Informatique et Algorithmique – Chapitre 6

Matrices avec Scilab

Scilab est un outil conçu pour effectuer des calculs numériques sur des matrices : ce sont les éléments de base du langage.

Depuis le début de l'année, nous avons déjà manipulé des cas particuliers de matrices : les vecteurs lignes et plus particulièrement les nombres que Scilab considère comme des matrices de taille 1×1.

I Création et modification d'une matrice en Scilab

1) Création d'une matrices en Scilab...

a) ... par extension

Pour construire une matrice par extension en Scilab, on donne la liste des éléments de la matrice entre crochets et en les séparant par des virgules, chaque ligne étant séparée par des points virgules. La commande `[]` construit une matrice vide.

On retrouve le cas particulier des matrices lignes qui sont les vecteurs que nous manipulons depuis le début de l'année. Si on donne une liste de nombres entre crochets et en les séparant par des points virgules, alors on obtient une matrice colonne.

Exemples :

```
-->A=[1,5,3;-2,0,-4;3,-1,6;5,-1,-1]
 A  =
    1.    5.    3.
  - 2.    0.  - 4.
    3.  - 1.    6.
    5.  - 1.  - 1.
```

```
-->v=[2;3;-1;4;0]
 v  =
    2.
    3.
  - 1.
    4.
    0.
```

```
-->typeof(A)
 ans  =
 constant

-->1==[1]
 ans  =
  T
```

La dernière commande illustre le fait que Scilab considère un nombre comme la matrice de taille 1×1 dont le seul coefficient est ce nombre.

b) ... par blocs

On peut également construire une matrice de taille $n \times p$ par blocs :

- en donnant la liste des n vecteurs constituant ses lignes entre crochets et en les séparant par des points virgules.

  Exemples :

```
-->L1=[3,4,-1,0]; L2=[1,-1,-2,5]; L3=[2,0,0,-3];
-->A=[L1;L2;L3]
 A  =
    3.    4.  - 1.    0.
    1.  - 1.  - 2.    5.
    2.    0.    0.  - 3.
```

- en donnant la liste des p vecteurs constituant ses colonnes entre crochets et en les séparant par des virgules.

Exemples :

```
-->C1=[1;3]; C2=[-1;0]; C3=[0;6]; C4=[-2;4]; C5=[3;-5];
-->A=[C1,C2,C3,C4,C5]
 A  =
    1.  - 1.    0.  - 2.    3.
    3.    0.    6.    4.  - 5.
```

- Les blocs peuvent être de tailles différentes. Mais dans ce cas, il faut bien faire attention à ce que chaque ligne (resp. chaque colonne) ait le même nombre d'éléments.

Exemples :

```
-->A=[[-3,5],1;[1,2;0,-4],[7;-6]]
 A  =
  - 3.    5.    1.
    1.    2.    7.
    0.  - 4.  - 6.
```

c) ... avec des fonctions et opérateurs prédéfinis

On a déjà vu quelques fonctions et opérateurs prédéfinis permettant de créer des vecteurs lignes : si `x` et `y` sont des réels, si `h` est un réel strictement positif et si `n` un entier strictement positif, alors

`x:y`	construit un vecteur constitué des nombres de `x` à `y` par pas de 1.
`x:h:y`	construit un vecteur constitué des nombres de `x` à `y` par pas de `h`.
`linspace(x,y,n)`	construit un vecteur constitué de `n` nombres entre `x` et `y`, régulièrement espacés.

Si `n` et `p` sont deux entiers strictement positif et si `v` est un vecteur, alors

`zeros(n,p)`	construit une matrice de taille `n`×`p` dont tous les coefficients sont nuls.
`ones(n,p)`	construit une matrice de taille `n`×`p` dont tous les coefficients sont égaux à 1.
`diag(v)`	construit une matrice diagonale dont les coefficients diagonaux sont les éléments du vecteur `v`.
`eye(n,p)`	construit une matrice de taille `n`×`p` dont tous les coefficients sont nuls sauf sur la diagonale où ils sont à 1.
`rand(n,p)`	construit une matrice de taille `n`×`p` dont tous les coefficients sont des nombres tirés *aléatoirement* dans $]0,1[$.

Exemples :

```
-->zeros(3,2)
 ans  =
    0.    0.
    0.    0.
    0.    0.

-->ones(2,5)
 ans  =
    1.    1.    1.    1.    1.
    1.    1.    1.    1.    1.

-->u=[1,-2,3]; diag(u)
 ans  =
    1.    0.    0.
    0.  - 2.    0.
    0.    0.    3.
```

```
-->eye(3,4)
 ans  =
    1.    0.    0.    0.
    0.    1.    0.    0.
    0.    0.    1.    0.

-->rand(4,3)
 ans  =
    0.2113249   0.6653811   0.8782165
    0.7560439   0.6283918   0.0683740
    0.0002211   0.8497452   0.5608486
    0.3303271   0.6857310   0.6623569
```

2) Accès aux éléments d'une matrices

Si `A` est une matrice et si `i` et `j` sont des entiers strictement positifs, alors

`size(A)`	renvoie la taille de la matrice (sous la forme d'un vecteur ligne).
`length(A)`	renvoie le nombre de coefficients de la matrice.
`A(i,j)`	renvoie le coefficient d'indice $(\mathtt{i},\mathtt{j})$ (ou un message d'erreur si l'indice ne correspond pas à la taille de la matrice).
`A(i,:)`	renvoie la $\mathtt{i}^{\text{ième}}$ ligne de la matrice.
`A(:,j)`	renvoie la $\mathtt{j}^{\text{ième}}$ colonne de la matrice.

Exemples :

```
-->A=[1,5,3,-2;0,-4,3,-1;6,2,-1,-1];
-->size(A)
 ans  =
    3.    4.

-->length(A)
 ans  =
    12.

-->A(2,3)
 ans  =
    3.
```

```
-->A(4,2)
      !--error 21
Index invalide.

-->A(3,:)
 ans  =
    6.    2.  - 1.  - 1.

-->A(:,1)
 ans  =
    1.
    0.
    6.
```

Plus généralement, si on désire extraire les coefficients de `A` qui se trouvent sur les lignes $\mathtt{i}_1,\ldots,\mathtt{i}_k$ et les colonnes $\mathtt{j}_1,\ldots,\mathtt{j}_\ell$, alors on utilise la commande $\boxed{\mathtt{A([i_1,\ldots,i_k],[j_1,\ldots,j_\ell])}}$. Notons que l'ordre des indices a son importance.

```
-->A([1,3],2)
 ans  =
    5.
    2.

-->A(1,2:4)
 ans  =
    5.    3.  - 2.
```

```
-->A([3,2],[1,2,3])
 ans  =
    6.    2.  - 1.
    0.  - 4.    3.

-->A([1,1],[3,4,1])
 ans  =
    3.  - 2.    1.
    3.  - 2.    1.
```

3) Modification d'une matrice

Soient `A` une matrice de taille $n \times p$, `L` un vecteur ligne de longueur p et `C` un vecteur colonne de longueur n. Soient `i` et `j` sont des entiers strictement positifs. Alors :

`A(i,j)=x`	remplace le coefficient d'indice (`i`,`j`) de `A` par le réel `x`.
`A=[A,C]`	ajoute la colonne `C` à la matrice `A` en dernière position.
`A=[C,A]`	ajoute la colonne `C` à la matrice `A` en première position.
`A=[A;L]`	ajoute la ligne `L` à la matrice `A` en dernière position.
`A=[L;A]`	ajoute la ligne `L` à la matrice `A` en première position.
`A(i,:)=[]`	supprime la $\mathtt{i}^{\text{ième}}$ ligne de la matrice `A`.
`A(:,j)=[]`	supprime la $\mathtt{j}^{\text{ième}}$ colonne de la matrice `A`.

Exemples :

```
-->A=[1,2,3,4;5,6,7,8]; A(2,3)=11
 A  =
    1.    2.    3.    4.
    5.    6.    11.    8.
```

```
-->A=[A,[-1;0]]
 A  =
    1.    2.    3.    4.  - 1.
    5.    6.    11.    8.    0.
```

```
-->A=[[-4,-3,-2,-1,0];A]
 A  =
 - 4.  - 3.  - 2.  - 1.    0.
    1.    2.    3.    4.  - 1.
    5.    6.    11.    8.    0.
```

```
-->A(2,:)=[]; A(:,4)=[]
 A  =
 - 4.  - 3.  - 2.    0.
    5.    6.    11.    0.
```

Plus généralement, si B est une matrice de taille $k \times \ell$, alors la commande

$$\boxed{\texttt{A([i}_1,\ldots,\texttt{i}_k\texttt{],[j}_1,\ldots,\texttt{j}_\ell\texttt{])=B}}$$

remplace la sous-matrice formée des lignes $\texttt{i}_1,\ldots,\texttt{i}_k$ et les colonnes $\texttt{j}_1,\ldots,\texttt{j}_\ell$ par la matrice B.

⚠ Lorsqu'on désire modifier plusieurs coefficients d'une matrice avec une seule commande, il faut faire très attention à la taille des matrices (qui doivent être de tailles égales de part et d'autre du symbole =).

Exemple :

```
-->B=ones(3,5); B([2,3],[2,4])=rand(2,2)
 B  =
    1.    1.          1.    1.          1.
    1.    0.2693125   1.    0.4051954   1.
    1.    0.6325745   1.    0.9184708   1
```

II Opérations sur les matrices

1) Opérations matricielles

Scilab permet de réaliser des opérations matricielles usuelles. Si A et B sont deux matrices, si x est un réel et si k est un entier naturel, alors :

A'	calcule la transposée de A.
x*B	calcule la multiplication de la matrice A par le scalaire x.
A+B	calcule la somme (si elle a un sens) des matrices A et B.
A*B	calcule le produit matriciel (si il a un sens) des matrices A et B.
A∧k	calcule la puissance $\texttt{k}^{\text{ième}}$ de la matrice A (si c'est une matrice carrée).

Exemples :

```
-->A=[1,2,-3,0;-1,0,2,4;0,1,3,-5];
-->size(A);
ans  =
    3.    4.

-->A', size(A')
ans  =
    1.  - 1.    0.
    2.    0.    1.
  - 3.    2.    3.
    0.    4.  - 5.
ans  =
    4.    3.

-->B=[1,0,3;2,4,-6];
-->C=[1,-1,5;10,3,7];
```

```
-->B+C
 ans  =
    2.   - 1.    8.
    12.    7.    1.

-->X=[1,0;-2,5;0,-2;4,3]; A*X
 ans  =
  - 3.     16.
    15.    8.
  - 22.  - 16.

-->X*A
    !--error 10
Multiplication incohérente.

-->D=[1,2;-1,3]; D^3
 ans  =
  - 9.     22.
  - 11.    13.
```

2) Opérations terme à terme

On peut également effectuer des opérations termes à termes sur des matrices A et B de même taille :

A.*B	effectue le produit terme à terme des matrices A et B.
A./B	effectue la division terme à terme de la matrice A par la matrice B.
A.∧k	élève chaque coefficient de la matrice A à la puissance k.
A.∧B	effectue la puissance terme à terme de la matrice A par la matrice B.

Les fonctions usuelles (`log, exp, sqrt, abs, floor, cos, sin, tan`) s'appliquent également aux vecteurs coordonnée par coordonnée.

Exemples :

```
-->E=[0,1;2,-4]; E.*D
 ans  =
    0.    6.
  - 2.  - 12.

-->E*D
 ans  =
  - 3.    9.
    6.  - 8.
```

```
-->E./D
 ans  =
    0.    1.5
  - 2.  - 1.3333333

-->D.^3
 ans  =
    1.    8.
  - 1.    27.
```

```
-->cos(D)
 ans  =
    0.5403023 - 0.4161468
    0.5403023 - 0.9899925

-->sqrt(abs(E))
 ans  =
    0.          1.7320508
    1.4142136   2.
```

3) Opérateurs de comparaison sur les matrices

Les opérateurs de comparaison et les opérateurs logiques s'appliquent aussi aux matrices de même taille. Les opérations se font terme à terme et le résultat est une matrice de même taille contenant des variables booléennes. Rappelons les opérateurs de comparaison :

==	test d'égalité	>	supérieur strict	<	inférieur strict
<>	différent	>=	supérieur ou égal	<=	inférieur ou égal

et les opérateurs permettant de réaliser des tests logiques :

&	opérateur et	|	opérateur ou	$\sim$	opérateur non

Exemples :

```
-->A=[2,0,-4;1,-2,3];

-->B=[-3,3,1;4,5,-2];
```

```
-->~(A<B)
 ans  =
  T F F
  F F T
```

Notons que l'on peut aussi comparer une matrice avec un nombre réel (chaque coefficient de la matrice est comparé avec le réel et le résultat est une matrice de même taille contenant des variables booléennes).

Si `M` est une matrice composée de variables booléennes, alors la commande `find(M)` renvoie un vecteur contenant les positions des coefficients de `M` ayant la valeur T.

Exemples :

```
-->A>0
 ans  =
  T F F
  T F T
```

```
-->find(A>0)
 ans  =
    1.    2.    6.
```

```
-->length(find(A>0))
 ans  =
    3.
```

Ainsi la commande `length(find(M))` renvoie le nombre de coefficients de `M` ayant la valeur T.

4) Fonctions spécifiques sur les matrices

Terminons par quelques fonctions prédéfinies très utiles sur les matrices :

`min(A)`	renvoie la valeur minimale des coefficients de `A`.
`max(A)`	renvoie la valeur maximale des coefficients de `A`.
`sum(A)`	calcule la somme de tous les coefficients de `A`.
`prod(A)`	calcule le produit de tous les coefficients de `A`.

Dans les fonctions précédentes, on peut ajouter l'option `'c'` pour que la fonction opère sur chaque ligne (et retourne un vecteur colonne), et l'option `'r'` pour que la fonction opère sur chaque colonne (et retourne un vecteur ligne).

Exemples :

```
-->A=[1,2,-1,6;7,-3,2,-4];
-->min(A)
 ans  =
  - 4.

-->max(A)
 ans  =
    7.
```

```
-->sum(A)
 ans  =
    10.

-->sum(A,'c')
 ans  =
    8.
    2.
```

```
-->sum(A,'r')
 ans  =
    8.  - 1.    1.    2.

-->prod(A)
 ans  =
  - 2016.
```

III Inversion de matrices et résolution de $AX = B$

1) Calcul de l'inverse d'une matrice

Si `A` est une matrice carrée inversible, alors la commande `inv(A)` calcule l'inverse de `A`. Si `A` n'est pas une matrice inversible, alors cette commande renvoie un message d'erreur.

Exemples :

```
-->A=[1,2,0,3;-3,-5,2,-3;2,0,1,2;-1,-7,3,-5];

-->inv(A)
 ans  =
  - 0.75  - 0.125    0.625  - 0.125
    3.5   - 1.25   - 2.75     1.75
    5.    - 1.5    - 3.5      2.5
  - 1.75    0.875    1.625  - 1.125

-->B=[1,-1,2;3,-4,0;4,-5,2]; inv(B)
      !--error 19
Le problème est singulier.
```

2) Rang et base du noyau

Si `A` est une matrice, alors

`rank(A)`	renvoie le rang de la matrice `A`.
`kernel(A)`	renvoie une base du noyau `A`.

Exemples :

```
-->rank(A), rank(B)
 ans  =
    3.
 ans  =
    2.
```

```
-->kernel(B)
 ans  =
    0.3764123
    0.0684386
    0.9239210
```

```
-->C=[1,5,4,0;-3,2,-1,1;-2,7,3,1];

-->rank(C)
 ans  =
    2.
```

```
-->kernel(C)
 ans  =
    0.0120886   0.5813933
  - 0.2680226   0.3877964
    0.3320062 - 0.6300939
    0.9043172   0.338493
```

IV Résolution de $AX = B$

Rappelons qu'un système de n équations à p inconnues est équivalent à résoudre $AX = B$ avec A la matrice associée au système et B un vecteur colonne de taille n.

Si `A` (resp. `B`) est la matrice A (resp. le vecteur colonne B) écrite en langage Scilab, alors la commande

```
[X0,noyau]=linsolve(A,-B)
```

affecte à la variable `X0` une solution particulière du système $AX = B$ (si il en existe) et à la variable `noyau` une base du noyau de la matrice `A`. Les solutions sont les vecteurs colonnes qui s'écrivent comme la somme de `X0` et d'une combinaison linéaire des colonnes du `noyau`.

Si A est inversible, alors il existe une unique solution qui est donnée par le vecteur $A^{-1}B$.

Exemples :

```
-->A=[1,2,0,3;-3,-5,2,-3;
          2,0,1,2;-1,-7,3,-5];
-->B=[2;-7;6;1]
-->[X0,noyau]=linsolve(A,-B)
 noyau  =
     []
 X0  =
    3.
    1.
    2.
  - 1.

-->inv(A)*B
 ans  =

    3.
    1.
    2.
  - 1.
```

```
-->A=[1,5,4,0;-3,2,-1,1;-2,7,3,1];
-->[X0,noyau]=linsolve(A,-[-2;1;3])
AVERTISSEMENT : Contraintes linéaires
en conflit.
 noyau  =
     []
 X0  =
     []

-->[X0,noyau]=linsolve(A,-[5;1;6])
 noyau  =
    0.5713227 - 0.1084187
    0.3239782 - 0.3424330
  - 0.5478034   0.4551459
    0.5182084   0.8147559
 X0  =
  - 0.0144928
    0.6666667
    0.4202899
    0.0434783
```