Algorithmique numérique

Résolution de systèmes linéaires

Gilles Marait & The IS104 team

12/04/2024

Plan du cours

I Méthodes de calcul numérique

II Résolution de systèmes linéaires

III Calcul des éléments propres

IV Équations non linéaires

V Méthodes d'interpolation et d'intégration

VI Équations différentielles

Plan

1 Résolution de systèmes linéaires

Plan

1. Résolution de systèmes linéaires
   - Présentation du problème
   - Méthodes directes
   - Méthodes itératives
   - Application à l'équation de la chaleur
   - Matrices creuses

Présentation du problème

Problème : trouver le vecteur x tel que :

$$A.x = b$$

avec,

- A, matrice carrée inversible $n \times n$
- b, vecteur de dimension n
- x, vecteur solution de dimension n

Remarque

A inversible donc unicité de la solution $x = A^{-1}b$

Présentation du problème

$$A.x = b$$

Solution naïve (1/2)

Inverser la matrice A pour calculer $A^{-1}b = x$.

Problème d'inversion

Équivalent à résoudre les systèmes :

$$A.x = e_i$$

pour i de 1 à n, où, e_i est le i-ème vecteur de base de l'espace vectoriel.

$\Longrightarrow$ Résolution de n systèmes linéaires.

$\Longrightarrow$ Complexité++ !

Formules de Cramer

Exemple avec n = 2

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 = b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 = b_2 \end{cases}$$

Une solution existe si et seulement si :

$$\Delta = \begin{vmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{vmatrix} = a_{1,1}a_{2,2} - a_{1,2}a_{2,1} \neq 0$$

et,

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{1,2} \\ b_2 & a_{2,2} \end{vmatrix}}{\Delta} \qquad x_2 = \frac{\begin{vmatrix} a_{1,1} & b_1 \\ a_{2,1} & b_2 \end{vmatrix}}{\Delta}$$

Formules de Cramer

Solution naïve (2/2)

Utiliser les formules de Cramer

Formule générale

$$x_k = \frac{det(B_k)}{det(A)} \text{ où } B_k = \begin{pmatrix} a_{1,1} & \dots & a_{1,k-1} & b_1 & a_{1,k+1} & \dots & a_{1,n} \\ a_{2,1} & \dots & a_{2,k-1} & b_2 & a_{2,k+1} & \dots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & \dots & a_{n,k-1} & b_n & a_{n,k+1} & \dots & a_{n,n} \end{pmatrix}$$

Complexité ?

Formules de Cramer

Solution naïve (2/2)

Utiliser les formules de Cramer

Complexité

Calcul de $n+1$ déterminants. Complexité du calcul d'un déterminant ? Pour un matrix C de dimensions $n \times n$:

$$det(C) = \sum_{\sigma \in S_n} \epsilon(\sigma) \prod_{i=1}^{n} C_{i,\sigma(i)}$$

où S_n est l'ensemble des permutations de $\{1, ..., n\}$ et $\epsilon(\sigma)$ la signature de la permutation σ.

- $\Theta(n!)$ additions
- $\Theta((n+1)!)$ multiplications

Plan

1. Résolution de systèmes linéaires
   - Présentation du problème
   - Méthodes directes
   - Méthodes itératives
   - Application à l'équation de la chaleur
   - Matrices creuses

Méthodes directes

Soit le système : $A.x = b$

Principe

Transformer le système de manière à se ramener au cas où la matrice est triangulaire. On recherche la matrice M telle que :

$$M.A.x = M.b$$

où $M.A$ est une matrice triangulaire (supérieure ou inférieure).

Algorithmes

- Gauss et Gauss-Jordan
- Factorisation LU
- Factorisation QR

Méthode de Gauss

Johann Carl Friedrich Gauß
1777 - 1855

Historique de la méthode de Gauss

Ier siècle av. J.C. premières références en Chine, dans le livre Les Neuf Chapitres sur l'art mathématique.

1720 redécouverte par *Newton*

1801 utilisation par *Gauss* pour retrouver Céres (méthode des moindres carrés)

1805 formalisation de la méthode par *Legendre*

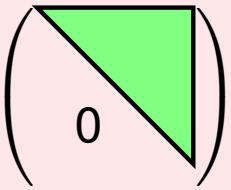
Méthode de Gauss

Passage à un système triangulaire

Itération de deux phases de calcul :

1. Échange de lignes (et/ou colonnes)
2. Éliminations de variables

Résolution triangulaire (remontée)



$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b'_1 \\ \vdots \\ b'_n \end{pmatrix}$$

$$T \qquad x \quad = \quad b'$$

où,

$$x_i = \frac{1}{t_{i,i}} \left(b'_i - \sum_{j=i+1}^{n} t_{i,j} x_j \right)$$

Méthode de Gauss - échange de lignes

A est inversible, donc il existe k tel que $A_{k,1} \neq 0$. On échange les lignes 1 et k pour éliminer la première variable.

On multiplie A à gauche par P_1 (pas stockée en pratique) :

$$P_1 = \begin{array}{c} \\ \\ \\ \\ k \rightarrow \\ \\ \\ \\ \end{array} \begin{pmatrix} 0 & \cdots & \cdots & 0 & 1 & & & \\ \vdots & 1 & & & 0 & & & \\ \vdots & & \ddots & & \vdots & & & \\ 0 & & & 1 & \vdots & & & \\ 1 & 0 & \cdots & \cdots & 0 & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{pmatrix}$$

$$A_1 x = b_1 \text{ avec } \begin{cases} A_1 = P_1 A \\ b_1 = P_1 b \end{cases}$$

Méthode de Gauss - élimination de variable

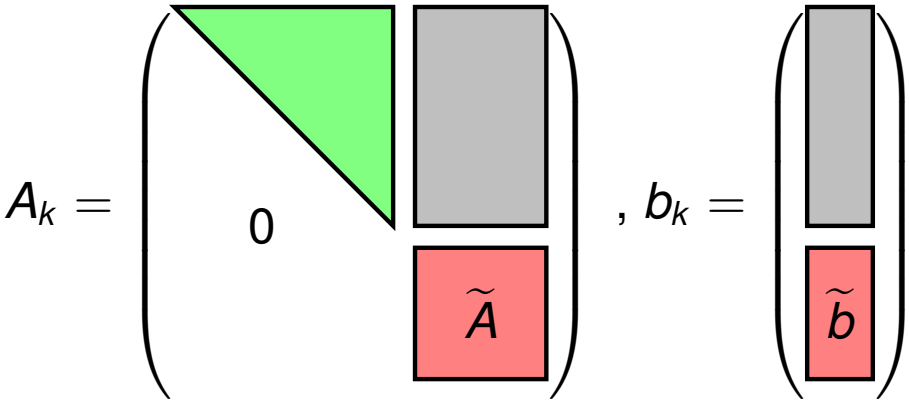
On multiplie A à gauche par la matrice d'élimination E_1 :

$$E_1 = \begin{pmatrix} 1 & & & & & \\ -\frac{a_{2,1}}{a_{1,1}} & 1 & & & & \\ \vdots & & \ddots & & & \\ -\frac{a_{k,1}}{a_{1,1}} & & & \ddots & & \\ \vdots & & & & \ddots & \\ -\frac{a_{n,1}}{a_{1,1}} & & & & & 1 \end{pmatrix}$$

$$A_2 x = b_2 \text{ avec } \begin{cases} A_2 = E_1 A_1 \\ b_2 = E_1 b_1 \end{cases}$$

La première colonne de A_2 est $\begin{pmatrix} a_{1,1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$.

Méthode de Gauss - récurrence



$$\begin{cases} A_{k+1} = E_{k+1} P_{k+1} A_k \\ b_{k+1} = E_{k+1} P_{k+1} b_k \end{cases}$$

Réduction de dimmensionnalité du problème : $\widetilde{A}\widetilde{x} = \widetilde{b}$

Convergence : dimension de $\widetilde{A}$ après $n-1$ itérations $= 1 \times 1$.

$A_n = M.A$, avec $M = E_{n-1} P_{n-1} \dots E_1 P_1$, est triangulaire supérieure.

Méthode de Gauss

Remarque 1

Par définition, on a :

- $det(E_k) = 1$
- $det(P_k) = \pm 1$ (dépend du nombre de permutations)

Donc $det(A) = \pm det(A_n)$. Or, A_n est triangulaire supérieure, donc,

$$det(A_n) = \prod_{i=1}^{n} a_{i,i}$$

Méthode de Gauss

Remarque 2

La stabilité dépend de la stratégie du choix du pivot :

- pivot immédiat : premier élément non nul
- pivot partiel : plus grand en norme dans la colonne
- pivot total : plus grand norme de la matrice entière

En général, on utilise le pivot partiel, car le *pivot immédiat* est trop instable, et le *pivot total* trop coûteux (nécessite la permutation de colonnes).

Méthode de Gauss

Remarque 3 - optimisation des calculs

On applique les mêmes opérations sur A et b. On peut donc travailler sur A à laquelle on adjoint la colonne b comme dernière colonne.

On travaille alors sur $(A|b)$, et on calcule $M(A|b) = (MA|Mb)$.

On améliore la *localité* des données et on simplifie le code !

Méthode de Gauss - Jordan

Remarque 4 - problème d'inversion

Si on veut calculer explicitement un inverse, on résout le système $AX = I$ avec la méthode de Gauss, en utilisant la remarque précédente :

$$M(A|I) \to (MA|MI)$$

On peut donc réaliser l'inversion matricielle sans changer la complexité (à un facteur multiplicatif près).

Besoin d'inversion

Attention : il ne faut pas calculer explicitement un inverse inutilement !

En effet, quand on voit apparaître la formule $A^{-1}u$ avec u un vecteur, il faut résoudre $Ax = u$ et **ne surtout pas** inverser A ! C'est beaucoup plus coûteux et moins stable.

Méthode de Gauss - complexité

	Additions	Multiplications	Échanges
P_k	-	-	$(n-k+1)$
E_k	$(n-k+2)$ $\times(n-k)$	$(n-k+2)$ $\times(n-k)$	-
Remontée k	$(n-k)$	$(n-k+1)$	-
Total	$\mathcal{O}(n^3/3)$	$\mathcal{O}(n^3/3)$	$\mathcal{O}(n^2/3)$

Complexité : $\mathcal{O}(n^3)$

Décomposition LU

Principe

Factoriser une matrix A :

$$A = P.L.U$$

avec,

- P : une matrice de permutation
- L : une matrice triangulaire inférieure, avec des 1 sur la diagonale
- U : une matrice triangulaire supérieure

En anglais :

- Triangulaire inférieure : Lower triangular
- Triangulaire supérieure : Upper triangular

Décomposition LU

Retour à la méthode de Gauss

On avait transformé A en A_n triangulaire supérieure :

$$E_{n-1}P_{n-1}\dots E_1P_1A = A_n$$

Idée : A_n est triangulaire supérieure comme U, donc pour avoir $A = PLU$:

$$\underbrace{E_{n-1}P_{n-1}\dots E_1P_1}_{L^{-1}P^{-1}?}A = \underbrace{A_n}_{U}$$

Peut-on réorganiser les E_k et P_k pour faire apparaître P et L ?

Décomposition LU

Conjugaison par une matrice de permutation

Soit P une matrice d'échange de lignes (i, j) et L_k une matrice triangulaire inférieure ne comportant que des 1 sur la diagonale et des coefficients extra-diagonaux non nuls que sur la k^{ime} colonne.

Alors si $i > k$ et $j > k$:

$$P.L_k = \tilde{L}_k.P$$

où $\tilde{L}_k$ est triangulaire inférieure, ne comportant que des 1 sur la diagonale, et ses coefficients extra-diagonaux non-nuls sont ceux de L_k dans lesquels on a échangé la ligne i et la ligne j.

Décomposition LU

Example (Conjugaison par une matrice de permutation)

Soit P une matrice d'échange de lignes ($i = 3, j = 4$), et $L_{k=1}$ une matrice triangulaire inférieure telles que :

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, L_{k=1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 0 & 0 & 1 \end{pmatrix}$$

On a $i > k$ et $j > k$, donc :

$$\tilde{L}_{k=1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 \end{pmatrix}$$

où $P.L_{k=1} = \tilde{L}_{k=1}.P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 0 & 0 & 1 \\ 3 & 0 & 1 & 0 \end{pmatrix}$

Décomposition LU

Construction de la décomposition PLU

Les matrices P_i ne font qu'échanger une ligne i avec une ligne $> i$. Donc on peut appliquer le lemme précédent à toute matrice E_i pour $j < i$.

$$\begin{aligned} A_n &= E_{n-1}P_{n-1}\dots E_1P_1A \\ &= E_{n-1}(P_{n-1}.E_{n-2})\dots(P_2.E_1).P_1.A \\ &= \tilde{E}_{n-1}\tilde{E}_1\dots P_{n-1}\dots P_1.A \\ \hline U &= L^{-1}.P^{-1}A \end{aligned}$$

$$L^{-1} = \tilde{E}_{n-1}\dots\tilde{E}_1$$

$$L = \tilde{E}_1^{-1}\dots\tilde{E}_{n-1}^{-1}$$

où $\tilde{E}_k^{-1}$ est une matrice triangulaire inférieure dans laquelle les éléments extra-diagonaux sont les opposés des éléments extra-diagonaux de E_k.

Décomposition LU - algorithme

Entrée A

Sortie L, U

Algorithme

- $U \leftarrow A$
- $L \leftarrow Id$
- **Pour** $k = 1, n$
- Pivotage des lignes
- **Pour** $j = k + 1, n$
- $mul \leftarrow U_{j,k} / U_{k,k}$
- $L_{j,k} \leftarrow mul$
- **Pour** $i = k, n$
- $U_{j,i} \leftarrow U_{j,i} - mul \times U_{k,j}$
- **Retourner** L, U

Décomposition LU - complexité

Nombre d'échanges	$\leq n+1$
Nombre de multiplications	$(n-k)$ pour L, $(n-k)(n-k+1)$ pour U
Nombre de soustractions	$(n-k)(n-k+1)$ pour U

Complexité ? Toujours $\mathcal{O}(n^3)$ (ce qui est rassurant). En évaluant la constante :

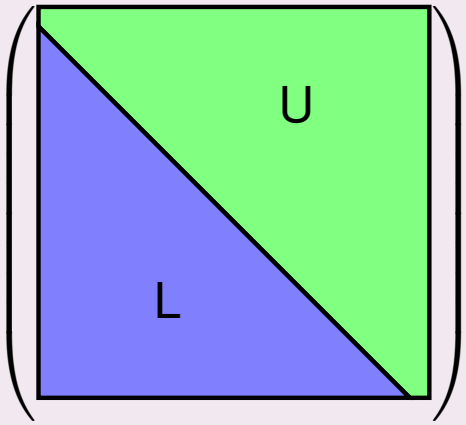
$$\text{Complexité : } \mathcal{O}(\tfrac{2}{3}n^3)$$

Décomposition LU

Remarque : stockage

Comme U est triangulaire supérieure et L triangulaire inférieure avec des 1 sur la diagonale, on peut stocker la décomposition LU dans une matrice $n \times n$.

$A \rightarrow$



Résolution d'un système avec LU

Pour résoudre le système $Ax = b$, on procède ainsi :

- On fait la décomposition LU : $\mathcal{O}(n^3)$
- On a alors $L.U.x = b$, on fait une résolution triangulaire $L.y = b$: $\mathcal{O}(n^2)$
- Puis une seconde résolution triangulaire $U.x = y$: $\mathcal{O}(n^2)$

Remarque

Si on veut résoudre des systèmes avec des seconds membres successifs $b_1, b_2, \ldots$, on fait alors une fois la décomposition LU (de plus grande complexité), puis on n'a plus qu'à résoudre les systèmes triangulaires les uns après les autres.

Remarque

On peut trouver des matrices sur lesquelles la factorisation LU est *instable*, mais on ne rencontre jamais de telles matrices dans les problèmes scientifiques réels.

Factorisation de Cholesky

Factorisation de Cholesky

Dans le cas où A est symétrique définie positive (SDP, ou *SPD* en anglais), on peut faire mieux en utilisant la décomposition de Cholesky :

$$A \to L^t L$$

où L est une matrice triangulaire inférieure.

André-Louis Cholesky (1875 - 1918)

Officier de l'armée française, il est le premier à formaliser la méthode (en 1910).

Factorisation de Cholesky

Factorisation de Cholesky

Idée : si A est SDP, prenons la décomposition LU de A.

$$A = L.U$$

Les coefficients diagonaux de U sont strictement positifs (car A est SDP). Prenons :

- $D = diag(U)$
- $\sqrt{D}$ la matrice diagonale dont les coefficients sont les racines carrées de ceux de D
- $V = D^{-1}U$
  On peut alors écrire :

$$A = L.D.V = (L\sqrt{D})(\sqrt{D}V)$$

$${}^tA = A = ({}^tV.\sqrt{D})(\sqrt{D}.{}^tL), \text{ d'où } V = {}^tL$$

Donc en prenant $\mathcal{L} = L.\sqrt{D}$, on a $A = \mathcal{L}^t\mathcal{L}$.

Factorisation de Cholesky

Remarque 1

Ce n'est évidemment pas à partir de la décomposition LU qu'on calcule la factorisation Cholesky (cf. projet).

Remarque 2

Contrairement à LU, on peut démontrer que cette factorisation est toujours stable.

Remarque 3

Si A est seulement symétrique inversible, mais pas SDP, il existe une décomposition $A = L.D.{}^tL$, où D est une matrice diagonale, et L triangulaire inférieure avec des 1 sur la diagonale.

Remarque 4

Une matrice A est SDP si et seulement si elle admet une factorisation de Cholesky.

Décomposition QR

Idée

Rarement utilisé pour résoudre un système linéaire, elle est utile pour les algorithmes avancés (cf. projet 3) ou la méthode des moindres carrés. Elle fait apparaître une matrice orthogonale Q.

La décomposition QR peut se faire sur une matrice rectangulaire. Prenons A de dimensions $m \times n$.

Factorisation QR pleine

$$\underset{(m,n)}{A} = \underset{(m,m)}{Q} \; \underset{(m,n)}{R}$$

- Q est une matrice orthogonale
- R est une matrice triangulaire supérieure

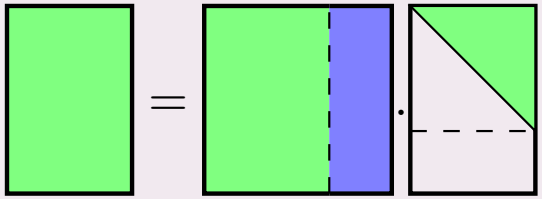
Décomposition QR

Factorisation QR pleine

$$\underset{(m,n)}{A} = \underset{(m,m)}{Q} \; \underset{(m,n)}{R}$$

- Q est une matrice orthogonale
- R est une matrice triangulaire supérieure

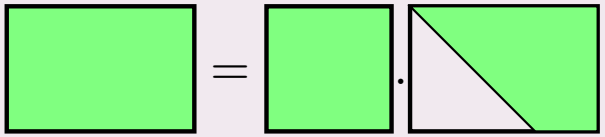
$m \geq n$:



$A \quad Q \quad R$

Q a des colonnes "inutiles" (multipliées par des 0 de R, elles permettent juste de garder la structure orthogonale de Q).

$m \leq n$:



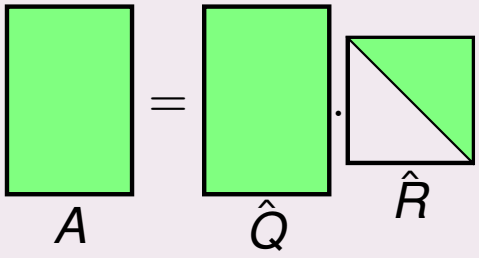
$A \quad Q \quad R$

Décomposition QR

Factorisation QR réduite

$$\underset{(m,n)}{A} = \underset{(m,n)}{\hat{Q}} \; \underset{(n,n)}{\hat{R}}$$

- $\hat{Q}$ est une matrice à colonnes orthonormales
- $\hat{R}$ est une matrice triangulaire supérieure



On n'a plus les colonnes "inutiles" de la décomposition pleine, mais $\hat{Q}$ n'est plus une matrice orthogonale (car pas carrée).

Décomposition QR

La matrice Q est orthogonale.

Rappels

- Chaque ligne / colonne est de norme 1, (on parle aussi de matrice unitaire, pour les complexes)
- Deux lignes / colonnes distinctes de Q sont orthogonales entre elles
- $\boxed{Q^t Q = Id}$, autrement dit $Q^{-1} = {}^tQ$
- Le conditionnement $\kappa(Q)$ vaut 1, donc multiplier par Q n'est pas source supplémentaire d'instabilité numérique.

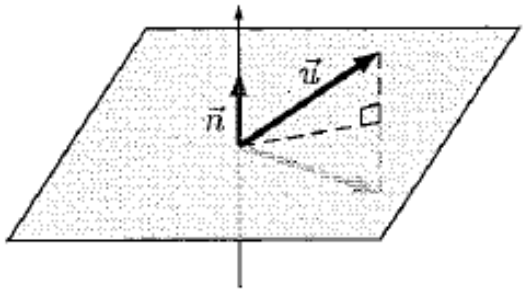
Construction de Q et R

- Orthogonalisation de Gram-Schmidt
- Utilisation de rotations successives : rotations de Givens
- Utilisation de réflexions successives : matrices de Householder

Dans la suite, on va détailler la méthode utilisant des matrices de Householder.

Matrice de Householder

Les matrices de Householder représentent une symétrie orthogonale par rapport à un hyperplan.



On prend $\vec{n}$ un vecteur unitaire normal à l'hyperplan. Pour un vecteur quelconque $\vec{u}$, l'opérateur $\mathcal{H}$ de Householder :

$$\mathcal{H}(\vec{u}) = \vec{u} - 2(\vec{u} \cdot \vec{n})\vec{n}$$

En écriture matricielle, la matrice de Householder H :

$$\begin{array}{rcll} Hu & = & u - 2({}^t un)n & \\ & = & u - 2n({}^t un) & \leftarrow \quad \textit{on déplace le scalaire} \\ & = & u - 2n({}^t nu) & \leftarrow \quad \textit{symétrie du produit scalaire} \\ & = & u - 2(n^t n)u & \leftarrow \quad \textit{associativité} \\ & = & (Id - 2n^t n)u & \end{array}$$

D'où

$$\boxed{H = Id - 2n^t n}$$

Matrice de Householder

$$\boxed{H = Id - 2n^t n}$$

Propriétés

- H symétrique : ${}^tH = H$
- H auto-inverse (logique pour une symétrie), vérification :

$$\begin{aligned} HH &= (Id - 2n^t n)(Id - 2n^t n) \\ &= Id - 4n^t n + 4n \underbrace{{}^t n n}_{1} {}^t n \\ &= Id \end{aligned}$$

- H est bien une matrice orthogonale

Matrice de Householder

Choix de l'hyperplan

On souhaite choisir n de manière à avoir une matrice $H_{u,v}$ qui projette un vecteur u sur un vecteur v de même norme :

$$H_{u,v}u = v$$

On a donc :

$$\begin{array}{lrcl} H_{u,v}u & u - 2({}^t un)n & = & v \\ \text{donc} & u - v & = & 2({}^t un)n \end{array}$$

$$\begin{cases} \text{Si } {}^t u.n = 0, u = v, H = Id \text{ convient.} \\ \text{Sinon, } n \text{ colinéaire à } u - v\text{, donc } \boxed{n = \pm \dfrac{u - v}{||u - v||}} \text{ pour } H \end{cases}$$

Calcul de la décomposition QR

Pour calculer la décomposition QR de la matrice A, prenons

- $A_0 = A$
- u la première colonne de A
- $v = \begin{pmatrix} ||u|| \\ 0 \\ \vdots \\ 0 \end{pmatrix}$

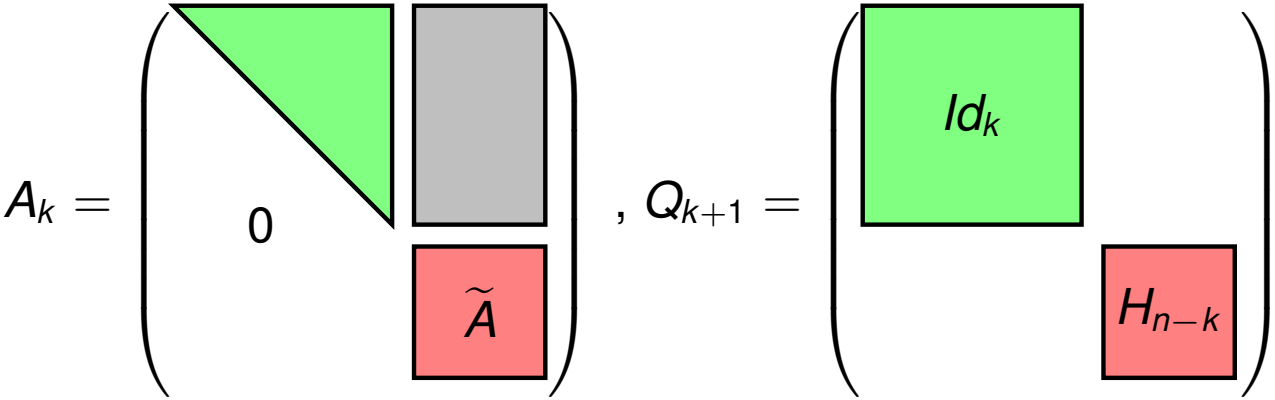
On prend alors

$$\begin{cases} Q_1 = H_{u,v} \\ A_1 = Q_1 A_0 \end{cases}$$

La première colonne de A_1 est donc v.

Calcul de la décomposition QR

On peut alors répéter le processus. À l'étape k :



$$A_n = (Q_n Q_{n-1} \dots Q_1) A$$

et les Q_k sont auto-inverses, donc :

$$A = \underbrace{Q_1 \dots Q_n}_{Q} \underbrace{A_n}_{R}$$

Décomposition QR

Complexité

Pour une matrice $m \times n$: $\mathcal{O}(\frac{4}{3}(n \times m \times min(n, m)))$, donc pour une matrice $n \times n$: $\mathcal{O}(\frac{4}{3}n^3)$.

Résolution du système

Si A carrée, pour résoudre $Ax = b$:

- décomposition QR : $QRx = b$
- calcul de $y = {}^tQb$: $Rx = {}^tQb = y$
- résolution triangulaire $Rx = y$

Méthode des moindres carrés

On considère le système $Ax = b$, avec A une matrice $m \times n$, où $m \gg n$. Cette fois-ci, on a plus d'équations que d'inconnues, donc aucune solution exacte. On cherche alors à minimiser les carrés des erreurs :

$$x = argmin(||b - Ax||^2)$$

Régression linéaire

En 2D, on peut penser à la régression linéaire de m points $(x_1, y_1), \ldots, (x_m, y_m)$:

$$A = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{pmatrix}, b = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

de solution $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$. La solution est alors la droite $D(x) = \alpha x + \beta$.

Résolution de la méthode des moindres carrés

Trois méthode de résolution, de la moins stable à la plus stable :

Factorisation de Cholesky

La première utilisée historiquement :

- Calcul de ${}^tAA$ et ${}^tAb$
- Factorisation de Cholesky ${}^tAA = R^tR$
- Résolution triangulaire ${}^tRw = {}^tAb$ pour w
- Résolution triangulaire $Rx = w$ pour x

Factorisation QR

La plus utilisée en pratique :

- Calcul de la décomposition réduite $A = \hat{Q}\hat{R}$
- Calcul du vecteur $y = {}^t\hat{Q}b$
- Résolution triangulaire $Rx = y$ pour x

SVD

Voir chapitre 3. Méthode coûteuse, à utiliser pour des problèmes difficiles.

Plan

1. Résolution de systèmes linéaires
   - Présentation du problème
   - Méthodes directes
   - Méthodes itératives
   - Application à l'équation de la chaleur
   - Matrices creuses

Directe VS itératif

Directe

Jusqu'à maintenant, on a vu des méthodes directes :

- nombre d'étapes de calcul fixe pour obtenir le résultat
- on peut espérer avoir une solution de l'ordre de la précision machine si le problème est bien conditionné

Itératif

Voyons maintenant les méthodes itératives :

- on part d'une solution initiale x_0, puis on applique des opérations pour calculer x_1 plus proche de la solution, et on répète le processus
- il faut choisir quand s'arrêter, pouvoir évaluer quand la solution est satisfaisante
- pas de garantie de convergence vers la solution
- plutôt adaptés aux très grands systèmes

Gauss

Gauss écrit à son étudiant Gerling, à propos d'une méthode itérative qu'il utilise, le 26 Décembre 1823 :

"[...] Je te recommande également cette méthode. Tu n'utiliseras plus jamais l'élimination directe, du moins quand tu as plus que 2 [inconnues]. La procédure indirecte peut être faite en étant à moitié endormi, ou en pensant à d'autres choses."

(traduction approximative d'une traduction anglaise réalisée par mes soins)

Méthodes de point fixe

On va étudier un type de méthodes itératives assez simple : les méthodes de point fixe.

Principe de la méthode

On cherche la solution $x \in \mathbb{R}^n$ de $x = Ux + C$. On va appliquer une méthode de point fixe sur f :

$$f : x \mapsto Ux + C$$

On calcule la suite de vecteur (x_n) :

$$\begin{cases} x_0 & \text{fixé (choisi arbitrairement)} \\ x_{n+1} & = f(x_n) \end{cases}$$

On a alors :

$$x_{n+1} - x = (Ux_n + C) - (Ux + C) = U(x_n - x)$$
$$x_n - x = U^n(x_0 - x)$$

Convergence si et seulement si $\|U\| < 1$ (pour au moins une norme matricielle donnée).

Description des algorithmes

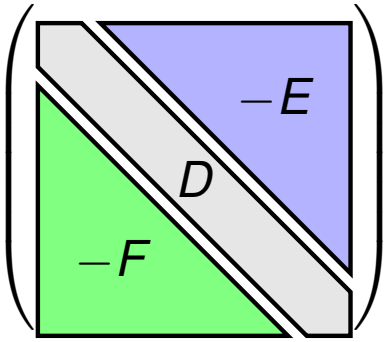
Reprenons le problème initial de la résolution de $A.x = b$.

On écrit : $A = M - N$, avec M (facilement) inversible. $Ax = b$ devient :

$$\begin{aligned} (M-N).x &= b \\ M.x &= N.x + b \\ x &= \underbrace{M^{-1}.N}_{U}.x + \underbrace{M^{-1}.b}_{C} \end{aligned}$$

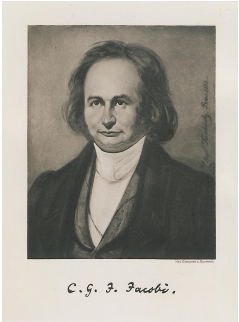
Concrètement, écrivons :

$$A = D - E - F =$$



et supposons D inversible.

Méthode de Jacobi



Charles Gustave Jacob Jacobi (1804 - 1851)

Méthode de Jacobi

On prend $M = D$ et $N = E + F$.
Une itération s'écrit alors :

$$\begin{cases} x_1^{(k+1)} = \frac{1}{a_{1,1}}(& -a_{1,2}x_2^{(k)} & -a_{1,3}x_3^{(k)} & \cdots & -a_{1,n}x_n^{(k)} + b_1) \\ x_2^{(k+1)} = \frac{1}{a_{2,2}}(-a_{2,1}x_1^{(k)} & & -a_{2,3}x_3^{(k)} & \cdots & -a_{2,n}x_n^{(k)} + b_2) \\ \cdots \\ x_n^{(k+1)} = \frac{1}{a_{n,n}}(-a_{n,1}x_1^{(k)} & \cdots & \cdots & -a_{n,n-1}x_{n-1}^{(k)} & +b_n) \end{cases}$$

Avantage : méthode parallèle, on peut calculer chaque $x_i^{(k+1)}$ indépendemment.

Méthode de Gauss-Seidel

Même méthode, mais à chaque étape on utilise les $x_i^{(k+1)}$ déjà calculés.

$$M = D - F \text{ et } N = E$$

$$\begin{cases} x_1^{(k+1)} = \frac{1}{a_{1,1}}(& & -a_{1,2}x_2^{(k)} & -a_{1,3}x_3^{(k)} & \cdots & -a_{1,n}x_n^{(k)} + b_1) \\ x_2^{(k+1)} = \frac{1}{a_{2,2}}(-a_{2,1}x_1^{(k+1)} & & & -a_{2,3}x_3^{(k)} & \cdots & -a_{2,n}x_n^{(k)} + b_2) \\ \cdots \\ x_n^{(k+1)} = \frac{1}{a_{n,n}}(-a_{n,1}x_1^{(k+1)} & \cdots & \cdots & -a_{n,n-1}x_{n-1}^{(k+1)} & & +b_n) \end{cases}$$

$$\boxed{x_{n+1} = D^{-1}(Fx_{n+1} + Ex_n + b)}$$

Convergence plus rapide, mais moins de parallélisme !

Fun fact

- Carl F. Gauss (1777 - 1855)
- Philip Ludwig Von Seidel (1821 - 1896)

La méthode de Gauss-Seidel n'était pas connue de Gauss, et n'était pas recommandée par Seidel.

Méthode de relaxation

Plus généralement, on peut utiliser un paramètre $\omega \in \mathbb{R}^+$:

$$\begin{cases} M = \frac{1}{\omega} D - F \\ N = \frac{1-\omega}{\omega} D + E \end{cases}$$

$$\begin{cases} x_1^{(k+1)} = \frac{1}{a_{1,1}}((1-\omega)a_{1,1}x_1^{(k)} & -\omega a_{1,2}x_2^{(k)} & \cdots & -\omega a_{1,n}x_n^{(k)} & +\omega b_1) \\ x_2^{(k+1)} = \frac{1}{a_{2,2}}(-\omega a_{2,1}x_1^{(k+1)} & +(1-\omega)a_{1,2}x_2^{(k)} & \cdots & -\omega a_{2,n}x_n^{(k)} & +\omega b_2) \\ \cdots \\ x_n^{(k+1)} = \frac{1}{a_{n,n}}(-\omega a_{n,1}x_1^{(k+1)} & \cdots & \cdots & +(1-\omega)a_{n,n-1}x_n^{(k)} & +\omega b_n) \end{cases}$$

$$\boxed{x^{(k+1)} = (1-\omega)x^{(k)} + \omega D^{-1}(Ex_n + Fx_{n+1} + b)}$$

Remarque

La méthode de Gauss-Seidel est la méthode de relaxation avec $\omega = 1$.

Résultats de convergence

Résultat théorique

Convergence si le rayon spectral $\rho(M^{-1}N) < 1$ (pas évident à calculer en pratique).

En pratique

- Si A est à diagonale strictement dominante : Gauss-Seidel et Jacobi convergent
- Si A est SDP : la méthode de relaxation converge $\iff$ $\omega \in]0, 2[$, Gauss-Seidel converge aussi
- Si A est tridiagonale : Gauss-Seidel converge $\iff$ Jacobi converge (et Gauss-Seidel converge plus vite).
- Si A est tridiagonale et SDP : les 3 méthodes convergent et ω est optimal pour :

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho(J)}}$$

où $\rho(J)$ est le rayon spectral pour la méthode de Jacobi.

Autres méthodes itératives

Il existe des centaines de méthodes itératives ! Parmi les plus utilisées, on peut noter la méthode du gradient conjugué et GMRES.

Gradient conjugué

Pour les matrices SDP, cette méthode revient à trouver le minimum d'une forme quadratique :

$$f(z) = \frac{1}{2}\,{}^t z.A.z - {}^t bz$$

dont le gradient est :

$$f'(z) = A.z - b$$

Méthode facile à implanter et efficace (cf. projet)

GMRES

Algorithme plus complexe, utilisable sur toutes les matrices.
Il fait partie de la famille plus vaste des méthodes de Krylov, qui nécessitent le stockage d'une base, que l'on enrichit d'un nouveau vecteur à chaque itération, ce qui peut devenir coûteux en mémoire si la convergence est lente.

Plan

1. Résolution de systèmes linéaires
   - Présentation du problème
   - Méthodes directes
   - Méthodes itératives
   - Application à l'équation de la chaleur
   - Matrices creuses

Équation de la chaleur

Comme application simple, on va modéliser l'équation de la chaleur en régime permanent avec l'équation suivante :

$$\Delta T + f = 0$$

où Δ est l'opérateur Laplacien, T le température, et f une fonction représentant la production de chaleur dans le volume.
En 2 dimensions, on peut réécrire l'équation :

$$\boxed{\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + f(x, y) = 0}$$

Discrétisation spatiale

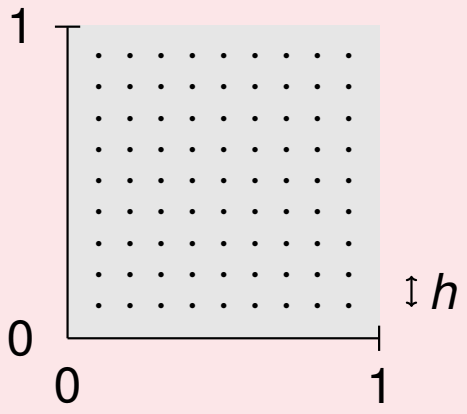
On veut modéliser le problème dans une carré $[0;1] \times [0,1]$. On va devoir discrétiser l'espace. Pour cela, on va utiliser la méthode des différences finies :

Différences finies

On va représenter la température en certains points d'une grille $N \times N$. La température en (x_i, y_j) est notée $t_{i,j}$, i et j étant des entiers allant de 1 à N.

$$t_{i,j} = T(x_i, y_j) \text{ avec } \begin{cases} x_i = \frac{i}{N+1} \\ y_j = \frac{j}{N+1} \end{cases}$$

on note la distance entre les points adjacents $h = \frac{1}{N+1}$.



Discrétisation numérique

Comment évaluer les dérivées partielles de T ? Avec la méthode des différences finies, on utilise le développement de Taylor à l'ordre 2 :

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \ldots + \frac{h^n}{n!}f^{(n)}(x) + o(h^n)$$
$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) + \ldots + \frac{(-1)^n h^n}{n!}f^{(n)}(x) + o(h^n)$$

En soustrayant, on peut ainsi estimer les dérivées partielles :

$$\left(\frac{\partial T}{\partial x}\right)_{i,j} \approx \frac{T(x_i+h,y_j)-T(x_i-h,y_j)}{2h} = \frac{t_{i,j+1}-t_{i,j-1}}{2h}$$
$$\left(\frac{\partial T}{\partial y}\right)_{i,j} \approx \frac{T(x_i,y_j+h)-T(x_i,y_j-h)}{2h} = \frac{t_{i+1,j}-t_{i-1,j}}{2h}$$

En sommant, on obtient pour les dérivées secondes :

$$\left(\frac{\partial^2 T}{\partial x^2}\right)_{i,j} \approx \frac{T(x_i+h,y_j)-T(x_i-h,y_j)-2T(x_i,y_j)}{h^2} = \frac{t_{i,j+1}-t_{i,j-1}}{h^2}$$
$$\left(\frac{\partial^2 T}{\partial y^2}\right)_{i,j} \approx \frac{T(x_i,y_j+h)-T(x_i,y_j-h)-2T(x_i,y_j)}{h^2} = \frac{t_{i+1,j}-t_{i-1,j}}{h^2}$$

Bordeaux INP AQUITAINE

Discrétisation numérique

On représente le champ de température $\bar{T}$ par un vecteur de taille N^2 :

$$\bar{T} = {}^t\left(\underbrace{t_{1,1}; t_{1,2}; \ldots; t_{1,N}}_{\text{ligne 1}}\ \underbrace{t_{2,1}; t_{2,2}; \ldots; t_{2,N}}_{\text{ligne 2}}\quad \cdots \quad \underbrace{t_{N,1}; \ldots; t_{N,N}}_{\text{ligne } N}\right)$$

L'opérateur Laplacien sera alors représenté par une matrice A $N^2 \times N^2$ (cf. projet) et la résolution de l'équation correspondra à la résolution d'un système linéaire :

$$A\bar{T} = \bar{F}$$

où $\bar{F}$ dépendra du flux de chaleur imposé en chaque point.

Plan

1. Résolution de systèmes linéaires
   - Présentation du problème
   - Méthodes directes
   - Méthodes itératives
   - Application à l'équation de la chaleur
   - Matrices creuses

Matrices denses et matrices creuses

Les systèmes d'équations aux dérivées partielles fournissent souvent des matrices comme celle décrite pour l'équation de la chaleur (à écrire dans le projet). On peut noter 2 choses :

- A est de grande dimension ($N^2 \times N^2$ dans notre exemple, mais $N^3 \times N^3$ pour du 3D...)
- A est principalement composée de 0

Il paraît assez inefficace de stocker toute la matrice au format dense, c'est à dire stocker tous les coefficients!

Matrices creuses

Un format de matrice plus adapté est celui des matrices creuses : on ne stocke que les coefficient non-zéro.

Exemple de Format de matrice creuses

Example (Matrice creuse)

Considérons la matrice : $\begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 5 & 6 \\ 0 & 7 & 0 & 8 \end{pmatrix}$

Format coordonnées (COO)

Un des formats les plus simples consiste en une structure de 3 tableaux de taille d le nombre de non-zéros :

- I : indice de ligne du coefficient
- J : indice de colonne du coefficient
- V : valeur du coefficient à cette position

Ainsi, un stockage possible de notre matrice dans ce format :

I	[	1,	1,	2,	2,	3,	3,	4,	4]
J	[	1,	2,	2,	3,	3,	4,	2,	4]
V	[	1.0,	2.0,	3.0,	4.0,	5.0,	6.0,	7.0,	8.0]

Matrices creuse

Il existe une multitude de formats, mais il est assez facile de se convaincre que la complexité des opérations avec de telles matrices devient vite avantageuse lorsque les dimensions augmentent. Elles dépendent du nombre de non-zéros d :

Addition	$A + B$	$\mathcal{O}\left(d_A + d_B\right)$
Produit matrice-vecteur	Ax	$\mathcal{O}(d_A)$
Produit matrice-matrice	AB	$\mathcal{O}(nd_A)$
Transposition	${}^tA$	$\mathcal{O}(d_A)$

Solveurs creux

Le problème de résoudre des très grands systèmes linéaires creux le plus efficacement possible est un grand enjeux scientifique et industriel !

Solveurs de référence

Solveurs denses

Les bibliothèques de référence respectent les interfaces :

- BLAS : *Basic Linear Algebra Subprograms*, pour les opérations de bases (produit scalaire, matrice-vecteur, matrice-matrice).
- LAPACK : *Linear Algebra Package* pour les opérations plus avancées (factorisation LU, QR, calculs de valeurs propres. . .)

Implémentations :

- OpenBLAS : ayant comme principal avantage d'être libre et open-source
- MKL : bibliothèque propriétaire d'*Intel*, la plus performante en théorie

Solveurs creux

Il existe une grande multitude de solveurs creux ayant chacun leur spécificité. Exemple de solveur creux direct : PaStiX (Inria ENSEEIRB), MUMPS (ENSEEIHT), . . .

Références

- Numerical linear algebra, Lloyd N. Trefethen; David Bau, II (plus pédagogique)
- Matrix computations 4th edition, Gene H. Golub; Charles F. Van Loan (plus complet)
- Iterative Methods for Sparse Linear Systems, Yousef Saad (plus orienté méthodes itératives)