Université René Descartes Paris 5 — Licence de mathématiques
UFR de mathématiques et informatique — Licence MASS

Méthodes numériques

Partiel du 26 mars 2004

Durée 2h
Documents autorisés : documents distribués et notes personnelles de cours
Calculettes et téléphones portables interdits

I

SCILAB exécute les instructions suivantes. Donner l'affichage obtenu à l'écran.

```
u=[2,2,3,5]
A=u*u'
B=u'*u

C=B;C(1,:)=C(1,:)/C(1,2);
for j=1:4, C(2,j)=C(2,j)/C(2,2); end,
for j=4:-1:1, C(3,j)=C(3,j)/C(3,2); end,
C

C([1,3],[2,4])=zeros(2,2)
C(2:4,2)=C(4,2:4)'

A=[1,2;3,4], [A,A,A], [A;A;A]
A^2, A.^2
L=[1:4],M=[2:5]
L^2,L.^2,L.*M,L*M
D=diag(u)
A=ones(4,4)
A*D
D*A
D*D

k=2,n=size(B,'c')
C=[eye(k,k),zeros(k,n-k);zeros(n-k,k-1),-B(k+1:n,k)/B(k,k),eye(n-k,n-k)]
```

II

Quelle est la complexité du calcul du produit de deux matrices A (de dimensions n et m) et B (de dimensions m et p) ?
Quelle est la complexité du calcul du produit de trois matrices A (de dimensions n et m), B (de dimensions m et p) et C (de dimensions p et q), selon que l'on calcul *(A* B)*C* ou *A*(B *C)* ? Lequel de ces deux calculs est-il préférable ?

III

Les 4 lignes suivantes d'instructions SCILAB donnent dans `s` le même résultat.

(1) `s=0,for i=1:n,u=1/i^2,s=s+u,end`
(2) `s=0;for i=1:n,u(i)=1/i^2;s=s+u(i);end`
(3) `for i=1:n,u(i)=1/i^2;end,s=sum(u)`
(4) `u=ones(1,n)./[1:n].^2,s=sum(u)`

1. Quel est ce résultat ?
2. Quelles sont les avantages et les inconvénients de chaque version ?
3. Quelle est (sont) la (les) versions à éviter quelles que soient les circonstances ? Pour les autres, dire dans quel contexte elles sont à privilégier ou à éviter. Pourquoi ?

IV

A est une matrice carrée de dimension n et D est une matrice diagonale $D = \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix}$

On multiplie A par D à droite. Exprimer les valeurs de $A*D$ en fonction de celles de A et des d_i.
On multiplie A par D à gauche. Exprimer les valeurs de $D*A$ en fonction de celles de A et des d_i.
Exprimer les valeurs de $D\hat{}2$ en fonction des d_i.
Trouver une matrice E dont le carré est égal à D.

V

Effectuer une décomposition LU de la matrice $\begin{bmatrix} 4 & -2 & -6 \\ -2 & 5 & 1 \\ -6 & 1 & 19 \end{bmatrix}$ par la méthode de Gauss.

VI

On cherche à obtenir une décomposition ${}^tCC$ d'une matrice A régulière c'est-à-dire trouver une matrice C telle que $A = {}^tC*C$ où C est une matrice triangulaire supérieure.

1. a) Montrer que ce problème n'est possible que si A est symétrique.
   b) Montrer que, si elle existe, la décomposition est alors unique, aux signes près.

2. a) Montrer que dans l'algorithme de Gauss `gauss0` (rappelé en annexe), avant l'étape k (au sens du cours), la sous-matrice $A(k:n, k:n)$ (notation SCILAB) est symétrique.
   b) Il n'est donc pas nécessaire de calculer, ni même de mémoriser les éléments de la partie inférieure de cette sous-matrice. Modifier le programme en conséquence.
   c) Calculer la complexité du programme ainsi écrit.

3. a) Montrer que dans les programmes `gauss1` et `LU` (rappelés en annexe), avant l'étape $k+1$,
   - $A(k,k)$ est égal au pivot de l'étape k,
   - la colonne $A(k+1:n, k)$ est égale à la ligne $A(k, k+1:n)$ multipliée par le pivot $A(k,k)$
   - la sous-matrice $A(k+1:n , k+1:n)$ est symétrique.

   b) Certaines valeurs pourront donc ne pas être calculées mais elles devront être mémorisées. Modifier le programme `LU` pour tenir compte de cette propriété.
   c) En déduire des relations entre les colonnes de L et les lignes de U de la décomposition LU.

4. a) Définir alors, à partir de la décomposition LU obtenue, une matrice simple D et une décomposition de la forme $A = {}^tU*D*U$, puis une matrice E et une décomposition de la forme $A = {}^t(E*U)*(E*U)$ qui est une décomposition ${}^tCC$.

5. Application à la matrice $\begin{bmatrix} 4 & -2 & -6 \\ -2 & 5 & 1 \\ -6 & 1 & 19 \end{bmatrix}$

Annexes

```
function X = gauss0(A,B)
n=size(A,'c') // calcul de la taille de A et vérif carrée
if ~ size(A,'r')==n  then printf('matrice non carree'), end
A=[A,B]
for k = 1 : n ,   // etape k
    if A(k,k)==0 then // recherche d'un pivot non nul
        printf('pivot nul a l''etape %d',k),
        if k<n then i=k+1,
                    while(A(i,k)==0 & i<n ), i=i+1,
                    end,
               else i=k,
        end,
        if A(i,k)==0 then X='matrice singuliere', return
                     else printf('on echange les lignes %d et %d',k,i),
                          A([k,i],:)=A([i,k],:),
        end,
    end,
    //  nouvelles lignes i
    for i = k+1 : n ,   // nouvelle ligne i
      A(i,k:n+1) = A(i,k:n+1) - A(i,k)*A(k,k:n+1)/A(k,k)
      end,
end,
// remontée, calcul des xi dans le vecteur X
X(n) = A(n,n+1)/A(n,n),
for i=[n-1:-1:1],   // remontee : calcul des xi
    X(i) = (A(i,n+1) - A(i,i+1:n)*X(i+1:n))/A(i,i),
end,
```

```
function X = gauss1(A,B)
n=size(A,'c') // calcul de la taille de A et vérif carrée
if ~ size(A,'r')==n  then printf('matrice non carree'), end
A(:,n+1) = B
for k = 1 : n ,   // etape k
    if A(k,k)==0 then // recherche d'un pivot non nul
        printf('pivot nul a l''etape %d',k),
        if k<n then i=k+1,
                    while(A(i,k)==0 & i<n ), i=i+1,
                    end,
               else i=k,
        end,
        if A(i,k)==0 then X='matrice singuliere', return
                     else printf('on echange les lignes %d et %d',k,i),
                          A([k,i],:)=A([i,k],:),
        end,
    end,
    A(k,k:n+1) = A(k,k:n+1)/A(k,k), // nouvelle ligne k
    for i = k+1 : n ,   // nouvelle ligne i
        A(i,k:n+1) = A(i,k:n+1) - A(i,k)*A(k,k:n+1)
    end,
end,
// remontée, calcul des xi  dans A(:,n+1)
for i=[n-1:-1:1],   // remontee : calcul des xi dans A(i,n+1)
    A(i,n+1) = (A(i,n+1)  - A(i,i+1:n)*A(i+1:n,n+1)),
end,
X=A(:,n+1)
```

```
function [L,U] = LU(A)
n=size(A,'c') // calcul de la taille de A et vérif carrée
if ~ size(A,'r')==n  then printf('matrice non carree'), return,end
for k = 1 : n ,   // etape k
    if A(k,k)==0 then
        printf('pivot nul à l''étape %d',k),
        L='pas de calcul',
        U='',
        return
    end,
   A(k,k+1:n) = A(k,k+1:n)/A(k,k), // nouvelle ligne k
   for i = k+1 : n ,   // nouvelle ligne i
       A(i,k+1:n) = A(i,k+1:n) - A(i,k)*A(k,k+1:n)
   end,
end,
L=zeros(n,n)
for i=1:n,L(i,1:i)=A(i,1:i), end
U=diag(ones(1,n))
for i=1:n,U(i,i+1:n)=A(i,i+1:n), end
```