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.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*
*CREATE A 2D FLOAT MATRIX
*/

float** createMatrix(int row, int column){
        float **mat=NULL;
        mat=malloc(row* sizeof(int*));
        if(mat==NULL){return NULL;}
        for (int i=0; i<row; i++){
                mat[i]=malloc(column* sizeof(int));
                if(mat[i]==NULL){
                        for(int j=0; j<i; j++){
                                free(mat[j]);
                                return NULL;
                        }
                }
        }
        return mat;
}

/*
*PRINT A 2D FLOAT MATRIX
*/

void printMatrix(float **mat, int row, int column){
        for (int i=0; i<row; i++){
                for(int j=0; j<column; j++){
                        printf("%f    ", mat[i][j]);
                }
                printf("\n");
        }
}

/*
*FREE A 2D FLOAT MATRIX
*/

void freeMatrix(float **mat, int row){
        for(int i=0; i<row;i++){
                free(mat[i]);
        }
        free(mat);
}

/*
*COMPLETE A 2D FLOAT MATRIX FROM USER INPUT
*/

void completeMatrix(float **mat, int row, int column){
        for (int i=0; i<row; i++){
                for(int j=0; j<column; j++){
                        printf("Coefficient at M_%d,%d:    ", i+1, j+1);
                        scanf("%f", &mat[i][j]);
                }
```

```c
	}
}

/*
*GENERATE A COLUMN VECTOR "B" FROM A 2D FLOAT MATRIX "A"
*/

void generateB(float **matA, float **matB, int row, int column){
	for(int i=0; i<row; i++){
		float sum=0;
		for(int j=0; j<column; j++){
			sum+=matA[i][j];
		}
		matB[i][0]=sum;
	}
}

/*
*PERFORM GAUSSIAN ELIMINATION ON A Ax=B MATRIX SYSTEM OF LINEAR EQUATIONS
*/

void gauss(float** matA, float** matb, int size){
	for(int k=0; k<size-1; k++){
		for(int i=k+1; i<size; i++){
			float alpha=matA[i][k]/matA[k][k];
			for(int j=k; j<size; j++){
				matA[i][j]=matA[i][j]-alpha*matA[k][j];
			}
			matb[i][0]=matb[i][0] - alpha*matb[k][0];
		}
	}
}

/*
*SOLVE A MATRIX SYSTEM OF LINEAR EQUATIONS USING BACKWARD SUBSTITUTION
*/
void resolution(float** matA, float** matb, float** matx, int size){
	matx[size-1][0]=matb[size-1][0]/matA[size-1][size-1];
	for (int i=size-2; i>=0; i--){
		float sum=0;
		for(int j=i+1; j<size; j++){
			sum+=matA[i][j]*matx[j][0];
		}
		matx[i][0]=(1/matA[i][i])*(matb[i][0]-sum);
	}
}

int main(){

	//A Matrix
	int rowA;
	int columnA;
	printf("\nRow count of matrix A : ");
	scanf("%d", &rowA);
	printf("\nColumn count of matrix A : ");
	scanf("%d", &columnA);

	float** Amatrix=createMatrix(rowA, columnA);
	completeMatrix(Amatrix, rowA, columnA);
	puts("\n                A matrix \n");
	printMatrix(Amatrix, rowA, columnA);

	//B Matrix
	float** Bmatrix=createMatrix(rowA, 1);
	generateB(Amatrix, Bmatrix, rowA, columnA);
	puts("\n                B matrix \n");
	printMatrix(Bmatrix, rowA, 1);

	//X Matrix
```

```c
        float** Xmatrix=createMatrix(rowA, 1);

        //Matrix Triangularization
        puts("\n                TRIANGULARIZATION \n");
        gauss(Amatrix, Bmatrix, rowA);
        puts("\n                A Matrix \n");
        printMatrix(Amatrix, rowA, columnA);
        puts("\n                B Matrix \n");
        printMatrix(Bmatrix, rowA, 1);

        //Solve the system
        puts("\n                SOLVING \n");
        resolution(Amatrix, Bmatrix, Xmatrix, rowA);
        puts("\n                SOLUTION VECTOR X \n");
        printMatrix(Xmatrix, rowA, 1);

        //Free
        freeMatrix(Amatrix, rowA);
        freeMatrix(Bmatrix, rowA);
        freeMatrix(Xmatrix, rowA);
        return 0;
}
```

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 0.0.1, 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:
   $a_{1,1}, a_{1,2}, ..., a_{1,n},$ puis $a_{2,1}, ...a_{2,n},$ jusque $a_{n,1}, ..., a_{n,n}$

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

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:

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

est représenté par l'entrée utilisateur:

Listing 1: User Input

```
Row count of matrix A : 3

Column count of matrix A : 3

                    FILL IN THE VALUE OF MATRIX A

Value for a_1,1:    3
Value for a_1,2:    0
Value for a_1,3:    4
Value for a_2,1:    7
Value for a_2,2:    4
Value for a_2,3:    2
Value for a_3,1:    -1
Value for a_3,2:    1
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

```
                    A matrix

3.000000    0.000000    4.000000
7.000000    4.000000    2.000000
-1.000000    1.000000    2.000000

                    B matrix

7.000000
13.000000
2.000000

                    TRIANGULARIZATION
                    A Matrix

3.000000    0.000000    4.000000
0.000000    4.000000    -7.333333
0.000000    0.000000    5.166667

                    B Matrix

7.000000
-3.333332
5.166667

                    SOLVING
                    SOLUTION VECTOR X

1.000000
1.000000
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

```
                A matrix

-3.000000    3.000000    -6.000000
-4.000000    7.000000    8.000000
5.000000    7.000000    -9.000000

                B matrix

-6.000000
11.000000
3.000000

                TRIANGULARIZATION
                A Matrix

-3.000000    3.000000    -6.000000
0.000000    3.000000    16.000000
0.000000    0.000000    -83.000000

                B Matrix

-6.000000
19.000000
-83.000000

                SOLVING
                SOLUTION VECTOR X

1.000000
1.000000
1.000000

Temps d'execution : 0.000250 secondes
```

Listing 4: $A_4X = B$ results

```
                A matrix

7.000000    6.000000    9.000000
4.000000    5.000000    -4.000000
-7.000000    -3.000000    8.000000
```

```

                 B matrix

22.000000
5.000000
-2.000000

                 TRIANGULARIZATION
                 A Matrix

7.000000    6.000000    9.000000
0.000000    1.571428    -9.142858
0.000000    0.000000    34.454552

                 B Matrix

22.000000
-7.571429
34.454548

                 SOLVING
                 SOLUTION VECTOR X

1.000001
0.999999
1.000000

Temps d'execution : 0.000231 secondes
```

Listing 5: $A_6X = B$ results

```
                 A matrix

-3.000000    3.000000    -6.000000
-4.000000    7.000000    8.000000
5.000000    7.000000    -9.000000

                 B matrix

-6.000000
11.000000
3.000000

                 TRIANGULARIZATION
                 A Matrix

-3.000000    3.000000    -6.000000
0.000000    3.000000    16.000000
0.000000    0.000000    -83.000000

                 B Matrix

-6.000000
19.000000
-83.000000

                 SOLVING
                 SOLUTION VECTOR X

1.000000
1.000000
1.000000

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.