

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

Cours n°2 : Calcul matriciel (1/2)

Stef Graillat

Université Pierre et Marie Curie (Paris 6)



Résumé du cours précédent

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

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

$$Ax = \lambda x$$

- 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

- 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 **transposée** de A : $b_{ij} = a_{ji}$
- $B = A^*$ signifie que B est la **transconjuguée** de A : $b_{ij} = \bar{a}_{ji}$
- 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 $\|\cdot\| : \mathbb{C}^n \rightarrow \mathbb{R}^+$ 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

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

Définition 2

Une *norme matricielle* est une fonction $\| \cdot \| : \mathbb{C}^{m \times n} \rightarrow \mathbb{R}^+$ 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 donné 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

$$Ax = x_1A(:, 1) + x_2A(:, 2) + \dots + x_nA(:, n)$$

```
[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 les transférer en cache (par conséquent, d'autres données doivent être effacées du cache). Pour qu'un algorithme soit efficace, il faut limiter le nombre de fois qu'une page est chargée en cache !

Supposons que la matrice soit stockée en mettant une colonne par page. Quel est l'algorithme que vous allez utiliser ?

Question à vous poser sur votre langage de programmation si votre algorithme utilise des matrices :

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 tâches 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 chaque 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)

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* (avec des 1 sur la diagonale)
- U est une *matrice triangulaire supérieure*

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 multiplieurs 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

$$A = \begin{pmatrix} 4 & 4 & 8 \\ 2 & 8 & 7 \\ 1 & 3 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 4 & 4 & 8 \\ 0 & 6 & 3 \\ 0 & 2 & 4 \end{pmatrix} \rightarrow \begin{pmatrix} 4 & 4 & 8 \\ 0 & 6 & 3 \\ 0 & 0 & 3 \end{pmatrix}$$

Au final

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

Que se passe-t-il pour

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

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 $PAx = Pb$

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

Pour résoudre $Ax = b$:

- 1 on résoud $Ly = Pb$
- 2 on résoud $Ux = y$

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

- 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 $i = n : -1 : 1$,

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

Limitations de la décomposition LU

Si la matrice A est **symétrique définie positive** : $A^T = A$ et $x^T Ax > 0$ 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

En MATLAB, utiliser la commande `chol`

Deux fois moins coûteux que la décomposition LU

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

Sensibilité 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

$$\varepsilon_A = \frac{\|\Delta A\|}{\|A\|}$$

$$\varepsilon_b = \frac{\|\Delta b\|}{\|b\|}$$

$$\kappa = \|A\| \|A^{-1}\| \quad \text{conditionnement de la matrice } A$$

Si $\kappa \varepsilon_A < 1$ alors

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