Résolution de systèmes linéaires

Skander Zannad et Judicaël Courant

2014-02-21

1 Motivations

Résolution d'une EDO : inconnue dans $E^{\mathbb{R}}$ (où E espace vectoriel).

Méthode d'Euler : recherche d'un nombre fini de valeurs ($y(t_0 + k\delta)$ où t_0 instant initial, δ pas), vérifiant certaines relations.

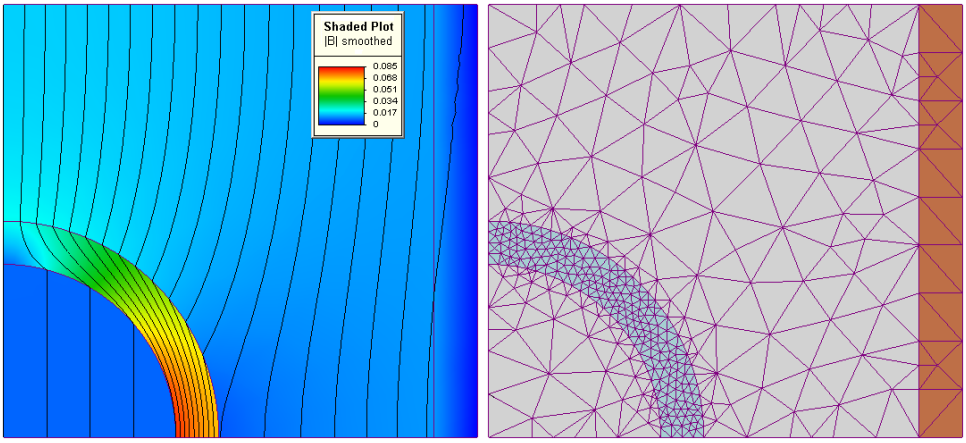
Principe : *discrétisation* du temps.

En physique : beaucoup d'équations aux dérivées partielles (EDP).

Résolution : discrétisation du temps et/ou de l'espace.

Applications :

- Diffusion de la chaleur dans un matériau ;
- Calcul du champ magnétique induit par une source électromagnétique ;
- Simulation de déformation d'une voiture en cas de choc.
- ...



Solution d'une équation magnétostatique. À droite, le maillage utilisé.

2 Méthode de Gauss-Jordan

Appelée aussi méthode du pivot de Gauss.
Cadre : équation $Ax = b$, où $A \in \mathcal{M}_n(\mathbb{K})$, $b \in \mathcal{M}_{n1}(\mathbb{K})$, inconnue $x \in \mathcal{M}_{n1}(\mathbb{K})$.

2.1 Cas où A est triangulaire

Si A est triangulaire (par ex. supérieure) et inversible ($\forall k \quad a_{kk} \neq 0$), la résolution est facile. Les solutions du système :

$$\begin{array}{ccccccccccc}
a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & \cdots & + & a_{1n}x_n & = & b_1 \\
 & & a_{22}x_2 & + & \cdots & + & \cdots & + & a_{2n}x_n & = & b_2 \\
 & & & \ddots & & & \vdots & & \vdots & & \vdots \\
 & & & & a_{kk}x_k & + & \cdots & + & a_{kn}x_n & = & b_k \\
 & & & & & \ddots & \vdots & & \vdots & & \vdots \\
 & & & & & & & & a_{nn}x_n & = & b_n
\end{array}$$

se calculent de proche en proche :

$$\begin{aligned}
x_n &= \frac{1}{a_{nn}}\,(b_n) \\
x_{n-1} &= \frac{1}{a_{n-1,n-1}}\,(b_{n-1} - a_{n-1,n}x_n) \\
&\vdots \\
x_k &= \frac{1}{a_{k,k}}\,(b_k - a_{k,k+1}x_{k+1} - \cdots - a_{k,n}x_n)
\end{aligned}$$

2.2 Cas général

On se ramène au cas triangulaire : pour cela on va effectuer des opérations sur les lignes du système $Ax = b$.

En numérotant les lignes de 0 à $n-1$ et les colonnes de 0 à $n-1$, et en notant L_i la ligne i pour $i \in [\![0, n[\![$, l'algorithme du pivot partiel est le suivant :

(i) Pour $i = 0, 1, \ldots, n-1$:
  (i) Trouver j dans $[\![i, n[\![$ tel que $|a_{ji}|$ soit maximal
  (ii) Échanger les lignes i et j dans A et dans b.
  (iii) Poser $p = a_{ii}$.
  (iv) Pour j de $i+1$ à $n-1$:
    (i) Remplacer b_j par $b_j - \frac{a_{ij}}{p} b_i$.
    (ii) Remplacer la ligne L_j de A par $L_j - \frac{a_{ji}}{p} L_i$.

(ii) A est maintenant triangulaire, calculer la solution avec la méthode précédente.

3 Implantation

A et b matrices, donnés sous forme d'un tableau bidimensionnel (type `array` de `numpy`). Intérêt d'`array` :

(i) opérations d'ajout de lignes et de multiplication d'une ligne par un scalaire déjà disponibles (sinon, il faut les programmer) ;

(ii) extractions de lignes faciles (tableau unidimensionnel).

NB : Il existe aussi un type `matrix`. Adapté pour des manipulations de matrices mais pas très pratique pour l'extraction de lignes.

```
from numpy import array, zeros

def matrice(m):
    """Construit une matrice à partir
    d'une liste de listes donnée en argument."""
    return array(m)

def mat_nulle(p, q):
    """Retourne, la matrice nulle de dimension p * q
    sous forme de tableau (array)."""
    return zeros(shape=(p, q))

def echange_lignes(A, i, j):
    """Échange les lignes i et j de la matrice,
    représentée comme liste de lignes de type array"""
    # on COPIE les éléments de la ligne j :
```

```
  t = A[j,:].copy()
  # avant de l'écraser:
  A[j,:] = A[i,:]
  A[i,:] = t

def resout_triang_inf(A, b):
  """Résout le système Ax = b.
  Préconditions: A triangulaire inférieure
  et b a autant de lignes que A."""
  n = len(A)
  p = len(b)
  assert n == p # même nombre de lignes dans A et b
  x = mat_nulle(len(A[0]), len(b[0]))
  for k in range(n-1, -1, -1):
    x[k,:] = 1./A[k,k] * (
     b[k,:] - sum(A[k,j]*x[j,:] for j in range(k+1, n)))
  return x

def cherche_pivot(A, i):
  """Cherche et retourne un j tel que
  abs(A[j][i]) est maximal, avec j>=i"""
  n = len(A)
  best = i
  for j in range(i+1, n):
    # invariant: pour tout k <= j
    # abs(A[best][i]) >= abs(A[k][i])
    if abs(A[j,i]) > abs(A[best,i]):
      best = j
  return best

def resout(A, b):
  """Applique la méthode du pivot pour résoudre Ax = b.
  Retourne la solution x trouvée.
  Précondition : A et b sont de type array,
  A est inversible, b a même nombre de lignes que A.
  Attention: cette fonction modifie A et b."""
  n = len(A)
  q = len(b)
  assert n == q # même nombre de lignes dans A et b
  A = A.copy()
  b = b.copy()

  for i in range(n):
    j = cherche_pivot(A, i)
    # on met en place la ligne du pivot :
    echange_lignes(A, i, j)
    echange_lignes(b, i, j)
```

```
    p = A[i, i] # le pivot
    for j in range(i+1, n):
      b[j,:] -= A[j,i] / p * b[i,:]
      A[j,:] -= A[j,i] / p * A[i,:]
  return resout_triang_inf(A, b)
```

4 Temps de calcul

Quel est le coût de l'algorithme du pivot ?

(i) Cas d'une matrice triangulaire : le calcul de la dernière composante de la solution requiert une division, la précédente une division, une multiplication, une soustraction, ..., la première composante une division, $n-1$ multiplications et $n-1$ soustractions. Au total : n divisions, $n(n-1)/2$ multiplications et $n(n-1)/2$ soustractions. Temps de calcul : $\Theta(n^2)$.

(ii) Cas général : à l'étape i, on cherche le pivot ($n-i-1$ comparaisons), un échange de lignes (n flottants à échanger), puis pour chacune des $n-i-1$ dernières lignes, on effectue une multiplication de la ligne i avant de soustraire le résultat. Il est clair que le nombre d'opérations avant d'arriver à une matrice triangulaire est un $O(n^3)$ et que c'est un $\Omega(n^3)$ (les $n/2$ premières étapes ont un coût supérieur à $n^3/8$).

L'algorithme du pivot est donc relativement coûteux.
Il est ici du même ordre que le produit (naïf) de deux matrices ($\Theta(n^3)$).
Voir plus loin les améliorations possibles.

5 Stabilité numérique : Problèmes possibles

Les calculs n'étant pas faits de façon exacte mais approchée, des problèmes peuvent survenir.

5.1 Incapacité à inverser une matrice

La matrice suivante est inversible :

$$\begin{pmatrix} 10^{20} & 10^{20} & 10 \\ 10^{19} & 1 & 0 \\ 10^{19} & 0 & 0 \end{pmatrix}$$

Après une étape de pivot, on arrive à :

$$\begin{pmatrix} 10^{20} & 10^{20} & 1 \\ 0 & 1-10^{19} & -1 \\ 0 & -10^{19} & -1 \end{pmatrix}$$

Mais en machine, $1 - 10^{19}$ va être arrondi (à la même valeur que -10^{19}) :

$$\begin{pmatrix} 10^{20} & 10^{20} & 1 \\ 0 & -10^{19} & -1 \\ 0 & -10^{19} & -1 \end{pmatrix}$$

Résultat : les deux dernières lignes sont égales et la matrice n'est plus inversible.

5.2 Capacité à inverser des matrices non inversibles

Considérons une matrice de la forme

$$\begin{pmatrix} a & b \\ a & b \end{pmatrix}$$

avec $a \neq 0$.

Après une étape de pivot, on obtient :

$$\begin{pmatrix} a & b \\ 0 & b - (b/a) \times a \end{pmatrix}$$

Si, en raison d'un arrondi, $b - (b/a) \times a$ ne donne pas 0, on obtient une matrice triangulaire sans zéro sur sa diagonale, donc inversible.

(en flottant ce problème se produit par exemple avec $b = 0,9999$ et $a = 1,9999$).

5.3 Conditionnement

Considérons la matrice $M = ((i + j - 1)^{-1})_{1 \leq i,j \leq 5}$ (matrice de Hilbert) et cherchons à résoudre $MX = B_1$ et $MX = B_2$ pour deux valeurs proches de B_1 et B_2 :

```
n = 5
M = matrice([[ 1/(i+j+1.) for j in range(n)]
               for i in range(n)])
u0 = matrice([[-0.76785474],
       [-0.44579106],
       [-0.32157829],
       [-0.25343894],
       [-0.20982264]])
s0 = resout(M, u0)
u1 = matrice([[-0.76784856],
       [-0.44590775],
       [-0.32107213],
       [-0.25420613],
       [-0.20944639]])
s1 = resout(M, u1)
```

Résultat :

```
>>> s0
array([[-0.4900022],
       [-0.2844282],
       [-0.2054472],
       [-0.1613528],
       [-0.1340892]])

>>> u1 / u0
array([[ 0.99999195],
       [ 1.00026176],
       [ 0.99842601],
       [ 1.00302712],
       [ 0.99820682]])

>>> s1
array([[   1.3877308],
       [ -35.7756354],
       [ 153.7403826],
       [-233.496746 ],
       [ 114.2981532]])

>>> s1 / s0
array([[   -2.83209096],
       [  125.78090147],
       [ -748.32065172],
       [ 1447.1192691 ],
       [ -852.40387145]])
```

Le problème n'est pas dû au calcul numérique : il est inhérent à la matrice M.

Étant donné un vecteur x, on peut définir sa norme de plusieurs façons. Pour fixer les idées, pour $x = (x_1, \ldots, x_n) \in \mathbb{R}^{\mathbb{N}}$, posons

$$\|x\| = \sqrt{x_1^2 + \ldots + x_n^2}$$

(appelée *norme 2*)

Étant donnés deux vecteurs $x \neq 0$ et x', on dit que l'erreur relative commise en prenant x' au lieu de x est $\frac{\|x-x'\|}{\|x\|}$.

On dit que M^{-1} est mal conditionnée car une petite erreur relative sur x peut se traduire par une grande erreur relative sur $M^{-1}x$.

Plus précisément, en posant

$$E_M = \left\{ \frac{\|Mx\|}{\|x\|} \;\middle|\; x \neq 0 \right\} = \{\, \|Mx\| \mid \|x\| = 1 \,\}$$

Le *conditionnement* c_M de la matrice M est la valeur

$$c_M = \frac{\sup E_M}{\inf E_M} = \frac{\max E_M}{\min E_M}$$

NB : on a aussi

$$c_M = \max E_M \times \max E_{M^{-1}} = c_{M^{-1}}$$

(et $\max E_M$ est souvent noté $|||M|||$). Alors, pour tout $x \neq 0$ et tout x' :

$$\frac{\|Mx - Mx'\|}{\|Mx\|} \leq c_M \times \frac{\|x - x'\|}{\|x\|}$$

et cette borne est serrée :

$$\exists x \neq 0 \exists x' \frac{\|Mx - Mx'\|}{\|Mx\|} = c_M \times \frac{\|x - x'\|}{\|x\|}$$

Donc si le conditionnement de M (donc de M^{-1}) est grand, une petite erreur relative sur x peut donner une grande erreur relative sur $M^{-1}x$.

6 Remarques sur l'algorithme

6.1 Résolution simultanée de plusieurs systèmes

(i) On peut résoudre plusieurs systèmes $Ax = b_1$, $Ax = b_2$, … $Ax = b_p$ en utilisant l'algorithme avec un b matriciel dont les colonnes sont b_1, …, b_p.

(ii) Si on prend pour membre droit I_n, alors le x trouvé sera A^{-1}.

6.2 Décomposition dite «LU»

Les différentes opérations effectuées sur le système consistent à le transformer en un système $UX = b'$ où U est triangulaire supérieure.

Ces opérations peuvent être exprimées comme des multiplications successives de la matrice A et de la matrice b à gauche par des matrices.

Ces matrices sont de deux types :

(i) Des matrices de permutation.

(ii) Des matrices de *transvection*.

6.2.1 Matrices de permutation

Déf $P_\sigma = (p_{ij})_{(i,j)\in[\![1,n]\!]^2}$ matrice de permutation associée à une permutation $\sigma : [\![1,n]\!] \to [\![1,n]\!]$ si $\forall (i,j) \in [\![1,n]\!] \quad m_{ij} = \delta_{i,\sigma(j)}$.

Prop P matrice de permutation si et seulement si P est carrée, ne contient que des 0 et des 1 avec exactement un 1 par ligne et par colonne.

Prop P de taille n matrice de permutation si et seulement si pour tout matrice M de taille $n \times p$, PM est la matrice M à permutation près des lignes (cette permutation étant celle associée à P : la ligne i de M est envoyée en $\sigma(i)$).

Rem $P_\sigma \circ P_{\sigma'} = P_{\sigma\circ\sigma'}$.

Rem P_σ inversible et $(P_\sigma)^{-1} = P_{\sigma^{-1}}$.

6.2.2 Matrices de transvection

Exemple :

$$T_\lambda = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \lambda & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Soit B matrice ayant 4 lignes L_1, L_2, L_3, L_4.

TB a les mêmes lignes que B sauf la 3^{e} ligne qui est $L_3 + \lambda L_1$.

Définition : matrice s'écrivant I_n plus une matrice avec un seul coefficient non-nul, situé sous la diagonale.

Rem :

(i) On a $T_\lambda T_\mu = T_{\lambda+\mu}$.

(ii) Donc T_λ inversible et $T_\lambda^{-1} = T_{-\lambda}$.

6.2.3 Décomposition PLU

La méthode du pivot consiste à obtenir une matrice U triangulaire supérieure en appliquant, en multipliant A à gauche par

– une matrice de permutation S_1 ;
– puis un produit de matrices de transvection G_1 ;
– ...
– une matrice de permutation S_n ;
– un produit de matrices de transvection G_n.

$$G_n S_n \cdots G_1 S_1 A = U$$

D'où :

$$A = S_1^{-1} G_1^{-1} \cdots S_n^{-1} G_n^{-1} U$$

On peut montrer qu'on peut simplifier ça en

$$A = PLU$$

où L est une triangulaire inférieure, P matrice de permutation. [1]

(On écrit parfois $P^{-1}A = LU$)

6.2.4 Algorithme

En pratique, on peut construire explicitement la matrice L et la permutation associée à P ou lors de l'exécution de l'algorithme du pivot et les retourner.

6.2.5 Avertissement

On parle de décomposition LU mais certaines matrices (bien qu'inversibles) n'admettent pas de décomposition LU. Par exemple

$$\begin{pmatrix} 0 & 2 \\ 7 & 8 \end{pmatrix}$$

n'en admet pas. Exercice : pourquoi ? (indication : s'intéresser au coefficient 0)

6.2.6 Applications

Résolution de $Ax = b$ On résout $PLUx = b$, en résolvant

$Py = b$ par application de la permutation réciproque aux lignes de b ;

$Lz = y$ qui est triangulaire ;

$Ux = z$ qui est triangulaire.

Calcul de l'inverse On prend $b = I_n$ et on résout $Ax = b$.

Calcul du déterminant On a $\det A = \det P \times \det L \times \det U$ et

– $\det P$ est la signature de la permutation associée à P

– $\det L$ et $\det U$ sont le produit des éléments diagonaux.

NB : En pratique, on peut souvent se passer de calculer explicitement L, U et P.

1. U pour «Upper», L pour «Lower».

7 Méthode des moindres carrés

7.1 Cadre

En physique ou en économie, il est fréquent qu'on soit dans la situation suivante :

(i) On sait/conjecture qu'une valeur b dépend linéairement de paramètres $(\alpha_1, \ldots, \alpha_q)$: il existe des constantes C_1, …, C_q telles que pour toute valeur des paramètres et toute valeur de b associée

$$b = \sum_{k=1}^{q} C_k \alpha_k = \begin{pmatrix} \alpha_1 & \cdots & \alpha_q \end{pmatrix} \times \begin{pmatrix} C_1 \\ \vdots \\ C_q \end{pmatrix}$$

(ii) On a mesuré des valeurs b_1, …, b_p et $(a_{11}, \ldots, a_{1q})$, …, $(a_{p1}, \ldots, a_{pq})$ les valeurs associées des paramètres.

(iii) On aimerait connaître C_1, …, C_q.

On a :

- nombre de mesures : p ;
- nombre de paramètres : q.

On prend $p \geq q$ (sinon on n'arrivera pas à connaître tous les paramètres) et même $p > q$ pour voir si ce modèle linéaire est réaliste, pour essayer de réduire l'effet des erreurs (aléatoires) de mesure.

Il s'agit de résoudre :

$$\begin{pmatrix} a_{11} & \cdots & a_{1q} \\ \vdots & & \vdots \\ a_{p1} & \cdots & a_{pq} \end{pmatrix} \times \begin{pmatrix} C_1 \\ \vdots \\ C_q \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_p \end{pmatrix}$$

d'inconnues C_1, …, C_q.

Problèmes :

(i) Erreurs de mesure : pas de solution.

(ii) La modélisation linéaire n'est en général qu'une approximation.

Pour s'en sortir :

Ne pas chercher à *résoudre*

$$Ax - b$$

mais plutôt à *minimiser*

$$\|Ax - b\|$$

On prendra la norme 2 (c'est le plus facile).

Donc on minimise le carré des écarts de Ax à b.

D'où le nom : «Méthode(s) des moindres carrés».

7.2 Matrices orthogonales

On dira qu'une matrice $M \in \mathcal{M}_{nm}(\mathbb{R})$ est *orthogonale* si l'application

$$\begin{array}{rcl} \mathbb{R}^m & \to & \mathbb{R}^n \\ x & \mapsto & Mx \end{array}$$

préserve la norme 2, i.e. $\forall x \in R^n \quad \|Mx\| = \|M\|$

Toute matrice orthogonale :

(i) préserve les produits scalaires car $M(\vec{u} + \vec{v}) = M\vec{u} + M\vec{v}$ et

$$\vec{u} \cdot \vec{v} = \frac{1}{2}\left(\|\vec{u} + \vec{v}\|^2 - \|\vec{u}\| - \|\vec{v}\|\right)$$

(ii) préserve les angles car

$$\cos(\vec{u}, \vec{v}) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \times \|\vec{v}\|}$$

7.3 Pseudo-inverse

Étant donné une matrice $A \in \mathcal{M}_{pq}(\mathbb{R})$, on peut définir son *pseudo-inverse de Moore-Penrose* A^+ (même si A non inversible et même si A non carrée).

C'est l'unique matrice vérifiant :

(i) $A^+ \in \mathcal{M}_{pq}(\mathbb{R})$

(ii) $AA^+A = A$

(iii) $A^+AA^+ = A^+$

(iv) AA^+ est une matrice orthogonale

(v) A^+A est une matrice orthogonale

7.4 Moindres carrés

L'ensemble des vecteurs x minimisant $\|Ax - b\|$ est le sous-espace affine $A^+b + \operatorname{Ker} A$.
Et parmi eux, A^+b est le vecteur de norme minimale.

7.5 Calcul du pseudo-inverse

Il existe plusieurs méthodes de calculs. Les plus utilisées passent par une décomposition implicite ou explicite de la matrice A :

Décomposition QR A est mis sous la forme QR où R triangulaire supérieure et Q orthogonale.

Décomposition en valeurs singulières[2] A est mis sous la forme $U\Sigma V$ où U et V orthogonales et Σ diagonale à coefficients positifs ou nuls.

8 Regard critique

8.1 Résumons

On travaille sur une matrice carrée A de taille n.

(i) On veut parfois des valeurs de n élevées : plusieurs milliers.

(ii) La complexité de ces méthodes est en général de l'ordre de la méthode du pivot : $\Theta(n^3)$.

(iii) C'est l'ordre de grandeur du coût de la multiplication naïve.

Peut-on faire mieux ?

- oui, il y a des algorithmes un peu plus efficaces pour obtenir les mêmes décomposition (mais c'est délicat).
- mais bien souvent en informatique, il vaut mieux se poser la question de la pertinence de ce qu'on fait.

8.2 Cadre physique

(i) interactions seulement entre des points proches

(ii) donc la matrice A est creuse (une majorité de zéros).

On peut améliorer la représentation des matrices et les algorithmes d'addition et de multiplication pour en tirer parti.
Problème : les méthodes de décomposition ont tendance à remplir les matrices.

8.3 Méthodes plus adaptées aux matrices creuses

Pour résoudre $Ax = b$ pour une matrice creuse,
Utilisation de méthodes itératives pour les matrices creuses :

(i) on part d'une solution approchée ;

(ii) on l'améliore ;

(iii) revenir au point précédent tant que nécessaire.

Intérêt de ces méthodes :

(i) Pas besoin de calculer de nouvelles matrices (on reste sur des matrices creuses) ;

(ii) On améliore peu à peu une solution : impact des erreurs de calcul a priori plus faible.