# ALLINU - Travail

## Sommaire



## 1 Introduction

Pour rappel, un système d'équation linéaire est un ensemble d'équations linéaires portant sur les mêmes inconnues. D'un point de vue général, un système linéaire s'écrit sous la forme suivante :

$$\begin{cases} \alpha_1 x_1 + \alpha_2 x_2 + \dots \alpha_n x_n = b_1 \\ \beta_1 x_1 + \beta_2 x_2 + \dots \beta_n x_n = b_2 \\ \vdots \\ eta a_1 x_1 + \eta_2 x_2 + \dots \eta_n x_n = b_n \end{cases} \tag{1}$$

Ce genre de système peut s'écrire sous la forme matricielle suivante :

$$A\vec{x} = \vec{b} \text{ avec } A = \begin{pmatrix} \alpha_1 & \alpha_2 & \dots & \alpha_n \\ \beta_1 & \beta_2 & \dots & \beta_n \\ \vdots & \vdots & \ddots & \vdots \\ \eta_1 & \eta_2 & \dots & \eta_n \end{pmatrix} \tag{2}$$

De nombreux problèmes font intervenir ce type de système d'équations, que ce soit en mathématique, dans des problèmes d'optimisation, ou en physique numérique, dans des domaines comme le traitement du signal, la simulation de circuits électriques, la résolution de systèmes mécaniques hamiltoniens, ...

De ce fait, il existe de nombreuses méthodes de résolutions de systèmes linéaires, que ce soit par la règle de Cramer, par la factorisation LU, par l'élimination de Gauss, ... . D'un point de vue général, on peut séparer ces méthodes en deux catégories : Les méthodes directes, permettant de calculer la solution de façon exacte, et les méthodes itératives, fonctionnant par approches successives de la solution.

Dans le cadre de ce travail, nous avons dû nous intéresser à deux autres méthodes de résolutions de systèmes linéaires, les méthodes d'Aasen et de Bunch-Parlett. Pour cela, j'ai dû écrire un programme en C permettant de résoudre différents systèmes linéaires à partir de ces deux méthodes. Je commencerai donc ce travail par un rappel des deux méthodes. Je décrirai ensuite brièvement mon code, pour enfin effectuer une brève analyse des résultats.

Pour information, les différents codes utilisés au cours de ce travail sont disponibles en annexe.

# 2 Méthodes de résolution de système linéaire

## 2.1 Méthode d'Aasen

La méthode d'Aasen est une méthode de résolution de système symétrique indéfini. Celle-ci consiste à décomposer la matrice $A$ de la façon suivante :

$$A = P^{\mathrm{T}} L T L^{\mathrm{T}} P \tag{3}$$

où $P$ est une matrice de permutation, $L$ est une matrice triangulaire inférieure inférieure unité et $T$ est une matrice symétrique tridiagonale, qui s'écrit donc comme :

$$T = \begin{pmatrix} \alpha_1 & \beta_1 & & \dots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{n-1} \\ 0 \dots & & \beta_{n-1} & \alpha_n & \end{pmatrix} \tag{4}$$

Nous avons ici considéré que la matrice $T$ est constitué d'éléments rééls, comme dans les systèmes que nous avons dû résoudre.

L'algorithme de factorisation de Aasen est disponible en annexe. Dans le cadre de ce travail, je n'ai pas eu à implémenter cette méthode numérique, je devais seulement utiliser une fonction déjà implémentée en C permettant de calculer les matrices $L$, $P$ et $T$ à partir de la matrice $A$. Cette fonction est disponible en annexe.

Une fois ces matrices obtenues, la solution du système linéaire $Ax = b$ s'obtient de la façon suivante :

**Algorithm 1** Calcul de la solution par la méthode d'Aasen

1: **function** SOLVE(P,L,T,b)
2: $b \leftarrow Pb$
3: Résoudre $Lz = b$
4: Résoudre $Tw = z$
5: Résoudre $L^{\mathrm{T}} y = w$
6: $x \leftarrow P^{\mathrm{T}} y$
7: **return** $x$

Pour implémenter ces étapes numériques, j'ai dû écrire ou utiliser les fonctions et sous-routines suivantes :
- Fonction gaxPD : Fonction permettant, à partir d'une matrice et d'un vecteur donné, d'obtenir le produit de cete matrice et ce vecteur. J'ai implémenter cette fonction afin de calculer les produits des étapes 1 et 5
- Sous-routine DTRTRS : Sous-routine de LAPack permettant de résoudre un système linéaire de la forme $Ax = b$, où $A$ est une matrice triangulaire. Cette sous-routine a été utilisée pour les étapes 2 et 4, $L$ étant une matrice triangulaire
- Sous-routine DGTSV : Sous-routine de LAPack permettant de résoudre un système linéaire de la forme $Ax = b$, où $A$ est une matrice tridiagonale. Cette sous-routine a donc été utilisée pour l'étape 3

Les informations complémentaires sur l'utilisation de gaxPD, DTRTRS et DGTSV sont disponibles en annexe

## 2.2 Méthode de Bunch-Parlett

La méthode de Bunch-Parlett est également une méthode de résolution de système symétrique indéfini. Dans ce cas-ci, nous décomposons la matrice $A$ de la façon suivante :

$$A = P^{\mathrm{T}} L D L^{\mathrm{T}} P \tag{5}$$

où $P$ est une matrice de permutation, $L$ est une matrice triangulaire inférieure unité et $D$ est une somme directe de matrice $1 \times 1$ ou $2 \times 2$. La solution du système est alors donnée de la façon suivante :

**Algorithm 2** Calcul de la solution par la méthode de Bunch-Parlett

1: **function** SOLVE(P,L,D,b)
2: $b \leftarrow Pb$
3: Résoudre $Lz = b$
4: Résoudre $Dw = z$
5: Résoudre $L^{\mathrm{T}} y = w$
6: $x \leftarrow P^{\mathrm{T}} y$
7: **return** $x$

Contrairement à la méthode précedente, il n'a pas été nécessaire d'implémenter un à un les différents produits et les différentes résolutions du système. La méthode de Bunch-Parlett peut-être appliquée à partir des deux sous-routines suivantes :

– Sous-routine DSYTRF : Sous-routine de LAPack permettant de calculer la factorisation de Bunch-Parlett à partir d'une matrice A donnée, en stockant les éléments de $L$ et de $D$ dans une même matrice, et renvoie également un vecteur contenant des informations sur la matrice $P$.
– Sous-routine DYTRS : Sous-routine de LAPack permettant de résoudre le système linéaire, à partir de la matrice et des informations sur $P$ obtenues par la sous-routine DSYTRF

Des informations supplémentaires sur l'utilisation de ces sous-routines sont disponibles en annexe.

# 3 Résultats obtenus

Au cours de ce programme, j'ai dû appliquer les deux méthodes implémentées en C à cinq systèmes appartenant à la collection de matrices Harwell-Boeing, afin de comparer les solutions obtenues. Les matrices utilisées sont reprises dans le tableau suivant :

| Système | lund_a | bfw398b | bcsstk19 | 1138_bus | bcsstm27 |
|---|---|---|---|---|---|
| Dimensions | $147 \times 147$ | $398 \times 398$ | $817 \times 817$ | $1138 \times 1138$ | $1224 \times 1224$ |
| $\|A\|_1$ | $2.85 \cdot 10^8$ | $3.40 \cdot 10^{-5}$ | $2.26 \cdot 10^{14}$ | $4.04 \cdot 10^7$ | $4.94 \cdot 10^3$ |
| $\|A\|_2$ | $2.24 \cdot 10^8$ | $3.15 \cdot 10^{-5}$ | $1.92 \cdot 10^{14}$ | $3.01 \cdot 10^4$ | $3.93 \cdot 10^3$ |
| $\|A\|_\infty$ | $2.85 \cdot 10^8$ | $3.40 \cdot 10^{-5}$ | $2.26 \cdot 10^{14}$ | $4.04 \cdot 10^4$ | $4.94 \cdot 10^3$ |
| $k_1(A)$ | $5.44 \cdot 10^6$ | $3.64 \cdot 10^1$ | $2.81 \cdot 10^{11}$ | $1.23 \cdot 10^7$ | $1.14 \cdot 10^{10}$ |
| $k_2(A)$ | $2.80 \cdot 10^8$ | $2.11 \cdot 10^1$ | $1.34 \cdot 10^{11}$ | $8.57 \cdot 10^6$ | $6.92 \cdot 10^9$ |
| $k_\infty(A)$ | $5.44 \cdot 10^6$ | $3.64 \cdot 10^1$ | $2.81 \cdot 10^{11}$ | $1.23 \cdot 10^7$ | $1.14 \cdot 10^{10}$ |

TABLE 1 – Ordres, normes et conditionnements des matrices utilisées

Ces matrices sont disponible à l'adresse suivante, sous le format de fichier .mtx :

`http://math.nist.gov/MatrixMarket/searchtool.html`

Lors de l'étude d'une technique de résolution numérique, il est important de ce concentrer sur trois facteurs importants :

1. Le nombre d'opérations arithmétiques nécesaires, en terme de FLOPS (Nombre d'opération à virgule flottante nécessaire pour le calcul). Dans notre cas, les deux méthodes ont un cout qui s'évalue en $\frac{n^3}{3}$, et nous ne pouvons donc pas comparer les méthodes à ce niveau là
2. Le nombre d'opérations de stockage néessaires. De nouveau, dans les deux méthodes utilisées, ce nombre est le même, de l'ordre de $\frac{n^2}{2}$, $n$ étant la dimension de la matrice.
3. Les erreurs dans les solutions, que nous allons analyser dans la suite.

Les matrices étant de grandes dimensions (dans certains cas, $m = n > 1000$), il serait ridicule de donner dans ce rapport l'ensemble des solutions obtenues. J'ai donc préféré me concentrer sur une analyse globale des erreurs obtenues. Pour cela j'ai, pour chaque système, calculer l'écart absolu entre chaque élément des solutions obtenus par les méthodes de Aasen et Bunch-Parlett et l'élement correspondant de la solution calculée avec MatLab, cette dernière faisant office de solution optimale, comme nous avons pu nous en rendre compte au cours des scéances de travaux pratique. A partir des écarts absolus, j'ai calculé les écart relatifs.

Le tableau ci-dessous reprends les valeurs des écarts moyens (absolu et relatif), des écarts quadratiques moyens et les valeurs maximales des écarts (absolus et relatifs) pour chaque système :

| Système | lund_a | bfw398b | bcsstk19 | 1138_bus | bcsstm27 |
|---|---|---|---|---|---|
| max(E.A.) Aasen | $7.46 \cdot 10^{-2}$ | $8.73 \cdot 10^{7}$ | $3.57 \cdot 10^{-3}$ | $1.51 \cdot 10^{3}$ | $9.73 \cdot 10^{6}$ |
| max(E.A.) Bunch | $1.33 \cdot 10^{-5}$ | $5.00 \cdot 10^{-1}$ | $2.79 \cdot 10^{-11}$ | $5.00 \cdot 10^{-4}$ | $1.48 \cdot 10^{0}$ |
| $\sigma$(E.A.) Aasen | $2.04 \cdot 10^{-3}$ | $4.08 \cdot 10^{5}$ | $3.43 \cdot 10^{-5}$ | $4.25 \cdot 10^{+1}$ | $1.42 \cdot 10^{5}$ |
| $\sigma$(E.A.) Bunch | $3.88 \cdot 10^{-7}$ | $8.00 \cdot 10^{-3}$ | $3.52 \cdot 10^{-13}$ | $8.61 \cdot 10^{-6}$ | $4.67 \cdot 10^{-2}$ |
| mean(E.A.) Aasen | $1.26 \cdot 10^{-2}$ | $3.10 \cdot 10^{6}$ | $6.00 \cdot 10^{-4}$ | $1.43 \cdot 10^{3}$ | $3.55 \cdot 10^{6}$ |
| mean(E.A.) Bunch | $2.43 \cdot 10^{-6}$ | $9.12 \cdot 10^{-2}$ | $6.33 \cdot 10^{-12}$ | $2.52 \cdot 10^{-4}$ | $4.09 \cdot 10^{-1}$ |
| max(E.R.) Aasen | $1.07 \cdot 10^{0}$ | $6.79 \cdot 10^{2}$ | $1.86 \cdot 10^{0}$ | $1.00 \cdot 10^{0}$ | $2.25 \cdot 10^{0}$ |
| max(E.R.) Bunch | $2.62 \cdot 10^{-4}$ | $4.46 \cdot 10^{-7}$ | $1.47 \cdot 10^{-4}$ | $4.06 \cdot 10^{-7}$ | $8.95 \cdot 10^{-5}$ |
| $\sigma$(E.R.) Aasen | $8.25 \cdot 10^{-2}$ | $1.84 \cdot 10^{0}$ | $3.47 \cdot 10^{-2}$ | $2.96 \cdot 10^{-2}$ | $2.86 \cdot 10^{-2}$ |
| $\sigma$(E.R.) Bunch | $1.57 \cdot 10^{-5}$ | $6.72 \cdot 10^{-9}$ | $1.91 \cdot 10^{-7}$ | $6.01 \cdot 10^{-9}$ | $8.28 \cdot 10^{-8}$ |
| mean(E.R.) Aasen | $1.00 \cdot 10^{0}$ | $6.38 \cdot 10^{0}$ | $9.91 \cdot 10^{-1}$ | $9.99 \cdot 10^{-1}$ | $1.00 \cdot 10^{0}$ |
| mean(E.R.) Bunch | $1.88 \cdot 10^{-4}$ | $1.00 \cdot 10^{-7}$ | $6.62 \cdot 10^{-7}$ | $1.76 \cdot 10^{-7}$ | $2.82 \cdot 10^{-7}$ |

TABLE 2 – Erreurs relatives et absolues

Comme nous pouvons le constater, la méthode de Bunch-Parlett donne des erreurs relatives, par rapport à la solution donnée par MatLab, très faibles tandis que la méthode de Aasen ne nous donne jamais la bonne solution. Étant donné le fait que, en terme de stockage et de nombre d'opérations arithmétiques, les deux méthodes sont identiques, il convient dès lors de choisir la méthode de Bunch-Parlett pour résoudre des systèmes linéaires symétriques indéfinis.

# 4 Conclusion

Comme je l'ai dit dans l'introduction, les systèmes d'équations linéaires sont souvent rencontrés dans des problèmes mathématiques ou de physique. Il est donc important de connaitre des méthodes de résolutions de ce type de système. Il existe actuellement de nombreux algorithmes permettant d'en calculer la solution, donc les deux méthodes que je vous ai présenté, à savoir la méthode de Aasen et la méthode de Bunch-Parlett. Comme nous avons pu le constater dans les rappels théoriques, ces méthodes permettent de factoriser une matrice $A$ en un produit de 5 matrices, dont une matrice de permutation, sa transposée, une matrice triangulaire et sa tranposée. Ces méthodes de factorisation permettent de simplifier les calculs nécessaires à la résolution de système linéaire $Ax = b$.

Comme dit précédemment, pour comparer deux méthodes, il est important de se pencher sur trois facteurs. Le cout en mémoire, le nombre d'opérations arithmétiques nécessaires et la précision du résultat. Les deux premiers facteurs sont identiques dans les deux méthodes étudiées ici. Cependant, au niveau de l'erreur sur la solution, les erreurs relatives avec une solution "théorique" (calculée avec MatLab) sont beaucoup plus importante dans le cas de la méthode de Aasen que dans le cas de la méthode de Bunch-Parlett, qui nous donne des valeurs fort proches des solutions de MatLab, comme le montre mes résultats.

Du fait qu'il existe de très nombreuses méthodes de résolution de système linéaire, il est difficile d'en déclarer une comme étant la meilleure, chaque méthode ayant ses avantages et ses caractéristiques propres. Néanmoins, dans le cas où nous devrions résoudre un système d'équation linéaire symétrique, la méthode de Bunch-Parlett est préférable à la méthode de Aasen.

# Codes écrits pour ce travail

## 4.1 ALLINU_EXAM

Ce script MatLab reprend toutes les calculs que j'ai effectué pour ce rapport, depuis les calculs de normes et de nombre de conditionnement jusqu'à l'analyse des erreurs.
Il permet aussi de convertir automatiquement les fichiers .mtx en fichier .dat et de lancer mon script principal (obtenu à partir du programme C compilé). Ce script utilise la fonction mmread, s'utilisant comme suivante :
– $A = mmread(NAME)$, où NAME est le nom du fichier .mtx contenant la matrice $A$

```
function [] = ALLINU_EXAM()
%% INITIALISATION DES MATRICES A TRAITER
    MAT_NAME = {'lund_a','bfw398b','bcsstk19','1138_bus','bcsstm27'};
%% CREATION DU FICHIER D'ENTREE ET DES MATRICES EN .dat
    FCND = fopen('CONDITION.dat','w');
    FIPT = fopen('INPUT.dat','w');
    fprintf(FIPT,'%d\n',numel(MAT_NAME));
    for INDX = 1:numel(MAT_NAME)
        NAME = MAT_NAME{INDX};
        fprintf(FIPT,'%s\n',NAME);
        MTRX = mmread([NAME '.mtx']);
        [POSX,POSY,VALR] = find(MTRX);
        FILE = fopen([NAME '.dat'],'w');
        fprintf(FILE,'%d\t%d\t%d\n',size(MTRX), length(POSX));
        fprintf(FILE,'%d\t%d\t%e\n',[POSX,POSY,VALR]');
        fclose(FILE);
%% ECRITURE DES TAILLES, DES CONDITIONNEMENTS ET DES NORMES DANS UN FICHIER
        fprintf(FCND,'%d\t',size(MTRX,1));
        MTRX = full(MTRX);
        fprintf(FCND,'%e\t',cond(MTRX,1),cond(MTRX,2),cond(MTRX,inf),       ...
                            norm(MTRX,1),norm(MTRX,2),norm(MTRX,inf));
        fprintf(FCND,'\n');
    end
    fclose(FCND);
    fclose(FIPT);
%% LANCEMENT DU PROGRAMME PRINCIPAL
    system('./ALLINU_EXAM.out');
%% CREATION DU FICHIER CONTENANT LES ANALYSES DES ERREURS
    FSOL = fopen('COMPARE.dat','w');
    for INDX = 1:numel(MAT_NAME)
        NAME = MAT_NAME{INDX};
%% RECUPERATION DU VECTEUR b ET DE LA MATRICE A
        VCTR = load(['rhs_' NAME '.dat']);
        MTRX = mmread([NAME '.mtx']);
        SLTN = [load(['result_' NAME '.dat']) MTRX\VCTR];
        FILE = fopen(['result_' NAME '.dat'],'w');
%% AJOUT DE LA SOLUTION MATLAB DANS LE FICHIER DES RESULTATS
        for POSX = 1:size(VCTR)
            for POSY = 1:3
                if(SLTN(POSX,POSY)>0)
                    fprintf(FILE,'+');
                end
                fprintf(FILE,'%0.10f\t',SLTN(POSX,POSY));
            end
            fprintf(FILE,'\n');
        end
        fclose(FILE);
%% CALCUL DES ERREURS ABSOLUES ET RELATIVES
        ERRA = (SLTN(:,1:2) - [SLTN(:,3) SLTN(:,3)]);
        ERRA = abs(ERRA(SLTN(:,3)~=0,:));
        ERRR = (SLTN(:,1:2) - [SLTN(:,3) SLTN(:,3)])./[SLTN(:,3) SLTN(:,3)];
        ERRR = abs(ERRR(SLTN(:,3)~=0,:));
%% ECRITURE DANS LE FICHIER DES ERREURS ABSOLUES ET RELATIVES
        fprintf(FSOL,'%e\t',max(ERRA), norm(ERRA(:,1))/size(MTRX,1),       ...
```

```
                            norm(ERRA(:,2))/size(MTRX,1), mean(ERRA),       ...
                            max(ERRR), norm(ERRR(:,1))/size(MTRX,1),        ...
                            norm(ERRR(:,2))/size(MTRX,1), mean(ERRR));
        fprintf(FSOL,'\n');
    end
    fclose(FSOL);                                                    % FERMETURE DU FICHIER
end
```

## 4.2 Programme principal (ALLINU_EXAM.c)

Programme principal. Ce programme va lire dans un fichier INPUT.dat les noms des matrices à traiter et va ensuite résoudre, pour chaque matrice, un système de la forme $Ax = b$, par les méthodes de Aasen et de Bunch-Parlet.

```
#include "ALLINU_EXAM.h"

int main(){
  double *A, *A2, *b, *b2, *L, *T, *P, *D, *SD1, *SD2, *PT, *WORK;
  int SIZE, m, POSX, POSY, NUML, NMBR, i, j, NRHS=1, INFO, *IPIV, NUMM;
  char NAME[20], FNAM[30], UPLO='L', TRANS='N', DIAG='U';
  FILE *FILI,*FILI2,*FILO, *FIL3;

    FIL3 = fopen("INPUT.dat","r");
// LECTURE DU NOMBRE DE MATRICE A TRAITER
    fscanf(FIL3,"%d",&NMBR);
    for(NUMM = 0; NUMM<NMBR;NUMM++){
// LECTURE DU NOM DE LA MATRICE ACTUELLE ET OUVERTURE DES FICHIERS
        fscanf(FIL3,"%s",NAME);
        FILI = fopen(NAME,"r");
        if(FILI==NULL){
            printf("Fichier %s introuvable !!!\n",NAME);
            continue;
        }
        sprintf(HEAD,"rhs_%s.dat",NAME);
        FILI2 = fopen(FNAM,"r");
        sprintf(HEAD,"result_%s.dat",NAME);
        FILO = fopen(FNAM,"w+");

// ALLOCATION DES DIFFERENTS TABLEAUX
        fscanf(FILI,"%d %d %d",&SIZE, &m, &NUML);
        A    = (double *) calloc(SIZE*SIZE,sizeof(double));
        A2   = (double *) calloc(SIZE*SIZE,sizeof(double));
        b    = (double *) calloc(SIZE,sizeof(double));
        b2   = (double *) calloc(SIZE,sizeof(double));
        P    = (double *) calloc(SIZE*SIZE,sizeof(double));
        PT   = (double *) calloc(SIZE*SIZE,sizeof(double));
        L    = (double *) calloc(SIZE*SIZE,sizeof(double));
        T    = (double *) calloc(SIZE*SIZE,sizeof(double));
        D    = (double *) calloc(SIZE,sizeof(double));
        SD1  = (double *) calloc(SIZE-1,sizeof(double));
        SD2  = (double *) calloc(SIZE-1,sizeof(double));
        WORK = (double *) calloc(SIZE,sizeof(double));
        IPIV = (int*)     calloc(SIZE,sizeof(int));

// LECTURE DES VALEURS DE A
        for(i=0;i<NUML;i++){
            fscanf(FILI,"%d %d",&POSX, &POSY);
            fscanf(FILI,"%lf",A+POSX-1+SIZE*(POSY-1));
            A2[(POSX-1)+SIZE*(POSY-1)] = A[(POSX-1)+SIZE*(POSY-1)];
        }

// LECTURE DES VALEURS DE b
        for(i=0;i<SIZE;i++){
            fscanf(FILI2,"%lf",b+i);
            b2[i] = b[i];
        }

// METHODE DE AASEN
```

```
55 //   CALCUL DES MATRICES T,L ET P
56         aasen(SIZE,A,T,L,P);
57
58 //   RESOLUTION DES AUTRES EQUATIONS
59         b = gaxPD(P,SIZE,SIZE,b);
60         for(i=0;i<SIZE;i++) D[i] = T[i+SIZE*i];
61         for(i=0;i<SIZE-1;i++) SD1[i] = T[i+SIZE*(i+1)], SD2[i] = T[i+1+SIZE*i];
62         for(i=0;i<SIZE;i++) for(j=0;j<SIZE;j++) PT[i+SIZE*j]=P[j+SIZE*i];
63
64         dtrtrs(&UPLO, &TRANS, &DIAG, &SIZE, &NRHS, L, &SIZE, b, &SIZE, &INFO);
65         dgtsv(&SIZE, &NRHS, SD1, D, SD2, b, &SIZE, &INFO);
66         TRANS = 'T';
67         dtrtrs(&UPLO,&TRANS,&DIAG,&SIZE,&NRHS,L,&SIZE,b,&SIZE,&INFO);
68
69 //   CALCUL DE LA SOLUTION FINALE
70         b = gaxPD(PT,SIZE,SIZE,b);
71
72 // METHODE DE BUNCH-PARLETT
73         dsytrf_(&UPLO, &SIZE, A2, &SIZE, IPIV, WORK, &SIZE, &INFO);
74         dsytrs_(&UPLO, &SIZE, &NRHS, A2, &SIZE, IPIV, b2, &SIZE, &INFO);
75
76 // ECRITURE DANS LE FICHIER DE SORTIE
77         for(i=0;i<SIZE;i++) fprintf(FILO,"%lf\t%lf\SIZE",b[i],b2[i]);
78
79 // LIBERATION DE LA MEMOIRE ET FERMETURE DES FICHIERS
80         free(A);free(b);free(A2);free(b2);free(P);free(PT);free(L);free(T);free(D);free(SD1);
81         free(SD2);
82         fclose(FILI); fclose(FILI2); fclose(FILO);
83     }
84     return(0);
85 }
```

## 4.3 Fonction gaxPD

Cette fonction permet de calculer le produit entre une matrice et un vecteur.

```
1  double* gaxPD(double* A, int m, int n, double* b){
2  /*
3    Fonction calculant le produit entre une matrice et un vecteur
4    INPUT  : A : Tableau unidimensionnel contenant les Ã©lÃ©ments de la matrice A
5             n : Nombre de ligne de la matrice
6             m : Nombre de colonne de la matrice
7             b : Tableau unidimensionnel contenant les Ã©lements du vecteur b
8    OUTPUT : c : Tableau unidimensionnel contenant les Ã©lÃ©ments du vecteur A*b
9  */
10   int i, j;
11   double *c = (double*) calloc(m,sizeof(double));
12     for(i=0;i<m;i++) for(j=0;j<n;j++) c[i]+=A[i+m*j]*b[j];
13     return c;
14 }
```

# Autres codes utilisés

## 4.4 Fonction aasen

```
function aasen(int n, double* A, double* T, double* L, double* P)
/*
Auteur : Charlotte Tannier
Date : Novembre 2015
But : Fonction qui permet de calculer la factorisation LTL^T
      de la matrice A (nxn). Implémentation de l'algorithme décrit dans
            le livre "Matrix computation" de Golub et Van Loan (2006, Section 4.4)
Input :  l'ordre n de la matrice A
         le tableau inidimensionnel A contenant les éléments de la matrice A
                les tableau unidimensionnels T, L et P
Output : le tableau unidimensionnel L contenant les élements de la matrice L
         le tableau unidimensionnel T contenant les éléments de la matrice T
                le tableau unidimensionnel P contenant les éléments de la matrice P
*/
```

Cette fonction se base sur l'algorithme suivant pour calculer la factorisation d'Aasen :

**Algorithm 3** Algorithme de Aasen

**function** AASEN(A)
  $L \leftarrow I_n$
  **for** $j = 1 : n$ **do**
    **if** $j = 1$ **then**
      $\alpha_1 \leftarrow a_{11}$
      $v(2:n) \leftarrow A(2:n, 1)$
    **else**
      $h_1 \leftarrow \beta_1 l_{j2}$
      **for** $k = 2 : j-1$ **do**
        $h_k \leftarrow \beta_{k-1} l_{j,k-1} + \alpha_k l_{jk} + \beta_k l_{j,k+1}$
      $h_j \leftarrow a_{jj} - L(j, 1:j-1)h(1:j-1)$
      $\alpha_j \leftarrow h_j - \beta_{j-1} l_{j,j-1}$
      $v(j+1:n) \leftarrow A(j+1:n, j) - L(j+1:n), 1:j)h(1:j)$
      **if** $j \leq n-1$ **then**
        $\beta_j \leftarrow v(j+1)$
        **if** $j \leq n-2$ **then**
          $L(j+2:n, j+1) \leftarrow v(j+2:n)/v(j+1)$
  **return** $L$,$T$

## 4.5 Sous-routine dtrtrs

Cette sous-routine provient de la librairie LAPack et permet de résoudre un système de la forme $Ax = B$ où $A$ est une matrice triangulaire. Des informations complémentaires peuvent être trouvées à cette adresse :

`http://www.netlib.org/lapack/explore-html/d6/d6f/dtrtrs_8f.html`

## 4.6 Sous-routine dgtsv

Cette sous-routine provient de la librairie LAPack et permet de résoudre un système de la forme $Ax = B$ où $A$ est une matrice tridiagonale. Des informations complémentaires peuvent être trouvées à cette adresse :

`http://www.netlib.org/lapack/explore-html/d1/db3/dgtsv_8f.html`

## 4.7 Sous-routine dsytrf

Cette sous-routine provient de la librairie LAPack et permet de calculer la factorisation de Bunch-Parlett d'une matrice symétrique. Des informations complémentaires peuvent être trouvées à cette adresse :

`http://www.netlib.org/lapack/explore-html/dd/df4/dsytrf_8f.html`

## 4.8 Sous-routine dsytrf

Cette sous-routine provient de la librairie LAPack et permet de résoudre un système de la forme $Ax = b$ à partir de la factorisation de Bunch-Parlet de la matrice $A$. Des informations complémentaires peuvent être trouvées à cette adresse :

`http://www.netlib.org/lapack/explore-html-3.3.1/d0/d9a/dsytrs_8f.html`