Rapport TP1 - Algorithmique Numerique
Partie 1 : Gauss

YMERAJ Eldis

10/2022

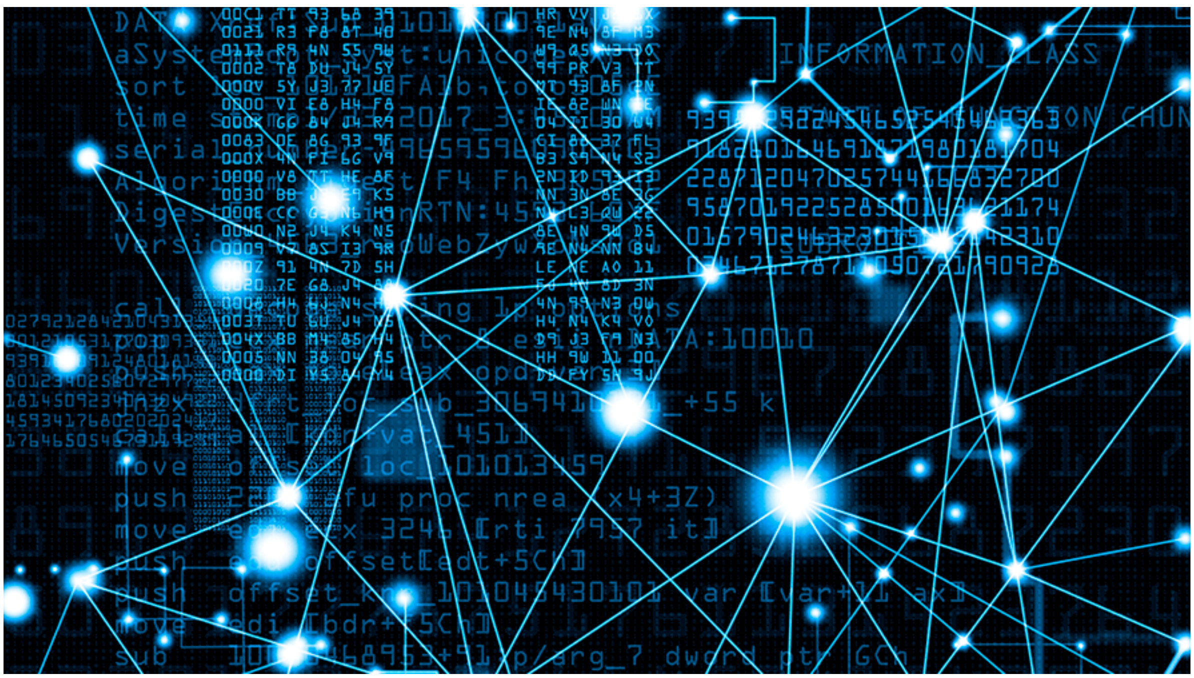

Figure 1 – Algorithmique numerique

Table des matières

Introduction aux Méthodes directes

L e problème de résolution des systèmes linéaires se rencontre très fréquemment dans beaucoup de problèmes ayant une origine physique. En effet, pour résoudre des problèmes plus compliqués d'analyse numérique (éqution aux dérivées partielles, par exemple), on est souvent ramené à résoudre un système linéaire de n éqution à n inconnues de la forme :

$$(S)\left\{\begin{array}{l} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \qquad\qquad\quad . \\ \qquad\qquad\quad . \\ \qquad\qquad\quad . \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{array}\right.$$

Figure 2 – Système linéaire S

avec a[i][j](1 <= n ; 1<=j <=n) et b[i](1 <= i <= n) sont des nobres réels données et x[i](1 <= i <= n) sont des inconnues. En utilisant la notation matricelle, le système linéaire (S) s'écrit d'une manière plus simple sous la forme Ax = b, avec A la matrice carrée formée par les éléments a[i][j], x et b les matrices colonnes formées respectivement par les éléments x[i] et b[i].

Dans la partie 1 de TP je vais implémenter la méthode de Gauss pour la résolution de systèmes linéaires dans le cas des matrices tests proposées dans le section 4 de TP.

1 Rappel - Méthode de Gauss

Elle permet le calcul d'une solution exacte en un nombre fini d'étapes (Méthode de Gauss simple). Des problèmes d'accumulation d'erreurs au cours de la résolution peuvent affecter la solution (Méthode de Gauss avec pivot). La méthode de Gauss consiste à transformer le système linéaire Ax = b en un système linéaire équivalent A'x = b' ayant la même solution et tel que A' soit triangulaire supérieure. Pour ce faire, on utilise les transformations élémentaires suivantes, qui laissent un système linéaire équivqlent à lui même :

- Permutation de deux lignes,
- Multiplication d'une ligne par une constante non nulle,
- Faire ajouter à une ligne une autre ligne multipliée par une constante non nulle.

On définit une matrice A = (a[i][j])1<=i<=n,1<=j<=n+1, associée au système (S) telle que a[1][n+1] = b[i]. Pour obtenir le système triangulaire équivalent, il suffit d'appliquer (n - 1) itérations de Gauss sur les lignes de la matrice A.

1.1 Algorithme de Gauss Simple

- Etape 1 : La triangularisation Passage du systéme Ax = b au système A'x = b' où A' est triangulaire supèrieure. Pour obtenir cette forme on applique l'algorithme suivant :

Pour $k = 1, \ldots, n-1$ **faire**

Pour $i = k+1, \ldots, n$ **faire**

Calculer le pivot

$$\alpha_i^{(k)} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$$

Pour $j = k, \ldots, n$ **faire**

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \alpha_i^{(k)} a_{kj}^{(k)}$$

FinPour j

$$b_i^{(k+1)} = b_i^{(k)} - \alpha_i^{(k)} b_k^{(k)}$$

FinPour i

FinPour k

FIGURE 3 – Algorithme de Gauss Simple

— Etape 2 : Résolution facile.

2 Implementation en langage C

Dans cette partie je vais vous montrer comment j'ai implementé la méthode de Gauss Simple en langage C et toutes les fonctions que j'ai construit pour réaliser ce travail.

2.1 Fonction allouerMat(n, m)

Cette fonction permet de nous allouer la memoire pour les matrices qu'on va utiliser.

```c
float ** allouerMat(int n, int m)
{
    int i;
    float ** mat = NULL;
    mat = malloc(n*sizeof(float *));
    if(mat == NULL){return NULL;}

    for(i=0;i<n;i++)
    {
        mat[i] = NULL;
        mat[i] = malloc(n *sizeof(float));
        if (mat[i] == NULL)
           {
               printf("erreur d'allocation");

             return NULL;}
    }
    return mat;
}
```

— La fonction prends une valeur en retour et il va nous retourner une matrice alloué.
— Prends comme argument les dimensions de la matrice qu'on va allouer.

— Si notre matrice est vide alors la fonction va nous retourner rien.

2.2 Fonction entrerMatrice(A, n, m)

Cette fonction permet à l'utilisateur d'entrer la valeur de matrice qu'il va tester. Dans notre cas sont les matrices A1, A2, A3, A4. Après qu'on a alloué l'espace pur la matrice qu'on va tester on demande à l'utilisateur de remplir la matrice.

```c
void entrerMatrice(float **A, int n, int m)
{
   printf("Entrez les valeurs de notre matrice : \n");
    for(int i =0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            scanf("%f", &A[i][j]);
        }
    }
}
```

— Cette fonction est de type void, qui ne retourne rien.
— Elle prend comme argument une matrice de type float qu'on a nommée A, et ses dimensions.
— Demande à l'utilisateur d'entrer les valeurs de matrice.

2.3 Fonction affMat(A, n, m)

Cette fonction va nous afficher la matrice après qu'on l'a entrée précédament avec la fonction entrerLMatrice(A, n, m). En plus il va nous servir d'afficher la matrice avant et après la triangularisation.

```c
void affMat(float **A, int n, int m){

    int i, j;
    for(i=0; i<n; i++){
        for(j=0; j<m; j++){
            printf("%.2f \t", A[i][j]);
        }
        printf("\n");
    }
    printf("\n");
}
```

— Cette fonction est de type void, qui ne retourne rien.
— Elle prend comme argument une matrice de type float qu'on a nommée A, et ses dimensions.
— Elle va nous afficher la matrice avec ses valeurs.

2.4 TriangGauss(A, n, m)

Cette fonction permet de nous triangulariser la matrice entrant avec la méthode de Gauss.

```c
void TriangGauss(float **A,  int n, int m ){

    float pivot = 0 ;
    for (int k = 0; k <= n-1; k++){

        for(int i = k+1; i < n; i++){
            pivot = (A[i][k]/A[k][k]);

            for (int j = k; j < n; j++){
                A[i][j]-= pivot * A[k][j];
            }
        }
    }
}
```

- Cette fonction est de type void, qui ne retourne rien.
- Elle prend comme argument une matrice de type float qu'on a nommée A, et ses dimensions.
- C'est simplement l'lgorithme de Gauss Simple donnée en cours pour triangulariser la matrice donnée qu'on a traduit en langage C.

2.5 Fonction trouveB(mat, int n, int m)

Cette fonction permet de nous trouver la matrice B qui est de dimension 3x1. Le vecteur B est choisie de sorte que la solution exacte du système d'équations est x[i]=1, pour i =1,...,n.

```c
float ** trouveB(float **mat, int n, int m)
{

    float **B=allouerMat(1,3);

    for(int i = 0; i < n; i++)
    {
       B[i][0] = 0;

        for(int j = 0; j < m; j++)
        {

            B[i][0]+= mat[i][j];
        }
    }

    return B;
}
```

- Cette fonction renvoit une matrice de type float.
- Elle prend comme argument la matrice test et ses dimensions.
- Elle permet de trouver le vecteur B telle que lq solution exacte est x[i] = 1.

2.6 remplMatA5(A5, n)

Cette fonction va nous servir pour les matrices test, elle permet de remplir la matrice A5 en suivant la régle donnée.

```
void remplMatA5(float ** A5, int n){
    int i, j;
    for(i = 0; i < n; i++)
    {
        A5[i][i] = 1;
    }
    for(j = 1; j < n; j++)
    {
        A5[1][j] = A5[j][1] = 1/((float)pow(2,(float)j)); //ici le pow permet de mettre 2 à la puissance j
    }
}
```

— Cette fonction est de type void, qui ne retourne rien.
— Elle prend comme argument la matrice test A5 qu'on a déclaré et alloué dans le main et le dimension n (on va utiliser que la dimension n car on utilise des matrices carrée nxn).
— Elle permet de remplir la matrice A5 en suivant la régle donnée dans la fiche de TP.

2.7 remplMatA6(A6, n)

Cette fonction va nous servir pour les matrices test, elle permet de remplir la matrice A6 en suivant la régle donnée.

```
void remplMatA6(float **A6, int n)
{
    for(int i =0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            if(i==j)
            {
                A6[i][j]=3;
            }
            else if(j==i+1 && i<n)
            {
                A6[i][j]=-1;
            }
            else if(j==i-1 && i>1)
            {
                A6[i][j]=-2;
            }else
            {
                A6[i][j]=0;
            }
        }
    }

}
```

— Cette fonction est de type void, qui ne retourne rien.

— Elle prend comme argument la matrice test A6 qu'on a déclaré et alloué dans le main et le dimension n (on va utiliser que la dimension n car on utilise des matrices carrée nxn).
— Elle permet de remplir la matrice A5 en suivant la régle donnée dans la fiche de TP.

3 La fonction main()

Dans le main on a testé les matrices test A1, A2, A3, A4, A5 et on a obtenu les résultats qu'on va expliquer après. On a fait la même chose pour tout les matrices test.

Les étapes qu'on a suivi sont :

— Premièrement on a déclaré les dimentions nxm et tant que on que des matrices carrée de dimensions 3x3 on a donné la valeur 3 à m et n.
— On a déclaré la matrice au'on va tester et on l'a alloué la memoire à l'qide de fonction allouerMat(n, m).
— Après à l'aide de la fonction entrerMatrice(A, n, m) on demande à l'utilisateur de remplir la matrice avec les valeurs de matrice test donnée à la fiche de TP.
— Après on affiche la matrice qu'on a entrée précédament.
— On fait appel à fonction TriangGauss(A, n, m) et on refait appel à fonction affLMat(A, n, m) pour afficher la matrice triangularisé.
— En plus on déclare le vecteur B et on lui alloue la memoire.
— Et à l'aide de fonction trouveB(A, n, m) on lui donne la valeur et on l'affiche.
C'est la même procédure pour tout les matrices test.

```c
int main()
{
    int n,m; n=3; m=3;
    printf("\n==================MATRICE TEST A1==================\n");
    // POUR MATRICE TEST A1
    float **A1 = allouerMat(3,3);
    entrerMatrice(A1, n, m);
    printf("Notre matrice A1 avant la triangularisation est: \n");
    affMat(A1, n, m);
    printf("Notre matrice A1 apres la triangularisation est : \n");
    TriangGauss(A1, n, m);
    affMat(A1, n, m);
    float **B1 = allouerMat(1,3);
   B1 = trouveB(A1, n, m);
    printf("La matrice B1 pour que xi est toujours egal a 1 est : \n");
    affMat(B1, 3, 1);


    printf("\n\n==================MATRICE TEST A2==================\n\n");
    // POUR MATRICE TEST A2
    float **A2 = allouerMat(3,3);
    entrerMatrice(A2, n, m);
    printf("Notre matrice A2 avant la triangularisation est: \n");
    affMat(A2, n, m);
    printf("Notre matrice A2 apres la triangularisation est : \n");
    TriangGauss(A2, n, m);
    affMat(A2, n, m);
    float **B2 = allouerMat(1,3);
   B2 = trouveB(A2, n, m);
    printf("La matrice B2 pour que xi est toujours egal a 1 est : \n");
    affMat(B2, 3, 1);


    printf("\n\n==================MATRICE TEST A3==================\n\n");
```

```c
    // POUR MATRICE TEST A3
    float **A3 = allouerMat(3,3);
    entrerMatrice(A3, n, m);
    printf("Notre matrice A3 avant la triangularisation est: \n");
    affMat(A3, n, m);
    printf("Notre matrice A3 apres la triangularisation est : \n");
    TriangGauss(A3, n, m);
    affMat(A3, n, m);
    float **B3 = allouerMat(1,3);
    B3 = trouveB(A3, n, m);
    printf("La matrice B3 pour que xi est toujours egal a 1 est : \n");
    affMat(B3, 3, 1);


    printf("\n\n====================MATRICE TEST A4====================\n\n");
    // POUR MATRICE TEST A4
    float **A4 = allouerMat(3,3);
    entrerMatrice(A4, n, m);
    printf("Notre matrice A4 avant la triangularisation est: \n");
    affMat(A4, n, m);
    printf("Notre matrice A4 apres la triangularisation est : \n");
    TriangGauss(A4, n, m);
    affMat(A4, n, m);
    float **B4 = allouerMat(1,3);
    B4 = trouveB(A4, n, m);
    printf("La matrice B4 pour que xi est toujours egal a 1 est : \n");
    affMat(B4, 3, 1);


    // POUR MATRICE TEST A5
    printf("\n\n====================MATRICE TEST A5====================\n\n");
    float **A5 = allouerMat(n,m);
    remplMatA5(A5,n);
    printf("Notre matrice A5 avant la triangularisation est: \n");
    affMat(A5, n, m);
    printf("Notre matrice A5 apres la triangularisation est : \n");
    TriangGauss(A5, n, m);
    affMat(A5, n, m);
    float **B5 = allouerMat(1,3);
    B5 = trouveB(A5, n, m);
    printf("La matrice B5 pour que xi est toujours egal a 1 est : \n");
    affMat(B5, 3, 1);


    // POUR MATRICE TEST A6
    printf("\n\n====================MATRICE TEST A6====================\n\n");
    float **A6 = allouerMat(n,m);
    remplMatA6(A6,n);
    printf("Notre matrice A6 avant la triangularisation est: \n");
    affMat(A6, n, m);
    printf("Notre matrice A6 apres la triangularisation est : \n");
    TriangGauss(A6, n, m);
    affMat(A6, n, m);
    float **B6 = allouerMat(1,3);
    B6 = trouveB(A6, n, m);
    printf("La matrice B6 pour que xi est toujours egal a 1 est : \n");
    affMat(B6, 3, 1);

    return 0;

}
```

4 Résultats obtenues

```
===============MATRICE TEST A1===============
Entrez les valeurs de notre matrice :
3 0 4 7 4 2 -1 1 2
Notre matrice A1 avant la triangularisation est:
3.00    0.00    4.00
7.00    4.00    2.00
-1.00   1.00    2.00

Notre matrice A1 apres la triangularisation est :
3.00    0.00    4.00
0.00    4.00    -7.33
0.00    0.00    5.17

La matrice B1 pour que xi est toujours egal a 1 est :
7.00
-3.33
5.17
```

Figure 4 – Matrice test A1

```
===============MATRICE TEST A2===============

Entrez les valeurs de notre matrice :
-3 3 -6 -4 7 8 5 7 -9
Notre matrice A2 avant la triangularisation est:
-3.00   3.00    -6.00
-4.00   7.00    8.00
5.00    7.00    -9.00

Notre matrice A2 apres la triangularisation est :
-3.00   3.00    -6.00
0.00    3.00    16.00
0.00    0.00    -83.00

La matrice B2 pour que xi est toujours egal a 1 est :
-6.00
19.00
-83.00
```

Figure 5 – Matrice test A2

```
===============MATRICE TEST A3===============

Entrez les valeurs de notre matrice :
4 1 1 2 -9 0 0 -8 6
Notre matrice A3 avant la triangularisation est:
4.00    1.00    1.00
2.00    -9.00   0.00
0.00    -8.00   6.00

Notre matrice A3 apres la triangularisation est :
4.00    1.00    1.00
0.00    -9.50   -0.50
0.00    0.00    6.42

La matrice B3 pour que xi est toujours egal a 1 est :
6.00
-10.00
6.42
```

Figure 6 – Matrice test A3

```
===============MATRICE TEST A4==============

Entrez les valeurs de notre matrice :
7 6 9 4 5 -4 -7 -3 8
Notre matrice A4 avant la triangularisation est:
7.00    6.00    9.00
4.00    5.00    -4.00
-7.00   -3.00   8.00

Notre matrice A4 apres la triangularisation est :
7.00    6.00    9.00
-0.00   1.57    -9.14
0.00    -0.00   34.45

La matrice B4 pour que xi est toujours egal a 1 est :
22.00
-7.57
34.45
```

Figure 7 – Matrice test A4

```
===============MATRICE TEST A5==============

Notre matrice A5 avant la triangularisation est:
1.00    0.00    0.00
0.00    0.50    0.25
0.00    0.25    1.00

Notre matrice A5 apres la triangularisation est :
1.00    0.00    0.00
0.00    0.50    0.25
0.00    0.00    0.88

La matrice B5 pour que xi est toujours egal a 1 est :
1.00
0.75
0.88
```

Figure 8 – Matrice test A5

```
===============MATRICE TEST A6==============

Notre matrice A6 avant la triangularisation est:
3.00    -1.00   0.00
0.00    3.00    -1.00
0.00    -2.00   3.00

Notre matrice A6 apres la triangularisation est :
3.00    -1.00   0.00
0.00    3.00    -1.00
0.00    0.00    2.33

La matrice B6 pour que xi est toujours egal a 1 est :
2.00
2.00
2.33
```

Figure 9 – Matrice test A6

Conclusion

Dans ce travaille pratique, on est arrivé à implementer la résolution de systèmes linéaires avec la méthode de Gauss Simple en programmation C. La méthode de Gauss Simple est une méthode qui consiste à transformer un sytème linéaire Ax = b en un système linéaire équivalent A'x = b' ayant la même solution et tel que A' soit triangulaire supérieure. Grace à cette méthode on peut résoudre facilement les systèmes d'équations linéaires sans avoir besoin de faire des calculs compliqué. L'algorithme de Gauss est simple et très efficace pour résoudre ces types de systéme d'équations linéaires.

Table des figures