

FACTORISATION DE MATRICES

8 novembre 2000

Nombreuses sont les applications où l'on doit résoudre des systèmes linéaires avec des matrices de grandes tailles, ou plus généralement manipuler ces matrices. Il peut alors être avantageux de factoriser une matrice comme un produit de deux ou plusieurs matrices ayant des propriétés agréables, comme celle d'être triangulaire, ou unitaire.

1 Factorisation LU

1.1 Résolution de systèmes tridiagonaux

But : Résoudre

$$Ax = b, \quad A \in \mathbb{C}^{N \times N}, x \in \mathbb{C}^N, b \in \mathbb{C}^N$$

On suppose que A est inversible.

1.1.1 Cas où A est triangulaire supérieure : remontée

La matrice $A = (a_{ij})_{i,j \in \{1, \dots, N\}}$ est triangulaire supérieure:

$$a_{ij} = 0 \quad \text{si } j < i$$

Comme A est inversible,

$$a_{ii} \neq 0 \quad \text{si } 1 \leq i \leq N.$$

Le système s'écrit

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1N}x_N &= b_1 \\ a_{22}x_2 + \dots + a_{2N}x_N &= b_2 \\ &\vdots \\ a_{NN}x_N &= b_N \end{aligned}$$

On résout le système en remontant

$$\begin{aligned} x_N &= b_N/a_{NN} \\ x_{N-1} &= (b_{N-1} - a_{N-1 N}x_N)/a_{N-1 N-1} \\ &\vdots \\ x_1 &= (b_1 - a_{12}x_2 - \dots - a_{1N}x_N)/a_{11} \end{aligned}$$

Coût du calcul de x_k :

$$x_k = (b_k - a_{k k+1}x_{k+1} - \dots - a_{kN}x_N)/a_{kk} \rightarrow$$

$(N - k)$ additions
 $(N - k)$ multiplications
 1 division

Coût total de la remontée:

$$\sim \sum_{k=1}^N (N - k) = \frac{1}{2}(N - 1)N \sim \frac{N^2}{2} \text{ additions+multiplications}$$

1.1.2 Cas où A est triangulaire inférieure : descente

La matrice $A = (a_{ij})_{i,j \in \{1, \dots, N\}}$ est triangulaire inférieure:

$$a_{ij} = 0 \quad \text{si } i < j$$

Comme A est inversible,

$$a_{ii} \neq 0 \quad \text{si } 1 \leq i \leq N.$$

Le système s'écrit

$$\begin{aligned} a_{11}x_1 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \\ &\vdots \\ a_{N1}x_1 + a_{N2}x_2 + \dots + a_{NN}x_N &= b_N \end{aligned}$$

On résout le système en descendant

$$\begin{aligned} x_1 &= b_1/a_{11} \\ x_2 &= (b_2 - a_{21}x_1)/a_{22} \\ &\vdots \\ x_N &= (b_N - a_{NN-1}x_{N-1} - \dots - a_{N1}x_1)/a_{NN} \end{aligned}$$

Coût total de la descente:

$$\sim \sum_{k=1}^N k = \frac{1}{2}(N - 1)N \sim \frac{N^2}{2} \text{ additions+multiplications}$$

1.2 Algorithme d'élimination de Gauss sans recherche de pivot, et interprétation matricielle

On suppose que A est inversible. On veut résoudre le système

$$Ax = b, \quad A \in \mathbb{C}^{N \times N}, x \in \mathbb{C}^N, b \in \mathbb{C}^N$$

Le But est de se ramener à un système triangulaire

Première étape : élimination de l'inconnue x_1 des lignes 2 à N . On va chercher un système équivalent (ayant la même solution x), où x_1 n'apparaît que dans la ligne 1.

Première Hypothèse:

$$a_{11} \neq 0 \rightarrow \text{on appelle premier pivot } \pi^1 = a_{11}.$$

On peut alors former une combinaison linéaire de la ligne $i > 1$ avec la ligne 1 pour éliminer l'inconnue x_1 dans la ligne i : la ligne i devient

$$0x_1 + (a_{i2} - \frac{a_{i1}}{\pi^1}a_{12})x_2 + \dots + (a_{iN} - \frac{a_{i1}}{\pi^1}a_{1N})x_N = b_i - \frac{a_{i1}}{\pi^1}b_1,$$

ce qu'on peut réécrire

$$a_{i2}^2 x_2 + \dots + a_{iN}^2 x_N = b_i^2, \quad \forall i > 1$$

en posant

$$\begin{aligned} a_{i2}^2 &= a_{i2} - \frac{a_{i1}}{\pi^1}a_{12}, \\ &\vdots \\ a_{iN}^2 &= a_{iN} - \frac{a_{i1}}{\pi^1}a_{1N}, \\ b_i^2 &= b_i - \frac{a_{i1}}{\pi^1}b_1. \end{aligned}$$

Si on pose aussi, pour $1 \leq j \leq N$,

$$a_{1j}^2 = a_{1j} \quad \forall 1 \leq j \leq N \quad \text{et} \quad b_1^2 = b_1,$$

on a obtenu le nouveau système équivalent

$$A^2 x = b^2,$$

avec

$$A^2 = \begin{pmatrix} a_{11}^2 & a_{12}^2 & \dots & a_{1N}^2 \\ 0 & a_{22}^2 & \dots & a_{2N}^2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{N2}^2 & \dots & a_{NN}^2 \end{pmatrix}$$

Le système s'écrit:

$$A^2 = M^1 A, \quad \text{et} \quad b^2 = M^1 b.$$

k ième étape: élimination de l'inconnue x_k des lignes $k + 1$ à N On suppose qu'on a pu itérer le procédé ci-dessus $k - 1$ fois, c'est à dire que les pivots apparus jusqu'à l'étape k sont non nuls:

$$\Pi^i = a_{ii}^i \neq 0, \quad \text{pour} \quad 1 \leq i \leq k - 1.$$

On obtient alors le système équivalent

$$A^k x = b^k$$

où A^k est de la forme

$$A^k = \begin{pmatrix} a_{11}^k & a_{12}^k & & \dots & & a_{1N}^k \\ 0 & a_{22}^k & & \dots & & a_{2N}^k \\ 0 & 0 & a_{33}^k & \dots & & a_{3N}^k \\ & & & \ddots & & \\ 0 & 0 & \dots & 0 & a_{kk}^k & \dots & a_{kN}^k \\ & & & & 0 & & \\ 0 & 0 & \dots & & \vdots & & \\ 0 & 0 & \dots & & 0 & & \end{pmatrix}$$

On va chercher un système équivalent (ayant la même solution x), où x_k n'apparaît pas dans les lignes $k + 1$ à N .

k -ième Hypothèse:

$$a_{kk}^k \neq 0 \rightarrow \text{on appelle } k\text{ième pivot } \pi^k = a_{kk}^k.$$

On introduit

$$M^k = \begin{pmatrix} 1 & 0 & & & \dots & & 0 \\ 0 & 1 & 0 & & & & 0 \\ & & \ddots & & & & \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & \dots & 0 & -\frac{a_{k+1,k}^k}{\pi^k} & 1 & \dots & 0 \\ 0 & \dots & 0 & \vdots & 0 & \ddots & 0 \\ 0 & \dots & 0 & -\frac{a_{N,k}^k}{\pi^k} & 0 & \dots & 1 \end{pmatrix}$$

Remarquons que l'inverse de M^k est L^k :

$$L^k = \begin{pmatrix} 1 & 0 & & & \dots & & 0 \\ 0 & 1 & 0 & & & & 0 \\ & & \ddots & & & & \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & \dots & 0 & \frac{a_{k+1,k}^k}{\pi^k} & 1 & \dots & 0 \\ 0 & \dots & 0 & \vdots & 0 & \ddots & 0 \\ 0 & \dots & 0 & \frac{a_{N,k}^k}{\pi^k} & 0 & \dots & 1 \end{pmatrix}$$

Soit

$$A^{k+1} = M^k A^k \quad \text{et} \quad b^{k+1} = M^k b^k,$$

alors

$$A^{k+1} x = b^{k+1},$$

et

$$A^{k+1} = \begin{pmatrix} a_{11}^k & a_{12}^k & & & \dots & & a_{1N}^k \\ 0 & a_{22}^k & & & & & a_{2N}^k \\ 0 & 0 & a_{33}^k & \dots & & & a_{3N}^k \\ & & & \ddots & & & \\ 0 & 0 & \dots & 0 & a_{kk}^k & \dots & a_{kN}^k \\ & & & & 0 & & \\ 0 & 0 & \dots & & \vdots & & \\ 0 & 0 & \dots & & 0 & & \end{pmatrix}$$

Après $N - 1$ étapes Si on itère $N - 1$ fois, et si les pivots apparus sont tous non nuls:
Hypothèses des $N - 1$ étapes :

$$\pi^i = a_{ii}^i \neq 0, \quad \text{pour } 1 \leq i \leq N - 1.$$

on arrive au système **triangulaire équivalent**

$$A^N x = b^N,$$

avec

$$A^N = \begin{pmatrix} \pi^1 & * & & \dots & * \\ 0 & \pi^2 & * & \dots & * \\ 0 & 0 & \ddots & & \\ 0 & \dots & 0 & \ddots & * \\ 0 & & \dots & 0 & \pi^N \end{pmatrix}$$

qui est inversible si de plus

Hypothèse :

$$\pi^N \neq 0.$$

Si on appelle L la matrice triangulaire inférieure avec des 1 sur la diagonale

$$L = L^1 \dots L^{N-1}$$

et U la matrice triangulaire supérieure

$$U = A^N,$$

on a

$$A = LU.$$

On dit qu'on a effectué une factorisation de Gauss ou factorisation LU de A .

Remarque 1 *Il est aussi possible avec le même algorithme d'obtenir la factorisation LU d'une matrice de $\mathbb{C}^{m \times n}$, avec $m \geq n$, si les pivots qui apparaissent sont non nuls.*

1.3 Coût de la méthode d'élimination de Gauss.

On estime le coût de l'élimination de x_k . Rappelons qu'à l'étape $k-1$, on obtient la matrice A_k

$$A^k = \begin{pmatrix} a_{11}^k & a_{12}^k & & \dots & & a_{1N}^k \\ 0 & a_{22}^k & & \dots & & a_{2N}^k \\ 0 & 0 & a_{33}^k & \dots & & a_{3N}^k \\ & & & \ddots & & \\ 0 & 0 & \dots & 0 & a_{k-1,k-1}^k & \dots & a_{k-1,N}^k \\ & & & & 0 & & \\ 0 & 0 & \dots & \vdots & & * & \\ 0 & 0 & \dots & 0 & & & \end{pmatrix}$$

C'est sur le bloc $*$ de A^k qu'il faut travailler.

Construction de M^k : $N - k$ divisions par le pivot.

$$M^k = \begin{pmatrix} 1 & 0 & & & \dots & & 0 \\ 0 & 1 & 0 & & & & 0 \\ & & \ddots & & & & \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & \dots & 0 & -\frac{a_{k+1,k}^k}{\pi^k} & 1 & \dots & 0 \\ 0 & \dots & 0 & \vdots & 0 & \ddots & 0 \\ 0 & \dots & 0 & -\frac{a_{N,k}^k}{\pi^k} & 0 & \dots & 1 \end{pmatrix}$$

Multiplication de A^k par M^k : $(N - k)^2$ adds+mults.

Total

$$\sum_{k=1}^N (N - k)^2 = \frac{1}{3}N(N - 1)(N - 2) \sim \frac{N^3}{3} \text{ additions+multiplications.}$$

$$\sum_{k=1}^N (N - k) = \frac{1}{2}N(N - 1) \text{ divisions.}$$

1.4 Utilisations de la factorisation LU

Calcul de déterminant Une utilisation de la factorisation LU est le calcul du déterminant de A : en effet, si A admet une factorisation LU , on a

$$\det(A) = \det(U) = \text{produit des pivots}$$

car $\det(L) = 1$. La factorisation LU permet donc de calculer le déterminant de A avec une complexité de l'ordre de $\frac{N^3}{3}$ adds + mults, plutôt que $N!$ avec la formule du déterminant!

Remarque 2 Calculer le déterminant d'une matrice d'ordre 100 avec la formule du déterminant prendrait un temps supérieur à l'âge de l'univers sur le calculateur le plus puissant disponible aujourd'hui.

Résolution de systèmes linéaires Si on a plusieurs systèmes (par exemple p) à résoudre avec la même matrice A , il est intéressant de calculer une fois pour toute et stocker la factorisation LU de $A = LU$, puis de résoudre les systèmes par descentes-remontées plutôt que d'utiliser l'algorithme d'élimination pour chaque système. On résout le système $Ax = b$ en résolvant d'abord

$$Ly = b,$$

soit une descente, puis

$$Ux = y,$$

soit une remontée. Si on a p systèmes à résoudre, la complexité est de l'ordre de $\frac{N^3}{3} + pN^2$ adds+mults plutôt que $p\frac{N^3}{3}$.

Remarque 3 Comme à la remarque précédente, calculer la solution du système linéaire en appliquant les formules de Cramer nécessiterait un temps inimaginable dès que N vaut quelques dizaines.

1.5 Algorithmes

Voici un algorithme réalisant la factorisation LU d'une matrice A et stockant cette factorisation dans la place mémoire occupée par A : (la matrice A est perdue, on dit qu'on écrase A).

```
for (int i=1;i<=N;i++)
{
  for(int j=1;j<=i;j++)
  {
    sum = 0;
    for (int k=1; k<j;k++)
      sum += A(i,k)*A(k,j);
    A(i,j) = A(i,j)-sum;
  }
  for(int j=i+1;j<=N;j++)
  {
    sum = 0;
    for (int k=1; k<i;k++)
      sum += A(i,k)*A(k,j);
    A(i,j) = (A(i,j) - sum)/A(i,i);
  }
}
```

Remarque 4 La diagonale de L (qui ne contient que des 1) n'est pas stockée.

Le programme pour la descente-remontée pour calculer la solution de $Ax = b$ en écrasant b est alors:

```
for (int i=1;i<=N;i++)
{
  sum=0.;
  for (int k=1; k<i;k++)
    sum += A(i,k)*b(k);
  b(i) = (b(i) - sum )/A(i,i);
}
for (int i=N;i>=1;i--)
{
  sum=0.;
  for (int k=i+1; k<=N;k++)
    sum += A(i,k)*b(k);
  b(i) -= sum ;
}
```

1.6 Condition nécessaire et suffisante d'existence, unicéité

Théorème 1 Soit A une matrice de $\mathbb{C}^{N \times N}$. Pour $1 \leq p \leq n$, on note A_p le bloc

$$A_p = \begin{pmatrix} a_{11} & \dots & \dots & a_{1p} \\ a_{21} & \dots & \dots & a_{2p} \\ \vdots & & & \\ a_{p1} & \dots & \dots & a_{pp} \end{pmatrix}$$

La matrice A admet une factorisation

$$A = LU$$

où L est triangulaire inférieure avec des 1 sur la diagonale et U est triangulaire supérieure et inversible si et seulement si tous les blocs A_p , $1 \leq p \leq N$, sont inversibles. De plus, cette factorisation est unique. De plus si A est réelle, alors L et U le sont aussi.

Démonstration de l'unicéité Soient deux factorisation LU de A : $A = LU = L'U'$ où L sont triangulaires inférieures avec des 1 sur la diagonale, et U sont triangulaires supérieures et inversibles. On a donc l'identité

$$L^{-1}L' = U'U^{-1}$$

Mais $L^{-1}L'$ est triangulaire inférieure avec des 1 sur la diagonale et $U'U^{-1}$ est triangulaire supérieure. On a donc

$$L^{-1}L' = U'U^{-1} = Id \Rightarrow \begin{cases} L = L' \\ U = U' \end{cases}$$

1.7 Cas de matrices bandes

Définition 1 Soit A une matrice de $\mathbb{C}^{n \times n}$. On appelle largeur de bande de A le plus petit entier $n_b \leq N$ tel que pour tout i , $1 \leq i \leq N$

$$\begin{aligned} a_{i,j} &= 0 & \text{si } N \geq j > i + n_b, \\ a_{j,i} &= 0 & \text{si } 0 < j < i - n_b, \end{aligned}$$

Les coefficients non nuls de la matrices A sont contenus dans une bande de largeur $2n_b + 1$, centrée sur la diagonale.

Dans le cas où la largeur de bande n_b de A est très inférieure à N , on parle de matrice bande. Si A est une matrice bande, sa factorisation LU demande moins d'opérations et de place mémoire :

Proposition 1 Si A , (de largeur de bande n_b) admet une factorisation LU , alors les largeurs de bandes de L et de U sont inférieures à n_b et la complexité nécessaire à effectuer la factorisation LU est inférieure à

$$\begin{aligned} \sum_{k=1}^N (n_b)^2 &= N(n_b)^2 \text{ additions+multiplications.} \\ \sum_{k=1}^N n_b &= N n_b \text{ divisions.} \end{aligned}$$

2 Méthode de Choleski

Dans le cas où A est hermitienne et définie positive, on peut toujours effectuer la factorisation décrite ci-dessus. De plus, on peut trouver une factorisation du type $A = LL^*$ moins gourmande en place mémoire.

2.1 Existence de la factorisation de Choleski

Théorème 2 *Si A est hermitienne et définie positive, alors A admet une unique factorisation LU , où L est triangulaire inférieure avec des 1 sur la diagonale et U est triangulaire supérieure et inversible.*

Démonstration Si A est hermitienne définie positive, ses blocs A_p $1 \leq p \leq N$, voir Théorème 1, le sont aussi, et on peut appliquer le Théorème 1.

Théorème 3 *Si A est hermitienne et définie positive, alors il existe une unique matrice L triangulaire inférieure et inversible, avec des coefficients réels positifs sur la diagonale, telle que*

$$A = LL^*$$

Cette factorisation porte le nom de factorisation de Choleski (qui était un colonel de l'armée de Napoléon). Si A est réelle symétrique définie positive, L est réelle.

Démonstration On part de la factorisation LU de A . Il existe une unique factorisation $A = L'U'$ où la matrice L' est triangulaire inférieure avec des 1 sur la diagonale et U' est triangulaire supérieure inversible. Appelons D la diagonale de U' . Comme pour tout p , $1 \leq p \leq N$, $\det(D_p) = \det(A_p) > 0$, tous les coefficients de D sont strictement positifs. Notons $U = D^{-\frac{1}{2}}U'$ et $L = L'D^{\frac{1}{2}}$. On a $A = LU$. Montrons que $U = L^*$. Comme A est hermitienne, $U^*L^* = A^* = A = LU$. On a donc $L^{-1}U^* = U(L^*)^{-1}$. Mais $L^{-1}U^*$ est triangulaire inférieure, avec des 1 sur la diagonale tandis que $U(L^*)^{-1}$ est triangulaire supérieure, avec des 1 sur la diagonale. Donc $U = L^*$, et on a $A = LL^*$. Pour l'unicité, on procède comme dans la démonstration de l'unicité pour la factorisation LU.

Remarque 5 *Il est important de comprendre que les matrices L dans les factorisation LU et de Choleski sont différentes.*

2.2 Algorithme

Considérons la k ième étape de la méthode de Choleski: A ce point, on suppose que l'on a déjà construit la matrice L^{k-1} et A^k telles que

$$L^{k-1} = \begin{pmatrix} \sqrt{\pi_1} & 0 & & \dots & 0 \\ * & \sqrt{\pi_2} & 0 & & 0 \\ & & \ddots & & \\ * & \dots & * & \sqrt{\pi_{k-1}} & 0 & \dots & 0 \\ * & \dots & * & * & 1 & \dots & 0 \\ * & \dots & * & \vdots & 0 & \ddots & 0 \\ * & \dots & * & * & 0 & \dots & 1 \end{pmatrix}$$

et

$$A^k = \begin{pmatrix} * & * & & \dots & * \\ 0 & * & & \dots & * \\ 0 & 0 & * & \dots & * \\ & & & \ddots & \\ 0 & 0 & \dots & 0 & a_{kk}^k & \dots & a_{kN}^k \\ & & & & \vdots & \ddots & \\ 0 & 0 & \dots & 0 & a_{Nk}^k & \dots & a_{NN}^k \end{pmatrix}$$

et $\pi_k = a_{k,k}^k$.

Remarque 6 Le bloc carré $k \rightarrow N$ de A^k est donc hermitien et défini positif.

Pour effectuer la k ème étape, on construit d'abord L^k en remplaçant la k ème colonne de L^{k-1} par le vecteur $l_k = (0, \dots, 0, \sqrt{\pi_k}, \frac{a_{k+1,k}^k}{\sqrt{\pi_k}}, \dots, \frac{a_{Nk}^k}{\sqrt{\pi_k}})^T$.

Observation 1 D'après la Remarque 6, $(0, \dots, 0, a_{kk}^k \dots a_{kN}^k) = \sqrt{\pi_k} l_k^*$.

On construit maintenant A^{k+1} en effectuant les éliminations de Gauss et d'après l'Observation 1, on a

$$A^{k+1} = A^k - l_k l_k^*. \quad (1)$$

On voit en fait que l'on peut ne construire que la partie triangulaire inférieure de la matrices A^{k+1} , ce qui économise la moitié des opérations. On aboutit à l'algorithme suivant :

```

for (k = 1; k <= n; k++)
{
  A(k,k)=sqrt(A(k,k) );
  for (i = k+1; i <= n ; i++)
    A(i,k)= A(i,k) / A(k,k);
  for (i = k + 1; i <=n; i++)
    for (j = i ; j <= n; j++)
      A(j,i)=A(j,i) - A(i,k)*A(j,k);
}

```

2.3 Complexité de la factorisation de Choleski

$$\begin{aligned} &\sim \frac{N^3}{6} \text{ additions+multiplications.} \\ &\frac{1}{2}N(N-1) \text{ divisions.} \\ &N \text{ évaluations de racines carrées.} \end{aligned}$$

Exercice Montrer l'évaluation précédente.

Remarque 7 L'intérêt de la factorisation de Choleski est qu'elle demande une place mémoire deux fois inférieure à celles de la factorisation LU ci-dessus, car on ne doit stocker que L et que sa complexité arithmétique est deux fois moindre.

3 Méthode de Gauss avec pivot partiel

La méthode de Gauss sans pivotage peut être bloquée si on tombe sur un pivot nul. Pour pallier cet inconvénient, on introduit une méthode qui permet d'éviter ce blocage en échangeant les lignes du système considéré. On obtient alors une méthode qui fonctionne pour toute matrice inversible: la méthode de Gauss avec pivot partiel.

On définit quelques notions utiles:

Définition 2 On appelle permutation de $\{1, \dots, N\}$ une bijection de $\{1, \dots, N\}$ sur $\{1, \dots, N\}$.

Définition 3 Soit σ une permutation de $\{1, \dots, N\}$, on associe à σ une matrice P dite de permutation d'ordre N par

$$P_{ij} = \delta_{\sigma(i)j},$$

c'est à dire

$$\begin{aligned} P_{ij} &= 1 & \text{si } j = \sigma(i), \\ P_{ij} &= 0 & \text{si } j \neq \sigma(i), \end{aligned}$$

Lemme 1 Soit P et Q deux matrices de permutation d'ordre N , alors PQ est une matrice de permutation d'ordre N .

Définition 4 Soient $1 \leq k < l \leq N$, on dit que la matrice de permutation P^{kl} définie par

$$\begin{aligned} P_{ij}^{kl} &= \delta_{ij} & \text{si } i \neq k \text{ et } i \neq l \\ P_{kj}^{kl} &= \delta_{lj} & \forall 1 \leq j \leq N \\ P_{lj}^{kl} &= \delta_{kj} & \forall 1 \leq j \leq N \end{aligned}$$

est appelée matrice de permutation élémentaire.

Observation 2 Le produit à gauche de A par P^{kl} conduit à échanger les lignes k et l de A , en laissant les autres lignes inchangées, c'est à dire si

$$B = P^{kl}A$$

$$\begin{aligned} B_{ij} &= A_{ij} & \text{si } i \neq k \text{ et } i \neq l \\ B_{kj} &= A_{lj} & \forall 1 \leq j \leq N \\ B_{lj} &= A_{kj} & \forall 1 \leq j \leq N \end{aligned}$$

Observation 3 Le déterminant d'une matrice de permutation élémentaire est ± 1 . En effet,

$$P^{kl}P^{kl} = Id.$$

Lemme 2 Toute matrice de permutation peut se décomposer en un produit de matrices élémentaires.

Corollaire 1 Le déterminant d'une matrice de permutation est ± 1 .

La méthode du pivot partiel est un algorithme dont l'application permet de prouver le théorème suivant. La méthode du pivot partiel a été présentée en cours sur un exemple et elle revue dans la preuve du théorème:

Théorème 4 Soit $A \in \mathbb{C}^{N \times N}$, inversible. Alors il existe

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

Comme

$$(P^{kr_k})^{-1} = P^{kr_k},$$

$$A^k = P^{kr_k} L^k A^{k+1},$$

ce qui implique

$$Q^{k-1} A = L^1 \dots L^{k-1} P^{kr_k} L^k A^{k+1}.$$

A ce point, on utilise le lemme:

Lemme 3 Soit P^{kr_k} la matrice de permutation élémentaire entre les lignes k et $r_k > k$, alors $\forall i < k$, si L^i s'écrit

$$L^i = \begin{pmatrix} 1 & 0 & & \dots & & 0 \\ 0 & \ddots & & & & \\ & 0 & \ddots & & & \vdots \\ & & 0 & 1 & & \\ \vdots & \vdots & \vdots & C_i & \ddots & 0 \\ 0 & \dots & 0 & & 0 & 1 \end{pmatrix} \quad \text{et} \quad C_i = \begin{pmatrix} \vdots \\ \alpha \\ \vdots \\ \beta \\ \vdots \end{pmatrix}$$

alors

$$L^i P^{kr_k} = P^{kr_k} L'^i,$$

où

$$L'^i = \begin{pmatrix} 1 & 0 & & \dots & & 0 \\ 0 & \ddots & & & & \\ & 0 & \ddots & & & \vdots \\ & & 0 & 1 & & \\ \vdots & \vdots & \vdots & C'_i & \ddots & 0 \\ 0 & \dots & 0 & & 0 & 1 \end{pmatrix} \quad \text{et} \quad C'_i = \begin{pmatrix} \vdots \\ \beta \\ \vdots \\ \alpha \\ \vdots \end{pmatrix}.$$

Donc,

$$Q^{k-1} A = L^1 \dots L^{k-2} P^{kr_k} L'^{k-1} L^k A^{k+1}$$

$$= P^{kr_k} L^1 \dots L'^{k-1} L^k A^{k+1}.$$

Par suite,

$$P^{kr_k} Q^{k-1} A = L^1 \dots L'^{k-1} L^k A^{k+1},$$

où

- $Q^k = P^{kr_k} Q^{k-1}$ est une matrice de permutation,
- $L^1 \dots L'^{k-1} L^k$ est triangulaire inférieure avec des 1 sur la diagonale.

Remarque 8 Dans la méthode de Gauss avec pivotage partiel, on ne doit pas stocker la matrice P mais seulement les r_k . Le coût mémoire et la complexité sont du même ordre que ceux de la factorisation LU . La résolution du système linéaire $Ax = b$ connaissant P , L et U se fait de la manière suivante :

$$y = Pb$$

$$Lz = y$$

$$Ux = z$$

Remarque 9 La méthode de Gauss avec pivotage partielle ne conserve pas l'aspect bande : si A est une matrice bande dont la largeur de bande est n_b , les matrices L et U n'ont pas en général cette propriété. La complexité et le coût mémoire de la méthode de Gauss avec pivotage partiel deviennent alors largement supérieurs que ceux de la factorisation LU.

Remarque 10 La méthode du pivot total consiste à échanger non seulement les lignes mais aussi les colonnes de manière à choisir comme pivot le plus grand coefficient en module du bloc carré restant à factoriser. Cette méthode conduit à l'existence de deux matrices de permutations P et Q telles que $PAQ = LU$.

Influence du pivotage sur la précision Le pivotage partiel ne sert pas seulement à garantir l'obtention d'une factorisation, il garantit aussi une bien meilleure stabilité que la factorisation LU, pour des matrices A mal conditionnées : prenons le système $Ax = b$ où

$$A = \begin{pmatrix} 10^{-9} & 1 \\ 1 & 1 \end{pmatrix} \quad \text{et} \quad b = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

dont la solution est $x = (\frac{1}{1-10^{-9}}, \frac{1-2 \cdot 10^{-9}}{1-10^{-9}})^T \sim (1,1)^T$. Supposons que la machine utilisée pour résoudre ce système ne garde que 8 chiffres significatifs : la factorisation LU calculée par la machine est

$$U = \begin{pmatrix} 10^{-9} & 1 \\ 0 & -10^9 \end{pmatrix} \quad \text{et} \quad L = \begin{pmatrix} 1 & 0 \\ 10^9 & 1 \end{pmatrix}$$

En utilisant cette factorisation, la machine retourne $x = (0,2)^T$!
Si on utilise le pivotage partiel, on obtient :

$$U = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \text{et} \quad L = \begin{pmatrix} 1 & 0 \\ 10^{-9} & 1 \end{pmatrix}$$

La machine retourne $x = (1,1)^T$.

4 La méthode de Householder

4.1 Les réflexions de Householder

Définition 5 Soit v un vecteur non nul de \mathbb{C}^n . On appelle réflexion de Householder ou matrice de Householder relative au vecteur v la matrice

$$H_v = I - 2 \frac{vv^*}{v^*v}. \tag{2}$$

Les réflexions de Householder ont les propriétés suivantes :

1. H_v est une matrice hermitienne.
2. H_v est une matrice unitaire.
3. $H_v - I$ est une matrice de rang un.
4. $H_{\lambda v} = H_v$, pour tout $\lambda \neq 0$.

Proposition 2 Soit e un vecteur unitaire de \mathbb{C}^N et x un vecteur non nul de \mathbb{C}^N . Il existe un vecteur non nul $v \in \mathbb{C}^N$ tel que $H_v x$ soit colinéaire à e .

Démonstration On cherche v sous la forme $v = x + \lambda e$. Dans le cas réel, ($x \in \mathbb{R}^N$ et $e \in \mathbb{R}^N$), les vecteurs $v = x \pm \|x\|_2 e$ (au moins l'un des deux est non nul) sont les seuls vecteurs de cette forme ayant la propriété demandée et on a $H_v x = \mp \|x\|_2 e$. Dans le cas général, on trouve aussi deux vecteurs v sous cette forme, et au moins l'un d'entre eux est non nul.

Remarque 11 Si x est presque colinéaire à e , on a intérêt en pratique à choisir le vecteur v pour que v^*v , qui est au dénominateur de (2), ne soit pas petit. En effet, en précision finie, il faut éviter les divisions par des nombres petits. En particulier si $v \in \mathbb{R}^n$, on préférera le vecteur $v^+ = x + \text{signe}(x^*e)\|x\|_2 e$ au vecteur $v^- = x - \text{signe}(x^*e)\|x\|_2 e$, car $\|v^-\|_2$ est petit.

4.2 Factorisations QR

Théorème 5 Soit A une matrice de $\mathbb{C}^{m \times n}$ avec $m \geq n$. Il existe une matrice unitaire $Q \in \mathbb{C}^{m \times m}$ et une matrice triangulaire supérieure $R \in \mathbb{C}^{m \times n}$, telles que

$$A = QR.$$

Démonstration On démontre ce résultat par récurrence : supposons qu'à l'aide de deux réflexions de Householder H_1 et H_2 , on ait obtenu

$$H_2 H_1 A = \begin{pmatrix} * & * & * & \dots & * \\ 0 & * & * & \dots & * \\ 0 & 0 & * & \dots & * \\ \vdots & \vdots & * & \dots & * \\ 0 & 0 & * & \dots & * \end{pmatrix}$$

On appelle x le vecteur de \mathbb{C}^{m-2} obtenu en prenant les $m-2$ derniers coefficients de la troisième colonne de $H_2 H_1 A$. Si x est non nul, on sait trouver $v_3 \in \mathbb{C}^{m-2}$, tel que

$$H_{v_3} x \text{ soit colinéaire à } \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

On prend alors H_3

$$H_3 = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & H_{v_3} \end{pmatrix}.$$

Si $x = 0$ on prend $H_3 = I$. Dans les deux cas H_3 est hermitienne et unitaire. On a

$$H_3 H_2 H_1 A = \begin{pmatrix} * & * & * & \dots & * \\ 0 & * & * & \dots & * \\ 0 & 0 & * & \dots & * \\ 0 & 0 & 0 & * & \dots & * \\ \vdots & \vdots & \vdots & * & \dots & * \\ 0 & 0 & 0 & * & \dots & * \end{pmatrix}$$

Comme $m \geq n$, on peut itérer ce procédé et construire n matrices H_i hermitiennes et unitaires, telles que $H_n \dots H_1 A$ soit une matrice triangulaire supérieure R . On note Q la matrice unitaire (mais pas forcément hermitienne) $Q = H_1 \dots H_n$, on a

$$A = QR.$$

Les éléments fondamentaux de l'algorithme de la factorisation QR à l'aide de réflexions de Householder sont

1. Le choix des vecteurs v_i , de manière à ce que $\|v_i\|$ ne soit pas petit, cf. Remarque 11.
 2. Le produit à gauche d'une matrice M par la matrice $H_m: H_m M$, doit être programmé intelligemment : on ne doit surtout pas stocker la matrice H_m . Il suffit de garder en mémoire le vecteur v_m , et d'opérer les réflexions de Householder à chaque colonne de M .
 3. On ne calcule donc pas Q en général, mais on garde en mémoire les vecteurs v_1, \dots, v_n .
 4. On peut écraser la matrice A que l'on factorise en stockant les vecteurs v_i dans la partie triangulaire inférieure stricte, et la matrice R dans la partie triangulaire supérieure à condition de normaliser les vecteurs v_i de manière à ce que leur premier coefficient soit 1.
- Cette factorisation QR a de nombreuses propriétés intéressantes:

1. Résoudre le système $Qy = b$ est facile car

$$Q^{-1} = Q^* = H_n \dots H_1.$$

La complexité de la résolution du système $Qy = b$ est donc $3 \sum_i^n (m - i)$ adds+mults. Si $m = n$, la complexité est de l'ordre de $\frac{3}{2}n^2$. Si la matrice A est carrée d'ordre n inversible, pour résoudre $Ax = b$, on résout d'abord $Qy = b$ comme ci-dessus, puis $Rx = y$ par une remontée. La complexité totale est de l'ordre de $2n^2$.

2. Si $A \in \mathbb{C}^{n \times n}$ est inversible, alors

$$\text{cond}_2(A) = \text{cond}_2(R),$$

car Q est unitaire.

3. la méthode de Householder permet de calculer $|\det(A)|$. En effet,

$$|\det(R)| = |\text{produit des coefficients diagonaux de } R| = |\det(A)|.$$

Théorème 6 Soit $A \in \mathbb{C}^{m \times m}$. On peut trouver une factorisation QR telle que tous les coefficients diagonaux de R sont réels positifs. Si A est inversible, cette factorisation est unique.

Démonstration On part de la factorisation QR de A obtenue par la méthode de Householder ci dessus: $A = Q'R'$ où la matrice Q' est unitaire et R' est triangulaire supérieure. Appelons D la matrice diagonale $D = \text{diag}(d_1, \dots, d_m)$ où

$$d_i = \begin{cases} \frac{r_{ii}}{|r_{ii}|} & \text{si } r_{ii} \neq 0, \\ 1 & \text{si } r_{ii} = 0, \end{cases}$$

La matrice D est clairement unitaire. Posons $Q = Q'D^{-1}$ et $R = DR'$. Ces matrices ont les propriétés désirées.

Supposons que A soit inversible et soient deux factorisation QR de A ,

$$A = Q_1 R_1 = Q_2 R_2$$

telles que tous les coefficients diagonaux de R_1 et R_2 sont réels positifs. Ils sont strictement positifs car A est inversible. On a alors $Q_2^* Q_1 = R_2 R_1^{-1}$. Mais $Q_2^* Q_1$ est unitaire tandis que $R_2 R_1^{-1}$ est triangulaire supérieure avec des coefficients diagonaux réels positifs strictement. On peut vérifier que l'Identité est la seule matrice unitaire et triangulaire supérieure avec des coefficients diagonaux réels positifs. Donc $Q_2 = Q_1$ et $R_2 = R_1$.

Remarque 12 Signalons pour finir qu'il est possible d'obtenir autrement des factorisation QR: on peut remplacer les réflexions de Householder par d'autres transformations unitaires comme des rotations.

```

//LU factorization
function [B]= LUfact(A)
[m,n]=size(A);
B=A;
for i=1:n,
    if (B(i,i)==0) then
        print(%io(2)," zero pivot")
    else
        B(i+1:m,i)=B(i+1:m,i)/B(i,i);
        for j=i+1:n,
            B(i+1:m,j)=B(i+1:m,j)-B(i+1:m,i)*B(i,j);
        end ;
    end ;
end;

// down sweep : solves an upper triangular system
function [y]=up_sweep(A,x)
[m,n]=size(A);
if (m~=n) then
    print(%io(2), "error, not a square matrix");
else
    y=x;
    y(n)=y(n)/A(n,n);
    for i=n-1:-1:1,
        y(i)=(y(i)-A(i,i+1:n)*y(i+1:n))/A(i,i);
    end;
end;

// down sweep : solves a lower triangular system with ones on the diagonal
function [y]=down_sweep(A,x)
[m,n]=size(A);
if (m~=n) then
    print(%io(2), "error, not a square matrix");
else
    y=x;
    for i=2:n
        y(i)=y(i)-A(i,1:i-1)*y(1:i-1);
    end;
end;

//assumes A contains the LU-factorization of A
function [y]=SolveLU(A,x)
[m,n]=size(A);
if (m~=n) then
    print(%io(2), "error, not a square matrix");
else
    y=down_sweep(A,x);
    y=up_sweep(A,y);
end;

```

end;

```

//Choleski factorization
//the matrix is stored in the lower part
function [B]= Choleski_fact(A)
[m,n]=size(A);
B=A;
for i=1:n,
    if (B(i,i)<=0) then
        print(%io(2)," non positive pivot")
    else
        B(i,i)=sqrt(B(i,i));
        B(i+1:m,i)=B(i+1:m,i)/B(i,i);
        for j=i+1:n,
            B(j:m,j)=B(j:m,j)-B(j:m,i)*B(j,i);
        end ;
    end ;
end ;

// up sweep : solves an upper triangular system
function [y]=up_sweep(A,x)
[m,n]=size(A);
if (m~=n) then
    print(%io(2), "error, not a square matrix");
else
    y=x;
    y(n)=y(n)/A(n,n);
    for i=n-1:-1:1,
        y(i)=(y(i)-(A(i+1:n,i))'*y(i+1:n))/A(i,i);
    end;
end;

// down sweep : solves a lower triangular system with ones on the diagonal
function [y]=down_sweep(A,x)
[m,n]=size(A);
if (m~=n) then
    print(%io(2), "error, not a square matrix");
else
    y=x;
    y(1)=y(1)/A(1,1);
    for i=2:n
        y(i)=y(i)-A(i,1:i-1)*y(1:i-1);
        y(i)=y(i)/A(i,i)
    end;
end;

//assumes A contains the Choleski-factorization of A

```

```
function [y]=SolveCholeski(A,x)
[m,n]=size(A);
if (m~=n) then
    print(%io(2), "error, not a square matrix");
else
    y=down_sweep(A,x);
    y=up_sweep(A,y);
end;
```

```

// householder reflexion : find Householder vector
//normalized so that v(1)=1

function [v]= householder_vector(x)
delta=1;
if (x(1)<0) then
    delta=-1;
end;
v=x;
beta=v(1)+delta* sqrt(x'*x);
v=v/beta;
v(1)=1;

// performs Householder reflexion of vector v on a vector x
function [y]= householder_reflexion(v,x)
beta=2*(v'*x)/(v'*v);
y=x-beta*v;

//performs QR factorisation of a matrix
//the R part is stored in the upper part of A
//the Q part (normalized so that the upper line is one)
//is stored in the strict lower part of A
function[B]=QRfactorization(A)

[m,n]=size(A);
B=A;
print(%io(2),m,n);
for i=1:n,
    v=householder_vector(B(i:m,i))
    //print(%io(2),v);
    delta=2/(v'*v);
    B(i,i)=B(i,i)-delta* (v'* B(i:m,i));
    for j=i+1:n,
        c=v'*B(i:m,j);
        B(i:m,j)=B(i:m,j)-c*delta*v;
    end ;
    if (i<m) then
        B(i+1:m,i)=v(2:m-i+1);
    end ;
    // print(%io(2),B);
end;

function [y]=up_sweep(A,x)
[m,n]=size(A);
if (m~=n) then
    print(%io(2), "error, not a square matrix");

```

```

else
    y=x;
    y(n)=y(n)/A(n,n);
    for i=n-1:-1:1,
        y(i)=(y(i)-A(i,i+1:n)*y(i+1:n))/A(i,i);
    end;
end;

//assumes A contains the QR-factorized part of A
function [y]=SolveQR(A,x)
[m,n]=size(A);
if (m~=n) then
    print(%io(2), "error, not a square matrix");
else
    y=x;

    for i=1:n-1,
        y(i:n)= householder_reflexion( [1;A(i+1:n,i)] , y(i:n));
    end ;
    y(n)=-y(n);

    y=up_sweep(A,y);
// print(%io(2), y);
end;

```