{pmatrix} y \\ \hdashline 0 \end{pmatrix}\right) =< B \cdot y, y >$. Par hypothèse de récurrence, il existe une matrice triangulaire inférieure M dont les coefficients diagonaux sont positifs telle que $B = M \cdot M^t$.

On pose $L = \left(\begin{array}{c:c} M & 0 \\ \hdashline b^t & \lambda \end{array}\right)$ où $b \in \mathbb{R}^n$ et $\delta > 0$.

On a :

$$L \cdot L^t = \left(\begin{array}{c:c} M \cdot M^t & M \cdot b \\ \hdashline b^t \cdot M^t & b^t \cdot b + \lambda^2 \end{array}\right)$$

Par identification avec la matrice A, on a $M \cdot b = a$ et $b^t \cdot b + \lambda^2 = \alpha$.
D'où $b = M^{-1} \cdot a$. et $a^t \cdot M^t \cdot M^{-1} \cdot a + \lambda^2 = \alpha$. C'est-à-dire
$a^t \cdot B^{-1} \cdot a + \lambda^2 = \alpha$. D'où $\lambda^2 = \alpha - a^t \cdot B^{-1} \cdot a$.
Le signe de $\alpha - a^t \cdot B^{-1} \cdot a$ est déterminé en choisissant $z = \left(\begin{array}{c} B^{-1} \cdot a \\ \hdashline -1 \end{array}\right)$
dans l'inégalité $A \cdot z \cdot z > 0$.
On obtient bien $\alpha - a^t \cdot B^{-1} \cdot a > 0$. D'où le choix de
$\lambda = \sqrt{\alpha - a^t \cdot B^{-1} \cdot a}$.
Finalement $L = \left(\begin{array}{c:c} M & 0 \\ \hdashline \left(M^{-1} \cdot a\right)^t & \lambda \end{array}\right)$ existe et permet de conclure
la récurrence.

Calcul de L

Soit $L = (\ell_{ij})_{ij}$ et $A = (a_{ij})_{ij}$.

$$A = L \cdot L^t \iff \boxed{a_{ij} = \sum_{k=1}^{\inf(i,j)} \ell_{ik}\ell_{jk} \quad \forall (i,j) \in [\![1,n]\!]^2}$$

Pour $j = 1$:

$$a_{i1} = \ell_{i1}\ell_{11}$$

$$\iff \begin{cases} a_{11} = \ell_{11}^2 \\ a_{i1} = \ell_{i1}\ell_{11} \quad \forall i \in [\![2,n]\!] \end{cases}$$

$$\iff \boxed{\begin{cases} \ell_{11} = \sqrt{a_{11}} \\ \ell_{i1} = \dfrac{a_{i1}}{\ell_{11}} \quad \forall i \in [\![2,n]\!] \end{cases}}$$

On suppose avoir calculé les $j-1$ premières colonnes de L. On a alors :

$$a_{jj} = \sum_{k=1}^{j} \ell_{jk}\ell_{jk}$$
$$= \sum_{k=1}^{j-1} \ell_{jk}^2 + \ell_{jj}^2$$

$$\boxed{\ell_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} \ell_{jk}^2}}$$

Et pour $i \in [\![j+1, n]\!]$:

$$a_{jj} = \sum_{k=1}^{j} \ell_{ik}\ell_{jk} = \sum_{k=1}^{j-1} \ell_{ik}\ell_{jk} + \ell_{ij}\ell_{jj}$$

$$\boxed{\ell_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} \ell_{ik}\ell_{jk}}{\ell_{jj}}}$$

D'où l'algorithme :

Algorithm 7: Factorisation de Cholesky

for j **from** 1 **to** n **do**
  $s = 0$ **for** k **from** 1 **to** $j-1$ **do**
    $s = s + a_{jk}^2$
    $a_{jj} = \sqrt{a_{jj} - s}$
    for i **from** $j+1$ **to** n **do**
      $s = 0$
      for k **from** 1 **to** $j-1$ **do**
        $s = s + a_{ik}a_{jk}$
      $a_{ij} = \dfrac{a_{ij} - s}{a_{jj}}$

Choleski conserve le profile de la matrice.

Coût de calcul de L :

$$\begin{aligned}\sum_{j=1}^{n}(2j-1)(n-j+1) &= 2n\sum_{j=1}^{n}j - 2\sum_{j=1}^{n}j^2 + 2\sum_{j=1}^{n}j + \sum_{j=1}^{n}(j-1) \\ &= 2n\frac{n(n+1)}{2} - 2\frac{n(n+1)(2n+1)}{6} \\ &= \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} \\ &\simeq \frac{n^3}{3}\end{aligned}$$

À titre de comparaison pour $n = 10$:

- Gauss : 700 opérations
- Cholesky : 350 opérations
- Cramer : 40 000 000 opérations

```python
def cholesky(A):
    n=A.shape[0]
    L=np.zeros((n,n),dtype=float)
    for j in range(n):
        s=np.dot(L[j,:j],L[j,:j])
        L[j,j] = np.sqrt(A[j,j]- s)
        for i in range(j+1,n):
            s=np.dot(L[i,:j],L[i,:j])
            L[i,j]=(A[i,j]-s)/L[j,j]
    return L
```

Normes matricielles

Norme

On appelle norme matricielle sur $\mathscr{M}_n(\mathbb{R})$ induite par la norme vectorielle de $\mathbb{R}^n$ l'application :

$$A \longmapsto \|A\| = \sup_{\|x\|=1} \|Ax\|$$

proposition

- $\|Ax\| \leqslant \|A\| \cdot \|x\|$
- $\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$
- $\|AB\| \leqslant \|A\| \cdot \|B\|$

Examples :

$$\|A\|_\infty = \max_i \sum_{i,j} \|a_{ij}\| \qquad \|A\|_1 = \max_j \sum_{i,j} |a_{ij}| \qquad \|A\|_2 = \sqrt{\rho(A \cdot A^t)}$$

Si A est symétrique $\|A\|_2 = \rho(A)$.

Normes matricielles

proposition

Soient $A \in \mathscr{M}_n(\mathbb{R})$ et $\varepsilon > 0$. Il existe une norme induite sur $\mathscr{M}_n(\mathbb{R})$ notée $\| \bullet \|_{A,\varepsilon}$ qui vérifie $\|A\|_{A,\varepsilon} \leqslant \rho(A) + \varepsilon$.

corollaire

On munit $\mathscr{M}_n(\mathbb{R})$ d'une norme $\| \cdot \|$. Soit $A \in \mathscr{M}_n(R)$. Alors $\rho(A) < 1$ si et seulement si $\lim\limits_{k \to \infty} A^k = 0$.

Si $\rho(A) < 1$ alors $\exists \varepsilon > 0$ tel que $\rho(A) < 1 - 2\varepsilon$.

- $\|A\|_{A,\varepsilon} \leqslant \rho + \varepsilon < 1 - \varepsilon < 1$
- $\|A^k\|_{A,\varepsilon} \leqslant \|A\|^k_{A,\varepsilon} < (1-\varepsilon)^k \to 0$

Donc $\lim\limits_{k \to \infty} \|A^k\| = 0$ en dimension finie.

Réciproquement, soit λ une valeur propre de A. On a $A^k x = \lambda^k x$. Donc si $\lim\limits_{k \to \infty} A^k = 0$, alors $\lim\limits_{k \to \infty} \lambda^k x = 0$ et donc $|\lambda| < 1$.

proposition

Soit $A \in \mathscr{M}_n(\mathbb{R})$ muni d'une norme matricielle $\|\cdot\|$. Alors :

$$\rho(A) \leqslant \|A\|$$

corollaire

Pour montrer que la suite $x^{(k+1)} = A \cdot x^{(k)}$ converge, il suffit de trouver une norme matricielle telle que $\|A\| < 1$.

Theorem

Si $\|A\| < 1$, *alors* $\mathrm{I} + A$ *inversible et on a :*

$$\left\|(I + A)^{-1}\right\| \leqslant \frac{1}{1 - \|A\|}$$

Si $\mathrm{I} + A$ *est singulière, alors* $\|A\| \geqslant 1$ *pour toute norme matricielle.*

Conditionnement

definition

Soit $A \in \mathscr{M}_n(\mathbb{R})$ une matrice inversible. On appelle conditionnement de A par rapport à la norme $\| \bullet \|$ le nombre :

$$\text{cond}(A) = \|A\| \cdot \|A^{-1}\|$$

proposition

- $\text{cond}(A) \geqslant 1$ car $\|\text{I}\| \leqslant \|A\| \cdot \|A^{-1}\|$
- $\text{cond}(\lambda A) = \text{cond}(A)$
- $\text{cond}(A \cdot B) \leqslant \text{cond}(A) \cdot \text{cond}(B)$

Exemple : Si A symétrique définie positive, alors $\text{cond}_2(A) = \frac{\lambda_n}{\lambda_1}$. Sinon, $\text{cond}_2(A) = \sqrt{\frac{\sigma_{11}}{\sigma_1}}$ où $\sigma_1 < \cdots < \sigma_n$ valeurs propres de $A \cdot A^t$.

Propriétés

- $\text{cond}_2(A) = 1$ si et seulement si $A = \alpha Q$ avec Q orthogonale
- Si $A = Q \cdot R$, alors $\text{cond}_2(A) = \text{cond}_2(R)$.
- Si A et B sont symétriques définies positives, alors $\text{cond}_2(A + B) \leq \max\{\text{cond}_2(A), \text{cond}_2(B)\}$

proposition

Si x solution de $Ax = b$ et $x + \delta x$ solution de $A(x + \delta x) = b + \delta b$, alors :

$$\frac{\|\delta x\|}{\|x\|} \leqslant \text{cond}(A) \cdot \frac{\|\delta b\|}{\|b\|}$$

Démo :

Nous avons $A \cdot x = b$ et $A(x + \delta x) = b + \delta b$. Donc $A \cdot \delta x = \delta b$. A est inversible, donc $\delta x = A^{-1} \cdot \delta b$. Par passage à la norme :

$\|\delta x\| \leqslant \|A^{-1}\| \cdot \|\delta b\|$.

D'autre part, $b = A \cdot x$, donc $\|b\| \leqslant \|A\| \cdot \|x\|$. Donc $\frac{1}{\|x\|} \leqslant \frac{\|A\|}{\|b\|}$.

Par multiplication des deux inégalités, nous obtenons $\frac{\|\delta x\|}{\|x\|} \leqslant \|A\|^{-1} \cdot \|A\| \cdot \frac{\|\delta b\|}{\|b\|}$.

proposition

Si x solution de $A \cdot x = b$ et $x + \delta x$ solution de $(A + \delta A)(x + \delta x) = b$, alors :

$$\frac{\|\delta x\|}{\|x + \delta x\|} \leqslant \text{cond}(A) \cdot \frac{\|\delta A\|}{\|A\|}$$

Démo :
On a $A \cdot x = b$ et $(A + \delta A)(x + \delta x) = b$. Donc $A\delta x = -\delta A(x + \delta x)$. Par passage à la norme, nous obtenons :

$$\|\delta x\| \leqslant \|A^{-1}\| \cdot \|\delta A\| \cdot \|x + \delta x\|$$

$$\frac{\|\delta x\|}{\|x + \delta x\|} \leqslant \|A^{-1}\| \cdot \|\delta A\|$$

$$\frac{\|\delta x\|}{\|x + \delta x\|} \leqslant \|A^{-1}\| \cdot \|A\| \cdot \frac{\|\delta A\|}{\|A\|}$$

Theorem

On suppose $\|\delta A\| < \frac{1}{\|A^{-1}\|}$ *et* $b \neq 0$. *Alors* $(A + \delta A)$ *est inversible et si* x *est solution de* $A \cdot x = b$ *et* $x + \delta x$ *de* $(A + \delta A)(x + \delta x) = b + \delta b$, *on a :*

$$\frac{\|\delta x\|}{\|x\|} \leqslant \frac{cond(A)}{1 - \|A^{-1}\| \cdot \|\delta A\|} \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|} \right)$$

Démo :

La matrice $I_d + A^{-1} \cdot \delta A$ est inversible car $\rho(A^{-1} \cdot \delta A) \leqslant \|A^{-1} \cdot \delta A\| \leqslant \|A^{-1}\| \cdot \|\delta A\|$. On applique alors le théorème.

La matrice $A + \delta A = A(I + A^{-1} \cdot \delta A)$ est donc inversible et il existe δx tel que :

$$(A + \delta A)(x + \delta x) = b + \delta b$$

Ainsi :

$$\delta A \cdot x + A \left(I_d + A^{-1} \cdot \delta A \right) \delta x = \delta b$$

Donc :

$$\frac{\delta x}{\|x\|} = \left(I_d + A^{-1} \cdot \delta A \right)^{-1} \cdot A^{-1} \left(\frac{\delta b}{\|x\|} - \frac{\delta A \cdot x}{\|x\|} \right)$$

Par passage à la norme :

$$\begin{aligned}\frac{\|\delta x\|}{\|x\|} &\leqslant \left\|(I_d + A^{-1}\cdot\delta A)^{-1}\right\| \cdot \|A^{-1}\| \left(\frac{\|\delta b\|}{\|x\|} + \|\delta A\|\right) \\ &\leqslant \left\|(I_d + A^{-1}\cdot\delta A)^{-1}\right\| \cdot \mathrm{cond}(A) \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|}\right)\end{aligned}$$

Par le théorème **??** :

$$\begin{aligned}\frac{\|\delta x\|}{\|x\|} &\leqslant \frac{\mathrm{cond}(A)}{1 - \|A^{-1}\cdot\delta A\|} \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|}\right) \\ &\leqslant \frac{\mathrm{cond}(A)}{1 - \|A^{-1}\| \cdot \|\delta A\|} \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|}\right)\end{aligned}$$

Exemple de système linéaire mal conditionné

Considérons le système

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix} \text{ de solution } \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Le système perturbé

$$\begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \begin{pmatrix} u_1 + \delta u_1 \\ u_2 + \delta u_2 \\ u_3 + \delta u_3 \\ u_4 + \delta u_4 \end{pmatrix} = \begin{pmatrix} 32,1 \\ 22,9 \\ 33,1 \\ 30,9 \end{pmatrix} \text{ de solution } \begin{pmatrix} 9,2 \\ -12,6 \\ 4,5 \\ -1,1 \end{pmatrix}$$

Exemple de système linéaire mal conditionné

Pourtant, la matrice est "bonne" (symétrique, de déterminant 1, donc loin de 0). Son inverse est d'ailleurs donnée par

$$A^{-1} = \begin{pmatrix} 25 & -41 & 10 & -6 \\ -41 & 68 & -17 & 10 \\ 10 & -17 & 5 & -3 \\ -6 & 10 & -3 & 2 \end{pmatrix}$$

Mais les valeurs propres de A sont

$$\lambda_1 \simeq 0,01015 < \lambda_2 \simeq 0,8431 < \lambda_3 \simeq 3,858 < \lambda_4 \simeq 30,2877$$

$$\text{cond}_2(A) = \frac{\lambda_4}{\lambda_1} \simeq 2984 >> 1$$

Exemple de système linéaire mal conditionné

D'autre part

$$u = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \delta u = \begin{pmatrix} 8,2 \\ -13,6 \\ 3,5 \\ -2,1 \end{pmatrix}, b = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}, \delta b = \begin{pmatrix} 0,1 \\ -0,1 \\ 0,1 \\ -0,1 \end{pmatrix}$$

de sorte que

$$\frac{\|\delta u\|_2}{\|u\|_2} \simeq 8,1985 \text{ et } \text{cond}_2(A)\frac{\|\delta b\|_2}{\|b\|_2} \simeq 9,9424$$

On n'est donc pas loin de l'égalité dans l'estimation

$$\frac{\|\delta u\|_2}{\|u\|_2} \leq \text{cond}_2(A)\frac{\|\delta b\|_2}{\|b\|_2}$$

Méthodes itératives

definition

La méthode itérative construit une suite récurrente $x^{(k)}$ qui converge pour tout $x_0^{(0)}$ vers x, solution de $A \cdot x = b$.

On décompose la matrice A sous la forme $A = P - N$. On a alors l'équivalence :

$$P \cdot x = N \cdot x + b$$

Si P est inversible, on définit la suite $\left(x^k\right)_{k\in\mathbb{N}}$ par :

$$\begin{cases} P \cdot x^{(k+1)} = N \cdot x^{(k)} + b \\ x^{(0)} \in \mathbb{R}^n \text{ donné} \end{cases}$$

Si la suite converge, alors elle converge vers $x = A^{-1}b$.
Dans la suite, on pose $B = P^{(-1)}N$ et $c = P^{-1} \cdot b$:

$$\boxed{\begin{cases} x^{(k+1)} = B \cdot x^{(k)} + c \\ x^{(0)} \in \mathbb{R}^n \text{ donné} \end{cases}}$$

Theorem

- *La suite* $\left(x^{(k)}\right)_{k\in\mathbb{N}}$ *converge vers* $x = A^{-1}\cdot b$ *ssi* $\rho(B) < 1$.
- *La suite* $\left(x^{(k)}\right)_{k\in\mathbb{N}}$ *converge vers* $x = A^{-1}\cdot b$ *si et seulement si il existe une norme telle que* $\|B\| < 1$.

Démo :
Nous avons $x^{(k+1)} - x = B\left(x^k - x\right)$. Donc $x^{(k)} - x = B^k\left(x^0 - x\right)$.

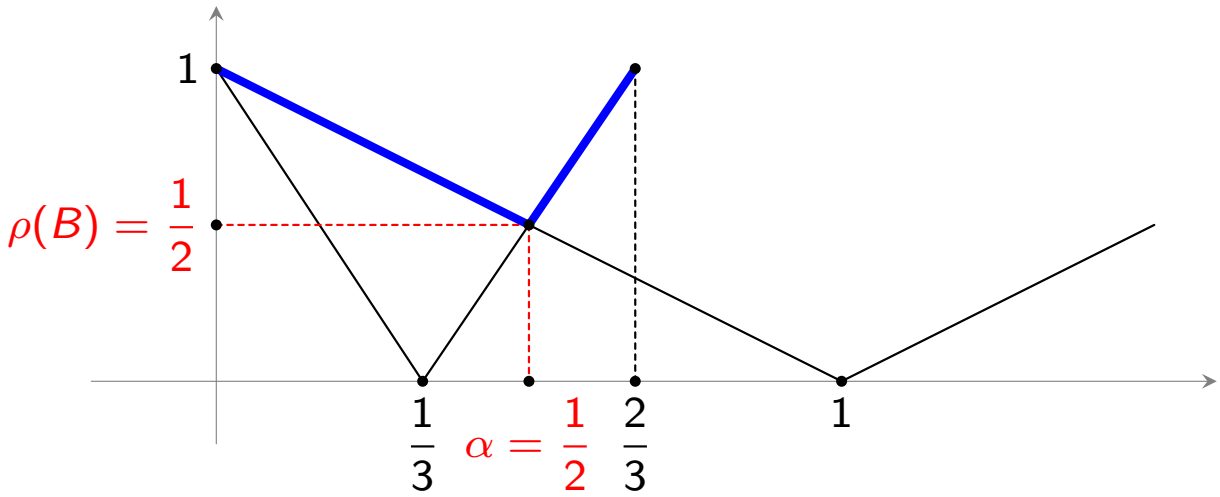
- D'après le corollaire , si $\rho(B) < 1$, alors $\lim\limits_{k\to\infty} B^k = 0$. D'où la convergence de la suite vers la solution.
- Si $\|B\| < 1$, alors $\|B^k\| \leqslant \|B\|^k$ et donc $\lim\limits_{k\to\infty} B^k = 0$. D'où la convergence de la suite.

example

$A = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$, $P = \mathrm{I}$ et $N = \mathrm{I} - A = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} = B$.
Les valeurs propres de B sont 0 et -2 ; $\rho(B) = 2$.
La suite $\begin{cases} x^{(k+1)} = Nx^{(k)} + b \\ x^{(0)} \in \mathbb{R}^n \text{ donné} \end{cases}$ ne converge pas.
Prenons $P = \beta \mathrm{I}$ et $N = P - A = \beta \mathrm{I} - A$.
Ainsi :

$$\begin{aligned} B &= I_d - \frac{A}{\beta} \\ &= I_d - \alpha A \\ &= \begin{pmatrix} 1-2\alpha & \alpha \\ \alpha & 1-2\alpha \end{pmatrix} \end{aligned}$$

où $\alpha = \frac{1}{\beta}$. Les valeurs propres de B sont $1 - \alpha\lambda$ (où λ valeur propre de A), $1 - \alpha$ et $1 - 3\alpha$. Donc $\rho(B) = \max\{|1-\alpha|, |1-3\alpha|\}$



Méthode converge $0 < \alpha < \frac{2}{3}$
La valeur optimale de $\rho(B)$ correspond à $\alpha = \frac{1}{2}$.

Méthode de Jacobi

Soit la décomposition de la matrice A en somme de trois matrices : $A = D - E - F$ où D représente la diagonale de A, $-E$ et $-F$ les parties triangulaires inférieures et supérieures.

$$D = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & & & a_{nn} \end{pmatrix}$$

$$-E = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ a_{n1} & & & 0 \end{pmatrix}; \quad -F = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & & & 0 \end{pmatrix}$$

La méthode de Jacobi est itérative telle que $P = D$ et $N = E + F$. Elle s'écrit donc :

$$\begin{cases} x^{(k+1)} = D^{-1}(E+F)x^{(k)} + D^{-1}b \\ x_0 \in \mathbb{R}^n \end{cases}$$

On note $B_J = D^{-1}(E+F)$ et $C_J = D^{-1}b$.

$$B_J = \begin{pmatrix} 0 & -\frac{a_{12}}{a_{11}} & \cdots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & \ddots & & \vdots \\ \vdots & & \ddots & 0 \\ \frac{a_{n1}}{a_{nn}} & & & 0 \end{pmatrix} \qquad C_J = \begin{pmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \vdots \\ \frac{b_n}{a_{nn}} \end{pmatrix}$$

La forme développée de la méthode s'écrit :

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(k)}}{a_{ii}} \qquad \forall i \in [\![1, n]\!]$$

où $x^{(0)}$ est un vecteur arbitraire de $\mathbb{R}^n$.

Algorithm 8: Méthode de Jacobi

for i **from** 1 **to** n **do**
- $s = b_i$
- **for** j **from** 1 **to** $i - 1$ **do**
  - $s = s - a_{ij} \cdot x_j$
- **for** j **from** $i + 1$ **to** n **do**
  - $s = s - a_{ij} \cdot x_j$
- $y_i = \frac{s}{a_{ii}}$

for i **from** i **to** n **do**
- $x_i = y_i$

```python
def jacobi(A,b,x0,N):
    x=x0
    for k in range(N):
        n=len(A)
        y=np.zeros(n)
        for i in range(n):
            y[i]=(b[i]-A[i,:i]@x[:i]-A[i,i+1:]@x[i+1:])/A[i,i]
        x=y.copy()
    return x
```

Example

Soit à résoudre le système $\begin{cases} 10x_1 + x_2 = 11 \\ 2x_1 + 10x_2 = 12 \end{cases}$ dont la solution est $x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

En partant de $x^{(0)} = 0$, on obtient successivement :

$$x^{(1)} = \begin{pmatrix} \frac{11}{10} \\ \frac{12}{10} \end{pmatrix}, \quad x^{(2)} = \begin{pmatrix} \frac{98}{100} \\ \frac{98}{100} \end{pmatrix}, \quad x^{(3)} = \begin{pmatrix} \frac{1002}{1000} \\ \frac{1004}{1000} \end{pmatrix}, \quad x^{(4)} = \begin{pmatrix} \frac{9996}{10000} \\ \frac{9992}{10000} \end{pmatrix}$$

La suite de la méthode de Jacobi semble converger vers la solution $x_1 = 1$, $x_2 = 1$.

Example

Soit à résoudre le système $\begin{cases} x_1 + 10x_2 = 11 \\ 10x_1 + 2x_2 = 12 \end{cases}$ dont la solution est aussi $x = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

En partant également de $x^{(0)} = 0$, on obtient :

$$x^{(1)} = \begin{pmatrix} 11 \\ 6 \end{pmatrix}, \quad x^{(2)} = \begin{pmatrix} -49 \\ -49 \end{pmatrix}, \quad x^{(3)} = \begin{pmatrix} 501 \\ 251 \end{pmatrix}, \quad x^{(4)} = \begin{pmatrix} -2499 \\ -2499 \end{pmatrix}, \quad .$$

La suite dans le second cas s'éloigne de la solution et semble diverger.

La méthode de Gauss-Seidel

Reprenons la méthode de Jacobi sous une forme développée :

$$a_{ii}x_i^{(k+1)} = -\sum_{j<i} a_{ij}x_j^{(k)} - \sum_{j>i} a_{ij}x_j^{(k)} + b_i \qquad \forall i \in [\![1, n]\!]$$

À l'étape i, les termes $\left(x_j^{(k+1)}\right)_{j<i}$ ont été déjà calculés. En général, ils sont plus proches de la solution que $\left(x_j^{(k)}\right)_{j<i}$ dans le cas d'une convergence : d'où l'idée de remplacer $x_j^{(k)}$ par $x_j^{(k+1)}$. Ainsi au lieu d'attendre une itération entière pour corriger chaque composante, la correction se fait au fur et à mesure :

$$\begin{cases} a_{ij}x_i^{(k+1)} = -\sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>i} a_{ij}x_j^{(k)} + b_i \\ x^{(0)} \in \mathbb{R}^n \end{cases}$$

La méthode de Gauss-Seidel

Cela revient à choisir une méthode itérative avec $P = D - E$ et $N = F$. La méthode de Gauss-Seidel s'écrit matriciellement : fait au fur et à mesure :

$$\begin{cases} (D - E)x^{(k+1)} = Fx^{(k)} + b \\ x^{(0)} \in \mathbb{R}^n \end{cases}$$

ou bien :

$$\begin{cases} x^{(k+1)} = B_{GS}x^k + c_{GS} \\ x^{(0)} \in \mathbb{R}^n \end{cases}$$

où $B_{GS} = (D - E)^{-1} \cdot F$ et $c_{GS} = (D - E)^{-1} \cdot b$.

Algorithm 9: Méthode de Gauss-Seidel

for i **from** 1 **to** n **do**
  $s = b_i$
  for j **from** 1 **to** n **do**
    $s = s - a_{ij} \cdot x_j$
  $x_i = x_i + \frac{s}{a_{ii}}$

```python
def GaussSeidel(A,b,x0,N):
    x=x0
    for k in range(N):
        n=len(A)
        y=np.zeros(n)
        for i in range(n):
            x[i]+=(b[i]-A[i]@x)/A[i,i]
    return x
```

La méthode de Gauss-Seidel

La programmation de la méthode de Gauss-Seidel ne nécessite pas deux tableaux X et Y comme dans la méthode de Jacobi. D'où le gain de stockage. En pratique, la méthode de Gauss-Seidel converge (ou diverge) souvent plus rapidement que celle de Jacobi car on utilise les nouvelles valeurs des composantes dès qu'elles sont calculées.

Theorem

Pour une matrice tridiagonale :

$$\rho(B_{GS}) = (\rho(B_J))^2$$

Donc la méthode de Gauss-Seidel converge ou diverge plus vite que celle de Jacobi.

proposition

Si A est une matrice symétrique définie positive, alors la méthode de Gauss-Seidel converge.

La méthode de Gauss-Seidel

Démo :

A étant symétrique définie positive, on a $a_{ii} = \exp_i^t \cdot A \cdot \exp_i > 0$. Donc les termes diagonaux de $D - E$ sont non nuls. D'où $P = D - E$ inversible et $B = (D - E)^{-1}$ a bien un sens.

On a $B = D^{-\frac{1}{2}} \cdot \left(I_d - D^{-\frac{1}{2}} \cdot E \cdot D^{-\frac{1}{2}}\right)^{-1} \cdot D^{-\frac{1}{2}} \cdot E^t$.

Posons $L = D^{-\frac{1}{2}} \cdot E \cdot D^{-\frac{1}{2}}$. On a
$B = D^{-\frac{1}{2}} \cdot (I_d - L) \cdot L^t \cdot D^{\frac{1}{2}} = D^{-\frac{1}{2}} \cdot B_1 \cdot D^{\frac{1}{2}}$ en posant $B_1 = (I - L) \cdot L^t$.

L'équivalence $B \cdot x = \lambda \cdot x \iff B_1 \left(D^{\frac{1}{2}} \cdot x\right) = \lambda \left(D^{\frac{1}{2}} x\right)$ prouve que B et B_1 ont les mêmes valeurs propres.

Soit λ une valeur propre de B_1 associée à un vecteur propre x tel que $\|x\| = 1$. On a $B_1 \cdot x = \lambda x$. Donc $(I - L)^{-1} \cdot L^t \cdot x = \lambda x$. Donc $L^t \cdot x = \lambda (I_d - L) \cdot x$. D'où $\lambda = \frac{x^t \cdot L \cdot x}{1 - x^t \cdot L \cdot x}$. Puis en développant l'inégalité $\left(D^{-\frac{1}{2}} \cdot x\right)^t \cdot A \cdot \left(D^{-\frac{1}{2}} \cdot x\right) > 0$, on trouve $x^t \cdot L \cdot x < \frac{1}{2}$. D'où $|\lambda| < 1$. Le rayon spectral de B étant strictement inférieur à 1, la méthode de Gauss-Seidel converge.

Les méthodes par blocs

En pratique, les systèmes linéaires obtenus par discrétisation des équations de la physique sont souvent tridiagonales ou tridiagonales par blocs et s'adaptent bien à une résolution par méthode itérative.
Soit $A \in \mathscr{M}_n(\mathbb{R})$ une matrice inversible. On suppose qu'elle se décompose par blocs :

$$A = \begin{pmatrix} A_{11} & A_{12} & & \\ A_{21} & & & \\ & & & A_{nn} \end{pmatrix}$$

où $A_{IJ} \in \mathscr{M}_{nI,nJ}(\mathbb{R})$ est une matrice inversible et $(n_1, n_2, \ldots, n_p)$ un p-uplet tel que $\sum_{I=1}^{p} n_I = n$.

On décompose la matrice A en une somme de trois matrices par bloc $A = D - E - F$ où :

$$D = \begin{pmatrix} A_{11} & & & \\ & A_{22} & & \\ & & & A_{nn} \end{pmatrix}$$

$$E = \begin{pmatrix} 0 & & & \\ -A_{21} & & & \\ -A_{31} & -A_{32} & & \\ & & & 0 \end{pmatrix} \qquad F = \begin{pmatrix} 0 & -A_{12} & & -A_{1n} \\ 0 & 0 & -A_{23} & \\ & & & \\ & & & 0 \end{pmatrix}$$

Soit $(X_1^t, X_2^t, \ldots, X_P^t)^t$ une décomposition par blocs de x adaptée à la décomposition de la matrice de A, c'est-à-dire $\forall I \in [\![1, P]\!],\ X_I \in \mathbb{R}^{n_I}$.

$$x = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_P \end{pmatrix} \qquad b = \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_P \end{pmatrix}$$

La méthode de Jacobi par blocs s'écrit :

$$\begin{cases} A_{II} X_I^{(k+1)} = -\sum_{j<i} A_{IJ} X_J^{(k)} - \sum_{j>i} A_{IJ} X_J^{(k)} + B_I \qquad I = 1, \ldots, P \\ x^{(0)} \in \mathbb{R} \end{cases}$$

Dans le cas particulier où $P = n$, chaque bloc A_{IJ} se réduit au coefficient a_{ij} et $X_I = x_i$, $B_I = b_i$. On retrouve la méthode de Jacobi par points.

La méthode de relaxation par blocs devient :

$$A_{ii} \cdot x_i^{(k+1)} = \omega \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} \right) + (1-\omega) A_{ii} x_i^{(k)} - \omega \sum_{j=i+1}^{p} A_{ij} x_j^k$$

Ce qui est équivalent à :

$$A_{ii} \left(x_i^{(k+1)} - x_i^{(k)} \right) = \omega \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i}^{p} A_{ij} x_j^k \right)$$

Méthode de relaxation

La méthode itérative de relaxation généralise les deux méthodes précédentes, Jacobi et Gauss-Seidel. Elle est définie par la relaxation :

$$\boxed{x_i^{(k+1)} = \omega \widetilde{x}_i^{(k+1)} + (1-\omega) x_i^{(k)}}$$

où le réel ω est un paramètre de la méthode et $\widetilde{x}_i^{(k+1)}$ est la composante obtenue à partir de $x_i^{(k)}$ par une méthode de Jacobi ou de Gauss-Seidel. Si la méthode auxiliaire est celle de Jacobi, on obtient pour $i \in [\![1, n]\!]$:

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

Soit matriciellement :

$$x^{(k+1)} = \left(I_d - \omega D^{-1} \cdot A \right) \cdot x^{(k)} + \omega D^{-1} \cdot b$$

Cette méthode est très peu utilisée car en général elle n'apporte aucun gain significatif.

Si la méthode de base choisie est celle de Gauss-Seidel, la méthode de relaxation est définie par :

$$x_i^{(k+1)} = \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) + (1-\omega) x_i^{(k)}$$

Ou bien en retranchant $x_i^{(k)}$ aux deux membres :

$$\boxed{x_i^{(k+1)} - x_i^k = \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)} \right)}$$

Soit matriciellement :

$$\boxed{x^{(k+1)} = \left(\frac{D}{\omega} - E \right)^{-1} \cdot b + \left(\frac{D}{\omega} - E \right)^{-1} \left[\left(\frac{1}{\omega} - 1 \right) \cdot D + F \right] \cdot x^{(k)}}$$

L'algorithme de la méthode de relaxation est aussi simple que celui de Gauss-Seidel :

Algorithm 10: Méthode de relaxation

for i **from** 1 **to** n **do**

$s = b_i$

for j **from** 1 **to** n **do**

$s = s - a_{ij} \cdot x_j$

$x_i = x_i + \omega \dfrac{s}{a_{ii}}$

Méthode du gradient

A matrice symétrique définie positive, $b \in \mathbb{R}^n$ et

$$J(x) = \frac{1}{2}(Ax, x) - (b, x)$$

Pour $h \in \mathbb{R}^n$ on a

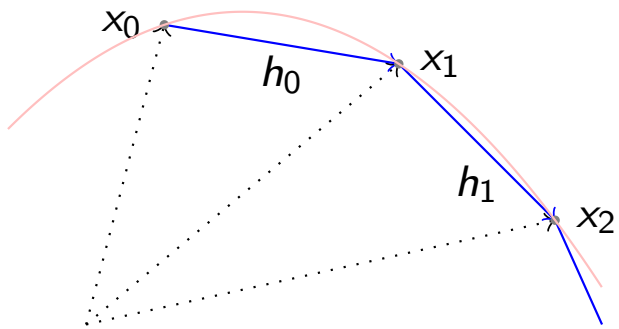
$$J(x+h) = J(x) + (Ax - b, h) + \frac{1}{2}(Ah, h)$$

Donc $\nabla J(x) = Ax - b$

$$J(\bar{x}) \text{ minimal } \Longrightarrow A\bar{x} - b = 0$$

Donc la solution de $Ax = b$ réalise le min de J :

$$J(\bar{x}) = \inf_{x \in \mathbb{R}^n} J(x)$$



$$J(x_0) > J(x_1) > J(x_2) > \cdots$$

On pose $x_1 = x_0 + h$. On a

$$J(x_1) = J(x_0 + h) = J(x_0) + (\nabla J, h) + \frac{1}{2}(Ah, h)$$

On choisit $h = -\alpha \nabla J$:

$$\boxed{x_{k+1} = x_k - \alpha \nabla J(x_k)}$$

Proposition

Soit $p \in \mathbb{R}^n$ donné et $f(\alpha) = J(x - \alpha p)$. Alors le minimum de f est réalisé en

$$\alpha = \frac{(\nabla J, p)}{(Ap, p)}$$

Démo :

$$J(x - \alpha p) = J(x) - \alpha(\nabla J, p) + \frac{\alpha^2}{2}(Ap, p)$$

$$\min \implies \frac{\mathrm{d}}{\mathrm{d}\alpha} J(x - \alpha p) = 0 \implies -(\nabla J, p) + \alpha(Ap, p) = 0$$

Méthode du gradient à pas optimal

On a $\nabla J(x) = Ax - b$ On pose $r = b - Ax$, on a donc

$$\begin{cases} r_k = b - Ax_k \\ \alpha_k = \frac{\|r_k\|^2}{(Ar_k, r_k)} \\ x_{k+1} = x_k + \alpha_k r_k \end{cases}$$

Algorithm 11: Méthode du gradient

x_0 donné

while $J(x_k) - J(x_{k+1}) > \varepsilon$ **do**

$r_k = b - Ax_k$

$\alpha_k = \frac{\|r_k\|^2}{(Ar_k, r_k)}$

$x_{k+1} = x_k + \alpha_k r_k$

```python
def gradient(A,b,x0,N):
    x=x0.copy()
    for k in range(N):
        r=b-A@x
        alpha=(r@r)/(A@r@r)
        x+=alpha*r
    return x
```

Méthode du gradient conjugué

Algorithm 12: Méthode du gradient conjugué

x_0 donné

$r_0 = b - Ax_0$

$p_0 = r_0$

for k **from** 0 **to** $n-1$ **do**

$$\alpha_k = \frac{\|r_k\|^2}{(Ar_k, r_k)}$$

$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k A p_k$$

$$\beta_k = \frac{\|r_{k+1}\|^2}{\|r_k\|^2}$$

$$p_{k+1} = r_{k+1} + \beta_k p_k$$

```python
def GradientConjugue(A,b,x0,N):
    x=x0.copy()
    r=b-A@x
    p=r.copy()
    for k in range(N):
        alpha=(r@r)/(A@r@r)
        x+=alpha*p
        r1=r.copy()
        r+=-alpha*A@p
        beta = (r@r)/(r1@r1)
        p=r+beta*p

    return x
```

Valeur et vecteur propre

Soit

$$A = \begin{pmatrix} 10 & 0 \\ -9 & 1 \end{pmatrix}$$

On fait

$$\begin{cases} x_{k+1} = Ax_k \\ x_0 \quad \text{arbitraire} \end{cases}$$

$$x_0 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \quad x_1 = \begin{pmatrix} 20 \\ 17 \end{pmatrix}, \quad x_2 = \begin{pmatrix} 200 \\ -197 \end{pmatrix}, \quad x_3 = \begin{pmatrix} 2000 \\ -1997 \end{pmatrix}$$

$$x_4 = \begin{pmatrix} 20000 \\ -19997 \end{pmatrix} \cdots$$

$$\Longrightarrow \begin{cases} x_k \quad // \begin{pmatrix} 1 \\ -1 \end{pmatrix} & : \text{vecteur propre} \\ x_{k+1} \simeq 10 x_k & : \text{valeur propre} \end{cases}$$

Valeur et vecteur propre

Si A diagonalisable, ici $\lambda_1 = 10$ associée au vecteur propre $v_1 = (1, -1)$ et $\lambda_2 = 10$ associée à $v_2 = (0, 1)$. D'où

$$A^k = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix}^k \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 10^k & 0 \\ -10^k + 1 & 1 \end{pmatrix}$$

Donc

$$x_k = A^k x_0 = \begin{pmatrix} 10^k & 0 \\ -10^k + 1 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \times 10^k \\ -2 \times 10^k + 3 \end{pmatrix}$$

Donc

$$x_k = 2 \times 10^k \begin{pmatrix} 1 \\ -1 + 1.5 \times 10^{-k} \end{pmatrix} \simeq 2 \times 10^k \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Algorithm 13: Puissances itérées

$q_0 \in \mathbb{C}^n$ tel que $\|q_0\| = 1$

for k **from** 1 **to** N **do**

$$x_k = Aq_{k-1}$$
$$\omega_k = \|x_k\|$$
$$q_k = \frac{x_k}{\omega_k}$$