

### 0.0.1 Implémentation grâce à une matrice augmentée

#### Code source

Voici le code source de mon implémentation du pivot de Gauss via le passage par la matrice augmentée.

C'est-à-dire que dans mon implémentation il y a une concaténation des matrices.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
/*
 * PRINT MATRIX WITH RIGHT FORMAT
 */
void printMatrix(float **matrix, int m, int p) {
    printf("PRINTING_MATRIX_FROM: %p_LOCATION: \n", matrix);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < p; j++) {
            (j <= p - 2) ? printf("%f_", matrix[i][j]) : printf("%f", matrix[i][j]);
        }
        puts("");
    }
}
/*
 * ALLOCATE MEMORY FOR MATRIX
 */
float **allocate(int m, int n) {
    float **T = malloc(m * sizeof *T);
    for (int i = 0; i < m; i++) {
        T[i] = malloc(n * sizeof *T[i]);
        if (T[i] == NULL) {
            for (int j = 0; j < i; j++) {
                free(T[j]);
            }
            free(T);
            puts("ALLOCATION_ERROR");
            exit(-1);
        }
    }
    return T;
}
/*
 * FILL MATRIX BY USER INPUT
 */
void fillM(int m, int p, float **T) {
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < p; j++) {
```

```

        T[i][j] = 0;
        printf("Enter coefficient for %p[%d][%d]", T, i, j);
        scanf("%f", &T[i][j]);
    }
}
/*
 * FREE MATRIX
 */
void freeAll(float **T, int m) {
    for (int i = 0; i < m; i++) {
        free(T[i]);
    }
    free(T);
}
/*
 * IMPLEMENTATION OF '.' OPERATOR FOR MATRIX
 */
float **multiplication(float **M1, float **M2, int m, int q) {
    float **R = allocate(m, q);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < q; j++) {
            for (int k = 0; k < q; k++) {
                R[i][j] += M1[i][k] * M2[k][j];
            }
        }
    }
    return R;
}
/*
 * BUILD AUGMENTED MATRIX
 */
float **AugmentedMatrix(float **M1, float **M2, int m, int n) {
    float **A = allocate(m, m + 1);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n + 1; j++) {
            (j != n) ? (A[i][j] = M1[i][j]) : (A[i][j] = M2[i][0]);
        }
    }
    return A;
}
/*
 * PERFORM GAUSS ALGORITHM ONLY ON AUGMENTED MATRIX
 */
void gauss(float **A, int m, int p) {
    if (m != p) {

```

```

    puts("La matrice doit etre carree!");
    return;
}
for (int k = 0; k <= m - 1; k++) {
    for (int i = k + 1; i < m; i++) {
        float pivot = A[i][k] / A[k][k];
        for (int j = k; j <= m; j++) {
            A[i][j] = A[i][j] - pivot * A[k][j];
        }
    }
}
}
/*
 * DETERMINE ALL UNKNOWN VARIABLES
 */
float *findSolutions(float **A, int m) {
    float *S = calloc(m, sizeof *S);
    S[m - 1] = A[m - 1][m] / A[m - 1][m - 1];
    for (int i = m - 1; i >= 0; i--) {
        S[i] = A[i][m];
        for (int j = i + 1; j < m; j++) {
            S[i] -= A[i][j] * S[j];
        }
        S[i] = S[i] / A[i][i];
    }
    return S;
}
int main() {
    int m, n, p, q;
    float **P, **Q, **B, **A, *S;
    clock_t start, end;
    double execution;
    puts("Nombre_de_ligne_suivit_du_nombre_de_colonne_pour_la_matrice_1:");
    scanf("%d%d", &m, &p);
    puts("Nombre_de_ligne_suivit_du_nombre_de_colonne_pour_la_matrice_2:");
    scanf("%d%d", &n, &q);
    start = clock();
    P = allocate(m, p);
    Q = allocate(n, q);
    fillM(m, p, P);
    fillM(n, q, Q);
    printMatrix(P, m, p);
    printMatrix(Q, n, q);
    B = multiplication(P, Q, m, n);
    printMatrix(B, m, q);
    A = AugmentedMatrix(P, B, m, p);

```

```

    gauss(A, m, p);
    printMatrix(A, m, m + 1);
    S = findSolutions(A, m);
    puts("SOLUTIONS");
    for (int i = 0; i < m; i++)
        printf("x%d=%f\n", i, S[i]);
    freeAll(P, m);
    freeAll(Q, n);
    freeAll(B, m);
    freeAll(A, m);
    free(S);
    end = clock();
    execution = ((double)(end - start) / CLOCKS_PER_SEC);
    printf("RUNTIME: %f_seconds", execution / 10);
    return 0;
}

```

### Commentaires du code

Mon implémentation utilise strictement l'algorithme de Gauss rappelé précédemment avec seulement quelques changements d'indices puisque au lieu de travailler sur une matrice carrée et un vecteur colonne, mon programme utilise une matrice augmentée ayant  $m$  lignes et  $m + 1$  colonnes,  $m \in \mathbb{N}^*$ .

#### Détail des fonctions non conventionnelles:

*Comme mentionné précédemment, je ne détaillerai pas les fonction gauss() et findSolutions() puisque ces fonctions permettent strictement que d'une part d'implémenter l'algorithme de Gauss et d'autre part à "remonter" la matrice échelonnée afin de récupérer les valeurs des inconnus.*

**-float \*\*AugmentedMatrix(float \*\*M1, float \*\*M2, int m, int n):** cette fonction permet de créer une matrice  $A \in \mathcal{M}_{m,m+1}$  à partir de la concaténation de  $M1 \in \mathcal{M}_{mm}$  et  $M2 \in \mathcal{M}_{m,1}$ . Soient  $a_{ij}$  les coefficients peuplant  $A$ ,  $b_{ij}$  les coefficients peuplant  $M1$  et  $c_{i0}$  les coefficients peuplant  $M2$ .

On obtient alors  $a_{ij} = b_{ij} \forall i \in \mathbb{N}_m, \forall j \in \mathbb{N}_m$  et  $a_{ij} = c_{i0}$  si  $j = m + 1$ .

Cette fonction renvoie alors  $A$ , la matrice de flottant créée dynamiquement.

### Inputs / Outputs

Mon programme demande d'abord 4 entiers  $m, p, n, q$  entiers qui correspondent aux dimensions de la première matrice  $A \in \mathcal{M}_{mp}$  et de la seconde matrice  $X \in \mathcal{M}_{nq}$ . Le but étant de résoudre le système  $AX = b$ , nous initialiserons  $X$  à 1. Ce choix de valeur permettra de contrôler la validité du programme, ainsi si à la fin du programme  $\forall x_i \neq 1, \forall i \in \mathbb{N}_n$ , on pourra affirmer que le programme est faux.

Sur le  $m \times p$  prochaines lignes, le programme demande les coefficients de  $A$ .

Sur les  $n \times q$  prochaines lignes, le programme demandera les coefficient du vecteur colonne  $X$ , que l'utilisateur initialisera à 1.

On peut alors automatiser les entrée en utilisant des fichiers.

Ainsi la matrice:  $M = \begin{pmatrix} 3 & 0 & 4 \\ 7 & 4 & 2 \\ -1 & 1 & 2 \end{pmatrix}$  et  $X = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$   
sont représentés par ce fichier d'entrée:

Listing 1: input.txt

```
3 3
3 1
3 0 4
7 4 2
-1 1 2
1 1 1
```

Pour ce qui est des résultats produits par mon programme, une fois injecté dans un fichier texte, une input "type" ressemble à ceci.

Listing 2: Gauss elimination with M and X matrix

```
PRINTING MATRIX FROM: 0x556d938672c0 LOCATION :
3.000000  0.000000  4.000000
7.000000  4.000000  2.000000
-1.000000  1.000000  2.000000
PRINTING MATRIX FROM: 0x556d93867340 LOCATION :
1.000000
1.000000
1.000000
PRINTING MATRIX FROM: 0x556d938673c0 LOCATION :
7.000000
13.000000
2.000000
PRINTING MATRIX FROM: 0x556d93867440 LOCATION :
3.000000  0.000000  4.000000  7.000000
0.000000  4.000000  -7.333333  -3.333332
0.000000  0.000000  5.166667  5.166667
SOLUTIONS
x0 = 1.000000
x1 = 1.000000
x2 = 1.000000
RUNTIME: 0.000002 seconds
```

On remarquera que le programme affiche dans cet ordre:

- La Matrice A
- La Matrice X
- La Matrice B trouvé avec les valeur de X
- La Matrice augmentée en triangle supérieur
- Les solutions
- Un timer permettant de contrôler le temps d'exécution approximatif de mon programme

## 0.0.2 Exemples d'exécutions

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

On obtient respectivement cette sortie:

Listing 3: Matrix 2 results

```
PRINTING MATRIX FROM: 0x55f604fb32c0 LOCATION :
-3.000000 3.000000 -6.000000
-4.000000 7.000000 8.000000
5.000000 7.000000 -9.000000
PRINTING MATRIX FROM: 0x55f604fb3340 LOCATION :
1.000000
1.000000
1.000000
PRINTING MATRIX FROM: 0x55f604fb33c0 LOCATION :
-6.000000
11.000000
3.000000
PRINTING MATRIX FROM: 0x55f604fb3440 LOCATION :
-3.000000 3.000000 -6.000000 -6.000000
0.000000 3.000000 16.000000 19.000000
0.000000 0.000000 -83.000000 -83.000000
SOLUTIONS
x0 = 1.000000
x1 = 1.000000
x2 = 1.000000
RUNTIME: 0.000002 seconds
```

Listing 4: Matrix 4 results

```
PRINTING MATRIX FROM: 0x55f7afd662c0 LOCATION :
7.000000 6.000000 9.000000
4.000000 5.000000 -4.000000
-7.000000 -3.000000 8.000000
PRINTING MATRIX FROM: 0x55f7afd66340 LOCATION :
1.000000
1.000000
1.000000
PRINTING MATRIX FROM: 0x55f7afd663c0 LOCATION :
22.000000
5.000000
-2.000000
PRINTING MATRIX FROM: 0x55f7afd66440 LOCATION :
7.000000 6.000000 9.000000 22.000000
0.000000 1.571428 -9.142858 -7.571429
```

```

0.000000 0.000000 34.454552 34.454548
SOLUTIONS
x0 = 1.000001
x1 = 0.999999
x2 = 1.000000
RUNTIME: 0.000002 seconds

```

Listing 5: Matrix 6 results

```

PRINTING MATRIX FROM: 0x557fdaa552c0 LOCATION :
3.000000 -1.000000 0.000000
0.000000 3.000000 -1.000000
0.000000 -2.000000 3.000000
PRINTING MATRIX FROM: 0x557fdaa55340 LOCATION :
1.000000
1.000000
1.000000
PRINTING MATRIX FROM: 0x557fdaa553c0 LOCATION :
2.000000
2.000000
1.000000
PRINTING MATRIX FROM: 0x557fdaa55440 LOCATION :
3.000000 -1.000000 0.000000 2.000000
0.000000 3.000000 -1.000000 2.000000
0.000000 0.000000 2.333333 2.333333
SOLUTIONS
x0 = 1.000000
x1 = 1.000000
x2 = 1.000000
RUNTIME: 0.000002 seconds

```

**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.