TP : feuille 5. Les Tableaux (corrigé)

Université Paris-Nord, Institut Galilée. DEUG MIAS1, 2002/03

Le pivot de Gauss.

Le but de cet exercice est d'implémenter un programme permettant de résoudre un système du type

$$AX = B,$$

où

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,N} \\ \vdots & \vdots & & \vdots \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N} \end{pmatrix}$$

et

$$B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix}.$$

sont connus.

Pour ce faire, on utilisera la méthode du pivot de Gauss. Par exemple, dans le cas du système

$$\begin{pmatrix} 1 & 2 & 0 \\ 2 & 1 & 2 \\ 0 & 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix},$$

on utilisera la matrice C constituée de A et B accolés, plus précisément:

$$C = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 2 & 1 & 2 & 2 \\ 0 & 2 & 1 & 3 \end{pmatrix}.$$

Puis, on utilise la première ligne $L_1 = (1, 2, 0, 1)$ de C pour faire apparaître des zéros dans la première colonne des autres lignes de C. Dans notre exemple, il suffit de remplacer L_2 par $L_2 - 2 * L_1$. On obtient la matrice

$$C = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 0 & -3 & 2 & 0 \\ 0 & 2 & 1 & 3 \end{pmatrix}.$$

On utilise ensuite L_2 pour faire apparaître un zéro sur la deuxième colonne de C, dans L_3. Pour cela, on remplace L_3 par $L_3 - \frac{2}{-3} * L_2$. On obtient

$$C = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 0 & -3 & 2 & 0 \\ 0 & 0 & \frac{7}{3} & 3 \end{pmatrix}.$$

On dit que l'on a ainsi triangularisé le système. On peut ensuite calculer ses solutions en commençant par

$$x_3 = \frac{3}{\frac{7}{3}} = \frac{9}{7} = 1.285714,$$

puis

$$x_2 = \frac{0 - 2 * x_3}{-3} = \frac{18}{21} = 0.857143$$

et enfin

$$x_1 = \frac{1 - 0 * x_3 - 2 * x_2}{1} = -\frac{15}{21} = -0.714285.$$

D'une façon générale:

- Pour triangulariser: pour chaque colonne j (de 1 à N)
  - On remplace toutes les lignes L_i (pour $i > j$) par la ligne

$$L_i \leftarrow L_i - \frac{c_{i,j}}{c_{j,j}} L_j \tag{1}$$

    où $c_{i,j}$ désigne l'élément de C situé sur la ligne i et la colonne j. Si l'un des $c_{j,j}$ est nul, on arrête la triangularisation.

- Pour résoudre le système triangularisé: On vérifie d'abord que tous les $c_{i,i}$ sont différents de 0. Si ce n'est pas le cas, on dira que le système n'a pas de solution. Si c'est le cas, pour chaque ligne i, pour i allant de N à 1
  - On calcule

$$somme = \sum_{j=i+1}^{N} c_{i,j} x_i = c_{i,i+1} x_{i+1} + c_{i,i+2} x_{i+2} + ... + c_{i,N} x_N.$$

  - On calcule

$$x_i = \frac{c_{i,N+1} - somme}{c_{i,i}}.$$

  Alors, le vecteur

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}$$

  est la solution du système $AX = B$.

1. Écrire un programme permettant à un utilisateur de saisir une matrice carrée et un vecteur de la taille de cette matrice. L'utilisateur devra dans un premier temps dire la taille de la matrice qu'il veut saisir. Cette matrice aura une taille maximum donnée par une constante TAILLEMAX.

   On enregistrera la matrice et le vecteur sous la forme d'une matrice similaire à C.

2. Compléter le programme afin qu'il triangularise le système. Afficher la matrice triangularisée et vérifier sur l'exemple donné en introduction que votre programme marche convenablement.

3. Compléter le programme afin qu'il calcule la solution du système.

4. Compléter le programme afin de pouvoir calculer le produit AX. Vérifier sur plusieurs exemples que vous trouvez bien le vecteur B entré par l'utilisateur. Il se peut que vous observiez des petites erreurs. On les appelle les erreurs numériques. La question suivante vise à les réduire.

5. Pour avoir plus de précision (et pour éviter le cas où le pivot $c_{j,j}$, de l'étape (1), est nul), on veut intervertir, avant d'effectuer (1), la ligne L_j avec la ligne L_k, avec $k \geq j$ tel que le coefficient $c_{k,j}$ soit, en valeur absolue, le plus grand possible.

   Inspirez vous de l'exercice 4 de la feuille de TD 10 pour modifier votre programme afin qu'il effectue correctement cette interversion.

```
#include <stdio.h>
#define TAILLEMAX 100
#define DEBUG 1

int main(){
  float matrice[TAILLEMAX][TAILLEMAX+1];
  float matrice_init[TAILLEMAX][TAILLEMAX];
  float lignetmp[TAILLEMAX+1];
  float solution[TAILLEMAX];
  float verification[TAILLEMAX];
  float tmp;
  int taille;
  int i,j,jtmp;
  int imax,ipivot;
  float max;
  int matriceInversible;

  /* saisie d'une matrice carree */

  printf("Quelle est la taille de la matrice carree?\n");
  scanf("%d",&taille);

  for(i=0;i<taille;i++)  for(j=0;j<taille;j++)
    {printf("matrice(%d,%d)=",i+1,j+1);
    scanf("%f",&matrice[i][j]);
    matrice_init[i][j]=matrice[i][j];
    }
  for(i=0;i<taille;i++)
    {printf("u(%d)= ",i+1);
    scanf("%f",&matrice[i][taille]);
    }



  /* affichage de la matrice */
  printf("La solution du systeme \n");
  for(i=0;i<taille;i++)
   {for(j=0;j<taille;j++)
     printf("%f \t",matrice[i][j]);
   printf("\t %f ",matrice[i][taille]);
   printf("\n");
   }

  /* pivot de Gauss */
  /*premiere etape: triangularisation */
  for(j=0;j<taille-1;j++)
    {/* trouve l'indice du meilleur pivot */
      max=matrice[j][j];
      imax=j;
```

```
    for(i=j;i<taille;i++)
      {if(max<matrice[i][j])
        {max=matrice[i][j];
        imax=i;
        }
      else if(max< -matrice[i][j])
        {max=-matrice[i][j];
        imax=i;
        }
      }
    if(max>0)
      {/* intervertie les lignes imax et j */
        for(jtmp=0;jtmp<=taille;jtmp++)
          lignetmp[jtmp]=matrice[j][jtmp];
        for(jtmp=0;jtmp<=taille;jtmp++)
          matrice[j][jtmp]=matrice[imax][jtmp];
        for(jtmp=0;jtmp<=taille;jtmp++)
          matrice[imax][jtmp]=lignetmp[jtmp];
        /* fait le pivot */
        for(i=j+1;i<taille;i++)
          {tmp=matrice[i][j]/matrice[j][j];
          for(jtmp=j;jtmp<=taille;jtmp++)
            matrice[i][jtmp]=matrice[i][jtmp]-tmp*matrice[j][jtmp];
          }
      }
  }
/* affichage de la matrice triangulaire */
printf("est la meme que celle du systeme \n");
for(i=0;i<taille;i++)
  {for(j=0;j<taille;j++)
    printf("%f \t",matrice[i][j]);
  printf("\t %f ",matrice[i][taille]);
  printf("\n");
  }


/*deuxieme etape: calcul de la solution */
matriceInversible=1;
for(i=0;i<taille;i++)
  if(matrice[i][i]==0)
    matriceInversible=0;

if(matriceInversible==1)
  for(i=taille-1;i>=0;i--)
    {tmp=0.;
    for(j=i+1;j<taille;j++)
      tmp=tmp+matrice[i][j]*solution[j];
    solution[i]=(matrice[i][taille]-tmp)/matrice[i][i];
    }
else
  printf("qui n'a pas de solution\n");

/* affichage de la solution */
printf("dont la solution est \n");
```

```
  for(j=0;j<taille;j++)
    printf("%f\n",solution[j]);

  /* produit de la matrice de depart avec la solution */
  if(DEBUG==1)
    {printf("VERIFICATION\n");
    for(i=0;i<taille;i++)
      {verification[i]=0.;

      for(j=0;j<taille;j++)
        verification[i]=verification[i]+matrice_init[i][j]*solution[j];

      printf("%f\n",verification[i]);
      }
    }

  return 0;
}
```