UNIVERSITÉ
DE REIMS
CHAMPAGNE-ARDENNE

Rapport MINF0402 TP1

Algèbre linéaire

2022/2023

AHNOUDJ Lina
KRUK Timothé

Sommaire

Exercice 4 :

Question 1 : RESOUINF

La fonction RESOUINF renvoie la solution X de l'équation AX=b, avec A, une matrice triangulaire inférieure inversible. Elle prendra en paramètres :

- A, une matrice carrée de taille n.
- b, un vecteur colonne à n lignes.
- n, la taille de la matrice.

Code et algorithme :

<table>
<tr><th>Algorithme</th><th>Code</th></tr>
<tr><td>Pour i allant de 1 à n
tmp <- 0

Pour j allant de 1 à i-1
tmp <- tmp +A(i,j)*X(j)
Fin Pour

X(i) <- (b(i)-tmp) / A(i,i)
Fin Pour</td><td>

```
function [x]=RESOUINF(A, b, n)
    x=[n] // Correspond au resultat
    for i = 1:n // Boucle qui parcout les lignes
        tmp=0
        for j = 1:(i-1) // Boucle qui parcout les colonnes
            tmp=tmp+(A(i, j)*x(j))
        end
        x(i)=(b(i)-tmp)/A(i,i)
    end
endfunction
```

</td></tr>
</table>

Méthode et raisonnement :

On suppose une matrice A(n, n), X(n, 1) et Y(n, 1) avec n un entier supérieur à 0, avec A une matrice triangulaire inférieure inversible. A et Y sont connus, on cherche X.

On a donc

$$A = \begin{pmatrix} a_{1,1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{pmatrix} \qquad X = \begin{pmatrix} x_{1,1} \\ \vdots \\ x_{n,1} \end{pmatrix} \qquad Y = \begin{pmatrix} y_{1,1} \\ \vdots \\ y_{n,1} \end{pmatrix}$$

Tel que $\begin{pmatrix} a_{1,1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{pmatrix} \times \begin{pmatrix} x_{1,1} \\ \vdots \\ x_{n,1} \end{pmatrix} = \begin{pmatrix} y_{1,1} \\ \vdots \\ y_{n,1} \end{pmatrix}$

On cherche la formule générale pour arriver à l'algorithme.

Pour la ligne 1.

On a $a_{1,1}x_{1,1}=y_{1,1}$

Donc $x_{1,1}=y_{1,1}/a_{1,1}$

Pour la ligne 2.

On a $a_{1,1}x_{1,1}+a_{1,2}x_{2,1}=y_{2,1}$

Donc $x_{2,1}$=($y_{2,1}$-$a_{1,1}x_{1,1}$)/$a_{1,2}$

Pour la ligne n.

On a $a_{n,1}x_{1,1}$+…+$a_{n,n-1}x_{n-1,1} + a_{n,n}x_{n,1}$=$y_{n,1}$

Donc $x_{n,1}$=($y_{n,1} - a_{n,1}x_{1,1}$+…+$a_{n,n-1}x_{n-1,1}$)/$a_{n,n}$

On remarque une récurrence

$$\begin{pmatrix} a_{1,1} & & \cdots & & 0 \\ \vdots & & \ddots & & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n-1} & a_{n,n} \end{pmatrix} \times \begin{pmatrix} x_{1,1} \\ x_{2,1} \\ \vdots \\ x_{n-1,1} \\ x_{n,1} \end{pmatrix} = \begin{pmatrix} y_{1,1} \\ y_{2,1} \\ \vdots \\ y_{n-1,1} \\ y_{n,1} \end{pmatrix}$$

En effet on remarque que pour calculer le terme $x_{n,1}$il faut prendre le terme rouge, donc le y a la ligne n ($y_{n,1}$) et lui soustraire tmp=$a_{n,1}x_{1,1}$+…+$a_{n,n-1}x_{n-1,1}$, c'est-à-dire on doit faire une opération où l'on utilise que les termes bleus. Les termes bleus qui sont pour la ligne n, les termes a allant du premier terme de la ligne n ($a_{n,1}$) jusqu'au terme de la colonne n-1 ($a_{n,n-1}$). Pour les termes bleus on a également besoin des termes x allant du premier terme ($x_{1,1}$) jusqu'au terme n-1 ($x_{n-1,1}$). Ensuite on a besoin tu terme rouge. Enfin la dernière opération consiste à diviser par le terme a d'indice (n, n) ($a_{n,n}$).

Il est à noter que même si nous cherchons les termes x, quand nous sommes à la ligne n, on connait déjà tous les termes allant de $x_{1,1}$ à $x_{n-1,1}$ ce qui permet d'avoir juste une équation à une inconnue. Pour la ligne 1, comme on obtient une équation à une inconnue ce qui nous permet de calculer $x_{1,1}$, puis à la ligne 2 on a donc tous les termes pour calculer $x_{2,1}$. Ceci permet d'arriver ensuite jusqu'au terme d'indice n ($x_{n,1}$).

Pour mettre en place l'algorithme on comprend qu'il nous faut d'abord une première boucle i allant de 1 à n (for i = 1 : n) puisque nous devons faire des « opérations » sur toute les lignes. Ainsi i représentera l'indice pour la ligne i.

Ensuite nous devons calculer tmp pour le soustraire à $y_{n,1}$, on a besoin d'une autre boucle j allant de 1 à (i-1) (for j = i-1) car on a vu que les termes bleus s'arrêtais à la colonne n-1. J représentera donc l'indice des colonnes.

Dans cette boucle j, on doit additionner les termes d'indice (i, j) entre eux. A noter la multiplication entre les termes $a_{i,j}$ et $x_{i,j}$ d'abord. Donc il faut faire tmp<-tmp+$a_{i,j} \times x_{i,j}$ (tmp=tmp+(A(i, j)*x(j))). Il faut également initialiser tmp à 0 avant de rentrer dans la boucle j (tmp=0).

Une fois sortit de la boucle j on peut calculer le terme $x_{i,j}$ il suffit de faire $x_{i,j}$<-($y_{i,1}$-tmp)/ $a_{i,i}$ (x(i)=(b(i)-tmp)/A(i,i)).

On retournera x le résultat que l'on a préalablement initialisé à x=[n] au début de la fonction.

Dans l'algorithme x et b qui représente respectivement X et Y sont des tableaux de taille n à une dimension.

Tests :

Pour vérifier le code on a utilisé la fonction créée d'une part et on a stockée le résultat (matrice X) dans la variable reponse, d'autre part on a exécuté la commande A*reponse qui doit donner donc b.

```
chdir("C:\Users\timot\Documents\Ecole\L2\MInfo402\tp1")
exec("Fonctions.sci")

/* VARIABLES */
tempsAttente = 2000 // Temps d attente en ms pour les sleep()
n = 5 // Taille des matrices
b = 100*rand(1, n) // Matrice a 1 ligne et n colonnes pour en faire un vecteur
A = 100*rand(n,n) // Matrice a n lignes et n colonnes
A=tril(A) // On transforme la matrice A en matrice inferieur

reponse=RESOUINF(A, b, n) // Matrice reponse correspon au resultat
disp("matrice A")
disp(A)
sleep(tempsAttente)
disp("matrice b")
disp(b') // On utilise la transpose de b pour l afficher en colonne
sleep(tempsAttente)
disp("matrice Reponse")
disp(reponse)
sleep(tempsAttente)
disp("Verification doit etre egale a b")
disp(A*reponse) // Puisqu on fait AX=b et que l on a trouve X qui est reponse
    // il faut faire A*reponse pour trouver b
```

Exécution :

On remarque que A*reponse donne effectivement b donc le code fonctionne correctement (reponse=X).

```
   62.839179   0.          0.          0.         0.
   84.974524   66.235694   0.          0.         0.
   68.573102   72.635068   21.646326   0.         0.
   87.821648   19.851438   88.338878   31.2642    0.
   6.8374037   54.425732   65.251349   36.16361   59.350947

  "matrice b"

   21.132487
   75.604385
   0.0221135
   33.032709
   66.538110

  "matrice Reponse"

   0.3362948
   0.7100084
  -3.4467821
   9.4001741
  -1.5069861

  "Verification doit etre egale a b"

   21.132487
   75.604385
   0.0221135
   33.032709
   66.538110
```

Question 2 : RESOUSUP

La fonction RESOUSUP renvoie la solution X de l'équation AX=b, avec A, une matrice triangulaire supérieure inversible. Elle prendra en paramètres :

- A, une matrice carrée de taille n.
- b, un vecteur colonne à n lignes.
- n, la taille de la matrice.

Code et algorithme :

<table>
<tr><th>Algorithme</th><th>Code</th></tr>
<tr><td>Pour i descendant de n à 1
tmp <- 0

Pour j descendant de n à i+1
tmp <- tmp +A(i,j)*X(j)
Fin Pour

X(i) <- (b(i)-tmp) / A(i,i)
Fin Pour</td><td><pre>function [x]=RESOUSUP(A, b, n)
    x=[n] // Correspond au resultat
    for i = n:-1:1
        tmp=0
        for j = n:-1:(i+1)
            tmp=tmp+(A(i, j)*x(j))
        end
        x(i)=(b(i)-tmp)/A(i,i)
    end
endfunction</pre></td></tr>
</table>

Méthode et raisonnement :

Pour résoudre Ax = b avec cette fois une matrice A inférieure, l'algorithme et la logique est la même, en revanche il faudra modifier le parcourt des boucles pour. En effet pour avoir une première équation avec une seule inconnue il faut donc parcourir en commençant par la dernière ligne jusqu'à la première ligne. Il faut faire une boucle Pour i allant de n à 1 par -1 (for i = n:-1:1).

Enfin la deuxième boucle j pour les colonnes devra la aussi se faire par la fin jusqu'au terme avant la diagonale donc ne pas prendre le terme (i,i). Il s'agit par conséquent d'une boucle Pour i allant de n à i+1 Par -1 (for j = n:-1:(i+1)).

Tests :

Pour tester le bon fonctionnement du code on utilise la même méthode expliquée précédemment (cf. RESOUINF)

```
exec("Fonctions.sci")

/* VARIABLES */
tempsAttente = 2000 // Temps d attente en ms pour les sleep()
n = 5 // Taille des matrice
b = 100*rand(1, n) // Matrice aleatoire a une ligne et n colonnes pour en faire un vec
teur
A = 100*rand(n,n) // Matrice aleatoire a n lignes et n colonnes
A=triu(A) // On transforme la matrice A en matrice superieur

reponse=RESOUSUP(A, b, n)
disp("matrice A")
disp(A)
sleep(tempsAttente)
disp("matrice b")
disp(b') // On utilise la transpose de b pour l afficher en colonne
sleep(tempsAttente)
disp("matrice Reponse")
disp(reponse)
sleep(tempsAttente)
disp("Verification doit etre egale a b")
disp(A*reponse) // Puisqu on fait AX=b et que l on a trouve X qui est reponse
    // il faut faire A*reponse pour trouver b
```

Exécution :

```
 91.847078   28.06498    68.56896    40.948255   58.961773
 0.          12.800585   15.312167   87.841258   68.539797
 0.          0.          69.708506   11.383597   89.062247
 0.          0.          0.          19.983377   50.422128
 0.          0.          0.          0.          34.936154

"matrice b"

 50.153416
 43.685876
 26.931248
 63.257449
 40.519540

"matrice Reponse"

 1.4832411
-3.0806511
-1.1345228
 0.2390494
 1.1598168

"Verification doit etre egale a b"

 50.153416
 43.685876
 26.931248
 63.257449
 40.519540
```

Exercice 5 :

Question 1 : REDUC

La fonction REDUC qui fait référence à « Réduction de Gauss » renvoie les matrices A et B, avec A, une matrice triangulaire supérieure inversible. Ax=b sera alors le nouveau système obtenu suite à la réduction de gausse. La fonction prendra en paramètres :

- A : une matrice carrée de taille n.
- b : un vecteur colonne à n lignes.
- n : taille de la matrice.

Code et algorithme :

<table>
<tr><th>Algorithme</th><th>Code</th></tr>
<tr><td>Pour i allant de 1 à n-1
  Pour j allant de i+1 à n
    tmp <- X(j,i)/X(i,i)
    X(i,k)<- 0
    Pour k allant de 1 à n
      X(j,k)<- X(j,k)-X(i,k)*tmp
  // autrement X(j,k)<- X(j,k)-X(i,k)/X(i,i)*X(j,i)
    Fin Pour
    b(j) <-b(j)- b(i)*tmp
    b(j) <-b(j)- b(i)/X(i,i)*X(j,i)
  Fin Pour
Fin Pour</td><td><pre>function [x, bNouveau]=REDUC(A, b, n)
    x=A // Correspond a la nouvelle matrice A
    bNouveau=b // Correspond a la nouvelle matrice B
    for i = 1:n-1
        for j = i+1 : n
            tmp = (1/x(i,i))*x(j,i)
            for k = 1:n
                x(j,k)=x(j,k)-x(i,k)*tmp // notre tmp ne fonction
ne pas, on ne sait pas pourquoi
                x(j,k)=x(j,k)-x(i,k)/x(i,i)*x(j,i)
            end
            bNouveau(j)=bNouveau(j)-bNouveau(i)*tmp
            bNouveau(j)=bNouveau(j)-bNouveau(i)/x(i,i)*x(j,i)
        end
    end
endfunction</pre></td></tr>
</table>

Méthode et raisonnement :

On suppose une matrice A(n, n), X(n, 1) et Y(n, 1) avec n un entier supérieur à 0, avec A une matrice triangulaire inférieure inversible. A et Y sont connus, on cherche X.

On a donc

$$A = \begin{pmatrix} a_{1,1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{pmatrix} \qquad X = \begin{pmatrix} x_{1,1} \\ \vdots \\ x_{n,1} \end{pmatrix} \qquad Y = \begin{pmatrix} y_{1,1} \\ \vdots \\ y_{n,1} \end{pmatrix}$$

Tel que $\begin{pmatrix} a_{1,1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{pmatrix} \times \begin{pmatrix} x_{1,1} \\ \vdots \\ x_{n,1} \end{pmatrix} = \begin{pmatrix} y_{1,1} \\ \vdots \\ y_{n,1} \end{pmatrix}$

On sait que l'on doit choisir pivot et les pivots ont pour indice, $a_{1,1}$, $a_{2,2}$, …, $a_{i,i}$, …, $a_{n,n}$. Les pivots représentent donc la diagonale de la matrice. Pour chaque pivot il faut donc que pour les lignes en dessous il n'y ait que des 0 sur la colonne du pivot.

Ainsi pour la ligne i avec 1 <= i <= n.

Le pivot sera donc $a_{i,i}$. Pour mettre un zéro sur la ligne en dessous à la colonne i. Il faudra donc faire L_{i+1}<--$L_{i+1} - \frac{L_i}{a_{i,i}} * a_{i+1,i}$. Ainsi après cette opération on aura $a_{i+1,i} = a_{i+1,i} - \frac{a_{i+i,i}}{a_{i+i,i}} * a_{i+1,i} = a_{i+1,i} - 1 * a_{i+1,i} = 0$. On continue pour toute les lignes en dessous.

Pour cette algorithme on comprend qu'il faudra déjà une première boucle pour i allant de 1 à n-1 (for i = 1 : n), ainsi cela permettra de manipuler tout les pivots. On n'a pas besoin d'aller jusqu'à n puisque arriver à la ligne n on aura donc plus de ligne en dessous, on aura donc terminer l'algorithme.

Il nous faut une seconde boucle pour j allant de i+1 à n (for i = i+1 : n), l'indice j représente les indices des lignes en dessous du pivot.

Enfin on a besoin d'une dernière boucle pour k allant de 1 à n où k représente les indices des colonnes. On a besoin de toute les colonnes pour faire l'opération L_{i+1}<--$L_{i+1} - \frac{L_i}{a_{i,i}} * a_{i+1,i}$. On devra donc faire l'instruction x(j,k)=x(j,k)-x(i,k)/x(i,i)*x(j,i) dans cette boucle. Expliquons le choix des indices :

> x(j,k) : j indice des lignes en dessous du pivot, k indice des colonnes à modifier
>
> x(i,k) : i indice de la ligne du pivot, donc k indice de la colonne où on est
>
> x(i,i) : terme qui est notre pivot
>
> x(j,i) : terme où le 0 sera placé

A la fin de cette boucle il restera à également modifier la matrice b, l'instruction est donc presque identique : bNouveau(j)=bNouveau(j)-bNouveau(i)/x(i,i)*x(j,i). On note juste que pour bNouveau est un vecteur ce qui permet de l'exclure de la boucle k.

On peut observer la récurrence d'opération x(i,i)*x(j,i), on peut donc calculer ceci avant de rentrer dans la boucle k pour optimiser. On fait tmp= (1/x(i,i))*x(j,i). Néanmoins nous n'utiliserons pas tmp dans le programme car ça ne marche alors que d'un point de vue mathématique cela devrait fonctionner.

On notera x la matrice A modifié après la méthode le pivot de Gauss, et bNouveau la matrice B également modifié.

Tests :

La fonction renvoie A et B après l'utilisation de la réduction de Gauss, pour tester la fonction on a stocké ses résultats respectivement dans les variables ANouveau et bNouveau, puis on compare bNouveau avec ANouveau*x qui est censé être égal à BNouveau, autrement dit A*x = b donc ANouveau*x = bNouveau, dans le test on a utilisé x' pour avoir un vecteur colonne afin de pouvoir réaliser la multiplication (car la matrice x créée correspond à une matrice ligne).

```
/* VARIABLES */
tempsAttente = 2000 // Temps d attente en ms pour les sleep()
n=5 // Taille des matrices
A = 100*rand(n,n) // Matrice aleatoire a n lignes et n colonnes
x = 100*rand(1,n) // Matrice aleatoire a 1 ligne et n colonnes po
ur en faire un vecteur

[ANouveau, bNouveau]=REDUC(A, A*x', n)
disp("matrice A")
disp(A)
sleep(tempsAttente)
disp("matrice x")
disp(x)
sleep(tempsAttente)
disp("matrice A apres avoir fait le pivot de Gauss")
disp(ANouveau)
sleep(tempsAttente)
disp("matrice b apres avoir fait le pivot de Gauss")
disp(bNouveau)
sleep(tempsAttente)
disp("matrice ANouveau*x doit etre egale a b reponse")
disp(ANouveau*x')
sleep(tempsAttente)
```

Exécution :

On remarque que les résultats sont identiques.

```
 "matrice x"

  28.553642   86.075146   84.941017   52.570608   99.312099

 "matrice A apres avoir fait le pivot de Gauss"

  38.737788   73.409406   53.762298   4.855662    58.787202
  0.         -148.61941  -116.00089   55.678879  -91.671905
  0.          0.         -7.7528561  -40.375757  -41.554511
  0.          0.          0.         -215.51414  -186.94862
  0.          0.          0.          0.         -127.72326

 "matrice b apres avoir fait le pivot de Gauss"

  18085.000
 -28822.727
 -6907.9793
 -29895.969
 -12684.465

 "matrice ANouveau*x doit etre egale a b reponse"

  18085.000
 -28822.727
 -6907.9793
 -29895.969
 -12684.465
```

Question 2 : GAUSS

La fonction GAUSS renvoie x la solution de l'équation Ax=b, à partir de la matrice triangulaire supérieure A et la matrice b en utilisant les fonctions REDUC et RESOUSUP. La fonction prendra en paramètres :

- A une matrice carré inversible aléatoire de taille n
- b un vecteur colonne de n lignes, aléatoires.

Code :

Code

`function X=GAUSS(A, b, n)`
`    [A,b]=REDUC(A,b,n)`
`    X=RESOUSUP(A,b,n)`
`endfunction`

Méthode et raisonnement :

Pour résoudre l'opération Ax=b, on devra d'abord transformer la matrice A en matrice triangulaire, donc on utilise la fonction REDUC : [ANouveau, bNouveau] =REDUC(A, A*x', n). A noter la transposé de la matrice x puisque x est une matrice (1, n) dans le programme. Enfin on peut réutiliser ma fonction RESOUSUP pour trouver la matrice X.

Tests :

Pour la vérification de la fonction GAUSS, on doit s'assurer que son exécution avec A et b(A*x') en paramètre donne la même matrice aléatoire X précédemment utilisée,

ou on peut le faire manuellement :

1. On exécute la fonction en utilisant deux matrice aléatoires A et b (GAUSS(A, b, n))
2. On calcul X manuellement
3. On compare les résultats obtenus.

```
disp("question 2")
disp("matrice x")
disp(x') //transpose pour avoir un vecteur colonne
sleep(tempsAttente)
XGauss=GAUSS(A, A*x', n)
disp("XGauss doit etre égale a la matrice x")
disp(XGauss)
sleep(tempsAttente)
```

Exécution :

```
"question 2"

"matrice x"

  67.446978
  91.528744
  2.8485976
  23.678415
  70.153436

"XGauss doit etre égale a la matrice x"

  67.446978
  91.528744
  2.8485976
  23.678415
  70.153436
```