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

Code source

Voici le code source de mon implémentation de l'algorithme de Gauss sans utiliser la matrice augmentée, c'est-à-dire en travaillant 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);
}
```

```
48 /*
   *COMPLETE A 2D FLOAT MATRIX FROM USER INPUT
50 */

52 void completeMatrix(float **mat, int row, int column){
     for (int i=0; i<row; i++){
54         for(int j=0; j<column; j++){
             printf("Coefficient at M_%d,%d:    ", i+1, j+1);
56             scanf("%f", &mat[i][j]);
           }
58   }
   }
60
   /*
62 *GENERATE A COLUMN VECTOR "B" FROM A 2D FLOAT MATRIX "A"
   */
64
   void generateB(float **matA, float **matB, int row, int column){
66   for(int i=0; i<row; i++){
       float sum=0;
68     for(int j=0; j<column; j++){
         sum+=matA[i][j];
70     }
       matB[i][0]=sum;
72       }
   }
74
   /*
76 *PERFORM GAUSSIAN ELIMINATION ON A Ax=B MATRIX SYSTEM OF LINEAR EQUATIONS
   */
78
   void gauss(float** matA, float** matb, int size){
80   for(int k=0; k<size-1; k++){
       for(int i=k+1; i<size; i++){
82       float alpha=matA[i][k]/matA[k][k];
         for(int j=k; j<size; j++){
84         matA[i][j]=matA[i][j]-alpha*matA[k][j];
         }
86       matb[i][0]=matb[i][0]-alpha*matb[k][0];
       }
88   }
   }
90
   /*
92 *SOLVE A MATRIX SYSTEM OF LINEAR EQUATIONS USING BACKWARD SUBSTITUTION
   */
94 void resolution(float** matA, float** matb, float** matx, int size){
     matx[size-1][0]=matb[size-1][0]/matA[size-1][size-1];
96   for (int i=size-2; i>=0; i--){
       float sum=0;
98     for(int j=i+1; j<size; j++){
         sum+=matA[i][j]*matx[j][0];
100    }
       matx[i][0]=(1/matA[i][i])*(matb[i][0]-sum);
102  }
   }
104
```

```
int main(){

    //A Matrix
    int rowA=3;
    int columnA=3;
    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);

    //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
    float** Xmatrix=createMatrix(rowA, 1);
    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 du code

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.

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 transforme cette matrice en une forme 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.

Exécution du code sur quelques matrices