

0.0.1 Implémentation de l'algorithme de Gauss en passant par le système d'équations linéaires

Code source

Voici mon implémentation de l'algorithme de Gauss, qui n'utilise pas la matrice augmentée. En effet, l'algorithme travaille directement avec le système d'équations linéaires $Ax=B$.

```
0 #include <stdio.h>
1 #include <string.h>
2 #include <stdlib.h>
3
4 /*
5 *CREATE A 2D FLOAT MATRIX
6 */
7
8 float** createMatrix(int row, int column){
9     float **mat=NULL;
10    mat=malloc(row* sizeof(int*));
11    if(mat==NULL){return NULL;}
12    for (int i=0; i<row; i++){
13        mat[i]=malloc(column* sizeof(int));
14        if(mat[i]==NULL){
15            for(int j=0; j<i; j++){
16                free(mat[j]);
17                return NULL;
18            }
19        }
20    }
21    return mat;
22 }
23
24 /*
25 *PRINT A 2D FLOAT MATRIX
26 */
27
28 void printMatrix(float **mat, int row, int column){
29     for (int i=0; i<row; i++){
30         for(int j=0; j<column; j++){
31             printf("%f ", mat[i][j]);
32         }
33         printf("\n");
34     }
35 }
36
37 /*
38 *FREE A 2D FLOAT MATRIX
39 */
40
41 void freeMatrix(float **mat, int row){
42     for(int i=0; i<row; i++){
43         free(mat[i]);
44     }
45     free(mat);
}
```

```

46 }
47
48 /*
49 *COMPLETE A 2D FLOAT MATRIX FROM USER INPUT
50 */
51
52 void completeMatrix(float **mat, int row, int column){
53     for (int i=0; i<row; i++){
54         for(int j=0; j<column; j++){
55             printf("Coefficient at M_%d,%d:   ", i+1, j+1);
56             scanf("%f", &mat[i][j]);
57         }
58     }
59 }
60
61 /*
62 *GENERATE A COLUMN VECTOR "B" FROM A 2D FLOAT MATRIX "A"
63 */
64
65 void generateB(float **matA, float **matB, int row, int column){
66     for(int i=0; i<row; i++){
67         float sum=0;
68         for(int j=0; j<column; j++){
69             sum+=matA[i][j];
70         }
71         matB[i][0]=sum;
72     }
73 }
74
75 /*
76 *PERFORM GAUSSIAN ELIMINATION ON A Ax=B MATRIX SYSTEM OF LINEAR EQUATIONS
77 */
78
79 void gauss(float** matA, float** matb, int size){
80     for(int k=0; k<size-1; k++){
81         for(int i=k+1; i<size; i++){
82             float alpha=matA[i][k]/matA[k][k];
83             for(int j=k; j<size; j++){
84                 matA[i][j]=matA[i][j]-alpha*matA[k][j];
85             }
86             matb[i][0]=matb[i][0]-alpha*matb[k][0];
87         }
88     }
89 }
90
91 /*
92 *SOLVE A MATRIX SYSTEM OF LINEAR EQUATIONS USING BACKWARD SUBSTITUTION
93 */
94 void resolution(float** matA, float** matb, float** matx, int size){
95     matx[size-1][0]=matb[size-1][0]/matA[size-1][size-1];
96     for (int i=size-2; i>=0; i--){
97         float sum=0;
98         for(int j=i+1; j<size; j++){
99             sum+=matA[i][j]*matx[j][0];
100     }

```

```

101         matx[i][0]=(1/matA[i][i])*(matb[i][0]-sum);
102     }
103 }
104
105 int main(){
106
107     //A Matrix
108     int rowA;
109     int columnA;
110     printf("\nRow count of matrix A : ");
111     scanf("%d", &rowA);
112     printf("\nColumn count of matrix A : ");
113     scanf("%d", &columnA);
114
115     float** Amatrix=createMatrix(rowA, columnA);
116     completeMatrix(Amatrix, rowA, columnA);
117     puts("\n          A matrix \n");
118     printMatrix(Amatrix, rowA, columnA);
119
120     //B Matrix
121     float** Bmatrix=createMatrix(rowA, 1);
122     generateB(Amatrix, Bmatrix, rowA, columnA);
123     puts("\n          B matrix \n");
124     printMatrix(Bmatrix, rowA, 1);
125
126     //X Matrix
127     float** Xmatrix=createMatrix(rowA, 1);
128
129     //Matrix Triangularization
130     puts("\n          TRIANGULARIZATION \n");
131     gauss(Amatrix, Bmatrix, rowA);
132     puts("\n          A Matrix \n");
133     printMatrix(Amatrix, rowA, columnA);
134     puts("\n          B Matrix \n");
135     printMatrix(Bmatrix, rowA, 1);
136
137     //Solve the system
138     puts("\n          SOLVING \n");
139     resolution(Amatrix, Bmatrix, Xmatrix, rowA);
140     puts("\n          SOLUTION VECTOR X \n");
141     printMatrix(Xmatrix, rowA, 1);
142
143     //Free
144     freeMatrix(Amatrix, rowA);
145     freeMatrix(Bmatrix, rowA);
146     freeMatrix(Xmatrix, rowA);
147     return 0;
148 }

```

Commentaires sur les fonctions usuelles de manipulation de matrices

Ce code implémente diverses fonctions pour travailler avec des matrices à coefficients en nombre flottants.

- La fonction ***createMatrix*** alloue dynamiquement de la mémoire pour créer une matrice de nombres flottants avec un nombre spécifié de lignes et de colonnes.
- La fonction ***printMatrix*** affiche les éléments d'une matrice de nombres flottants.
- La fonction ***freeMatrix*** libère la mémoire allouée pour une matrice de nombres flottants.
- La fonction ***completeMatrix*** permet à l'utilisateur de saisir des valeurs pour remplir les éléments d'une matrice de nombres flottants.
- La fonction ***generateB*** génère un vecteur colonne B en fonction de la somme des éléments de chaque ligne de la matrice A .

Commentaires sur les fonctions résolvant notre système linéaire $Ax = B$ à l'aide de l'algorithme de Gauss

Dans le cadre de notre résolution de systèmes d'équations linéaires, deux fonctions jouent des rôles clefs dans ce code : la fonction ***gauss*** et la fonction ***resolution***.

- La fonction ***gauss*** joue un rôle important dans la préparation de la résolution de notre système d'équations linéaires. En effectuant l'élimination de Gauss sur la matrice A , elle la transforme en une matrice triangulaire supérieure. Cela signifie que les éléments sous la diagonale principale de la matrice deviennent tous des zéros, simplifiant ainsi la résolution du système. De plus, la fonction met également à jour la matrice B en conséquence, garantissant que notre système $Ax = B$ reste équilibré.
- La fonction ***resolution***, quant à elle, prend en charge la résolution effective du système linéaire une fois que la matrice A a été triangulée par la fonction ***gauss***. Elle utilise la méthode de substitution pour calculer la solution et stocke le résultat dans le vecteur X . Cette étape finale permet d'obtenir les valeurs des variables inconnues du système, fournissant ainsi la solution recherchée pour le problème initial.

En combinant ces deux fonctions avec celles citées dans la sous-section ??, le code réalise un processus complet de résolution de systèmes d'équations linéaires de manière efficace et précise (aux erreurs d'arrondies près).

Entrées utilisateur

En premier lieu dans notre programme, nous avons besoin de spécifier le système $Ax = B$ à l'ordinateur. Pour ce faire, nous allons dans l'ordre :

1. Allouer une matrice A en mémoire. Cette matrice verra sa taille définie par la première entrée utilisateur du programme (nous demanderons consécutivement le nombre de lignes, puis le nombre de colonnes de la matrice).

2. Définir les coefficients de la matrice A . Il s'agira de la deuxième entrée utilisateur de notre programme.

Par définition de notre fonction ***completeMatrix***, nous remplirons la matrice dans l'ordre suivant:

$a_{1,1}, a_{1,2}, \dots, a_{1,n}$, puis $a_{2,1}, \dots, a_{2,n}$, jusque $a_{n,1}, \dots, a_{n,n}$

3. Allouer une matrice B en mémoire. À noter que la taille de B est définie automatiquement en fonction de la taille de A . Nous avons $A \in \mathcal{M}_{n,p} \Rightarrow B \in \mathcal{M}_{n,1}$.

4. Définir les coefficients de la matrice B . Chaque coefficient prendra la valeur de la somme des éléments de la ligne respective de la matrice A .

Nous avons donc:

Soient $A \in \mathcal{M}_{n,p}$ et $B \in \mathcal{M}_{n,1}, \forall i \in \{1, n\}, b_{i,1} = \sum_{j=1}^p a_{i,j}$.

5. Allouer une matrice X en mémoire. Cette matrice aura la même taille que la matrice B . Ces coefficients ne seront pas définis pour le moment.

En guise d'exemple, le système matriciel $AX = B$ suivant:

$$\begin{pmatrix} 3 & 0 & 4 \\ 7 & 4 & 2 \\ -1 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 13 \\ 2 \end{pmatrix} \quad (1)$$

est représenté par l'entrée utilisateur:

Listing 1: User Input

```

0 Row count of matrix A : 3
1
2 Column count of matrix A : 3
3
4             FILL IN THE VALUE OF MATRIX A
5
6 Value for a_1,1: 3
7 Value for a_1,2: 0
8 Value for a_1,3: 4
9 Value for a_2,1: 7
10 Value for a_2,2: 4
11 Value for a_2,3: 2
12 Value for a_3,1: -1
13 Value for a_3,2: 1
14 Value for a_3,3: 2

```

Une fois toutes les matrices initialisées et complétées, nous pouvons attaquer la résolution du système par la triangularisation du système. Ceci fait, nous résolverons le système obtenu pour obtenir notre vecteur X solution.

Affichage Console

Dès lors le système $AX = B$ connu par l'ordinateur, ce dernier peut retrouver les valeurs de la matrice X . Voici l'affichage produit par notre programme en console:

Listing 2: Console Display of the Gauss elimination for the AX

```

0      A matrix
1
2      3.000000    0.000000    4.000000
3      7.000000    4.000000    2.000000
4      -1.000000    1.000000    2.000000
5
6      B matrix
7
8      7.000000
9      13.000000
10     2.000000
11
12     TRIANGULARIZATION
13     A Matrix
14
15     3.000000    0.000000    4.000000
16     0.000000    4.000000    -7.333333
17     0.000000    0.000000    5.166667
18
19     B Matrix
20
21     7.000000
22     -3.333332
23     5.166667
24
25     SOLVING
26     SOLUTION VECTOR X
27
28     1.000000
29     1.000000
30     1.000000

```

Il est à repérer que le programme affiche dans cet ordre:

- La Matrice A
- La Matrice B
- La Matrice A une fois triangulée supérieure
- La Matrice B une fois mise à jour en conséquence pour que le système reste équilibré
- La Matrice X solution du système

Remarque: le temps d'exécution de ce programme a été de 0.000237 secondes

Exemples d'exécutions

Soient les matrices suivantes données dans le TP:

$$A_2 = \begin{pmatrix} -3 & 3 & -6 \\ -4 & 7 & 8 \\ 5 & 7 & -9 \end{pmatrix}, A_4 = \begin{pmatrix} 7 & 6 & 9 \\ 4 & 5 & -4 \\ -7 & -3 & 8 \end{pmatrix}, A_6 = \begin{pmatrix} -3 & 3 & -6 \\ -4 & 7 & 8 \\ 5 & 7 & -9 \end{pmatrix}$$

On obtient respectivement les résultats suivants:

Listing 3: $A_2X = B$ results

```

0      A matrix
1
2      -3.000000    3.000000    -6.000000
3      -4.000000    7.000000    8.000000
4      5.000000    7.000000    -9.000000
5
6      B matrix
7
8      -6.000000
9      11.000000
10     3.000000
11
12     TRIANGULARIZATION
13     A Matrix
14
15     -3.000000    3.000000    -6.000000
16     0.000000    3.000000    16.000000
17     0.000000    0.000000    -83.000000
18
19     B Matrix
20
21     -6.000000
22     19.000000
23     -83.000000
24
25     SOLVING
26     SOLUTION VECTOR X
27
28     1.000000
29     1.000000
30     1.000000
31
32     Temps d'execution : 0.000250 secondes

```

Listing 4: $A_4X = B$ results

```

0      A matrix
1
2      7.000000    6.000000    9.000000
3      4.000000    5.000000    -4.000000
4      -7.000000    -3.000000    8.000000
5
6      B matrix
7
8      22.000000
9      5.000000
10     -2.000000
11
12     TRIANGULARIZATION
13     A Matrix
14
15     7.000000    6.000000    9.000000
16     0.000000    1.571428    -9.142858
17     0.000000    0.000000    34.454552
18
19     B Matrix
20
21     22.000000
22     -7.571429
23     34.454548
24
25     SOLVING
26     SOLUTION VECTOR X
27
28     1.000001
29     0.999999
30     1.000000
31
32     Temps d'execution : 0.000231 secondes

```

Listing 5: $A_6X = B$ results

```

0      A matrix
1
2      -3.000000    3.000000    -6.000000
3      -4.000000    7.000000    8.000000
4      5.000000    7.000000    -9.000000
5
6      B matrix
7
8      -6.000000
9      11.000000
10     3.000000
11

```

```

12          TRIANGULARIZATION
13      A Matrix
14
15      -3.000000    3.000000    -6.000000
16      0.000000    3.000000    16.000000
17      0.000000    0.000000    -83.000000
18
19          B Matrix
20
21      -6.000000
22      19.000000
23      -83.000000
24
25          SOLVING
26      SOLUTION VECTOR X
27
28      1.000000
29      1.000000
30      1.000000
31
32      Temps d'execution : 0.000246 secondes

```

On remarquera que sur le calcul de A_4 , on tombe sur des valeur extrêmement proche de 1. Ceci est provoqué à cause des erreurs d'arrondis provoqués par l'encodage des nombres flottants.