© *J.-B.A.K. <jean-baptiste.apoung@math.u-psud.fr>*

La méthode de réduction cyclique pour systèmes tridiagonaux : récursion et liste doublement chaînée en langage C

L'objectif de ce projet est de proposer et d'évaluer en langage de **programmation C** *une implémentation de la méthode de* **réduction cyclique**. *Il portera sur la résolution des* **systèmes linéaires tridiagonaux**, *issus de la discrétisation par* **différences finies** *des équations aux dérivées partielles (edp) de type* **convection-diffusion-réaction**. *Sous une hypothèse non restrictive que la taille du système est de la forme* $n = 2^p - 1$, *on évaluera la possibilité de d'utiliser une structure de données combinant* **arbres** *et* **liste double chaînée** *pour représenter le système tridiagonal pour mettre en oeuvre les phases de réduction (ou descente) et de résolution (ou remontée) de la méthode de réduction cyclique.*
Unités d'enseignement requises : Math 312, Math 315 et Math 325.

Partie - 1 *Motivations et besoins*

Le calcul scientifique a entre autres objectifs, la résolution efficace et rapide des équations aux dérivées partielles. Dans bon nombres de ces équations, la possibilité d'exhiber une solution analytique est hors d'atteinte. On est alors contraint du point de vue théorique à simplement s'assurer de l'existence et de l'unicité de la solution de ces équations. Le calcul scientifique vient alors exhiber une solution représentable par ordinateur. Mais avant d'y parvenir et de proposer ce qu'il y a de mieux, il a besoin de deux informations supplémentaires de l'étude théorique de ces équations : la dépendance continue de la solution vis-à-vis des données du problème (appelée aussi stabilité) et la régularité de la solution. Cette dernière étant déterminante dans le choix de la méthode d'approximation.

Lorsque ces ingrédients sont fournis, le calcul scientifique peut alors s'engager à proposer une **approximation consistante** de la solution de l'équation aux dérivées partielles. Elle suit alors les trois étapes suivantes :

1. La **discrétisation** de l'équation aux dérivées partielles. Il commence par identifier les points dans le domaine de résolution où il est capable de fournir une approximation (ponctuelle ou en moyenne suivant la régularité de la solution) de la solution de l'edp. Ces points peuvent être imposés (capteurs météo etc.) ou choisis si l'on dispose des informations plus précises sur le comportement de la solution (existence d'extéma locaux etc.). Ces points sont ensuite connectés entre-eux pour définir un réseau de transfert d'information d'un point à l'autre (c'est le maillage). Suivant la forme du domaine et la régularité de la solution, le maillage peut conditionner le choix d'une méthode. Ainsi lorsque le domaine est cartésien, le maillage cartésien et la solution présentant une régularité permettant sont évaluation ponctuelle, l'une des méthodes de choix de par sa simplicité de mise en oeuvre est la méthode des **différences finies.** Le caractère cartésien de la grille facilite alors l'approximation des dérivées partielles au moyen des **différences divisées**. La méthode des différences finies peut souffrir si l'edp présente des dérivés partielles d'ordre élevé. Mais là encore il existe des techniques pour contourner cette difficulté. Ainsi cette méthode, bien que supplantée par d'autres dans certains contextes est le premier choix dans des configurations favorables (problèmes posés en une dimension d'espace). Dans le domaine du calcul scientifique, elle a encore des années devant elle. *Dans le présent projet l'edp que nous considérons est de type convection-diffusion-réaction en une dimension d'espace, dont nous admettrons l'existence l'unicité et la stabilité de la solution ainsi que la bonne régularité de la solution, et nous opterons pour une méthode de différences finies.*
2. **La résolution du problème matriciel :** Il est question de déterminer effectivement les valeurs des inconnus introduites par la discrétisation. Ce seront ces valeurs approchées de la solution aux les points placés dans le domaine (points de discrétisation). Cette étape conduit la plupart du temps à la résolution des systèmes non linéaires de la forme $F(U) = 0$ ou à des systèmes linéaires de la forme $AU = f$. La résolution efficace et rapide de ces systèmes est donc un enjeu important pour le calcul scientifique. (*Le présent projet se focalise sur ce point et considère un système linéaire dont la matrice a une structure particulière dite tridiagonale.*)
3. **La convergence :** Il est ici question de justifier théoriquement que la solution calculée approche bien celle du problème continu de départ. (*Nous n'aborderons pas cette partie dans le présent projet, nous comparerons néanmoins des résultats obtenus à la solution exacte dans les cas simples où celle-ci existe*).

Partie - 1-1 *Origine du système linéaire tridiagonale*

Un système linéaire tridiagonale a la forme suivante :

$$\begin{pmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & & 0 & a_n & b_n \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix} \tag{1}$$

où seules les trois diagonales principales de la matrices contiennent des valeurs non nulles.
Ces systèmes s'obtiennent généralement par discrétisation des équations aux dérivées partielles posées en une dimension d'espace. en particulier si la méthode employée est du type différences finies et la grille considérée a les les sommets (ou points) numérotés de manière lexicographique (i.e. selon l'ordre croissante des abscisses).

Donnons en un exemple qui nous servira de problème modèle.

On considère l'équation de convection diffusion suivante :

$$\begin{cases} -\kappa(x)u''(x) + \beta(x)u'(x) + r(x)u(x) &= f(x) \text{ dans }]x_L, x_R[\\ u(x_L) &= g_L \\ u(x_R) &= g_R \end{cases} \tag{2}$$

où sans nuire à la généralité,
- $x_L < x_R$ sont deux réels donnés,
- κ est de classe C^1 sur $[x_L, x_R]$ avec $\exists \kappa_0, \kappa_1$ tels que $0 < \kappa 0 \leq \kappa(x) \leq \kappa_1$,
- $\beta(x) etr(x)$ sont de classe C^1 dur $[x_L, x_R]$ et positives,
- g_L et g_R sont des réels donnés et f est continue dans $[x_L, x_R]$.

On peut montrer par des techniques variationnelles que la solution de cette équation existe, est unique, et dépend continue des données f, g_L, g_R. Pour ce qui est de la régularité on peut montrer que la solution est au moins dans $C([x_L, x_R])$ ce qui autorise une méthode de différences finies pour son approximation.

Q-1-1 **:** Montrer que la discrétisation par différences finies du problème (2) conduit à un système linéaire tridiagonale de la forme (1)

On pourra remarquer qu'en deux dimensions, pour une grille dont les sommets sont numérotés de manière lexicographique, les blocs diagonaux obtenus par discrétisation par différences finies sont aussi tridiagonaux. Ceux-ci peuvent servir de *préconditionnement* si leur inversion efficace est possible. Dans certains problèmes où il n'est pas possible de construire explicitement la matrice faute d'espace mémoire, un préconditionnement est parfois cherché sous forme d'approximation de la partie tridiagonale de la matrice u système. C'est le cas par exemple des problèmes associés au *complément de Schur* (voire méthode d'Uzawa pour l'équation de Stokes, méthode de sous-structuration en décomposition de domaines etc.).

Les systèmes tridiagonaux peuvent aussi apparaître dans la résolution des équations aux dérivées partielles non stationnaires. Considérons en effet l'équation aux dérivées partielles suivante :

$$\begin{cases} \dfrac{\partial u(t,x)}{\partial t} - \kappa(t,x)\dfrac{\partial^2 u(t,x)}{\partial x^2}(x) + \beta(t,x)\dfrac{\partial u(t,x)}{\partial x} + r(t,x)u(x) &= f(t,x) \text{ dans }]0,T[\times]x_L, x_R[\\ u(t,x_L) &= g_L(t) \quad \text{sur } [0,T] \\ u(t,x_R) &= g_R(t) \quad \text{sur } [0,T] \\ u(0,x) &= u_0(x) \quad \text{sur } [x_L, x_R] \end{cases} \tag{3}$$

où sans nuire à la généralité, $\kappa, \beta, r, f, g_L, g_R, u_0$ sont des fonctions suffisamment régulières (disons C^2) de leurs arguments, avec l'existence des réels α, β tels que $0 < \kappa_1 \leq \kappa(t,x) \leq \kappa_2$ et une relation de compatibilité $g_L(0) = u_0(x_L), g_R(0) = u_0(x_R)$.

Q-1-2 **:** En effectuant une semi-discrétisation en temps par un schéma implicite, montrer qu'à chaque pas de temps on est amené à résoudre un problème du type (2)

Partie - 1-2 *Résolution du système tridiagonal*

Bien que nous nous proposons dans ce projet de résoudre un système tridiagonal par une méthode de réduction cyclique, mentionnons néanmoins que pour les problèmes modèles considérés ici, il existe des méthodes de résolution assez efficaces. L'algorithme de Thomas en est une illustration. Celui-ci est basé sur une factorisation *LU* de la matrice et sa mise en oeuvre ne présente pas de difficulté majeure.
La méthode de réduction cyclique que nous verrons dans la suite ne se distingue des autres méthodes que par son efficacité en calcul parallèle. Par ailleurs son extension à des problèmes multidimensionnels est possible moyennant un traitement par bloc. La théorie dans ce cas montre que lorsque les coefficients sont constants, les différents blocs de matrices intervenant à chaque stade ont une formule explicite.

Nous n'aborderons pas tout cela dans ce projet, qui se focalise simplement sur l'exploration d'une structure de données pour mettre en oeuvre cet algorithme. Néanmoins afin de nous convaincre de la bonne résolution des systèmes tridiagonaux, il est demandé de

Q-1-3 **:** Programmer l'algorithme de Thomas pour la résolution du système linéaire tridiagonal (1). On supposera la matrice tridiagonale stockée par trois vecteurs a, b, c représentant respectivement ses diagonales inférieure, principale et supérieure.

Partie - 2 *Méthode de réduction cyclique*

Le principe de la méthode de réduction cyclique pour un système tridiagonal de taille $n = 2^p - 1$ consiste en une procédure de deux phase :

- **Phase de réduction ou descente :**(*voir illustration ci-dessous*)
  On commence par éliminer les inconnues d'indice impair dans le système d'équation. L'élimination est telle que le système résultant est encore tridiagonal. On répète alors le procédé jusqu'à ce qu'il ne reste plus qu'un système avec une seule équation à une inconnue. *Les éliminations peuvent se faire soit par renumérotation et résolution par bloc, soit par combinaison linéaire, à coefficients bien choisis, des équations prises par paquets de 3.*
- **Phase de résolution ou remontée :**(*voir illustration ci-dessous*)
  On résout le dernier système obtenu et par une procédure de remontée (substitution), on résout de proche en proche, dans l'ordre inverse de leur obtention, les différents systèmes tridiagonaux.

Pour illustrer le principe, considérons les cas où $p = 3$, c'est-à-dire que le système (1) est de taille $n = 7$. Il s'écrit alors

$$\left\{\begin{array}{ccccccccccccccc}
b_1u_1 & + & c_1u_2 & & & & & & & & & & & = & f_1 \\
a_2u_1 & + & b_2u_2 & + & c_2u_3 & & & & & & & & & = & f_2 \\
 & & a_3u_2 & + & b_3u_3 & + & c_3u_4 & & & & & & & = & f_3 \\
 & & & & a_4u_3 & + & b_4u_4 & + & c_4u_5 & & & & & = & f_4 \\
 & & & & & & a_5u_4 & + & b_5u_5 & + & c_5u_6 & & & = & f_5 \\
 & & & & & & & & a_6u_5 & + & b_6u_6 & + & c_6u_7 & = & f_6 \\
 & & & & & & & & & & a_7u_6 & + & b_7u_7 & = & f_7
\end{array}\right. \tag{4}$$

Phase de réduction

Q-2-1 **:** Elimination de l'inconnue u_1, u_3 : On considère les 3 première équations que l'on désigne par le paquet $(1, 2, 3)$. On multiple la première équation par α et la troisième par γ et on les ajoute à la deuxième équation. Montrer que l'on peut choisir convenablement les réels α et γ pour que le système résultant soit de la forme

$$b'_1u_2 + c'_1u_4 = f'_1 \tag{5}$$

où b'_1, c'_1 et f'_1 sont des constantes que l'on peut encore stocker respectivement dans les variables b_2, c_2 et f_2.

Q-2-2 **:** En appliquant la procédure précédente sur les paquets d'équations $(3, 4, 5), (5, 6, 7)$, en déduire que l'on obtient le nouveau système tridiagonal

$$\left\{\begin{array}{ccccccc}
b'_1u_2 & + & c'_1u_4 & & & = & f'_1 \\
a'_2u_2 & + & b'_2u_4 & + & c'_2u_6 & = & f'_2 \\
 & & a'_3u_4 & + & b'_3u_6 & = & f'_3
\end{array}\right. \tag{6}$$

où a'_2, b'_2, c'_2, f'_2 sont des constantes que l'on peut encore stocker dans a_4, b_4, c_4, f_4 et $a'_3, b3', f'_3$ sont des constantes que l'on peut encore stocker dans a_6, b_6, f_6.

Q-2-3 : Appliquer la même démarche au système (6) et en déduire que l'on obtient un système sous la forme :

$$\left\{ \begin{array}{rcl} b''_1 u_4 & = & f''_1 \end{array} \right. \tag{7}$$

où b''_1, f''_1 sont des constantes que l'on peut encore stocker dans b_4, f_4.

Phase de résolution ou Remontée

Q-2-4 : Montrer que la résolution dans l'ordre des systèmes (7) puis (6) et enfin (4) est possible et fournit la solution $u_1, u_2, u_3, u_4, u_5, u_6, u_7$ du système de départ (4).

Algorithme sous forme d'un programme en langage C

L'illustration précédente peut se généraliser de manière directe au cas $n = 2^p - 1$ avec p quelconque. Il en résulte un algorithme itératif que nous présentons ici sous forme d'un script en langage C. Et pour simplifier sa compréhension, nous considérons un stockage plein de la matrice tridiagonale A. Le Listing 1 représente les structures de données et le programme principal. Pour une telle structure de données, l'algorithme de réduction est donné dans le Listing 2 et celui de la résolution est fourni dans le Listing3.

Listing 1 – Reduction cyclique pour un stockage plein de la matrice

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h >
typedef struct Problem{
 int n;       //< de taille 2^p -1
 double** A; //< matrice tridiagonale stockage non optimal
 double* F;  //< second membre du systeme
 double* mesh; // maillage du domaine
}Problem, *pProblem;

void pb_alouer(pProblem pb, int n);
void pb_construire(pProblem pb);
void pb_liberer(void* pb);
void cyclic_reduire(pProblem p);
void cyclic_remonter(pProblem p);

int main(int argc, char** argv){
  int n = pow(2,7) - 1;
  Problem pb={0,0,0};
  pb_construire(&pb, n);
  cyclic_reduire(&pb);
  cyclic_resoudre(&pb);
  cyclic_ecrire_gnuplot(&pb,"sol.dat");
  pb_liberer(&pb);
  return 0;
}
```

Listing 2 – Phase de réduction pour un stockage plein de la matrice

```c
void cyclic_reduire(pProblem p){

  int size  = p->n;
  int i,j, index1,index2, decallage;
  double alpha, gamma;

  for(i = 0; i < log2( size + 1) -1 ; i++) {
  for(j = pow(2,i+1) -1; j < size;  j = j + pow(2,i+1)){

   decallage = pow(2,i);
   index1 = j - decallage;
   index2 = j + decallage;
   alpha  = A[j][index1]/A[index1][index1];
   gamma  = A[j][index2]/A[index2][index2];

   for(k = 0; k < size; k++){
    A[j][k] -= alpha * pb->A[index1][k] + gamma * pb->A[index2][k];
   }
   F[j] -= alpha * pb->F[index1] + gamma * pb->F[index2];
  }
  }
}
```

Listing 3 – Phase de remontée pour un stockage plein de la matrice

```
void cyclic_remonter(pProblem p){
  int size  = p->n;
  int i, j,
      index, index1,index2, decallage;

  index = (size -1)/2;
  pb->x[index] = pb->F[index]/pb->A[index][index];

  for(i = log2( size + 1) -2; i >= 0 ; i--) {
  for(j = pow(2,i+1) -1; j < size;  j = j + pow(2,i+1)){

   decallage = pow(2,i);
   index1 = j - decallage;
   index2 = j + decallage;

   pb->x[index1] = pb->F[index1];
   pb->x[index2] = pb->F[index2];

   for(k = 0; k < size; k++){
     if(k != index1)
       pb->x[index1] -= A[index1][k] * pb->x[k];
     if(k != index2)
       pb->x[index2] -= A[index2][k] * pb->x[k];
   }
   pb->x[index1] = pb->x[index1]/pb->A[index1][index1];
   pb->x[index2] = pb->x[index2]/pb->A[index2][index2];
  }
  }
}
```

Q-2-5 : Compléter le script ci-dessus puis tester l'algorithme de réduction cyclique sur un problème modèle.

Q-2-6 : Modifier les scripts proposés afin de traiter le cas où la matrice tridiagonale est stockée de manière optimale à l'aide de trois vecteurs a, b, c.

Q-2-7 : Comparer l'algorithme proposé avec l'algorithme de Thomas. On pourra évaluer les nombres d'opérations et quantifier les temps d'exécution.

Partie - 3 *Une structure de données pour la méthode de réduction cyclique*

On rappelle encore une fois que cette méthode (de réduction cyclique) ne démontre son efficacité véritable que dans une programmation parallèle. Néanmoins dans le soucis de mettre en pratique les enseignements reçus, notamment dans le cours de Math 312 "algorithmique et programmation en C", nous envisageons ici, d'exploiter la récursion qui semble découler de manière naturelle de la méthode. Pour cela nous allons proposer une structure de données combinant les structures de données **arbre** et **liste doublement chaînée** afin de :

1. stocker la matrice tridiagonale et plus généralement le système linéaire,
2. mettre en oeuvre les deux phases de la méthode de réduction cyclique.

A notre connaissance une telle structure est novatrice et n'a pour l'instant pas encore été explorée.

Partie - 3-1 *Proposition de représentation du système tridiagonal dans une algorithme de réduction cyclique*

On fait de choix de créer une structure de données décrite dans le Listing12. La Figure1 illustre cette structure de données dans le cas de l'exemple ci-dessus où $n = 2^3 - 1$.. Avec cette structure de donnée en arbre, le système linéaire tout entier est identifié par la racine, que nous désignerons dans toute la suite par le mot **root** en anglais.

Listing 4 – une représentation de la matrice tridiagonale dans la réduction cycique

```
typedef struct tdiag
```

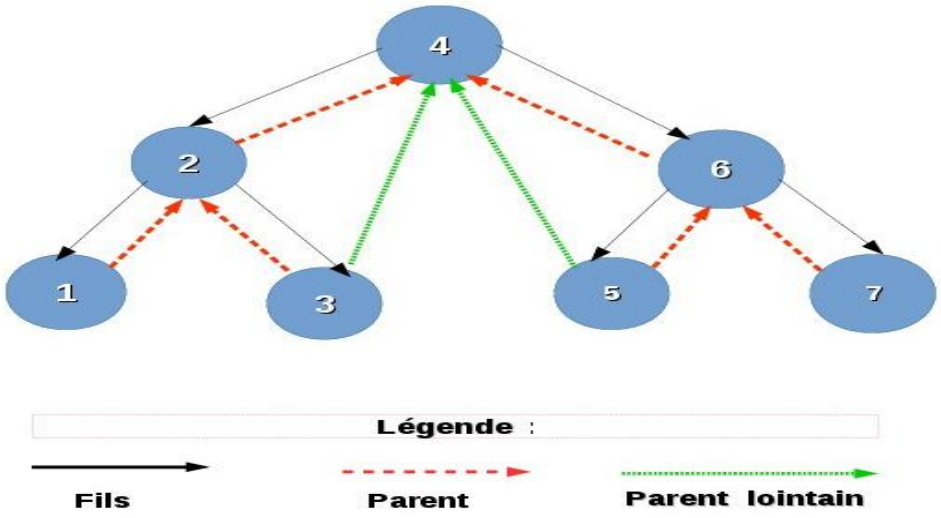

FIGURE 1 – Illustration lorsque $n = 2^3 - 1$ de la structure de données proposée pour la représentation du système tridiagonal dans l'algorithme de réduction cyclique. Chaque noeud (rond) représente une ligne du système linéaire, et porte le numéro de la ligne. Les flèches indiquent les relations de chaînage. L'information sur le parent lointain n'est pas stockée et peut être obtenue sans difficulté.

```
{
  int level;            /*!< niveau dans la hierarchie cyclique                          */
  int rank;             /*!< ne numero de l'equation  (ligne i du systeme de depart)     */
  double a, b, c;       /*!< les coefficients de la ligne;                               */
  double f;             /*!< le second membre de l'equation pour la ligne courante       */
  double x;             /*!< la composante de la solution                                */

  struct tdiag *father; /*!< geniteur dans la hierarchie  utile dans la remontee         */
  struct tdiag *lson;   /*!< le fils gauche dans la hierarchie                            */
  struct tdiag *rson;   /*!< le fils droit dans la hierarchie                             */

  int n;                /*!< nombre total d'equations (variable significative seulement dans
                           la racine) n'est pas forcement sous la forme 2^p -1 !!!      */
} tdiag, *ptdiag;
```

Petit commentaire

- les lignes de numéro 1,3,5,7 sont les feuilles et sont au niveau 0,
- les lignes de numéro 2, 6, sont de au niveau 1. La ligne 2 par exemple a pour fils les lignes 1 et 3 (ces lignes vont s'éliminer à l'étape 0 de la réduction (ou niveau 0) pour propulser la ligne 2 au prochain niveau),
- la ligne 4 est au niveau 2 et a pour fils les lignes 2 et 6.
- la ligne 3 a pour parent (immédiat) la ligne 2. Elle aura besoin de la ligne 2 pour calculer sa solution lors de la remontée. Elle aura aussi besoin de la ligne 4 qu'on aurait pu aussi considérer comme son parent. Mais comme nous avons limité la relation de parent au niveau supérieur immédiat, la ligne 4 apparaît ainsi comme parent éloigné (ou lointain) de la ligne 3. De même la ligne 5 a contribuer à propulser les lignes 4 et 6 au niveau supérieur, mais la ligne 6 est sont parent (immédiat) et la ligne 4 est sont parent éloigné. La ligne 5 aura néanmoins besoin de la solution des lignes 4 et 6 pour calculer sa solution dans la phase de la remontée.

Implémentation et validation de la structure de données

On décrit a présent les fonctions à fournir pour rendre vivante cette structure de données. Pour cela fournir :

- Une fonction permettant de créer et de détruire une structure de type **`tdiag`**

Listing 5 – Gestion de cycle de vie

```
ptdiag tdiag_create ();     /*!< allocation de memoire pour une structure tdiag */
void tdiag_free (void *);   /*!< liberation de memoire pour une structure tdiag */
```

- Une fonction (la plus importante) permettant de générer toute l'arbre si appelée sur la racine.

Listing 6 – Fonction de génération des fils

```
/*! Pour un neoud de niveau donne genere ses fils sur toutes les generation*/
void tdiag_ajout_fils(ptdiag root);
```

Il suffira par exemple pour générer l'arbre de l'exemple ci-dessus, de créer un noeud racine **`root`**, de lui affecter un niveau ici 2 (on suppose comme indiquer que les feuille de constituent le niveau 0), et d'appeler cette fonction.
Pour charger le système tridiagonale fournir les fonctions suivantes :

Listing 7 – Fonctions permettant de charger le système tridiagonal

```
/* Ici root represente le systeme lineaire. */
/*! Charge la diagonale principale du systeme lineaire */
void tdiag_charge_diagonal (ptdiag root, const double *b, int n);
/*! Charge la diagonale superieure du  systeme lineaire */
void tdiag_charge_sur_diagonal (ptdiag root, const double *c, int n);
/*! Charge la diagonale inferieure du  systeme lineaire */
void tdiag_charge_sous_diagonal (ptdiag root, const double *a, int n);
/*! Charge le second membre du systeme lineaire */
void tdiag_setRhs (ptdiag root, const double *f, int n);
```

Pour valider la mise en oeuvre, fournir les fonctions ci-dessous qui programment le produit matrice vecteur pour une matrice tridiagonale. Le vecteur d'entrée est de taille n ce paramètre n'est important que lorsque la taille du système de départ n'est pas sous la forme $n = 2^p - 1$ (Voir partie 3 la gestion de ce cas.)

Listing 8 – Impémentation du produit matrice vecteur

```
/*! Ici root est la racine du systeme lineaire de matrice A. */
void tdiag_gaxpy (ptdiag root, const double *x, double *y, int n); /*!< y = a*x + y */
void tdiag_gaxy (ptdiag root, const double *x, double *y, int n); /*!< y = a *x  */
```

Partie - 3-2 *Implémentation des deux phases de réduction cyclique*

Phase de la réduction

Pour l'implémentation de la phase de réduction fournir une fonction

Listing 9 – Impémentation de la phase de réduction

```
void tdiag_reduction (ptdiag root); /*!<  reduction */
```

Remarque sur la mise en oeuvre de la réduction ou descente.
On observera, au moyen de l'exemple proposé, que dans le soucis d'être optimal en utilisation mémoire, la structure de données ne stocke qu'une seule fois chaque équation du système, malgré le fait que celle-ci apparaisse à plusieurs niveaux de la procédure. Par exemple l'équation de numéro 4 apparaît au niveau 0 (i.e. dans le triplet $(3, 4, 5)$ *) et au niveau 1 dans le triplet* $(2, 4, 6)$*. Pourtant elle est stockée comme appartenant au niveau 2. Il faudra donc être attentif dans la mise oeuvre, car malgré le fait qu'une équation soit stockée à un niveau* l*, elle intervient à tous les niveaux inférieurs à* l*. Nous conseillons donc de fournir une fonction de mise à jour du système dans un niveau donné de la réduction. De sorte qu'en supposant cette fonction de prototype* **`tdiag_reduction_de_niveau(int niveau, ptdiag root)`**, *l'algorithme de réduction puisse s'écrire :*

Listing 10 – Impémentation de la phase de réduction

```
void tdiag_reduction (ptdiag root)
{
  int level = root->level;
  for (int i = 0; i < level; ++i)   /*  Compiler avec l'option -std=c99  */
     tdiag_reduction_de_niveau(i, root);
}
```

Phase de la remontée

Q-3-1 : Pour la phase de remontée dans la méthode de réduction cyclique, fournir une fonction

Listing 11 – Impémentation de la phase de remontée

```
void tdiag_remontee (ptdiag root); /*!<  remontee */
```

Partie - 3-3 *Post-traitement*

A fin de représenter graphiquement la solution obtenue et de calculer les erreurs commises dans l'approximation, il semble judicieux de fournir la solution calculée dans un vecteur. Pour cela fournir une fonction

Listing 12 – Résupération de la solution

```
void tdiag_fournir_solution (ptdiag root, double *x, int n);
```

Partie - 3-4 *Validations*

Valider votre mise en oeuvre sur un problème du type (2) où l'on choisira convenablement les données pour que la solution analytique soit connue. On pourra comment par prendre les coefficients constants (par exemple $K \equiv 1, \beta =\equiv 0, r \equiv 1, x_0 = 0, x_L = 1$.

Q-3-2 : Evaluer votre implémentation sur ce problème modèle

Q-3-3 : Comparer les temps d'exécution avec la solution obtenue pas l'algorithme de Thomas.

Partie - 4 *Extension au cas où n n'est pas sous la forme $2^p - 1$.*

Si à ce stade du projet vous disposez encore d'un peu de temps. Essayer d'étendre ce qui précède au cas n quelconque.

Pour cela :

1. chercher le plus petit entier p tel que $n \leq 2^p - 1$,
2. ajouter au système tri-diagonal les équations triviales :

$$u_i = 0, \quad i = n+1, \ldots, 2^p - 1, \tag{8}$$

3. appliquer la méthode au système obtenu.

Q-4-1 : Modifier votre code pour prendre en compte le cas n quelconque.
les modifications sont minimes, on fera bon usage du paramètre n qui figure dans les fonctions fournis ci-dessus).

Q-4-2 : Validez votre nouveau programme sur des exemples simples.