Programmation Linéaire

Implémentation de Gauss et Jacobi

Clément Payard
Mathieu Laurençot

Encadrant : M. Suzanne Élodie

2021

Table des matières

1 TODO Finir le boulot!! [2/5]

1.1 TODO Présenter les programmes

1.2 DONE Le commenter

1.3 DONE Test les matrices [0/2] :

1.3.1 TODO Test les matricesau dos de la feuille

1.3.2 TODO Tester pour des matrices de grandes tailles

1.3.3 TODO Tester pour des matrices avec + de 70% de 0

:CREATED : *<2021-09-24 ven. 10 :52>*

1.4 TODO Résultats modifiés ?

1.5 TODO Mettre la complexité

2 Rappel rapide des méthodes

2.1 Méthode de Gauss

Cette méthode permet de trouver une solution exacte au système $Ax = b$ en un nombre fini d'étape.

Pour ce faire, cette méthode se fait en plusieurs étapes :

1. La triangularisation On doit passer du système $Ax = b$ au système $A'x = b'$ où A' est une matrice triangulaire supérieure. L'algorithme utilisé est disponible dans le programme.
2. La résolution facile `Nécessite aucun 0 sur la diagonale de A`

2.2 Méthode de Jacobi

Cette méthode fait partie des méthodes itératives, où l'on cherche à se rapprocher, avec une suite d'itération définie, à une solution exacte.

Pour cette méthode, nous devons tout d'abord décomposer A sous la forme A = D -E -F

1. D est la matrice nul de taille A, sauf sur sa diagonale où D possède les coefficients de A.
2. -E est la matrice triangulaire inférieure de A
3. -F est la matrice triangulaire supérieur de A

De plus, on pose $M = D$ et $N = E + F$

On obtient donc le système :

$$Ax = b \iff Dx^{k+1} = (E + F)x^k + b$$

pour l'itération $k + 1$

De plus, l'algorithme de Jacobi s'écrit avec une précision ϵ :

3 Présentation des programmes commentés

3.1 Présentation Général :

3.1.1 Les différents fichiers utilisés

Pour effectuer ce travail, nous avons décidé de séparer notre programme en plusieurs fichiers :

1. main.c, qui est notre fichier appelant les divers fonctions présentent dans
2. fonction.c, puis
3. fonction.h, permettant de définir les différentes structures et les headers des fonctions, et enfin
4. main.h, où les différentes bibliothèques sont déclarées
5. De plus, il y a un Makefile, qui nous permet de compiler et tester notre programme efficacement

3.1.2 Les structures ainsi que les fonctions de base

3.2 Gauss

3.3 Jacobi

```
  matrice *Jacobi(matrice *A, matrice *B, float Eps, int nombremaxinte)
{
  /* gestion des cas d'erreur pouvant faire echouer la methode jacobi*/
  if ((A->largeur != A->longueur) || (A->longueur != B->longueur) ||
      (B->largeur != 1))
  {
    printf(
"Les matrice ne sont pas de la taille nécessaire a leurs résolution.");
    return B;
  }
  else
  {
    for (int i = 0; i < A->longueur; i++)
    {
      int verifieur = 0;
      for (int j = 0; j < A->longueur; j++)
      {
if (j != i)
{
  verifieur += fabsl(A->Mat[i][j]);
}
      }
      if (verifieur > A->Mat[i][i])
      {
printf("La matrice n'est pas à diagonale dominante et ne vas donc pas "
       "converger...\n");
return B;
      }
    }
  }

  /* création pour résoudre le système */
  matrice *x = creerMatrice(1, A->longueur);
  matrice *D = creerMatrice(A->largeur, A->longueur);
  matrice *E = creerMatrice(A->largeur, A->longueur);
  matrice *F = creerMatrice(A->largeur, A->longueur);
  matrice *N = creerMatrice(A->largeur, A->longueur);
```

```
  /* initialisation de D E et F */
  for (int i = 0; i < A->longueur; i++)
  {
    for (int j = 0; j < A->largeur; j++)
    {
      if (i == j)
      {
D->Mat[i][j] = A->Mat[i][j];
E->Mat[i][j] = 0;
F->Mat[i][j] = 0;
      }
      else if (i < j)
      {
D->Mat[i][j] = 0;
E->Mat[i][j] = -(A->Mat[i][j]);
F->Mat[i][j] = 0;
      }
      else
      {
D->Mat[i][j] = 0;
E->Mat[i][j] = 0;
F->Mat[i][j] = -(A->Mat[i][j]);
      }
      N->Mat[i][j] = E->Mat[i][j] + F->Mat[i][j];
    }
  }

  // initialisation de x
  /* for (int i = 0; i < x->longueur; ++i) */
  /* { */
  /* x->Mat[0][i] = 0; */
  /* } */

  float erreur = Eps + 1;
  InversematriceD(D->longueur, D);

  /* while ((pow(sigma, k)) >= sigma) */
  while (erreur > Eps)
  {
    // nouvelle valeur de x selon la formule
    x = multiplicationMatrice(
*D, *additionMatrice(*(multiplicationMatrice(*N, *x)), *B));

    // TODO: retirer cette ligne qui annule juste la boucle infini
    erreur = Norme(soustractino(*multiplicationMatrice(*A, *x), *B));
  }
  return x;
}
```

3.4 Programme final

Les différentes fonctions sont appelées au fur et à mesure du main.c, en laissant le choix à l'utilisateur de son choix. Ceci est regroupé dans un switch.

4 Présentation des matrices test au dos de la feuille

5 Conclusion ssur les méthodes

5.1 TODO Comparaison

5.2 TODO Cadre d'utilisation

5.3 TODO Stabilité

Cette méthode a un coût de l'ordre de 3n2+2n par itération. Elle converge moins vite que la méthode de Gauss-Seidel, mais est très facilement parallélisable.