Modélisation et résolutions numérique et symbolique de problèmes via les logiciels Maple et MATLAB (MODEL)

Cours n°7 : Calcul matriciel (1/2)

Stef Graillat

Université Pierre et Marie Curie (Paris 6)

Résumé des cours précédents

- Présentation du cours : algorithme numérique et symbolique
- Introduction au logiciel MATLAB
- Introduction à la Symbolic Math Toolbox de MATLAB

Objectifs

On considère des calculs utilisant des matrices denses (c'est à dire avec peu d'éléments nuls)

Nous avons besoin de

1. manipuler efficacement ces matrices sur un ordinateur
2. choisir la décomposition adéquate pour résoudre un problème donné

Problèmes classiques en calcul matriciel

Résolution de systèmes linéaires : étant donné une matrice A de taille $n \times n$ et un vecteur b de taille n, trouver x tel que

$$Ax = b$$

Résolution de problèmes de moindres carrés : étant donné une matrice A de taille $m \times n$ (avec $m > n$) et un vecteur b de taille m, résoudre

$$\min_x \|b - Ax\|$$

Résolution de problèmes aux valeurs propres : étant donné une matrice A de taille n, trouver un vecteur $x \neq 0$ et un scalaire λ tels que

$$Ax = \lambda x$$

Plan du cours

1. Manipulation des matrices
   1. comment sont stockées les matrices en machine ?
   2. outils de base pour la manipulation de matrice : les BLAS
2. Décomposition de matrice et applications
   1. LU
   2. QR
   3. Réduction (diagonalisation, etc.)
   4. SVD (décomposition en valeurs singulières)
3. Logiciels

Bibliographie

- Scientific Computing with Case Studies, D. O'Leary, SIAM, 2009
- Applied Numerical Linear Algebra, J. Demmel, SIAM, 1997
- Numerical Linear Algebra, L. N. Trefethen et D. Bau, SIAM, 1997
- Matrix Computations, G. Golub et C. Van Loan, Johns Hopkins University Press, 1996
- Algèbre linéaire numérique, G. Allaire et S. M. Kaber, Ellipses, 2002
- Introduction à l'analyse numérique matricielle et à l'optimisation, P. G. Ciarlet, Dunod, 2006
- Analyse numérique matricielle appliquée à l'art de l'ingénieur, P. Lascaux et R. Théodor, Dunod, 2004

Notations

- les vecteurs sont des vecteurs colonnes
- les matrices sont notées en majuscule et les vecteurs et scalaires en minuscule
- les éléments d'une matrice A en position (i,j) sont notés a_{ij} ou $A(i,j)$
- I est la matrice identité et e_i le i-ème vecteur unité
- $B = A^T$ signifie que B est la tranposée de A : $b_{ij} = a_{ji}$
- $B = A^*$ signifie que B est la transconjuguée de A : $b_{ij} = \overline{a}_{ji}$
- on utilise aussi la notation MATLAB. Par exemple $A(i:j, k:l)$ représente la sous-matrice de A avec les lignes de i à j et les colonnes de k à l
- une matrice orthogonale U vérifie $U^T U = I$
- une matrice unitaire U vérifie $U^* U = I$
- deux matrices A et B sont dites semblables s'il existe X inversible telle que $B = XAX^{-1}$

Normes vectorielles et matricielles

Définition 1

Une norme vectorielle est une fonction $\|\cdot\| : \mathbb{C}^n \to \mathbb{R}^+$ *vérifiant*

1. $\|x\| = 0$ *ssi* $x = 0$
2. $\|\alpha x\| = |\alpha|\|x\|$ *pour tout* $\alpha \in \mathbb{C}$ *et* $x \in \mathbb{C}^n$
3. $\|x + y\| \leq \|x\| + \|y\|$ *pour tout* $x, y \in \mathbb{C}^n$

Exemple 1

- $\|x\|_1 = |x_1| + \cdots + |x_n| = \sum_{i=1}^{n} |x_i|$
- $\|x\|_2 = (|x_1|^2 + \cdots + |x_n|^2)^{1/2} = (\sum_{i=1}^{n} |x_i|^2)^{1/2}$
- $\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$

Normes vectorielles et matricielles (suite)

Définition 2

Une norme matricielle est une fonction $\|\cdot\| : \mathbb{C}^{m\times n} \to \mathbb{R}^+$ *vérifiant les mêmes propriétés que pour les normes vectorielles*

Exemple 2

Norme subordonnée à une norme vectorielle : $\|A\| = \sup_{x\neq 0} \frac{\|Ax\|}{\|x\|}$

- $\|A\|_1 = \sup_{x\neq 0} \frac{\|Ax\|_1}{\|x\|_1} = \max_{1\leq j\leq n} \sum_{i=1}^{m} |a_{ij}|$
- $\|A\|_\infty = \sup_{x\neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} = \max_{1\leq i\leq m} \sum_{j=1}^{n} |a_{ij}|$
- $\|A\|_2 = \sup_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2}$

Stockage des matrices

Pour programmer efficacement des algorithmes sur les matrices, il est très important de savoir comment sont stockées les matrices en mémoire !

Exemple : produit matrice-vecteur

Étant donnés une matrice A de taille $m \times n$ et un vecteur x de longueur n, on veut calculer $y = Ax$

- Le vecteur y est défini par des produits scalaire entre les lignes de A et x

$$y_i = A(i,:)x$$

```
[m,n]=size(A);
y = zeros(m,1);
for i=1:m,
    for j=1:n,
        y(i) = y(i) + A(i,j)*x(j);
    end
end
```

Stockage des matrices (suite)

- On peut exprimer Ax en utilisant plutôt les colonnes de A

$$Ax = x_1 A(:,1) + x_2 A(:,2) + \cdots + x_n A(:,n)$$

Ce qui revient à calculer la transposée de $(Ax)^T$

```
[m,n]=size(A);
y = zeros(m,1);
for j=1:n,
    for i=1:m,
        y(i) = y(i) + A(i,j)*x(j);
    end
end
```


Stockage des matrices (suite)

Les deux algorithmes font mn multiplications et mn additions !

- L'ordinateur stocke les informations dans des pages mémoire.
- Une partie des informations est stockée dans des caches
- Ce qui ne rentre pas dans le cache est stocké dans la mémoire centrale (plus lente d'accès)
- Enfin ce qui ne tient pas en mémoire centrale est stocké sur le disque dur

Pour utiliser des données, il faut transférer en cache la page contenant les données (par conséquent, d'autres données doivent être effacées du cache). Ces pages étant au mieux dans la mémoire vive (la plupart du temps lorsque les calculs sont raisonnables) ou au pire dans le disque dur (par exemple pour la multiplication de matrices carrées de plusieurs millions de lignes). Pour qu'un algorithme soit efficace, il faut limiter le nombre de fois qu'une page est chargée en cache !

Stockage des matrices (suite)

Question à se poser lorsque l'on veut multiplier une matrice et un vecteur : Quel est l'algorithme que vous allez utiliser pour que l'algorithme soit efficace ?

La vraie question à se poser est en fait :

Est-ce que les matrices sont rangées par ligne ou par colonne ?

Langage	schéma de stockage
C, C++	par ligne
Fortran	par colonne
Java	par ligne
MATLAB	par colonne

Outils de base pour manipuler des matrices : les BLAS

Il y a des taches qui sont effectuées dans presque toutes les opérations matricielles !

Des fonctions ont été développées pour éviter que les programmeurs n'aient à les reprogrammer à chaques fois

Ces Basic Linear Algebra Subroutines ou BLAS sont maintenant accessibles pour presque tous les langages de programmation

Les BLAS sont partitionnées par niveau. Les BLAS de niveau k effectuent $\mathcal{O}(n^k)$ opérations (ici n est la dimension des vecteurs ou des matrices)

Outils de base pour manipuler des matrices : les BLAS (suite)

Exemple 3

- *BLAS de niveau 1*
  1. *sscal calcule ax où a est un scalaire et x un vecteur*
  2. *saxpy calcule $ax + y$*
  3. *sdot calcule $x^* y$*
- *BLAS de niveau 2*
  1. *produit matrice-vecteur*
  2. *solution d'un système linéaire triangulaire*
- *BLAS de niveau 3*
  1. *produit matrice-matrice*
  2. *solution de multiples systèmes linéaires triangulaires*

Quand une BLAS existe, il est important de l'utiliser dans votre programme

MATLAB utilise automatiquement les BLAS. Dans les autres langages, on doit les appeler explicitement

Décompositions matricielles et leurs applications

Nous allons étudier les 4 décompositions suivantes :

1. LU
2. QR
3. Réduction (diagonalisation, etc.)
4. SVD (décomposition en valeurs singulières)

Petit rappel

Une matrice de permutation est une matrice qui vérifie les propriétés suivantes :

- Les coefficients sont 0 ou 1
- Il y a un et un seul 1 par ligne
- Il y a un et un seul 1 par colonne

Multiplier à gauche par une matrice de permutation revient à permuter les lignes de la matrice A. Par exemple :

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{pmatrix}$$

Décomposition LU

Définition 3

La décomposition LU d'une matrice A inversible de taille $n \times n$ est définie par $PA = LU$ où

- *P est une matrice de permutation*
- *L est une matrice triangulaire inférieure (Low) (avec des 1 sur la diagonale)*
- *U est une matrice triangulaire supérieure (Up)*

$$P \begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ \vdots & & \vdots \\ a_{n,1} & \dots & a_{n,n} \end{pmatrix} = \begin{pmatrix} 1 & & \\ \vdots & \ddots & \\ \ell_{n,1} & \dots & 1 \end{pmatrix} \begin{pmatrix} u_{1,1} & \dots & u_{1,n} \\ & \ddots & \vdots \\ & & u_{n,n} \end{pmatrix}$$

Cela correspond à l'algorithme d'élimination de Gauss

Comment calculer une décomposition LU

Principe de l'algorithme

- La matrice A est réduite en une matrice triangulaire supérieure en mettant des zéros en dessous de la diagonale, colonne par colonne, en soustrayant un multiple du pivot courant à toutes lignes situées en dessous
- Les multiplicateurs forment les entrées de L
- Pour la stabilité, il est nécessaire de pivoter ou d'interchanger des lignes. Les changements sont stockés dans P

En MATLAB : `[L,U,P]=lu(A)` ou `[PtL,U]=lu(A)` qui calcule $P^T L$ et U

Coût : $n^3/3$ multiplications

Les détails seront vus en TD/TME

Un exemple de décomposition LU

$$L_2^{-1} A = \begin{pmatrix} 1 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/4 & 0 & 1 \end{pmatrix} \begin{pmatrix} 4 & 4 & 8 \\ 2 & 8 & 7 \\ 1 & 3 & 6 \end{pmatrix} = \begin{pmatrix} 4 & 4 & 8 \\ 0 & 6 & 3 \\ 0 & 2 & 4 \end{pmatrix}$$

$$L_1^{-1} L_2^{-1} A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1/3 & 1 \end{pmatrix} \begin{pmatrix} 4 & 4 & 8 \\ 0 & 6 & 3 \\ 0 & 2 & 4 \end{pmatrix} = \begin{pmatrix} 4 & 4 & 8 \\ 0 & 6 & 3 \\ 0 & 0 & 3 \end{pmatrix} = U$$

Au final

$$A = L_2 L_1 U = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/4 & 1/3 & 1 \end{pmatrix} \begin{pmatrix} 4 & 4 & 8 \\ 0 & 6 & 3 \\ 0 & 0 & 3 \end{pmatrix} = LU$$

Pourquoi une matrice de permutation (1)

Que se passe-t-il pour

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} ?$$

Le pivot est nul, on factorise alors la matrice :

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

.

Pourquoi une matrice de permutation (2)

Que se passe-t-il pour

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix} ?$$

On a alors $A = LU = \begin{pmatrix} 1 & 0 \\ 10^{20} & 1 \end{pmatrix} \begin{pmatrix} 10^{-20} & 1 \\ 0 & 1 - 10^{20} \end{pmatrix}$

Si la mantisse pour représenter les flottants est de 16 chiffres décimaux, alors $1 - 10^{20}$ est représenté par le flottant -10^{20}.
La factorisation devient alors

$$\widetilde{L}\widetilde{U} = \begin{pmatrix} 1 & 0 \\ 10^{20} & 1 \end{pmatrix} \begin{pmatrix} 10^{-20} & 1 \\ 0 & -10^{20} \end{pmatrix} = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 0 \end{pmatrix}$$

Utiliser la matrice de permutation pour utiliser l'élément le plus grand en valeur absolue comme pivot.

Utilisation de la décomposition LU

On peut utiliser la décomposition LU pour résoudre un système d'équations linéaires

$$Ax = b,$$

étant donnés A et b.

Si on factorise A en $PA = LU$ alors on se ramène à résoudre

$$PA = LUx = Pb$$

En posant $y = Ux$, on a $Ly = Pb$

Pour résoudre $Ax = b$:

1. on résout $Ly = Pb$
2. on résout $Ux = y$

En MATLAB, le backslash `x=A\b` résout le système en utilisant une décomposition LU

Utilisation de la décomposition LU (suite)

- Pour résoudre le système $Ax = b$, on résoud d'abord le système $Ly = Pb$
  Pour $i = 1 : n$

$$y_i = (Pb)_i - \sum_{j=1}^{i-1} l_{ij} y_j$$

- Puis on résoud le système $Ux = y$
  Pour $i = n : -1 : 1$,

$$x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij} x_j \right) / u_{ii}$$

Cas des matrices symétriques définies positives

Une matrice $n \times n$, A est symétrique définie positive si $A^T = A$ et $x^T A x > 0$ pour tout $x \neq 0$.
Si ω est un vecteur de taille $n-1$ et K une matrice de taille $(n-1) \times (n-1)$, une étape du pivot de Gauss sur $\frac{1}{\alpha}A$ avec $a_{1,1} = \alpha^2$ donne :

$$A = \begin{pmatrix} \alpha^2 & \omega^T \\ \omega & K \end{pmatrix} = \begin{pmatrix} \alpha & 0 \\ \omega/\alpha & I \end{pmatrix} \begin{pmatrix} \alpha & \omega^T/\alpha \\ 0 & K - \omega\omega^T/\alpha^2 \end{pmatrix}$$

En factorisant la matrice U comme :

$$\begin{pmatrix} \alpha & \omega^T/\alpha \\ 0 & K - \omega\omega^T/a_{1,1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & K - \omega\omega^T/a_{1,1} \end{pmatrix} \begin{pmatrix} \alpha & \omega^T/\alpha \\ 0 & I \end{pmatrix}$$

En combinant les deux opérations on a :

$$A = \begin{pmatrix} \alpha & 0 \\ \omega/\alpha & I \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & K - \omega\omega^T/a_{1,1} \end{pmatrix} \begin{pmatrix} \alpha & \omega^T/\alpha \\ 0 & I \end{pmatrix} = LDL^T$$

Méthode de Choleski

Si la matrice A est symétrique définie positive : $A^T = A$ et $x^T A x > 0$ pour tout $x \neq 0$, alors il vaut mieux utiliser la décomposition de Choleski,

$$A = LL^T$$

avec L triangulaire inférieure, ou

$$A = LDL^T$$

avec L triangulaire inférieure avec des 1 sur la diagonale et D diagonale

Deux fois moins coûteux que la décomposition LU
En MATLAB, utiliser la commande `chol`

Existence, unicité des solutions d'un système linéaire

- Si A est inversible alors le système linéaire $Ax = b$ admet une unique solution
- Si A est singulière alors x est solution de $Ax = b$ si b peut s'exprimer comme une combinaison linéaire des colonnes de A. Dans ce cas, tout vecteur $x + y$ est encore solution si $Ay = 0$.

Sensiblité des solutions d'un système linéaire

Supposons que l'on perturbe un peu le système de façon à ce que nous ayons à résoudre

$$(A + \Delta A)y = b + \Delta b.$$

On veut savoir à quelle distance la solution y du système perturbé se trouve de la solution x du système initial
Posons

$$\begin{aligned} \varepsilon_A &= \frac{\|\Delta A\|}{\|A\|} \\ \varepsilon_b &= \frac{\|\Delta b\|}{\|b\|} \\ \kappa &= \|A\|\|A^{-1}\| \quad \text{conditionnement de la matrice } A \end{aligned}$$

Si $\kappa\varepsilon_A < 1$ alors

$$\frac{\|x - y\|}{\|x\|} \leq \frac{\kappa}{1 - \kappa\varepsilon_A}(\varepsilon_A + \varepsilon_b)$$