.1 Méthode de Jacobi

Rappelons que la méthode de **Jacobi** est itérative et ne garantit pas toujours un résultat. La méthode est définie si A est définie positive.
L'algorithme permet de trouver un résultat si la matrice est dites à diagonale strictement dominante. Autrement dit, Soit $[a_{ij}]_{0\leq i,j\leq n}$ les coefficients réels peuplant $A \in \mathcal{M}_{n,n}(\mathbb{R})$, alors si :
$\forall i, |a_{i,i}| < \sum_{i\neq j} |a_{ij}|$, on a que Jacobi converge vers l'unique solution du système $Ax = b$.

.1.1 Principe de la méthode

On veut résoudre $Ax = b$ avec $A \in \mathcal{M}_{n,n}(\mathbb{R})$, $n \in \mathbb{N}$, x la vecteur colonne contenant les inconnus et b le vecteur colonne des solution.
On pose $D \in \mathcal{M}_{n,n}(\mathbb{R})$ la matrice contenant les coefficients $[a_{i,j}]_{0\leq i=j\leq n}$ de A.
On pose aussi E et F avec E la matrice triangulaire opposée inférieure de A et F la matrice supérieure opposée de A.
On obtient alors :

$$Ax = b \tag{1}$$
$$(D - E - F)x = b \tag{2}$$
$$Dx - (E + F)x = b \tag{3}$$
$$x = D^{-1}(E + F)x + D^{-1}b \tag{4}$$
$$x^{k+1} = D^{-1}(E + F)x^k + D^{-1}b \tag{5}$$

Ce qui donne l'algorithme suivant :
Soit ϵ L'erreur maximale, un point initial x^0 et $k = 0$
avec $\epsilon^0 = ||Ax^0 - b||$
On obtient :

Tant que $(\epsilon^{(k)} \leq \epsilon)$
$x_i^{k+1} = \frac{1}{a_{ii}}[b_i - \sum_{j\neq i} a_{ij}x_j^{(k)}], i = 1, \ldots, n$
$\epsilon^{k+1} = ||Ax^{k+1} - b||$
$k = k + 1$
FIN JACOBI

Remarque, on ajoutera aussi un nombre d'iterations maximum afin de ne pas être dans le cas d'une boucle infinies (si jacobi diverge alors l'erreur augmente).

.1.2 Résolution manuelle

Nous en détaillerons seulement une itération
Soit $A = \begin{pmatrix} 4 & 1 & 0 \\ -1 & 3 & 6 \\ -2 & -5 & -3 \end{pmatrix}$, $b = \begin{pmatrix} 8 \\ 3 \\ 8 \end{pmatrix}$,$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$ et $x^{(0)} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$
On a $A = \underbrace{\begin{pmatrix} 4 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & -2 \end{pmatrix}}_{D} - \underbrace{\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 2 & 5 & 0 \end{pmatrix}}_{E} - \underbrace{\begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -6 \\ 0 & 0 & 0 \end{pmatrix}}_{F}$
On a donc $x^{k+1} = D^{-1}[(E + F)x^k + b]$
Dans le cas présent on obtient alors :
$x^1 = \begin{pmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & -\frac{1}{2} \end{pmatrix} \left[\left[\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 2 & 5 & 0 \end{pmatrix} + \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -6 \\ 0 & 0 & 0 \end{pmatrix} \right] \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 8 \\ 3 \\ 8 \end{pmatrix} \right]$

$x^1 = \begin{pmatrix} 2 \\ 1 \\ -4 \end{pmatrix}$

et

$$\epsilon^{(1)} = ||Ax^{(1)} - b||$$
$$\epsilon^{(1)} = || \begin{pmatrix} 4 & 1 & 0 \\ -1 & 3 & 6 \\ -2 & -5 & -3 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \\ -4 \end{pmatrix} - \begin{pmatrix} 8 \\ 3 \\ 8 \end{pmatrix} ||$$
$$\epsilon^{(1)} = || \begin{pmatrix} 1 \\ -26 \\ -5 \end{pmatrix} ||$$
$$\epsilon^{(1)} = \sqrt{1^2 + (-26)^2 + (-5)^2}$$
$$\epsilon^{(1)} = \sqrt{(702)}$$

.1.3 Implémentation

Pour l'implémentation de cette méthode, nous utiliserons ϵ comme suit :

$$\epsilon^{(k)} = p^k = \text{Max}_{i=1,\ldots,n} |\bar{x_i} - \tilde{x_i}^k|$$

Où $\bar{x_i}$ est les résultat attendu et $\tilde{x_i}^k$ est l'approximation trouvée à l'étape k.
De plus on utilisera aussi une limite d'occurrence, pour pouvoir gérer les matrices où **Jacobi** diverge. En l'occurrence on se fixera un $\epsilon = 10^{-6}$ et un nombre d'itération maximum fixé à 1000

.1.4 Code

On ne détaillera ici seulement les fonctions dites "non triviales" c'est-à-dire les fonctions étant en lien directe avec l'algorithme décrit.
De plus chaque fonction présentée sera dépouillée de toute fonction d'affichage permettant de produire des chiffres liés à l'utilisation du programme (pour une question de lisibilité).

Listing 1 – jacobi.c

```
int jacobi(float **A, float *vector, float *b, float *S, int n, float minErr,int bound) {
  float epsilon = epsi(S, vector, n);
  int k = 0;
  float *cp = malloc(n * sizeof *cp);
  while (k < bound && epsilon >= minErr) {
    fprintf(stderr, "%f ", epsilon);
    // printf("EPSILON : %f\n", epsilon);
    copy(cp, vector, n);
    for (int i = 0; i < n; i++) {
      vector[i] = (1 / A[i][i]) * (b[i] - jacobiSum(A, cp, n, i));
    }
    epsilon = epsi(vector, S, n);
    k++;
  }
  free(cp);
  return k;
}
```

Commentaires : Nous remarquerons que l'implémentation de l'algorithme de Jacobi est similaire à ce que a été présenté ultérieurement. Pour des questions de lisibilité, $\epsilon^{(k)}$ est calculée par la fonction *epsi()*, de même pour **jacobiSum()**, le calcul a été séparé afin de faciliter la compréhension du programme.

Listing 2 – jacobiSum() function in "source.h"

```
float jacobiSum(float **A, float *V, int m, int i) {
  float s = 0;
  for (int j = 0; j < m; j++) {
    if (j != i)
      s += A[i][j] * V[j];
  }
  return s;
}
```

Commentaire : Permet de calculer de façon lisible est clair ceci :

$$\sum_{j \neq i} a_{ij} x_j^{(k)}, i = 1, \dots k$$

Listing 3 – epsi() function in "source.h"

```
float epsi(float *V, float *S, int n) {
  float max = Fabs(V[0] - S[0]);
  for (int i = 1; i < n; i++) {
    if (Fabs(S[i] - V[i]) > max)
      max = Fabs(S[i] - V[i]);
  }
  return max;
}
```

Commentaires : Utilise la fonction *Fabs()* que nous avons implémenter, elle renvoie la valeur absolue d'un nombre flottant.
Cette fonction *epsi()* permet donc de calculer l'erreur entre deux itération de la façon suivante :

$$\epsilon^{(k)} = \text{Max}_{i=1,\dots,n} |\bar{x_i} - \tilde{x_i}^k|$$

Listing 4 – conv() function in "source.h"

```
int conv(float **A, int m) {
  float sl = 0;
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < m; j++) {
      if (i != j)
        sl += fabs(A[i][j]);
    }
    if (A[i][i] - sl <= 0)
      return 0;
  }
  return 1;
}
```

Commentaire *conv()* est une fonction utilitaire permettant d'effectuer une prediction sur la convergeance potentielle d'une matrice par jacobi.
Pour cela on vérifie si la matrice sur laquelle on effectue jacobi est à diagonale strictemennt dominante. Ce qui se vérifie par cette formule :

$$\forall i, |a_{ii}| > \sum_{j \neq i} |a_{ij}|$$

.1.5 Entrées / Sorties

Entrées

Le programme prend en entrée 2 paramètres soient

1. minErr, ou l'erreur minimale tolérée
2. bounds, qui désigne la limite d'occurrence du programme (en cas de divergeance)

L'utilisateur peut ensuite décider de remplir manuellement sa matrice où de rediriger le flux d'un fichier comme suit :

Listing 5 – A4.txt

```
3 3
7 6 9
4 5 -4
-7 -3 8
```

Avec sur la première ligne : les dimensions de la matrice suivit, sur les ligne suivantes, des coefficients de la matrice

.1.6 Sorties

Le flux d'erreur sera réservé afin de produire des données engendrant des graphiques (des données formatées). Il s'agit de l'énumération de tout les ϵ calculés suivit du nombre d'itération.
Sur les autre lignes seront inscrit des messages facilitant la prise en main du programme pour l'utilisateur.
Il figuera ensuite la prédiction quant à la convergence de la méthode.
Enfin la dernière lignes représentera le nombre de ϵ calculés.

.1.7 Exécution

Voici l'exécution sur les 12 matrices de la méthode de jacobi avec $\epsilon = 10^{-4}$.
La redirection de flux de sortie du programme est la suivante : >

Listing 6 – Execution with A1 matrix

```
A matrix dimensions:
Enter coefficient for 0x55ce91727300[0][0]Enter coefficient for 0x55ce91727300[0][1]Enter coefficient fo
 ===== CONVERGENCE PREDICTION: may not conv =====
PRINTING VECTOR FROM: 0x55ce917272e0 LOCATION :
-nan -48517597648316769037382673682594267136.000000 inf
EPSILON CALCULATED 742
```

Listing 7 – Execution with A2 matrix

```
A matrix dimensions:
Enter coefficient for 0x55ea0288b300[0][0]Enter coefficient for 0x55ea0288b300[0][1]Enter coefficient fo
 ===== CONVERGENCE PREDICTION: may not conv =====
PRINTING VECTOR FROM: 0x55ea0288b2e0 LOCATION :
-nan -nan -inf
EPSILON CALCULATED 252
```

Listing 8 – Execution with A3 matrix

```
A matrix dimensions:
Enter coefficient for 0x5579594bf300[0][0]Enter coefficient for 0x5579594bf300[0][1]Enter coefficient fo
 ===== CONVERGENCE PREDICTION: may not conv =====
PRINTING VECTOR FROM: 0x5579594bf2e0 LOCATION :
1.000052 1.000013 0.999954
EPSILON CALCULATED 13
```

Listing 9 – Execution with A4 matrix

```
A matrix dimensions:
Enter coefficient for 0x560374865300[0][0]Enter coefficient for 0x560374865300[0][1]Enter coefficient fo
 ===== CONVERGENCE PREDICTION: may not conv =====
```

```
PRINTING VECTOR FROM: 0x5603748652e0 LOCATION :
1.000078 0.999903 1.000078
EPSILON CALCULATED 24
```

Listing 10 – Execution with A5 dimensions : 3x3

```
A matrix dimensions:
Enter coefficient for 0x56131ea84300[0][0]Enter coefficient for 0x56131ea84300[0][1]Enter coefficient fo
 ===== CONVERGENCE PREDICTION: may not conv =====
PRINTING VECTOR FROM: 0x56131ea842e0 LOCATION :
1.000068 1.000045 1.000023
EPSILON CALCULATED 17
```

Listing 11 – Execution with A5 dimensions : 6x6

```
A matrix dimensions:
Enter coefficient for 0x55b573d18300[0][0]Enter coefficient for 0x55b573d18300[0][1]Enter coefficient fo
 ===== CONVERGENCE PREDICTION: may not conv =====
PRINTING VECTOR FROM: 0x55b573d182e0 LOCATION :
0.999950 0.999927 0.999963 0.999982 0.999991 0.999995
EPSILON CALCULATED 18
```

Listing 12 – Execution with A5 dimensions : 8x8

```
A matrix dimensions:
Enter coefficient for 0x56527ea95320[0][0]Enter coefficient for 0x56527ea95320[0][1]Enter coefficient fo
 ===== CONVERGENCE PREDICTION: may not conv =====
PRINTING VECTOR FROM: 0x56527ea952f0 LOCATION :
0.999949 0.999924 0.999962 0.999981 0.999991 0.999995 0.999998 0.999999
EPSILON CALCULATED 18
```

Listing 13 – Execution with A5 dimensions : 10x10

```
A matrix dimensions:
Enter coefficient for 0x55cfc4470320[0][0]Enter coefficient for 0x55cfc4470320[0][1]Enter coefficient fo
 ===== CONVERGENCE PREDICTION: may not conv =====
PRINTING VECTOR FROM: 0x55cfc44702f0 LOCATION :
0.999949 0.999924 0.999962 0.999981 0.999990 0.999995 0.999998 0.999999 0.999999 1.000000
EPSILON CALCULATED 18
```

Listing 14 – Execution with A6 dimensions : 3x3

```
A matrix dimensions:
Enter coefficient for 0x55774b4d7300[0][0]Enter coefficient for 0x55774b4d7300[0][1]Enter coefficient fo
 ===== CONVERGENCE PREDICTION: may not conv =====
PRINTING VECTOR FROM: 0x55774b4d72e0 LOCATION :
0.999956 0.999941 0.999911
EPSILON CALCULATED 24
```

Listing 15 – Execution with A6 dimensions : 6x6

```
A matrix dimensions:
Enter coefficient for 0x562479ac8300[0][0]Enter coefficient for 0x562479ac8300[0][1]Enter coefficient fo
 ===== CONVERGENCE PREDICTION: may not conv =====
PRINTING VECTOR FROM: 0x562479ac82e0 LOCATION :
0.999989 0.999967 0.999949 0.999917 0.999917 0.999926
EPSILON CALCULATED 61
```

Listing 16 – Execution with A6 dimensions : 8x8

```
A matrix dimensions:
Enter coefficient for 0x55a7d738a320[0][0]Enter coefficient for 0x55a7d738a320[0][1]Enter coefficient fo
 ===== CONVERGENCE PREDICTION: may not conv =====
PRINTING VECTOR FROM: 0x55a7d738a2f0 LOCATION :
0.999994 0.999985 0.999968 0.999954 0.999927 0.999918 0.999904 0.999936
EPSILON CALCULATED 84
```

Listing 17 – Execution with A6 dimensions : 10x10

```
A matrix dimensions:
Enter coefficient for 0x55cc8327d320[0][0] Enter coefficient for 0x55cc8327d320[0][1] Enter coefficient fo
 ===== CONVERGENCE PREDICTION: may not conv =====
PRINTING VECTOR FROM: 0x55cc8327d2f0 LOCATION :
0.999997 0.999992 0.999983 0.999974 0.999955 0.999942 0.999918 0.999912 0.999902 0.999934
EPSILON CALCULATED 104
```

.1.8 Remarque sur les résultats

On remarquera que lorsque Jacobi renvoie **NaN, inf ou des valeurs proche de la limite d'encodage du type int (4 octets)**, Il s'agit du cas où cette méthode diverge, donc aucun résultat fiable n'est envisageable.
On rappelera aussi que dans le cadre de ce programme, si la méthode renvoie des valeurs **extremêment proches** de 1, alors le programme a trouvé une solution.