David Michel
david.michel@ens-rennes.fr
perso.eleves.ens-rennes.fr/people/David.Michel

ECE 1
Lycée Saint-Vincent
2016-2017

TP6 : Matrices – Suite – Corrigé

Q.1 On pourrait entrer les coefficients un à un comme on l'a vu au TP précédent, mais ce serait fastidieux. On utilise plutôt l'une des fonctions définies dans l'énoncé, en l'occurrence, la fonction **zeros** qui permet de construire une matrice nulle. Il faut alors remplir cette matrice de façon adéquate. Pour cela, on exprime les coefficients de A :

$$a_{i,j} = \begin{cases} 1 & \text{si } j = i+1,\ i \in [\![1, n-1]\!] \\ 0 & \text{sinon} \end{cases}$$

On peut maintenant définir A dans Scilab :

```
A=zeros(10,10)
for i=1:10
   for j=1:10
      if j=i+1 then
         A(i,j)=1
      end
   end
end
```

ou, plus simplement,

```
A=zeros(10,10)
for i=1:9
   A(i,i+1)=1
end
```

Q.2 L'instruction **[n,p]=size(A)** attribue effectivement la valeur 10 aux variables **n** et **p**.

Q.3 L'exécution des instructions, avec **b=[1 ;1]** produit le même résultat : le vecteur colonne égal à une valeur approchée de $\frac{1}{7}\begin{pmatrix}1\\5\end{pmatrix}$, qui est bien solution de $AX = \begin{pmatrix}1\\1\end{pmatrix}$.

Q.4 On commence par définir une matrice de taille 1000 ne contenant que des 1, puis on remplace les coefficients diagonaux par 0.

```
A=ones(1000,1000)
for i=1:1000
   A(i,i)=0
end
b=ones(1000,1)
```

L'exécution de la série d'instruction permet de constater que résoudre une équation par **A\b** est la méthode la plus rapide. Ceci s'explique par le fait qu'on n'a pas besoin de calculer l'inverse de A pour résoudre le système. Diverses méthodes existent. Dans la suite du TP, on s'intéresse à la méthode dite du pivot de Gauss.

Q.5 Examinons un exemple en dimension 3. On veut résoudre

$$\begin{cases} a_{1,1}x_1 & + & a_{1,2}x_2 & + & a_{1,3}x_3 & = & b_1 \\ & & a_{2,2}x_2 & + & a_{2,3}x_3 & = & b_2 \\ & & & & a_{3,3}x_3 & = & b_3 \end{cases}$$

où la matrice $A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & a_{2,2} & a_{2,3} \\ 0 & 0 & a_{3,3} \end{pmatrix}$ est inversible, c'est-à-dire $a_{1,1}a_{2,2}a_{3,3} \neq 0$. En commençant par la dernière ligne, on résout facilement ce système en écrivant :

$$\begin{cases} x_3 & = & \frac{b_3}{a_{3,3}} \\ x_2 & = & \frac{b_2 - a_{2,3}x_3}{a_{2,2}} \\ x_1 & = & \frac{b_1 - a_{1,2}x_2 - a_{1,3}x_3}{a_{1,1}}. \end{cases}$$

De manière générale, pour $A \in \mathcal{M}_n(\mathbb{R})$ triangulaire supérieure inversible de taille n quelconque, on peut écrire les mêmes formules pour résoudre $Ax = b$:

$$\begin{cases} x_n & = & \dfrac{b_n}{a_{n,n}} \\ x_{n-1} & = & \dfrac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}} \\ \vdots & & \\ x_i & = & \dfrac{b_i - \displaystyle\sum_{j=i+1}^{n} a_{i,j}x_j}{a_{i,i}} \qquad \forall i \in \{n-1, n-2, \ldots, 3, 2, 1\} \\ \vdots & & \\ x_1 & = & \dfrac{b_1 - \displaystyle\sum_{j=2}^{n} a_{1,j}x_j}{a_{1,1}} \end{cases}$$

Dans Scilab, cela donne :

```
function x=SolTriangulaireSup(A,b)
    n=length(b)     //on détermine la dimension en jeu
    x=zeros(n,1)    //on définit globalement notre vecteur solution comme un vecteur colonne
                    //de taille n, initialement rempli de 0 et qu'il va maintenant falloir
                    // modifier.
    for i=n:-1:1   //on part de la dernière ligne
    //on commence par calculer le numérateur de la fraction
       x(i)=b(i)   //premier terme
       for j=i+1:n
           x(i)=x(i)-A(i,j)*x(j)
       end
       //puis on divise par le dénominateur
       x(i)=x(i)/A(i,i)
    end
endfunction
```

On teste le programme, par exemple avec $A = I_3$ et $b = \begin{pmatrix}1\\1\\1\end{pmatrix}$ et on obtient bien $x = \begin{pmatrix}1\\1\\1\end{pmatrix}$. Si $A = \begin{pmatrix}1 & 1 & 1\\0 & 2 & 1\\0 & 0 & 1\end{pmatrix}$ et $b = \begin{pmatrix}1\\0\\1\end{pmatrix}$ on obtient $x = \begin{pmatrix}0,5\\-0,5\\1\end{pmatrix}$ et on vérifie que $Ax = b$.

Q.6 Il s'agit de parcourir la k-ième colonne, de la ligne k à la ligne n et de trouver qui a la plus grande valeur absolue. On va donc garder en mémoire le numéro de ligne qui, au cours du parcours de la colonne, a son coefficient plus grand que les autres en valeur absolue.

```
function numLigne=pivot(B,k)
    [n,p]=size(B)    //on a besoin de connaître le nombre n de lignes
    numLigne=k       //on commence à la ligne k. Pour l'instant, le coefficient en ligne k
                     //est le plus grand des coefficients parcourus
                     //(on n'en a pas parcouru d'autre...)
    for i=k+1:n    //on parcourt le reste de la colonne
        if abs(B(i,k))>abs(B(numLigne,k)) then
            numLigne=i    //si on trouve un coefficient plus grand, on met à jour la
                          //variable contenant le numéro de la ligne de plus grand coefficient
        end
    end
endfunction
```

On teste sur quelques exemples en dimension 3. Si $B = \begin{pmatrix}5 & 5 & -2\\1 & 2 & 0\\2 & -3 & 0\end{pmatrix}$ alors **pivot(B,1)** renvoie bien 1 (5 est en ligne 1 et est plus grand que 1 et 2), **pivot(B,2)** renvoie bien 3 (-3 est en ligne 3 et est plus grand en valeur absolue que 2 (on ne parcourt la colonne qu'à partir de la ligne 2)) et **pivot(B,3)** renvoie bien 3 puisqu'on ne parcourt la colonne qu'à partir de la ligne 3.

On aurait pu utiliser la fonction maximum de Scilab qui renvoie deux valeurs : la valeur du maximum et l'indice où il est atteint. Pour procéder ainsi, il faut indiquer à Scilab de comparer les valeurs absolue des indices $(k+1, k)$ à (n, k) grâce à **abs(B(k+1 :n,k))** :

```
function numLigne=pivot(B,k)
    [n,p]=size(B)
    [m,i]=max(abs(B(k+1:n,k)))
    numLigne=k+i
endfunction
```

Q.7 On suit l'énoncé et on copie A dans une nouvelle matrice B puis on fait le changement de coefficients sur les lignes à échanger :

```
function B=Echange(A,i,k)
    [n,p]=size(A)    //on va devoir parcourir des lignes, donc on doit connaître p
    B=A              //on copie A
    for j=1:p            //on parcourt...
        B(i,j)=A(k,j)    //...la ligne i et on y met les valeurs de la ligne k
```

```
        B(k,j)=A(i,j)    //...la ligne k et on y met les valeurs de la ligne i
    end
endfunction
```

Scilab permet de faire des opérations sur les lignes facilement : on aurait pu écrire

```
function B=Echange(A,i,k)
    [n,p]=size(A)
    B=A
    B(i,:)=A(k,:)     //on place la ligne k dans la ligne i
    B(k,:)=A(i,:)     //on place la ligne i dans la ligne k
endfunction
```

où les " :" signifient qu'on parcourt toute la colonne (on aurait pu écrire **1:p**).

Q.8 On suit l'algorithme de l'énoncé. On doit modifier les lignes L_{k+1} à L_n. On écrit donc une boucle portant sur l'indice i de la ligne qu'on modifie. Puis, on fait la modification décrite par l'énoncé en parcourant les colonnes de la matrice.

```
function C=Elimine(B,k)
    C=B
    [n,p]=size(B)    //on aura besoin de connaître le nombre de lignes et de colonnes
    for i=k+1:n     //on parcourt les lignes numérotées de k+1 à n
        for j=1:p   //on parcourt maintenant les colonnes
            C(i,j)=C(i,j)-B(i,k)/B(k,k)*C(k,j)   //on fait les modifications demandées
        end
    end
endfunction
```

En utilisant les capacités de Scilab à manipuler les lignes, on préférera écrire

```
function C=Elimine(B,k)
    C=B
    [n,p]=size(B)
    for i=k+1:n
        C(i,:)=C(i,:)-B(i,k)/B(k,k)*C(k,:)
    end
endfunction
```

Q.9 Il suffit maintenant de mettre les fonctions précédentes les unes après les autres, selon l'algorithme de l'énoncé.

```
function B=Gauss(M)
    B=M
    [n,p]=size(B)
    for k=1:n-1
        i0=pivot(B,k)
        B=Echange(B,i0,k)
```

```
        B=Elimine(B,k)
    end
endfunction
```

Q.10 Examinons un exemple de résolution de système linéaire, en taille 3.

$$\begin{cases} a_{1,1}x_1 & + & a_{1,2}x_2 & + & a_{1,3}x_3 & = & b_1 \\ a_{2,1}x_1 & + & a_{2,2}x_2 & + & a_{2,3}x_3 & = & b_2 \\ a_{3,1}x_1 & + & a_{3,2}x_2 & + & a_{3,3}x_3 & = & b_3 \end{cases}$$

Appliquons l'algorithme du pivot de Gauss. Disons qu'on choisit le premier pivot comme étant $a_{1,1}$. Il faut donc éliminer les coefficients $a_{2,1}$ et $a_{3,1}$. Pour cela, on soustrait à la deuxième équation la première multipliée par $\frac{a_{2,1}}{a_{1,1}}$, puis on soustrait à la troisième équation la première multipliée par $\frac{a_{3,1}}{a_{1,1}}$. On fait donc les mêmes opérations sur les coefficients $(a_{i,j})$ que sur les (b_i). On applique en fait l'algorithme du pivot de Gauss sur la matrice $\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & b_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & b_2 \\ a_{3,1} & a_{3,2} & a_{3,3} & b_3 \end{pmatrix}$. À la fin de l'algorithme, on obtient une matrice de la forme $\begin{pmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ 0 & p_{2,2} & p_{2,3} & p_{2,4} \\ 0 & 0 & p_{3,3} & p_{3,4} \end{pmatrix}$ et sait que résoudre $Ax = b$ est équivalent à résoudre $Tx = V$ où $T = (p_{i,j})_{1 \leq i,j \leq 3}$ et $V = \begin{pmatrix} p_{1,4} \\ p_{2,4} \\ p_{3,4} \end{pmatrix}$. Mais T est triangulaire supérieure, donc on peut appliquer la fonction précédente.

```
function x=Solution(A,b)
    n=length(b)   //on a besoin de connaître la dimension n

//Comme indiqué dans l'énoncé et motivé ci-dessus, on accole la colonne b à la matrice A
//dans une nouvelle matrice C ∈ M_{n,n+1}(ℝ)
    C=zeros(n,n+1)   //on commence par définir une matrice de taille nx(n+1)
    for i=1:n   //on copie ensuite les valeurs de A et b où il faut
        for j=1:n
           C(i,j)=A(i,j)
        end
        C(i,n+1)=b(i)
    end

//On applique l'algorithme de Gauss et on extrait les matrices T et V
    P=Gauss(C)
    T=zeros(n,n)
    V=zeros(n,1)
    for i=1:n
        for j=1:n
            T(i,j)=P(i,j)
        end
        V(i)=B(i,n+1)
    end
```

```
//On résout le système triangulaire Tx=V
    x=SoltriangulaireSup(T,V)
endfunction
```