

---

**SEBAOUNE Ahmed**

N\_ETUDIANT : 22206789

# Rapport TP: Calcul Numérique

13 janvier 2023

Exo 1:

1-

On suppose que  $\varepsilon > 0$  et soit  $h \in \mathbb{R}$

$$|x + \varepsilon| \leq h$$

Utilisant le développement de Taylor  
à l'ordre 2 on aura :

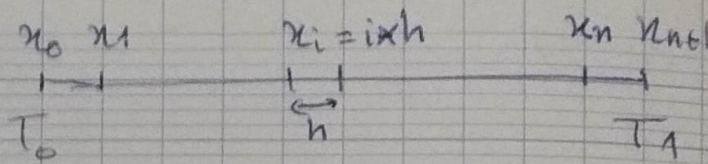
$$\begin{cases} T(x+h) = T(x) + h \frac{\partial T}{\partial x} + \frac{h^2}{2} \frac{\partial^2 T}{\partial x^2} + o(h^2) \\ T(x-h) = T(x) + h \frac{\partial T}{\partial x} + \frac{h^2}{2} \frac{\partial^2 T}{\partial x^2} + o(h^2) \end{cases}$$

$$T(x+h) + T(x-h) = 2T(x) + h^2 \frac{\partial^2 T}{\partial x^2} + o(h^2)$$

$$- \frac{\partial^2 T}{\partial x^2} = \frac{-T(x-h) + 2T(x) - T(x+h)}{h^2}$$

2/ On note pour  $\forall i \in [0, n+1]$

$$\begin{cases} T(x_i) = U_i \\ A \cdot x \cdot U = f \end{cases}$$



$$x_{N+1} = (N+1)h = \frac{N+1}{N+1} = 1$$

alors

$$h = \frac{1}{N+1}$$

$$\begin{cases} T(x_0) = T_0 = U_0 \\ T(x_{N+1}) = T(1) = T_1 = U_{N+1} \end{cases}$$

$$K \frac{-U_{i-1} + 2U_i - U_{i+1}}{h^2} = g_i \quad \forall i \in [1, N]$$

Pour  $i=1$

$$K \frac{-U_1 + 2U_2 - U_3}{h^2} = g_2$$

$$\boxed{\cancel{U_1} + 2\cancel{U_2} - \cancel{U_3} = \frac{h^2}{K} \cancel{g_2}}$$



Pour  $i=2$

$$K = \frac{-U_1 + 2U_2 - U_3}{h^2} = g_2$$

on déduit que :

$$-U_1 + 2U_2 - U_3 = \frac{h^2}{K} g_2$$

Pour  $i=N$

$$\begin{cases} K - U_{N-1} + 2U_N - U_{N+1} = g_N \\ T(x_{N+1}) = T_1 - U_{N+1} \end{cases}$$

soit alors :

$$-U_{N-1} + 2U_N = \frac{h^2}{K} g_N + T_i$$

Alors le système linéaire :

$$\begin{cases} A \times U = f \\ A \in \mathbb{R}^{n \times n}, U \text{ et } f \in \mathbb{R}^n \end{cases}$$

---

## Les matrices General Band

### EXO 3 :

#### 1. creation et allocation de matrice pour CBLAS et LAPACK

Une matrice doit être stockée sous forme “General Band” c’est-à-dire on stocke les sous diagonales comme des lignes ou des colonnes.

Row Major : les éléments contigus appartiennent à la même ligne,

Column Major: les éléments contigus appartiennent à la même colonne.



```
1 double* AB = (double *) malloc(sizeof(double)*lab*la);
2
```

#### 2. LAPACK\_COL\_MAJOR

signifie que les matrices sont stockées en mémoire en suivant un ordre de colonne. Cela signifie que les éléments d'une colonne de la matrice sont stockés les uns à la suite des autres dans la mémoire.

- **dgbmv** : Double General Banded Matrix-Vector elle permet de calculer le produit d'une matrice bande générale double précision par un vecteur. La fonction modifie le vecteur y pour lui donner la solution du système linéaire d'équations  $A \cdot x = y$ . Les paramètres :
  - trans: Cette variable spécifie la forme de la matrice A. Elle peut prendre les valeurs 'N' pour une matrice non transposée ou 'T' pour une matrice transposée.
  - m: C'est le nombre de lignes de la matrice A.

- 
- n: C'est le nombre de colonnes de la matrice A.
  - kl: C'est le nombre de bandes inférieures de la matrice A. Il s'agit du nombre de colonnes de la matrice qui sont situées en dessous de la diagonale principale.
  - ku: C'est le nombre de bandes supérieures de la matrice A. Il s'agit du nombre de colonnes de la matrice qui sont situées au-dessus de la diagonale principale.
  - alpha: C'est un scalaire qui multiplie la matrice A.
  - A: C'est un pointeur vers la matrice A. La matrice A est stockée en mémoire sous forme de bande générale, c'est-à-dire que seuls les coefficients de la diagonale principale et des diagonales adjacentes sont stockés en mémoire.
  - lda: C'est la taille de l'espacement entre les lignes de la matrice A.
  - x: C'est un pointeur vers le vecteur x.
  - incx: C'est l'espacement entre les éléments consécutifs du vecteur x.
  - beta: C'est un scalaire qui multiplie le vecteur y avant de l'ajouter à la matrice A multiplié par x.
  - y: C'est un pointeur vers le vecteur y.
  - incy: C'est l'espacement entre les éléments consécutifs du vecteur y.
- 
- **dgbtrf** : Double General Banded Matrix LU factorization. elle calcule la factorisation LU de la matrice bande générale A. Cela signifie qu'elle décompose la matrice A en deux matrices triangulaires L et U tels que  $A = LU$ . Les matrices L et U remplacent les coefficients de A dans la mémoire. les paramètres :
    - n: C'est la dimension de la matrice bande générale A.
    - kl: C'est le nombre de bandes inférieures de la matrice A. Il s'agit du nombre de colonnes de la matrice qui sont situées en dessous de la diagonale principale.
    - ku: C'est le nombre de bandes supérieures de la matrice A. Il s'agit du nombre de colonnes de la matrice qui sont situées au-dessus de la diagonale principale.
    - A: C'est un pointeur vers la matrice A. La matrice A est stockée en mémoire sous forme de bande générale, c'est-à-dire que seuls les

---

coefficients de la diagonale principale et des diagonales adjacentes sont stockés en mémoire.

- `lda`: C'est la taille de l'espacement entre les lignes de la matrice A.
  - `ipiv`: C'est un pointeur vers un tableau de taille n qui contiendra les permutations de pivot de Gauss.
  - `info`: C'est un entier qui contiendra un code d'erreur après l'exécution de la fonction. Si `info` = 0, la factorisation a réussi. Si `info` < 0, il y a eu une erreur.
- **`dgbtrs`** : utilisée pour résoudre un système linéaire d'équations matricielles de la forme  $Ax = B$ , où A est une matrice de bande générale à double précision de dimension (n x n), La fonction `dgbtrs` utilise la méthode de LU (décomposition en matrices triangulaires) pour résoudre le système linéaire.
    - `trans (input)` : spécifie la forme de la matrice de bande. Peut prendre les valeurs 'N' (non transposée), 'T' (transposée) ou 'C' (conjuguée transposée).
    - `n (input)` : nombre de lignes de la matrice B.
    - `kl (input)` : nombre de bandes inférieures de la matrice A.
    - `ku (input)` : nombre de bandes supérieures de la matrice A.
    - `nrhs (input)` : nombre de colonnes de la matrice B.
    - `ab (input)` : matrice de bande A de dimension (kl+ku+1) x n.
    - `ldab (input)` : longueur de la première dimension de la matrice A, c'est-à-dire la longueur de la ligne.
    - `ipiv (input)` : vecteur d'indice de permutation de dimension n.
    - `b (input/output)` : matrice de dimension n x nrhs qui contient les termes indépendants (input) et les solutions (output).
    - `ldb (input)` : longueur de la première dimension de la matrice B.
    - `info (output)` : variable d'état qui contient 0 si la résolution s'est déroulée avec succès, sinon un entier positif indiquant la raison de l'échec.
  - **`dgbtsv`** : est une routine de LAPACK qui résout un système linéaire de type  $Ax = b$ . La routine utilise un algorithme de décomposition LU de bande pour résoudre ce système linéaire. Ensuite, on résout le système linéaire en deux étapes en utilisant les matrices triangulaires L et U.
    - `N` : la taille de la matrice A et des vecteurs x et b.
    - `KL` : le nombre de bandes inférieures de la matrice A.

- 
- KU : le nombre de bandes supérieures de la matrice A.
  - A : la matrice bande A, stockée dans l'ordre COL MAJOR.
  - LDA : la taille de l'allocation mémoire de la matrice A.
  - IPIV : un tableau qui stocke les permutations de ligne.
  - B : le vecteur b.
  - LDB : la taille de l'allocation mémoire du vecteur b.

8.

1. Calculer le vecteur résidu  $r = b - Ax$ .
2. Calculer la norme euclidienne du vecteur résidu en utilisant la fonction BLAS "dnrm2" .
3. Calculer la norme euclidienne de b en utilisant la même fonction BLAS.
4. Calculer la norme du résidu relatif en divisant la norme du résidu par la norme de b, c'est-à-dire:  $\|r\| / \|b\|$ .



## EXO 4 :

### 1. Écriture de stockage GB en priorité colonne pour la matrice de Poisson 1D



```
1 set_GB_operator_colMajor_poisson1D(AB, &lab, &la, &kv);
2 write_GB_operator_colMajor_poisson1D(AB, &lab, &la, "DATA/DIRECT/AB/AB.dat");
3
```



```
1 void set_GB_operator_colMajor_poisson1D(double* AB, int *lab, int *la, int *kv){
2     int ii, jj, kk;
3     for (jj=0;jj<(*la);jj++){
4         kk = jj*(*lab);
5         if (*kv>=0){
6             for (ii=0;ii< *kv;ii++){
7                 AB[kk+ii]=0.0;
8             }
9         }
10        AB[kk+ *kv]=-1.0;
11        AB[kk+ *kv+1]=2.0;
12        AB[kk+ *kv+2]=-1.0;
13    }
14    AB[0]=0.0;
15    if (*kv == 1) {AB[1]=0;}
16
17    AB[(*lab)*(*la)-1]=0.0;
18 }
```

## 2. Utilisation la fonction BLAS dgbmv avec cette matrice

```
1  cblas_dgbmv(CblasColMajor,CblasConjTrans,la,la,kl,ku,1.0,AB+1,lab,EX_SOL,1,0.0,RHS,1);  
2
```

## 3 .Proposez une méthode de validation

On peut calculer l'erreur relative et on vérifi

$$ERR = \frac{EX\ SOL - RHS}{RHS}$$

Le résultat obtenu après exécution est : 2.692638e-16 ,donc la fonction dgbmv donne des valeurs cohérentes.

## EXO 5 :

1. Résolution du système linéaire par une méthode directe en faisant appel à LAPACK

### A.Partie DGBTRF, DGBTRS :

Utilisant les fonctions DGBTRF pour la factorisation et DGBTRS pour résoudre le système  $Ax = B$  on aura se code :

```
1 // DGBMV and DGBTRF DGBTRS to find RHS
2
3 // double* VECTOR_dgbm
4 cblas_dgbmv(CblasColMajor,CblasConjTrans,la,la,kl,ku,1.0,AB+1,lab,EX_SOL,1,0.0,RHS,1);
5 write_vec(RHS, &la, "DATA/RHS/RHS_dgbmv.dat");
6
7 printf("Solution with LAPACK\n");
8 /* LU Factorization */
9 info=0;
10 ipiv = (int *) calloc(la, sizeof(int));
11 LAPACK_dgbtrf(&la, &la, &kl, &ku, AB, &lab, ipiv, &info);
12
13 /* LU for tridiagonal matrix (can replace dgbtrf) */
14 ierr = dgbtrftridiag(&la, &la, &kl, &ku, AB, &lab, ipiv, &info);
15 write_GB_operator_colMajor_poisson1D(AB, &lab, &la, "DATA/AB/AB_dgbtrftridiag.dat");
16
17 /* Solution (Triangular) */
18 if (info==0){
19     LAPACK_dgbtrs("N", &la, &kl, &ku, &NRHS, AB, &lab, ipiv, RHS, &la, &info);
20     write_vec(RHS, &la, "DATA/RHS/RHS_dgbtrs.dat");
21     if (info!=0){printf("\n INFO DGBTRS = %d\n",info);}
22 }else{
23     printf("\n INFO = %d\n",info);
24 }
25
```

les résultats seront stockés dans le répertoire DATA/DIRECT :

DATA

DIRECT

AB

RHS

SOL

X\_grid.dat

ITERATIVE

RHS

RHS\_dgbmv.dat

RHS\_dgbsv.dat

RHS\_dgbtrs.dat

RHS.dat

1 -3.888889

2 -2.777778

3 -1.666667

4 -0.555556

5 0.555556

6 1.666667

7 2.777778

8 3.888889

## B.Partie DGBSV :

1 set\_GB\_operator\_colMajor\_poisson1D(AB, &lab, &la, &kv);

2 set\_dense\_RHS\_DBC\_1D(RHS,&la,&T0,&T1);

3

4

5 /\* It can also be solved with dgbsv (dgbtrf+dgbtrs) \*/

6 LAPACK\_dgbsv(&la, &kl, &ku, &NRHS, AB, &lab, ipiv, RHS, &la, &info);

7

RHS

RHS\_dgbmv.dat

RHS\_dgbsv.dat

RHS\_dgbtrs.dat

RHS.dat

1 0.111111 -3.888889

2 0.222222 -2.777778

3 0.333333 -1.666667

4 0.444444 -0.555556

5 0.555556 0.555556

6 0.666667 1.666667

7 0.777778 2.777778

8 0.888889 3.888889

9

---

## 2. Evaluation de performances :

en calculant le temps d'exécution on aura :

```
● ahmed@ahmed-UX550VD:~/Desktop/CHPS/Claculs/Rapp_V2$ make run_tpPoisson1D_direct
bin/tpPoisson1D_direct
----- Poisson 1D -----

Solution with LAPACK
Execution time of DGBTRF and DGBTRS 0.004077 seconds
Execution time of DGBSV 0.000360 seconds
The relative forward error is relres = 2.692638e-16


----- End -----
```

On voit que les performances de DGBSV sont meilleures que celles de DGBTRF + DGBTRS.



## EXO 6 :

1.Implémentation de la méthode de factorisation LU pour les matrices tridiagonales avec le format GB.



```
1 void LU_Facto(double* AB, int *lab, int *la, int *kv){
2     int i, j, k, k1 = 3;
3     if (*kv>=0){
4         k1 = 4;
5         for (i=0;i< *kv;i++){
6             AB[i]=0.0;
7         }
8     }
9     AB[*kv+2]/=AB[*kv+1];
10    for (j=1;j<(*la);j++){
11        k = j*(lab);
12        if (*kv>=0){
13            for (i=0;i< *kv;i++){
14                AB[k+i]=0.0;
15            }
16        }
17
18        printf("kv+2 = %lf\n",AB[(k-3)+ *kv+2]);
19
20        AB[k+ *kv+1]-=AB[k+ *kv]*AB[(k-k1)+ *kv+2];
21        AB[k+ *kv+2]/=AB[k+ *kv+1];
22    }
23
24 }
```

## 2.Méthode de validation :

On va remplacer la fonction de dgbtrf par notre fonction LU\_Facto.

```
1
2 LU_Facto(AB, &lab, &la, &kv);
3
4 if (info==0){
5 // ierr = dgbtrs_("N", &la, &kl, &ku, &NRHS, AB, &lab, ipiv, RHS, &la, &info);
6 LAPACK_dgbtrs("N", &la, &kl, &ku, &NRHS, AB, &lab, ipiv, RHS, &la, &info);
7 write_vec(RHS, &la, "DATA/DIRECT/RHS/RHS_dgbtrs.dat");
8 if (info!=0){printf("\n INFO DGBTRS = %d\n",info);}
9 }else{
10 printf("\n INFO = %d\n",info);
11 }
```

On calculant l'erreur relative entre le vecteur solution et le vecteur RHS qui est le résultat de la résolution du système  $Ax = B$  avec la fonction dgbtrf et on aura l'erreur :

3.776012e-01, une valeur négligeable donc le résultat est valide.

$$ERR = \frac{EX\ SOL - RHS}{RHS}$$

## Méthode de résolution itérative:

1. L'implantation de Richardson avec des matrices au format GB

la fonction pour calculer le alpha optimale :

```
1 double richardson_alpha_opt(int *la){
2
3     return 2 / (eigmax_poisson1D(la) + eigmin_poisson1D(la));
4 }
```

et on a créé les deux fonctions sigma max et sigma min:

```
1 double eigmax_poisson1D(int *la){
2     double eigmax;
3     eigmax=sin(*la *M_PI_2*(1.0/(*la+1)));
4     eigmax=4*eigmax*eigmax;
5     return eigmax;
6 }
7
8 double eigmin_poisson1D(int *la){
9     double eigmin;
10    eigmin=sin(M_PI_2*(1.0/(*la+1)));
11    eigmin=4*eigmin*eigmin;
12    return eigmin;
13 }
```

et voici le code de l'itération de Richardson :

```
1 void richardson_alpha(double *AB, double *RHS, double *X, double *alpha_rich, int *lab,
2                       int *la,int *ku, int*kl, double *tol, int *maxit, double *resvec, int *nbite){
3     int nb_ittertions =0;
4
5     cblas_dgbmv(CblasColMajor, CblasConjTrans, *la, *la, *kl, *ku, 1.0, AB, *lab, X, 1, 0.0, resvec, 1);
6     cblas_dscal(*la, -1.0, resvec, 1);
7     cblas_daxpy(*la, 1.0, RHS, 1, resvec, 1);
8
9     double releres = cblas_dnorm(*la, resvec, 1) / cblas_dnorm(*la, RHS, 1);
10
11    while (releres > (*tol) && *nbite < *maxit)
12    {
13        cblas_daxpby(*la, *alpha_rich, resvec, 1, 1.0, X, 1);
14
15
16        cblas_dgbmv(CblasColMajor, CblasConjTrans, *la, *la, *kl, *ku, 1.0, AB, *lab, X, 1, 0.0, resvec, 1);
17        cblas_dscal(*la, -1.0, resvec, 1);
18        cblas_daxpy(*la, 1.0, RHS, 1, resvec, 1);
19        *nbite += 1;
20        releres = cblas_dnorm(*la, resvec, 1) / cblas_dnorm(*la, RHS, 1);
21        nb_ittertions++;
22    }
23    printf("\nLe nombre d'iteration %d\n",nb_ittertions);
24 }
25
```

---

après l'exécution on se trouve avec 125 itération dans notre cas de figure.

```
● ahmed@ahmed-UX550VD:~/Desktop/CHPS/Claculs/Rapp_V2$ make run_tpPoisson1D_iter
bin/tpPoisson1D_iter
----- Poisson 1D -----

Optimal alpha for simple Richardson iteration is : 0.500000
Le nombre d'iteration 125

----- End -----
```

## 2. Calcule d'erreur par rapport au résultat analytique

on calculant l'erreur relative :

```
ahmed@ahmed-UX550VD:~/Desktop/CHPS/Claculs/Rapp_V2$ make run_tpPoisson1D_iter
bin/tpPoisson1D_iter
----- Poisson 1D -----

Optimal alpha for simple Richardson iteration is : 0.500000
Le nombre d'iteration 125

The relative forward error is relres = 1.770223e+00

----- End -----
```

$\text{err\_rel} = 1.770223\text{e}+00$

### 3. La courbe de convergence utilisant le fichier RESVEC.dat

