SeaTech École d'ingénieurs

Modélisation et Calculs Fluides et Structures

Compte Rendu de TP: Calculs Numérique Avancés

Résolution de l'équation de la chaleur de 1D

Enseignant: Mr.Golay

Etudiant: Dupont Ronan

Année Universitaire 2019-2020

I) Introduction

Dans ce TP, nous nous intéressons à la simulation d'un problème physique : la diffusion de la chaleur en 1D. Pour simuler ce problème, nous utiliserons différentes méthodes numériques comme les méthodes explicite, implicite et de Cranck-Nicolson. Nous coderons ces méthodes en fortran 90.

II) Modèle physique

En 1D, la diffision de la chaleur est représentée par l'équation suivante:

$$\rho C_p \frac{\partial T}{\partial t} = div(k \nabla T) + r$$

On impose une température pariétale $T = Tp$ ainsi qu'un flux pariétale $-k\frac{\partial T}{\partial n} = \Phi_p = h(T_p - T_e xt)$.
Par un changement de variable, on se ramène à l'équation de la chaleur que nous connaissons mieux sous cette forme:

$$\frac{\partial T}{\partial t} - \frac{\partial^2 T}{\partial^2 x} = 0$$

III) Modèle Numérique

Afin de pouvoir simuler la diffusion thermique, nous avons besoin de poser les conditions limites du problème.
La problème a donc été posé par l'ennoné de la manière suivant:

$$\begin{cases} \dfrac{\partial T}{\partial t} - \dfrac{\partial^2 T}{\partial^2 x} = 0 & \forall (x,t) \in]-1,1[^2 \\ T(x, t=0) = 1 - 4(x - \dfrac{1}{2})^2 & \forall x \in]-1,1[\\ \dfrac{\partial T}{\partial t}(0,t) = \dfrac{\partial T}{\partial x}(1,t) = 0 & \forall t \in]-1,1[\end{cases}$$

Afin d'avoir stabilité du schéma explicite, nous devons respecter la condition CFL: $2\Delta t \leq (\Delta x)^2$.

Dans tout le problème, nous programmerons de sorte à obtenir la température au bout d'une seconde.

1) Méthode explicite

Afin de résoudre ce problème, nous allons dans un premier temps utiliser la méthode d'approximation explicite. Nous posons dans un premier temps: $T(x_i, t_n) = T_i^n$, $h = \frac{1}{n-1}$ où n est le nombre de subdivision dans l'intervalle]0,1[. En utilisant le développement de Taylor au premier ordre et second ordre, on se ramène facilement à:

$$\frac{\partial T}{\partial t}(x_i, t_n) \approx \frac{T_i^{n+1} - T_i^n}{dt} \qquad \frac{\partial^2 T}{\partial^2 x}(x_i, t_n) \approx \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2}$$

On peut donc ramener le problème à l'équation suivant:

$$\frac{T_i^{n+1} - T_i^n}{dt} = \frac{\partial^2 T}{\partial^2 x}(x_i, t_n) \approx \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2}$$

$$\Rightarrow \; T_i^{n+1} = T_i^n + \frac{dt}{h^2}(T_{i+1}^n - 2T_i^n + T_{i-1}^n) \tag{1}$$

Cette méthode étant explicite, nous avons seulement besoin de calculer pas à pas les différents T_i jusqu'à obtenir les derniers T_i.

On initialise donc dans un premier temps une liste A contenant les T_i^0 comme nous pouvons voir sur Figure 2. On l'initialise simplement à l'aide de l'équation de conditions initiale: $T(x, t = 0) = 1 - 4(x - \frac{1}{2})^2$.

Etant donné que nous utilisons une méthode itérative, nous avons besoin d'une deuxième liste (Figure 1). Nous aurons donc une liste stockant les T_i^n et une les T_i^{n+1}. Nous initialisons donc tout ceci au préalable du programme:

- n le nombre de discrétisation
- h le pas de discrétisation spatial
- dt le pas de discrétisation temporel
- A la liste des T_i^n
- B la liste des T_i^{n+1}

Il nous reste plus qu'à démarrer les itérations tout en respectant la condition:

$$\frac{\partial T}{\partial x}(0, t) = \frac{\partial T}{\partial x}(1, t) = 0$$

Une solution simple pour régler cette condition est d'égaliser les deux premiers termes et les deux derniers termes du vecteur Ti.
Nous démarrons donc $\frac{1}{dt}$ itérations avec le schéma (1) pour obtenir le résultat à 1 secondes comme nous pouvons le voir Figure 3.

Cette méthode est très simple car elle est explicite, elle ne demande pas de résolution de système pour être codé. Cependant, elle demande d'avoir un pas h très faible qui doit respecter la condition de stabilité CFL: $2\Delta t \leq (\Delta x)^2$.

2) Méthode implicite

Une seconde approche pour résoudre ce problème est d'utiliser la méthode implicite. De même que précedemment, nous posons: $T(x_i, t_n) = T_i^n$. En utilisant le développement de Taylor au premier ordre:

$$\frac{\partial T}{\partial t}(x_i, t_n) \approx \frac{T_i^{n+1} - T_i^n}{dt} \qquad \frac{\partial^2 T}{\partial^2 x}(x_i, t_n) \approx \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2}$$

On peut ensuite ramener le problème à l'équation suivant:

$$\frac{T_i^{n+1} - T_i^n}{dt} = \frac{\partial^2 T}{\partial^2 x}(x_i, t_{n+1}) \approx \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{h^2}$$

$$\Rightarrow \quad -T_{i+1}^{n+1} + (\frac{h^2}{dt} + 2)T_i^{n+1} - T_{i-1}^{n+1} = \frac{h^2}{dt}T_i^n \tag{2}$$

Ceci étant une méthode implicite, nous posons ceci sous force de système matriciel afin de le résoudre et obtenir les valeurs des T_i.

Après écriture du système obtenu par (2), nous posons donc le système suivant: $Ax = b$ avec :

$$A = \begin{pmatrix} 1 + \frac{2dt}{h^2} & -\frac{dt}{h^2} & & & (0) \\ -\frac{dt}{h^2} & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -\frac{dt}{h^2} \\ (0) & & & -\frac{dt}{h^2} & 1 + \frac{2dt}{h^2} \end{pmatrix} \quad b = \begin{pmatrix} T_1^n \\ \vdots \\ \vdots \\ \vdots \\ T_n^n \end{pmatrix}$$

Algorithmiquement parlant, nous avons juste à déclarer les variables du programme (Figure 4), d'initialiser la matrice A ainsi que le vecteur b (Figure 5).

Nous démarons ensuite la méthode itérative résolvant $1/dt$ fois le système matriciel. Pour résoudre ce système matriciel, la méthode est plus coûteuse car nous faisons appel à un programme de résolution de système matriciel utilisant la méthode du pivot de Gauss. Cette méthode triangularise la matrice mise en argument. Il est donc important d'effectuer une copie de la matrice A avant d'appeler le programme de résolution.
Une fois le vecteur solution x obtenu, on réaffecte b à celui-ci. Nous changeons la première et dernière valeur de ce vecteur pour satisfaire la condition $\frac{\partial T}{\partial t}(0, t) = \frac{\partial T}{\partial x}(1, t) = 0$ (Figure 6).

Contrairement à la méthode explicite, cette méthode n'a pas à satisfaire la condition CFL. Elle fonctionne donc quelque soit h et dt. Cependant, elle est beaucoup plus coûteuse en temps de calcul car elle appel un programme de résolution utilisant le pivot de Gauss ayant une complexité en $o(n^3)$

3) Méthode de Cranck-Nicolson

La dernière méthode : Cranck-Nicolson est une méthode semi-implicite. Pour coder cette méthode, nous avons besoin de coder la théta méthode. La méthode Cranck-Nicolson est la théta méthode si l'on prend $\theta = 0.5$.
La méthode utilise le schéma suivant avec $\theta \in]-1, 1[$:

$$\frac{T_i^{n+1} - T_i^n}{dt} = \theta \frac{\partial^2 T}{\partial^2 x}(x_i, t_{n+1}) + (1-\theta)\frac{\partial^2 T}{\partial^2 x}(x_i, t_n) \approx \theta \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{h^2} + (1-\theta)\frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{h^2}$$

Pour cette méthode, nous obtenons le système d'équation suivant:

On pose: $M = \begin{pmatrix} 2 & -1 & & & (0) \\ -1 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ (0) & & & -1 & 2 \end{pmatrix}$

On a alors:

$$\underbrace{(I_d + \frac{\theta dt}{h^2} M)}_{\text{A}} U^{n+1} = \underbrace{(I_d - \frac{(1-\theta)dt}{h^2} M)}_{\text{P}} U^n \tag{3}$$

On obtient la simplification:

$$AU^{n+1} = PU^n$$

Comme précédemment, on déclare les variables (Figure 6) ainsi que les matrices définies précédemment (Figure 7):

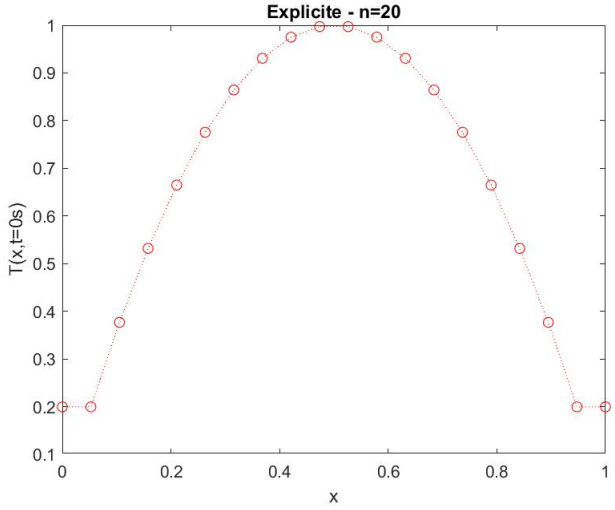
Comme pour la méthode implicite, nous démarrons les iterations pour satisfaire le système (3) ainsi que la condition limite en fixant le premier terme égal au deuxième ainsi que le dernier égal à l'avant dernier (Figure 9).

Cette méthode a l'avantage de regrouper les 3 méthodes abordés en une seule. Cependant, elle ne règle pas le problème de la condition de CFL pour la méthode explicite. De plus, elle est aussi coûteuse que la méthode implicite car elle fait appel à un programme de résolution de système matriciel en $o(n^3)$.

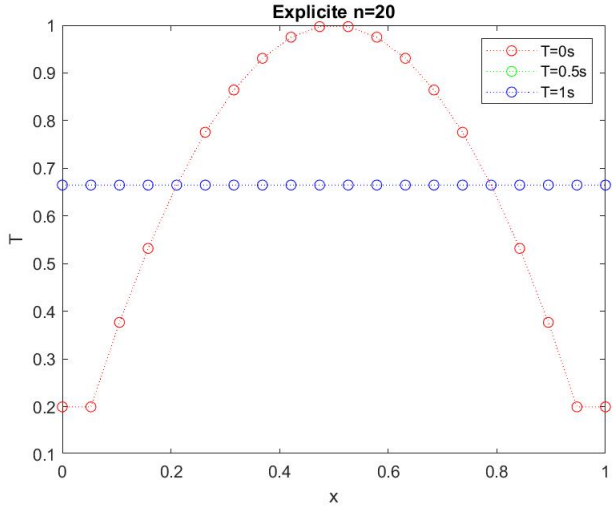
IV) Résultats

Dans cette partie de résultats, nous allons découper l'intervalle]0,1[en un maillage de $n = 20$. Nous regarderons les résultats avec cette donnée et nous regarderons les conditions de convergences sur dt.

Quelque soit le schéma que nous utiliserons, nous aurons cette condition initiale:

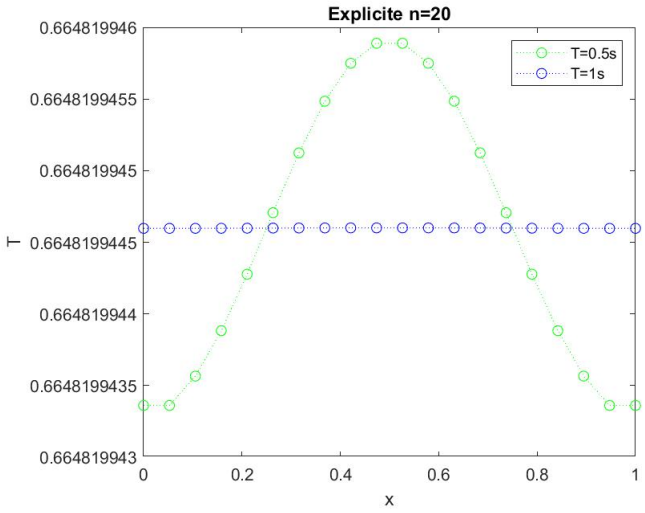


Puis nous verrons que les valeurs évoluent de la manière suivante:

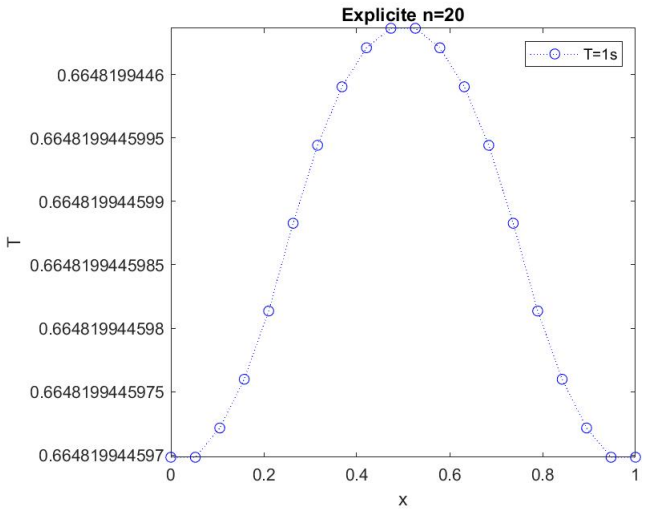


On voit que dés même 0.5s, la température tend vers une valeur constante vers 0.6. On peut voir que les courbes à $t = 0.5$ et $t = 1s$ semblent confondues.

Si l'on zoom sur les courbes à $t = 0.5s$ et $t = 1s$, on obtient ceci:



On remarque l'allure de la courbe qui est très similaire à celle obtenue par les conditions initiales. Elle a comme été "applatie" de par la condition de tangente au bord nulle.
Si l'on trace que la courbe à $t = 1s$:



On voit qu'on obtient exactement la même allure mais cette fois-ci avec une précision meilleure.

1) Méthode explicite

En faisant varier notre dt, nous remarquons qu'il converge quelque soit n à partir du moment où la condition CFL est respecté.

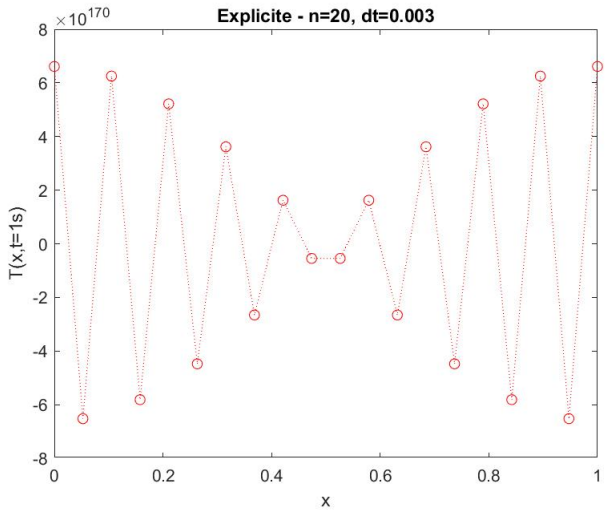
On a donc pour $n = 20$, $dt < \frac{h^2}{2} = \frac{1}{2(n-1)^2} = 0.00277$

On obtient les valeurs suivantes:

```
0.66481939311145788
0.66482048856365650
0.66481942299253827
0.66482042961657384
0.66481950939770063
0.66482031811023945
0.66481964296360485
0.66482016612809502
0.66481980921631278
0.66481999013976389
0.66481999013976389
0.66481980921631278
0.66482016612809502
0.66481964296360485
0.66482031811023945
0.66481950939770063
0.66482042961657384
0.66481942299253827
0.66482048856365650
0.66481939311145788
```

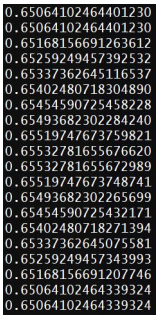
L'avantage de cette méthode est que la complexité est très faible: en $o(\frac{n}{dt})$. Donc les calculs se font très rapidement.

Cependant, il est nécessaire de satisfaire la CFL: $2\Delta t \leq (\Delta x)^2$ sinon nous pouvons avec des divergences comme on peut voir ci-dessous.



2) Méthode implicite

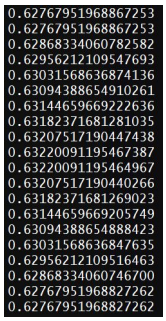
Pour ce qui est de la méthode implicite, nous observons une convergence vers la valeur 0.6 environ quand $dt < 0.000005$ pour n=20. Nous obtenons les valeurs ci-dessous:



L'avantage de cette méthode est qu'on peut utiliser n'importe quelle dt et elle ne divergera pas. Cependant elle ne convergera pas plus vite. De plus, le temps de calcul est bien plus long car la complexité est de l'ordre de $o(\frac{n^3}{dt})$. Il sera donc intéressant d'utiliser cette méthode lorsque nous aurons un petit maillage.

3) Méthode de Cranck-Nicolson

De même que la méthode implicite, nous observons une convergence vers la valeur 0.6 environ quand $dt < 0.000005$ pour n=20. Nous obtenons les valeurs ci-dessous:



Les avantages de cette méthode sont les mêmes que celle de la méthode implicite. Sauf que cette méthode nous permet aussi en paramétrant θ d'utiliser les méthodes explicites et implicites.
Il n'est pas utile d'utiliser cette méthode avec $\theta = 0$ pour obtenir la méthode explicite car celle-ci dépendra toujours de la condition CFL et sa complexité sera de $o(\frac{n^3}{dt})$ contre $o(\frac{n}{dt})$ pour la méthode explicite.
Cependant, il peut être intéressant d'utiliser cette méthode avec $\theta = 1$ car nous avons exactement la même méthode que celle implicite. En effet, nous avons vérifié et nous obtenons exactement les mêmes valeurs pour les mêmes

paramètres.
Cette méthode semble plus précise que la méthode implicite mais tout aussi complexe ($o(\frac{n^3}{dt})$).

V) Conclusion

Dans ce TP nous avons pu voir différentes méthodes pour étudier la diffusion thermique à l'aide de l'équation de la chaleur en 1D. Nous avons vu :

- Un schéma explicite: très intéressant car son coût de calcul est moindre. Cependant il présente une condition de convergence.
- Un schéma implicite: pratique car il n'a pas de condition de convergence. Cependant il a une complexité assez élevée.
- Un schéma semi-implicite: polyvalent car il regroupe la méthode implicite et explicite dans une même méthode. Cependant son coût de calcul est de même assez élevé.

Bien que la méthode implicite et semi-implicite soient plus coûteuse en calculs, nous pouvons optimiser ceci en utilisant des algorithmes de résolution de systèmes plus optimisés comme le programme de résolution de système matriciel : gradient conjugué matrice creuse.
Ces schémas numériques que nous avons codés en Fortran 90 nous ont permis de comprendre les différentes méthodes obtenues par différences finies, leurs limites ainsi que mieux manipuler le langage Fortran 90.

VI) Annexe

1) Méthode explicite

```
 !-- Declarations
  implicit none
  integer                 :: i,j
  integer                 :: n,nt
  real(kind=kind(0.d0))   :: dt,h
  real(kind=kind(0.d0)), dimension(:),allocatable :: A
  real(kind=kind(0.d0)), dimension(:),allocatable :: B

n=20
h=1.d0/(n-1)
dt=0.000001
nt=1/dt
allocate(A(n))
allocate(B(n))
```

Figure 1: Méthode explicite: Déclaration des variables

```

! Condition initiales
DO i=1,n
        A(i)=1-4.d0*((i-1)*h-1.d0/2)**2
ENDDO
```

Figure 2: Méthode explicite: Conditions initiales

```
 DO i=1,1000000
        B(1)=A(1)+dt/h/h*(-2*A(1)+2*A(2))
        DO j=2,n-1
                B(j)=A(j)+dt/h/h*(A(j-1)-2*A(j)+A(j+1))
        ENDDO
        B(n)=A(n)+dt/h/h*(2*A(n-1)-2*A(n))
        A(1:n)=B(1:n)
ENDDO
```

Figure 3: Méthode explicite: Itérations

2) Méthode implicite

```
 !-- Declarations
  implicit none
  integer                 :: i,j,ii,jj
  integer                 :: n,nt
  real(kind=kind(0.d0)) :: dt,h
  real(kind=kind(0.d0)), dimension(:,:),allocatable :: A
  real(kind=kind(0.d0)), dimension(:,:),allocatable :: Aaux
  real(kind=kind(0.d0)), dimension(:),allocatable :: b
  real(kind=kind(0.d0)), dimension(:),allocatable :: x

n=20
h=1.d0/(n-1)

dt=0.000005
nt=1/dt
allocate(A(n,n))
allocate(Aaux(n,n))
allocate(B(n))
allocate(x(n))
```

Figure 4: Méthode implicite: Déclaration des variables

```
 ! Mise en place du systeme matriciel Ax=b

! Vecteur b
DO i=1,n
        b(i)=1-4.d0*((i-1)*h-1.d0/2)**2
ENDDO

! Matrice A

DO i=2,n
        A(i,i)=1+2*dt/h/h
        A(i,i-1)=-dt/h/h
        A(i-1,i)=-dt/h/h
ENDDO
A(1,1)=1+2*dt/h/h
```

Figure 5: Méthode implicite: Conditions initiales, déclaration des vacteurs/-matrices

```
 DO i=1,nt
        Aaux(1:n,1:n)=A(1:n,1:n)
        CALL SOLP(0,Aaux,B,1,1,n,x)
        b(1:n)=x(1:n)
        b(1)=b(2)
        b(n)=b(n-1)
ENDDO
```

Figure 6: Méthode implicite: Itérations

3) Méthode de Cranck-Nicolson

```
 !-- Declarations
  implicit none
  integer                 :: i,j,ii,jj
  integer                 :: n,nt
  real(kind=kind(0.d0)) :: dt,h,teta
  real(kind=kind(0.d0)), dimension(:,:),allocatable :: A,M,Id
  real(kind=kind(0.d0)), dimension(:,:),allocatable :: Aaux,P
  real(kind=kind(0.d0)), dimension(:),allocatable :: b,un
  real(kind=kind(0.d0)), dimension(:),allocatable :: x

n=20
h=1.d0/(n-1)

dt=0.000005
nt=1/dt
teta=0.5

allocate(A(n,n))
allocate(M(n,n))
allocate(Id(n,n))
allocate(P(n,n))
allocate(Aaux(n,n))
allocate(un(n))
allocate(b(n))
allocate(x(n))
```

Figure 7: Méthode de Cranck-Nicolson: Déclaration des variables

```
 ! Matrice M
DO i=2,n
        M(i,i)=2
        M(i,i-1)=-1
        M(i-1,i)=-1
        Id(i,i)=1
ENDDO
M(1,1)=2
Id(1,1)=1

! Matrice A et P, vecteur un
DO i=1,n
        un(i)=1-4.d0*((i-1)*h-1.d0/2)**2
        DO j=1,n
                A(i,j)=Id(i,j)+dt/h/h*M(i,j)
                P(i,j)=Id(i,j)-(1-teta)*dt/h/h*M(i,j)
        ENDDO
ENDDO
```

Figure 8: Méthode de Cranck-Nicolson: Conditions initiales,déclaration des vacteurs/matrices

```
 ! Iterations
DO i=1,nt
        do ii=1,n
                b(ii)=0.d0
                do jj=1,n
                        b(ii)=b(ii)+P(ii,jj)*un(jj)
                enddo
        enddo
        Aaux(1:n,1:n)=A(1:n,1:n)
        CALL SOLP(0,Aaux,B,1,1,n,x)
        un(1:n)=x(1:n)
        un(1)=un(2)
        un(n)=un(n-1)
ENDDO
```

Figure 9: Méthode de Cranck-Nicolson: Itérations