Projet MA0844

ERTAS Elif

2022 - 2023

Table des matières

1. Solveur LU

1.1. Principe de fonctionnement

Dans cette première étape, on cherche à résoudre un système linéaire $Ax = b$. On rappelle que tous les mineurs principaux de la matrice A doivent être inversibles pour qu'elle admette une décomposition LU.

A est une matrice de taille NxN, x un vecteur de taille N et b un vecteur de taille N.

La factorisation LU consiste à décomposer la matrice A en produit de deux matrices : L et U de taille NxN. L est une matrice triangulaire inférieure à diagonale unité et U est une matrice triangulaire supérieure. Pour cela j'utilise ma fonction factorisation_LU (Annexe 1).

Une fois la matrice A décomposée, pour résoudre le système linéaire initiale on procède à une descente du sytème $Lx = b$ avec ma fonction descente (Annexe 2).

Puis à une remontée du système Uy = b, y étant le résultat de la descente effectuée précédemment avec ma fonction remontee (Annexe 3).

On obtient alors x, la solution finale à notre système linéaire Ax = b.

1.2. Application

On pose une matrice A de taille 4x4 et un vecteur b de taille 4x1 :

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \quad et \quad b = \begin{pmatrix} 0 \\ 4 \\ 6 \\ 5 \end{pmatrix}$$

On cherche à résoudre le système linéaire $Ax = b$. C'est-à-dire on cherche un unique x qui permet d'avoir l'égalité.
Dans mon programme solveurLU.c, il y a la fonction Solveur_LU_TriDiagonal (Annexe 4) qui permet de résoudre le système linéaire en passant par toutes les étapes expliquées dans le fonctionnement. C'est-à-dire la décomposition, la descente et enfin la remontée.
Faisons tourner le programme avec notre exemple :

Lorsqu'on applique la décomposition LU sur notre matrice A on obtient

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -0.5 & 1 & 0 & 0 \\ 0 & -0.2857 & 1 & 0 \\ 0 & 0 & -0.2692 & 1 \end{pmatrix} \quad et \quad U = \begin{pmatrix} 2 & -1 & 0 & 0 \\ 0 & 3.5 & -1 & 0 \\ 0 & 0 & 3.7142 & -1 \\ 0 & 0 & 0 & 1.7307 \end{pmatrix}$$

Puis en appliquant la descente puis la remontée on obtient au final la solution x du système $Ax = b$:

$$x = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

1.3. Validation

Pour vérifier si cette solution est bonne, on peut soit calculer directement $A * x$ en comparant avec b, soit calculer $\|Ax - b\|_\infty$. Pour que cette solution soit acceptée, il nous faut un résultat proche de 0. Après calcul on a comme résidu : $\|Ax - b\|_\infty = 1.776357e - 015$. On peut considérer que c'est 0, donc on valide cette unique solution.

2. Problème elliptique 1D et MDF

2.1 Schéma aux différences finies

Soit le maillage de [0, L] à n points : $0 = x_0 < ... < x_{n-1} = L$ où $x_i = i * h \quad \forall i \in 0, ..., n-1$ de pas $h = L/(n-1)$. Le schéma aux différences finies classique du problème elliptique, où u_i approxime $u(x_i)$ est donnée par :

$$\begin{cases} \epsilon/h^2(-u_{i-1} + 2u_i - u_{i+1}) + u_i = f(x_i) \forall i = 1, ..., n-2 \\ u_0 = a \quad ; \quad u_{n-1} = b \end{cases}$$

La formulation matricielle $AU^h = B$ correspondante est :

$$A = \begin{pmatrix} 1+2\epsilon/h^2 & -\epsilon/h^2 & 0 & \cdots & 0 \\ -\epsilon/h^2 & 1+2\epsilon/h^2 & -\epsilon/h^2 & & \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & -\epsilon/h^2 \\ 0 & \cdots & 0 & -\epsilon/h^2 & 1+2\epsilon/h^2 \end{pmatrix} \quad ; \quad U^h = \begin{pmatrix} u_1 \\ \vdots \\ \vdots \\ u_{n-2} \end{pmatrix} \quad ; \quad B = \begin{pmatrix} f(x_1) + u_0 * \epsilon/h^2 \\ \vdots \\ \vdots \\ f(x_{n-2}) + u_{n-1} * \epsilon/h^2 \end{pmatrix}$$

2.2 Application

Prenons un exemple :

$$\begin{cases} -10^{-5}u''(x) + u(x) = 0 \quad dans \quad]0, 1[\\ u(0) = 1 \quad ; \quad u(1) = 0 \end{cases}$$

On construit alors les matrices A et B pour ce schéma et on résout le système linéaire $AU^h = B$. Cependant, il faut choisir le pas $h = 1/N$ où N est le nombre d'intervalles que l'on fixe. On résout donc ce schéma pour différent N que l'on va fixé dans notre fichier NI.txt

On décide de fixer N = 50, 100, 200 et 400.
Dans mon programme solveurMDF.c, il y a la fonction Solveur_MDF_1D (Annexe 5) qui permet d'assembler les matrices A et B puis de résoudre le schéma avec la fonction Solveur_LU_Tridiagonal vu dans l'étape 1 et enfin de calculer l'erreur avec la solution exacte du problème.

Dans notre exemple, après avoir éxecuter le programme MDF.c on obtient alors l'erreur pour chaque N que l'on a fixé :

```
 ------------>> DEBUT DU PROGRAMME SOLVEURMDF <<--------------

N = 50  h = 0.020000  einf = 2.203127e-002
N = 100  h = 0.010000  einf = 4.159100e-002
N = 200  h = 0.005000  einf = 2.869491e-002
N = 400  h = 0.002500  einf = 8.821658e-003

 ------------>> FIN DU PROGRAMME SOLVEURMDF <<--------------
```

Figure 1: Sortie Python après éxecution de main.py

2.3 Enregistrement des résultats

Dans cette partie, on voudrait enregistrer les différents résultats des étapes précédentes dans des fichiers textes afin de les utiliser à posteriori pour l'étude de la convergence en maillages.

Tout d'abord on veut enregistrer le maillage dans un fichier X.txt, c'est à dire les points $x_1, ..., x_{n-2}$. On ne prend pas les points aux extrémités car on connait déjà leurs valeurs (conditions limites).
Pour cela on utilise la fonction fopen en C, avec l'option w+ qui permet l'ouverture (ou la création s'il n'existe pas) du fichier X.txt mais qui en plus écrase le contenu déjà présent. Puis pour chaque itération de calcul on y enregistre $x = i * h$ (Annexe 6).
On veut également enregistrer la solution exacte dans un fichier U.txt, pour cela on procède de la même manière que précédemment, à chaque calcule de la solution exacte on l'écrit dans le fichier U.txt (Annexe 7). Même chose pour Uh.txt (Annexe 8).

Pour le fichier Convergence.txt, on veut un affichage par ligne tel que : [N, h, einf]. Pour cela il faut faire une écriture pour chaque itération qui calcule Solveur_MDF_1D(N) pour chaque N du fichier NI.txt que l'on va stocker dans une liste après lecture du fichier (Annexe 9). On peut alors enregistrer chaque résultat dans le fichier Convergence.txt (Annexe 10).

3. Etude graphique de la convergence en maillages

3.1 Préparation des données

Dans cette partie on s'intéresse à la visualisation graphique de la convergence en maillages. On travaillera donc en parallèle avec Python.

A partir des étapes précédentes, on a maintenant les fichiers NI.txt, X.txt, U.txt, Uh.txt et Convergence.txt d'enregistrer dans notre espace de travail. On peut alors faire une lecture de ces fichiers pour stocker leurs contenus dans une liste. Exemple pour le fichier X.txt dans l'annexe 11.

On aimerait laisser le choix à l'utilisateur pour les intervalles étudiés. On lui demande alors s'il veut changer ou non, si oui on prend les nouvelles valeurs données par l'utilisateur puis on les stock dans notre liste N=[] mais il ne faut pas oublier également de changer les valeurs présente dans le fichier NI.txt, pour cela on fais une ouverture de fichier en écriture (Annexe 12).

Une fois qu'on a défini le fichier NI.txt, on peut passer au traitement, c'est à dire qu'on peut faire appel au programme éxécutable solveurMDF.exe via le programme Python main.py. Pour cela on utilise la fonction subprocess.run et on affiche dans la console la sortie du programme (Annexe 13).
Une fois éxécuté, les fichiers .txt sont mis à jour en fonction des valeurs de N que l'utilisateur a fixé.

On va alors de la même manière que pour la liste N, stocker le contenu des fichiers .txt dans des listes. Une fois les valeurs stockées, on peut alors les utiliser pour les représentation graphique.

3.2 Visualisation graphique

On trace la solution exacte et la solution approchée en fonction du maillage :

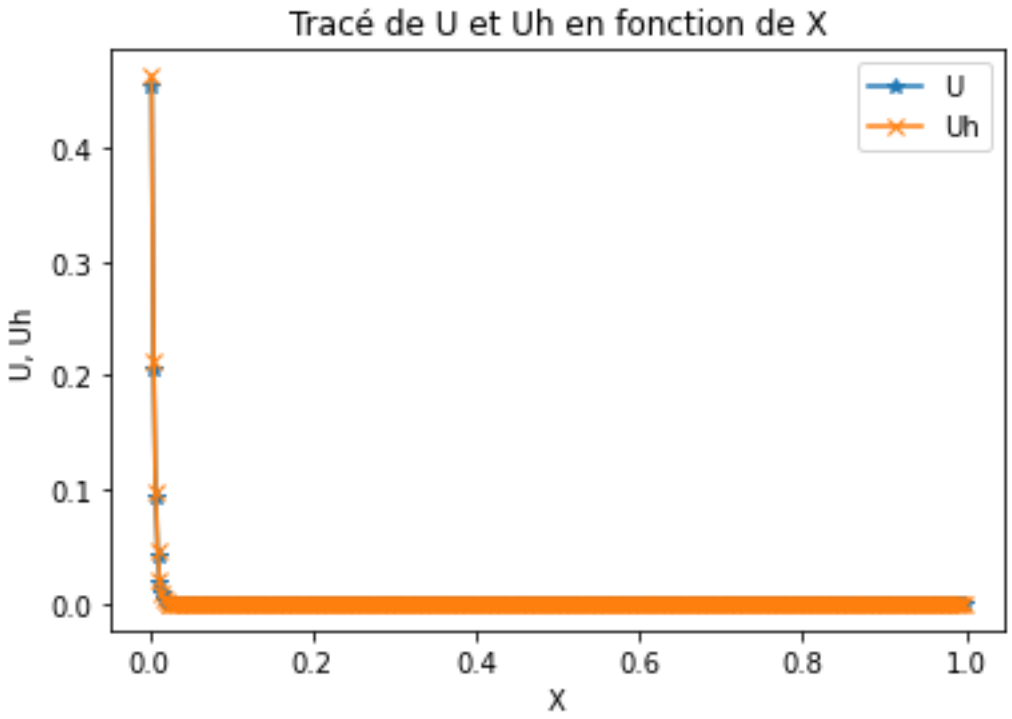

Figure 2: Tracé de U et Uh en fonction de X

On observe une chute brutale vers 0.

On trace l'erreur $|U - U^h|_\infty$:

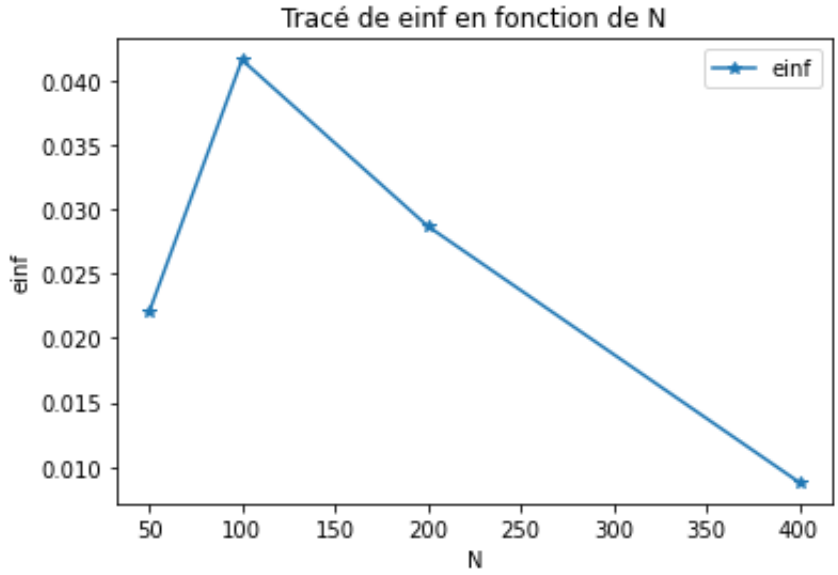

Figure 3: Tracé de l'erreur en fonction de N

On observe une erreur qui augmente lorsqu'on double N de 50 à 100, le résultat n'est pas très convaincant mais traçons tout de même la courbe de convergence en maillage :

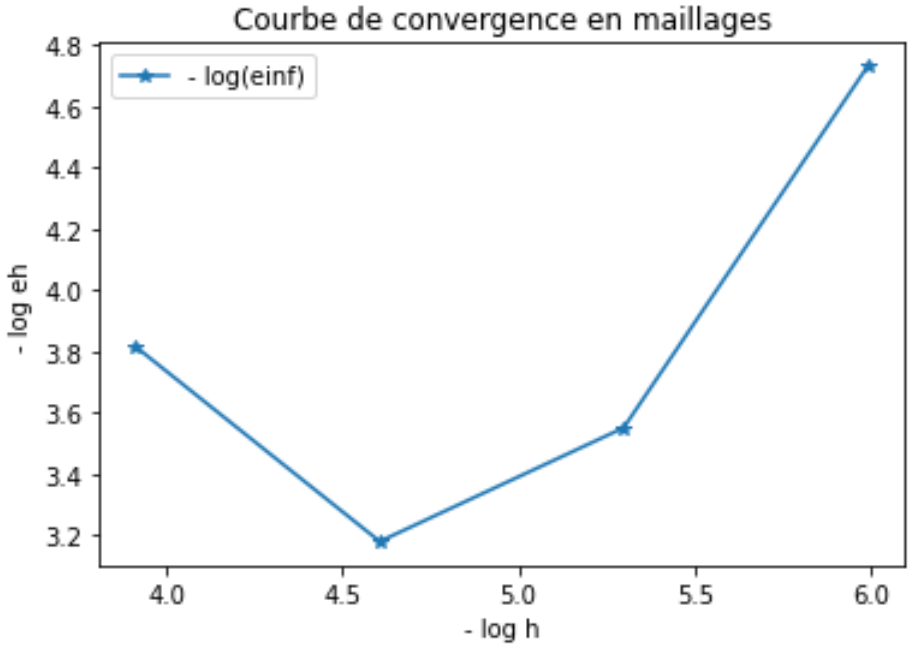

Figure 4: Courbe de convergence en maillages

Les points ne sont pas alignés, on ne peut donc pas calculer l'ordre de convergence p qui est normalement la valeur de la pente.

Essayons alors d'augmenter notre maillage, prenons par exemple N = 400, 800, 1600 et 3200 On obtient :

```
 -------------->> DEBUT DU PROGRAMME SOLVEURMDF <<--------------

N = 400  h = 0.002500  einf = 8.821658e-003
N = 800  h = 0.001250  einf = 2.326777e-003
N = 1600  h = 0.000625  einf = 5.966055e-004
N = 3200  h = 0.000313  einf = 1.496293e-004

 -------------->> FIN DU PROGRAMME SOLVEURMDF <<--------------
```

Figure 5: Sortie Python après éxecution de SolveurMDF

Cela permet d'ajouter plus de points et donc on observe mieux ce qu'il se passe lors de la chute brutale vers 0 :

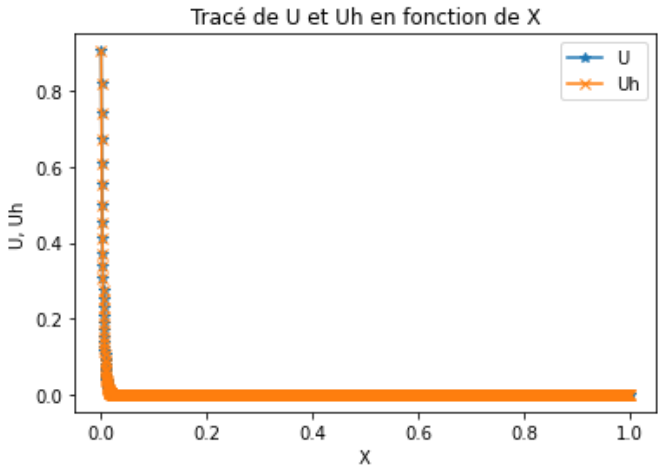

Figure 6: Tracé de U et Uh en fonction de X

On observe bien une erreur qui diminue lorsque N augmente.

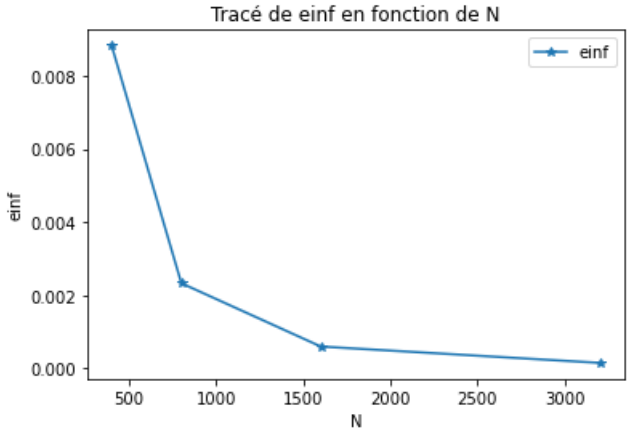

Figure 7: Tracé de l'erreur en fonction de N

La courbe de convergence obtenue est une droite où les points sont alignés, donc on peut calculer la pente.

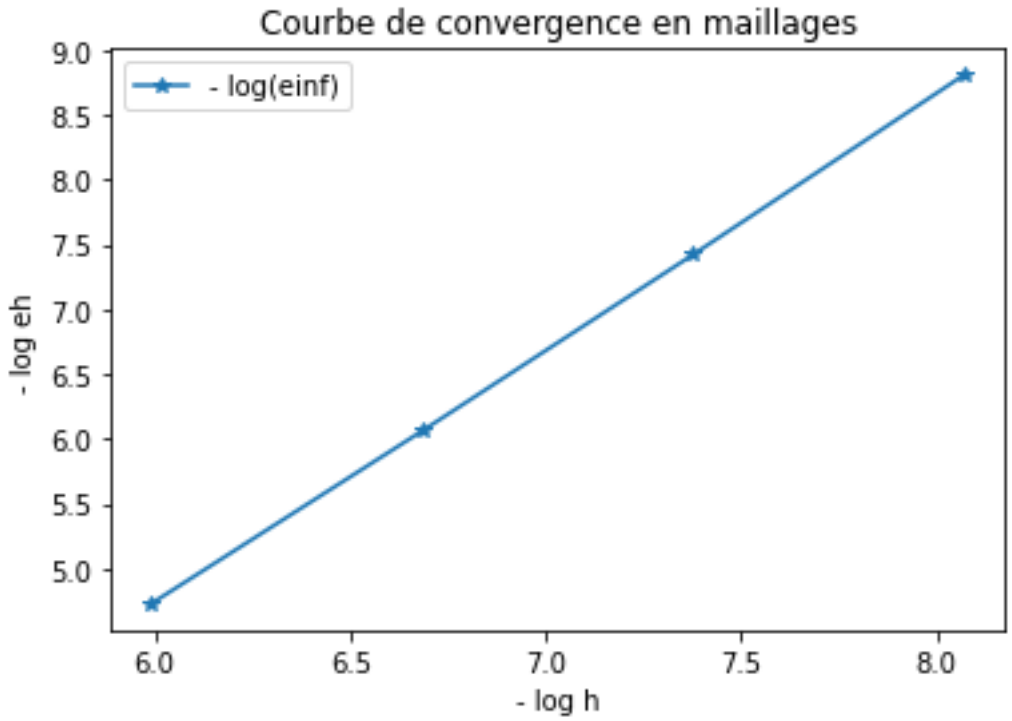

Figure 8: Courbe de convergence en maillages

Après calcul avec les valeurs du graphique, on obtient une pente p = 1.96.

On calcule aussi $p \approx 1/ln(2) * ln(e_h/e_{h/2})$ pour chaque étape, on obtient :

$$p = [1.9227168823652405, 1.9634837656306396, 1.9953845702002935]$$

On a alors bien une convergence vers 2.

Annexes

Annexe 1

```
void factorisation_LU(double** A, double** L, double** U, int n){
    int i,j;
    for(i=0; i<(n-1); i++){
            L[i+1][i] = A[i+1][i] / A[i][i];
            for(j=i; j<i+2; j++){
                    U[i][j] = A[i][j];}
            A[i+1][i+1] = A[i+1][i+1] - L[i+1][i] * U[i][i+1];}
    U[n-1][n-1] = A[n-1][n-1];}
```

Annexe 2

```
Vecteur * descente(double ** L, Vecteur * b, int n){
  int i;
  Vecteur * y;
  y = alloc_Vecteur(n);
  y->tab[0] = b->tab[0];
  for (i=1; i<n; i++){
      y->tab[i] = b->tab[i] - y->tab[i-1] * L[i][i-1];}
  return y;}
```

Annexe 3

```
Vecteur * remontee(double ** U, Vecteur * y, int n){
  int i;
  Vecteur * x = NULL;
  x = alloc_Vecteur(n);
  x->tab[n-1] = y->tab[n-1] / U[n-1][n-1];
  for(i=n-2; i>=0; i--){
    x->tab[i]=(y->tab[i]-U[i][i+1]*x->tab[i+1])/U[i][i];}
  return x;}
```

Annexe 4

```
Vecteur * Solveur_LU_TriDiagonal(double ** A, Vecteur * b){
    Vecteur * y = NULL;
    Vecteur * x = NULL;
    double ** L = NULL;
    double ** U = NULL;
    int n = b->dim;
    y = alloc_Vecteur(n); x = alloc_Vecteur(n);
    init_Vecteur(y); init_Vecteur(x);
    L = alloc_Matrice(n);
    U = alloc_Matrice(n);
    init_matrice(L,n,n);
    init_matrice(U,n,n);
    factorisation_LU(A,L,U,n);
    y = descente(L,b,n);
    x = remontee(U,y,n);
    free_vecteur(y);
    free_matrice(L,n);
    free_matrice(U,n);
    return x;}
```

Annexe 5

```
float Solveur_MDF_1D(int N){
    double ** A = NULL;
    Vecteur * B = NULL;
    Vecteur * Uh = NULL;
    Vecteur * U = NULL;
    Vecteur * y = NULL;
    int i;
    int n = (N+1.0);
    float L = 1.0; float h = L/N; float eps = pow(10,-5);
    float u0 = 1.0; float u1 = 0.0;
    float einf = 0.0;
    U = alloc_Vecteur(n-2); init_Vecteur(U);
    y = alloc_Vecteur(n-2); init_Vecteur(y);
    A = assemblage_A(eps,h,n);
    B = assemblage_B(u0,u1,eps,h,n);
    Uh = Solveur_LU_TriDiagonal(A,B); /* Resolution du systeme Uh? \ A1*Uh = B */
    /* Solution analytique du probleme */
    for (i=0; i<(n-2); i++){
        U->tab[i]=(1.0/(1.0-exp(-2.0/sqrt(eps))))*exp(-((i+1)(1.0/(double)N))/sqrt(eps
                +(1.0/(1.0-exp(2.0/sqrt(eps))))*exp(((i+1)(1.0/(double)N))/sqrt(eps));
    }
    /* Calcul de l'erreur ||U - Uh||_inf */
    for (i=0; i<(n-2); i++){
        y->tab[i] = U->tab[i] - Uh->tab[i] ;
    }
    einf = NORM_INF(y);
    free_matrice(A,n-2);
    free_vecteur(B);
    free_vecteur(Uh);
    free_vecteur(U);
    free_vecteur(y);
    return einf;}
```

Annexe 6

```
            f3 = fopen("./X.txt","w+");
            for (i=0; i<(n-2); i++){
                fprintf(f3,"%f\n",(i+1)*h);}
            fclose(f3);
```

Annexe 7

```
for (i=0; i<(n-2); i++){
    U->tab[i]=(1.0/(1.0-exp(-2.0/sqrt(eps)))) * exp(-((i+1)*(1.0/(double)N))/sqrt(eps))
        + (1.0/(1.0-exp(2.0/sqrt(eps)))) * exp(((i+1)*(1.0/(double)N))/sqrt(eps));}
f2 = fopen("./U.txt","w+"); /* ouverture ou creation + ecriture en supprimant dans U.t
for (i=0; i<(n-2); i++){
    fprintf(f2,"%e\n",U->tab[i]);}
fclose(f2);
```

Annexe 8

```
            f2 = fopen("./Uh.txt","w+");
            Uh = Solveur_LU_TriDiagonal(A,B);
            for (i=0; i<(n-2); i++){
                fprintf(f2,"%e\n",Uh->tab[i]);}
            fclose(f2);
```

Annexe 9

```
    /* Lecture du fichier NI.txt pour stocker les intervalles dans NI[] */
    f1 = fopen("./NI.txt","r"); /* lecture du fichier NI.txt */
    if (f1==NULL) {printf("Fichier␣Inexistant!\n");exit(1);}
    NI = alloc_Vecteur(4);
    while ( 1 ) {
        nbl = fscanf(f1,"%d",&N); /* :lecture(s) */
        if ( nbl==EOF ){break;}
        NI->tab[nbNI] = N;
        nbNI = nbNI + 1; /*nb d'intervalles*/
    }
    fclose(f1);
```

Annexe 10

```
/* On calcule einf = Solveur_MDF_1D pour chaque intervalle qui se trouve dans NI[] */
    f2 = fopen("./Convergence.txt","w+");
    einf = alloc_Vecteur(nbNI);
    for (i=0; i<nbNI; i++){
        einf->tab[i] = Solveur_MDF_1D((int)NI->tab[i]);
        fprintf(f2,"%d␣%f␣%e␣\n",(int)NI->tab[i], 1.0/NI->tab[i], einf->tab[i]);
    }
    fclose(f2);
```

Annexe 11

```
# Lecture du fichier X.txt et stockage du contenu dans un tableau
f1 = open('X.txt', 'r')
X = []
lignes = []
lignes = f1.readlines()
# Parcours de chaque ligne du fichier et ajout des chiffres dans le tableau
for i in lignes:
    # Suppression des espaces en debut et fin de ligne (ici les '\n')
    i = i.strip()
    # Ajout du contenu de chaque ligne dans liste X
    X.append(float(i))
f1.close()
```

Annexe 12

```
reponse = input("? Voulez-vous changez les intervalles ? Oui = 1 / Non = 0\n")
if reponse == "1":
    new_N = input('Entrez les nouvelles valeurs , separees par des espaces')
    new_N = [int(val) for val in new_N.split()]
    f1 = open('NI.TXT', 'w')
    for val in new_N:
        f1.write(str(val) + '\n')
    f1.close()
    f1 = open('NI.TXT', 'r')
    N = []
    lignes = f1.readlines()
    for i in lignes:
        i = i.strip()
        if i.isdigit():
            N.append(int(i))
    f1.close()
    print("--> Les nouvelles valeurs ont ete enregistrees dans le fichier NI.TXT")
    print("--> Voici la liste des nouveaux nombres d'intervalles etudies :",N)
else:
    print("Le contenu du fichier NI.TXT n'a pas ete modifie.")
```

Annexe 13

```
# Execution du programme solveurMDF.exe
result = subprocess.run(["./solveurMDF.exe"], capture_output=True)
# Affichage de la sortie du programme en C
print(result.stdout.decode())
```