SeaTech École d'ingénieurs

Modélisation et Calculs Fluides et Structures

Compte Rendu de TP n2: Calculs Numérique Avancés

Résolution d'une équation de diffusion thermique non linéaire en 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 d'une flamme en 1D. Pour simuler ce problème, nous utiliserons la méthode de Newton-Raphson. Nous coderons ces méthodes en fortran 90.

II) Modèle physique

Ce problème de diffusion thermique est représenté par la figure suivante:

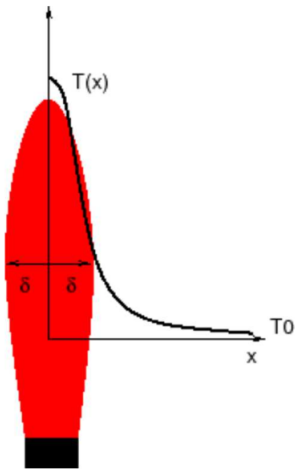


Figure 1: Profil de température transversal dans une flamme

On peut observer que la figure 2 représente l'évolution de la température en fonction d'un x donné. On a δ qui représente l'espacement sur l'axe des x (avec x dans [0,1]) de la source.
Pour étudier ce modèle nous prendrons en compte uniquement la diffusion par rayonnement.
On modélise donc le problème par l'équation suivante:

$$\underbrace{-\frac{\partial}{\partial}(\lambda\frac{\partial T}{\partial x})}_{\text{diffusion}}+\underbrace{\sigma(T^4-T_0^4)}_{\text{rayonnement}}=\underbrace{Q}_{\text{source}}$$

Ici Q représente l'énergie produite par une flamme de largeur 2δ avec une constante de rayonnement σ.
Etant donné que les variations de températures entre la flamme et l'extérieur sont importantes, on a un coefficient de condution λ dépendant de la température avec la lois suivante: $\lambda(T)=\lambda_0 T^q$. Pour le terme source, nous choisissons d'imposer une température seulement sur la flamme et d'imposer une température nulle à l'extérieure.
On impose ensuite les conditions conditions limites suivante:

$$\frac{\partial T}{\partial x}(x=0) \quad T(x=L)=T_0$$

La première condition traduit la symétrie de la flamme en 0. La seconde traduit la température dans la pièce.

On a L >> δ. On pose donc $u=\frac{T}{T_0}=0$ et le problème s'écrit alors:

$$\begin{cases} -\frac{\partial}{\partial}(K(u)\frac{\partial u}{\partial x})+\sigma(u^4-1)=Q(x) \\ \frac{\partial u}{\partial x}(0)=0 \quad u(1)=1 \end{cases} \tag{1}$$

III) Modèle Numérique

1) Discrétisation par différence finies

Afin de pouvoir simuler la diffusion thermique 1, nous utiliserons une discrétisation par différence finies sur [0,1]. On aura un maillage de n subintervalles de longueur $dx = \frac{1}{n-1}$. Nous utilisons la discrétisation centré suivante:

$$\frac{\partial u}{\partial x}(x) \approx \frac{u_i - u_{i-1}}{dx} \qquad \frac{\partial^2 u}{\partial^2 x}(x) \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{dx^2} = \frac{(u_{i+1} - u_i) - (u_i - u_{i-1})}{dx^2}$$

Étant donné que K dépend de u, nous moyennons celui-ci sur chaque les intervalles qui nous avons séparés ci-dessus ($[u_i, u_{i+1}]$ et $[u_{i-1}, u_i]$). On a alors:

$$\begin{cases} K_{i+1/2} = \frac{K(u_{i+1})+K(u_i)}{2} \\ K_{i-1/2} = \frac{K(u_{i-1})+K(u_i)}{2} \end{cases}$$

On obtient donc la discrétisation suivante:

$$\frac{\partial}{\partial}(K(u)\frac{\partial u}{\partial x}) = \frac{K_{i+1/2}(u_{i+1} - u_i) - K_{i-1/2}(u_i - u_{i-1})}{dx^2} \tag{2}$$

Avec 2, l'équation régissant la diffusion 1 devient:

$$-\frac{K_{i+1/2}(u_{i+1} - u_i) - K_{i-1/2}(u_i - u_{i-1})}{dx^2} + \sigma(u_i^4 - 1) = Q_i \tag{3}$$

2) Méthode Newton-Raphson

Pour cette méthode, nous réutiliserons l'équation 1). Celle-ci discrétisé nous fait arriver au système suivant:

$$\begin{cases} F_1(u_0^k, ..., u_N^k) = 0 \\ F_2(u_0^k, ..., u_N^k) = 0 \\ ... \\ F_N(u_0^k, ..., u_N^k) = 0 \end{cases}$$

Où on a :

$$F_i(u_0^k, ..., u_N^k) = Q_i + \frac{K_{i+1/2}(u_{i+1} - u_i) - K_{i-1/2}(u_i - u_{i-1})}{dx^2} - \sigma(u_i^4 - 1)$$

En généralisant la formule de Newton, on arrive à:

$$u_i^{k+1} = u_i^k - (F'(u_i^k))^{-1} F(u_i^k)$$

Et donc à l'équation matricielle suivante:

$$[u_i^{k+1}] = [u_i^k] - [J^k]_{i,j}^{-1}[F_j(u_0^k, ..., u_N^k)] \tag{4}$$

Avec donc

$$J_{i,j}^k = \frac{\partial F_i}{\partial u_j}(u_0^k, ..., u_N^k)]$$

En multipliant à gauche et à droite l'équation 4 par $[J^k]_{i,j}^{-1}$, on arrive à un système de la forme $Ax = b$:

$$[J^k][u_i^{k+1} - u_i^k] = -F_j(u_0^k, ..., u_N^k)] \tag{5}$$

Avec donc $dUn = [u_i^{k+1} - u_i^k]$ ce qui nous mène à : $u_i^{k+1} = u_i^k + dUn$.

Il suffit donc de résoudre k fois le système 5 en réinitialisant à chaque fois la valeur de u^k par la méthode de Newton-Raphson et ainsi cela nous donnera une solution approchée des u_i^k.

3) Résolution numérique

D'un point de vu numérique, il ne nous reste plus qu'à bien poser numériquement le problème puis appliquer la méthode de Newton-Raphson.

3).1 Calcul de J

Dans un premier temps, nous allons calculer les coefficients de la matrice jacobienne:

$$J_{i,j}^k = \frac{\partial F_i}{\partial u_j}(u_0^k, ..., u_N^k)] = \begin{pmatrix} a_0 & 2b_0 & & & (0) \\ c_1 & a_1 & b_1 & & \\ & c_2 & \ddots & \ddots & \\ & & \ddots & a_{n-1} & b_{n-1} \\ (0) & & & 0 & 1 \end{pmatrix}$$

On a :

$$\begin{aligned} ai =& \frac{\partial}{\partial u_i}(Q_i + \frac{K_{i+1/2}(u_{i+1} - u_i) - K_{i-1/2}(u_i - u_{i-1})}{dx^2} - \sigma(u_i^4 - 1)) \\ =& \frac{\frac{1}{2}(\frac{\partial K_i}{\partial u_i})(2u_i^n - u_{i-1}^n - u_{-+1}^n)}{dx^2} + \frac{K_{i+1/2} + K_{i-1/2}}{dx^2} + 4\sigma(u_i^n)^3 \approx \frac{K_{i+1/2} + K_{i-1/2}}{dx^2} + 4\sigma(u_i^n)^3 \\ bi =& \frac{\partial}{\partial u_{i+1}}(Q_i + \frac{K_{i+1/2}(u_{i+1} - u_i) - K_{i-1/2}(u_i - u_{i-1})}{dx^2} - \sigma(u_i^4 - 1)) \\ =& - - \frac{\frac{1}{2}(\frac{\partial K_{i+1}}{\partial u_{i+1}})(u_{i+1}^n - u_i^n)}{dx^2} - \frac{K_{i+1/2}}{dx^2} \approx - \frac{K_{i+1/2}^n}{dx^2} \\ ci =& \frac{\partial}{\partial u_{i+1}}(Q_i + \frac{K_{i+1/2}(u_{i+1} - u_i) - K_{i-1/2}(u_i - u_{i-1})}{dx^2} - \sigma(u_i^4 - 1)) \\ =& - - \frac{\frac{1}{2}(\frac{\partial K_{i-1}}{\partial u_{i-1}})(u_{i-1}^n - u_i^n)}{dx^2} - \frac{K_{i-1/2}}{dx^2} \approx - \frac{K_{i-1/2}^n}{dx^2} \end{aligned}$$

On a donc ces les termes suivant dans la matrice:

$$\boxed{ai \approx \frac{K_{i+1/2}^n + K_{i-1/2}^n}{dx^2} + 4\sigma(u_i^n)^3 \quad bi \approx - \frac{K_{i+1/2}^n}{dx^2} \quad ci \approx - \frac{K_{i-1/2}^n}{dx^2}}$$

δ =ecartement de la source
β=température de la flamme

3).2 Calcul de Q

Nous devons aussi poser de manière numérique le terme source Q. On sait qu'on impose une température de B sur l'intervalle $[0, \delta]$. Ceci se résume à représenter Q par:

$$Q(x) = \beta H(\delta - x)$$

Avec H la fonction d'Héviside modélisé par:

$$H(x) = \begin{cases} 0 \ si \ x < 0 \\ 1 \ si \ x \geq 0 \end{cases}$$

On arrive donc à:

$$Q(x) = \begin{cases} 0 \ si \ x > \delta \\ 1 \ six \leq \delta \end{cases}$$

3).3 Calcul de la solution u

Comme nous l'avons vu précédemment, la méthode de Newton-Raphson consiste à résoudre k fois le système 5.
Afin d'optimiser notre programme, nous imposerons une condition d'arrêt en fonction de l'erreur que nous calculons avec la formule suivante:

$$erreur = \sqrt{\frac{1}{N}\sum_{i=1}^{N} F_i(u_1^n, ...u_N^n)^2}$$

L'idée générale du programme est donc de démarrer k itérations (dépendant de la précision demandée) puis dans la boucle:

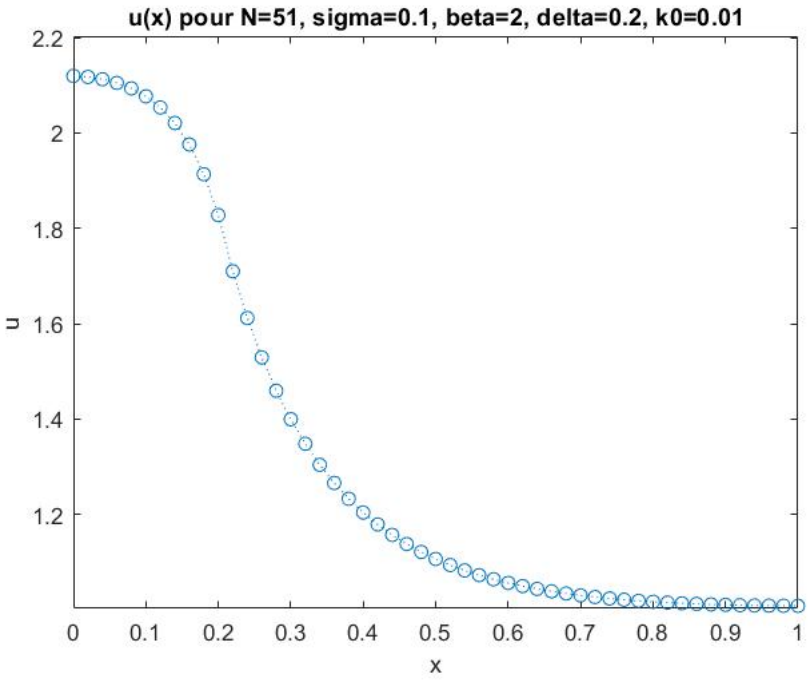
- Calculer ou recalculer les $K_{i+1/2}$,$K_{i-1/2}$
- Calculer ou recalculer le second membre F
- Calculer ou recalculer la matrice Jacobienne
- Résoudre le système de type $Jx = F$ pour obtenir dUn
- Calculer les u_i^{k+1} à l'aide de la formule de Newton-Raphson

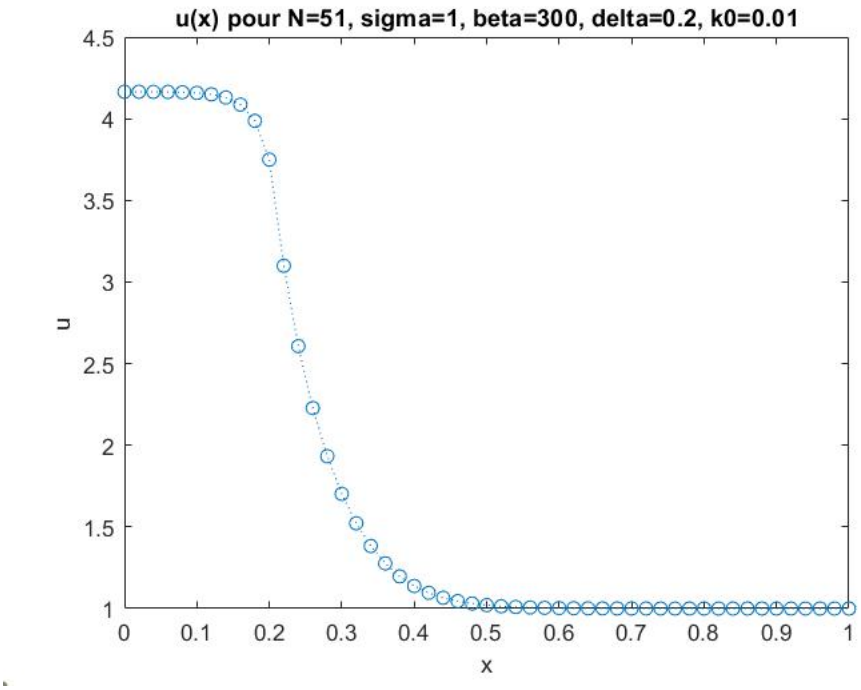
On codera tout ceci sous Fortran 90 comme on peut voir en annexe sur les figures ... .

IV) Résultats

1) Interprétation des courbes

Après avoir écrit notre programme, nous traçons dans un premier temps les deux figures demandée dans l'énnoncé:





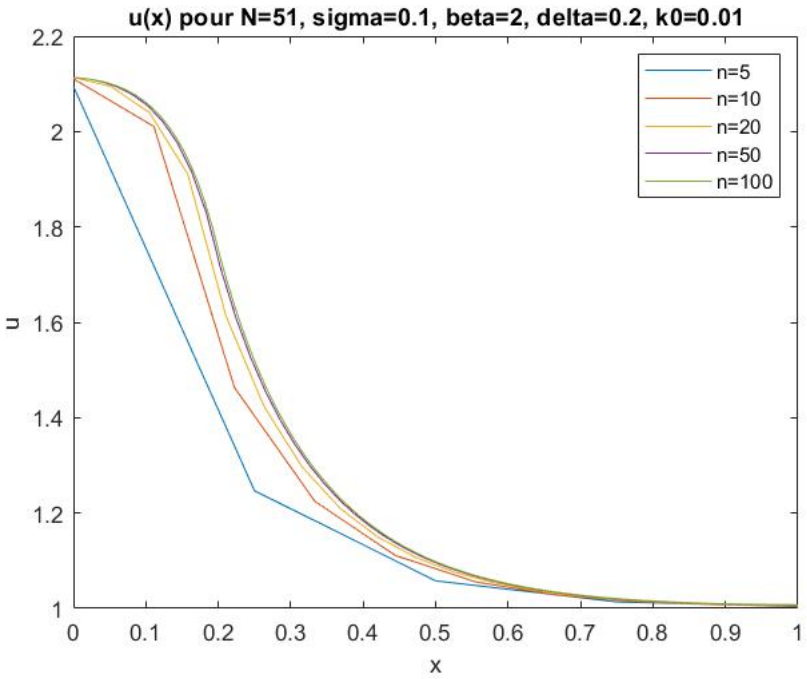
Sur les deux courbes, on remarque que la condition $\delta = 0.2$ est bien respecté car dans les deux cas on voit que la température descend radicalement à partir de x=0.2.
On remarque de même que les deux conditions de 1 sont respectée:

- La tangente à l'origine est bien nulle
- On a bien u(1)=1

Pour ce qui est de l'inclinaison de la pente passé $\delta = 0.2$, on peut justifier que pente est plus grande sur la deuxième figure à cause du coefficient sigma qui est 10 fois supérieur à celle de la figure 1. En effet, ce coefficient représente le rayonnement, on a donc un rayonnement plus fort sur la figure 2, ce qui fait que la température chute plus rapidement après δ passé.

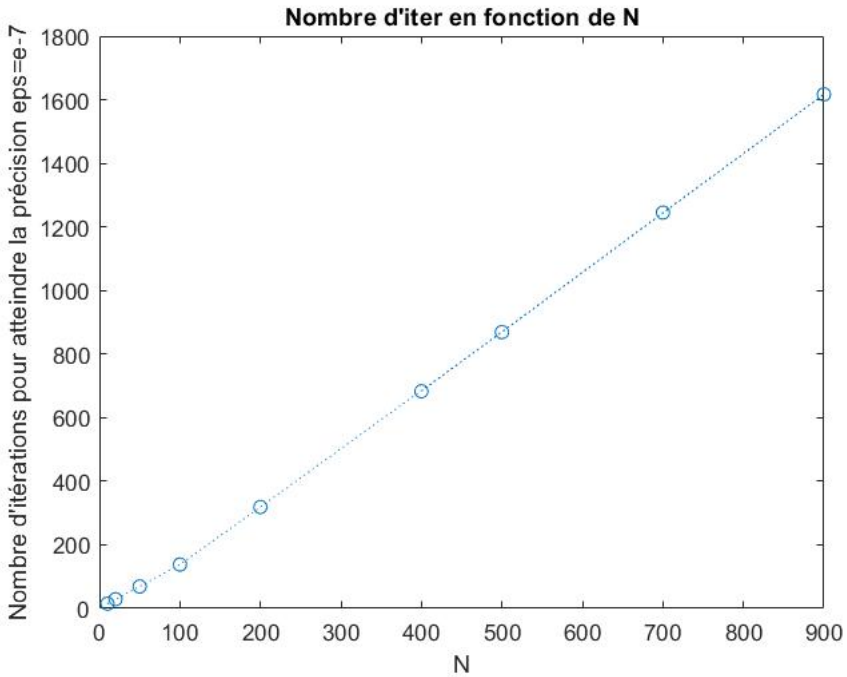
2) Convergence

On a ensuite tracé différentes courbes pour différents N et obtenu la courbe suivante:



On remarque qu'à partir de n=20, la solution semble être assez précise.

On a ensuite cherché à savoir pour combien d'itérations on arrivait à une précision $\epsilon = 10e-7$. On a donc tracé pour plusieurs n le nombre d'itérations nécessaires pour atteindre cette précision:



On remarque que le nombre d'itérations nécessaire en fonction de n est quasi linéaire. On peut en déduire un coefficient d'environ 1.8.

V) Conclusion

Dans ce TP, nous avons vu la méthode de Newton-Raphson que nous avons programmé en fortran 90 sur l'exemple de diffusion thermique d'une flamme.
D'après le sujet du TP, cette méthode cette être la plus optimisée car elle atteint 12 fois plus rapidement une précision à e-8 que la méthode implicite et 148 fois plus rapidement que la méthode explicite.

Annexe

```
 !-- Declarations
  implicit none
  integer                  :: i,j,ii,jj,iter
  integer                  :: n,nt,c
  real(kind=kind(0.d0)) :: dt,h,k0,sigma,beta,esp,dx,u,y,y1,y2,y3,delta,er
  real(kind=kind(0.d0)), dimension(:,:),allocatable :: Jac
  real(kind=kind(0.d0)), dimension(:,:),allocatable :: Jacaux
  real(kind=kind(0.d0)), dimension(:),allocatable :: Un,Kp12,Kn12,F
  real(kind=kind(0.d0)), dimension(:),allocatable :: dUn,Q

n=51
dx=1.d0/(n-1)

dt=0.0000005
nt=1/dt
k0=0.01
sigma=0.1
beta=1
esp=0.0000001
delta=0.2
er=1
c=0
allocate(Jac(n,n))
allocate(Jacaux(n,n))
allocate(Un(n))
allocate(dUn(n))
allocate(Kp12(n))
allocate(Kn12(n))
allocate(F(n))
allocate(Q(n))
! ------------- Corps du programme ------------------

! Initialisation des listes
DO i=1,n
        Un(i)=1
        dUn(n)=0
        F(i)=0
        IF (((i-1)*dx)<=delta) THEN
                Q(i)=beta
        ELSE
                Q(i)=0
        END IF
ENDDO
```

Figure 2: Déclarations, initialisation des vecteurs

```
 DO WHILE (er>esp)
  c=c+1
  ! Initialisation des Ki+1/2 Ki-1/2
  DO i=2,n-1
          CALL K(K0,Un(i-1),y1)
          CALL K(K0,Un(i),y2)
          CALL K(K0,Un(i+1),y3)
          Kp12(i)=(y2+y3)/2
          Kn12(i)=(y2+y1)/2
  ENDDO

  i=1
  CALL K(K0,Un(i),y2)
  CALL K(K0,Un(i+1),y3)
  Kp12(i)=(y2+y3)/2

  i=n
  CALL K(K0,Un(i-1),y1)
  CALL K(K0,Un(i),y2)
  Kn12(i)=(y2+y1)/2

  ! On cree le vecteur F
  DO i=2,n-1
    F(i)=Q(i)-sigma*(Un(i)**4-1)+1/dx/dx*(Kp12(i)*(Un(i+1)-Un(i))-Kn12(i)*
  (Un(i)-Un(i-1)))
  ENDDO
  i=1
  F(i)=Q(i)-sigma*(Un(i)**4-1)+1/dx/dx*(Kp12(i)*(Un(i+1)-Un(i))-Kn12(i)*
(Un(i)-Un(i)))
  i=n
  F(i)=Q(i)-sigma*(Un(i)**4-1)+1/dx/dx*(Kp12(i)*(Un(i)-Un(i))-Kn12(i)*
(Un(i)-Un(i-1)))

  ! On cree la matrice J

  DO i=1,n
          Jac(i,i)=1/dx/dx*(Kp12(i)+Kn12(i))+4*sigma*Un(i)**3
          Jac(i,i-1)=-1/dx/dx*Kp12(i)
          Jac(i,i+1)=-1/dx/dx*Kn12(i+1)
  ENDDO

  ! On resout
  Jacaux(1:n,1:n)=Jac(1:n,1:n)
  CALL SOLP(1,Jacaux,F,1,1,n,dUn)
  Un(1:n)=Un(1:n)+dUn(1:n)
  CALL norm(F,n,y)
  nf=n
  er=y/sqrt(nf)
ENDDO
```

Figure 3: Méthode de Newton-Raphson: Itérations

```
 SUBROUTINE SOLP   (NSYM,VKG,VFG,IKG,IFG,NEQ,VU)
!============================ DEBUT DES DECLARATIONS ====================
implicit none
real(kind=8)::vzero=0.d0,cl,rpiv
real(kind=8),allocatable,dimension(:,:)::vkg
real(kind=8),allocatable,dimension(:)::VFG,VU
integer :: NSYM,IKG,IFG,NEQ, i,j,ij,n1,ij1,is,is1,ii
!-------  Triangularisation
      IF(IKG.EQ.0.AND.IFG.EQ.0) GOTO 9999
      N1=NEQ-1
      DO 50 IS=1,N1
      RPIV=VKG(IS,IS)
      IF (RPIV) 10,800,10
10    IS1=IS+1
      DO 50 II=IS1,NEQ
      IF(NSYM.EQ.1) CL=VKG(II,IS)
      IF(NSYM.NE.1) CL=VKG(IS,II)
      IF(CL.EQ.VZERO) GOTO 50
      CL=CL/RPIV
      IF(IFG.EQ.1) VFG(II)=VFG(II)-CL*VFG(IS)
      IF(IKG.EQ.0) GOTO 50
      IF(NSYM.EQ.1) GOTO 32
      DO 30 IJ=II,NEQ
      VKG(II,IJ)=VKG(II,IJ)-CL*VKG(IS,IJ)
30    VKG(IJ,II)=VKG(II,IJ)
      GOTO 50
32    DO 40 IJ=IS1,NEQ
40    VKG(II,IJ)=VKG(II,IJ)-CL*VKG(IS,IJ)
50    CONTINUE
!-------  RESOLUTION DU SYSTEME TRIANGULAIRE
      IF(VKG(NEQ,NEQ).NE.VZERO) GOTO 55
      IS=NEQ
      GOTO 800
55    IF(IFG.EQ.0) GOTO 9999
      VU(NEQ)=VFG(NEQ)/VKG(NEQ,NEQ)
      DO 70 II=1,N1
      IS1=IS1-1
      CL=VZERO
      IJ1=IS1+1
      DO 60 IJ=IJ1,NEQ
60    CL=CL+VKG(IS1,IJ)*VU(IJ)
70    VU(IS1)=(VFG(IS1)-CL)/VKG(IS1,IS1)
!-------  Fin SUBROUTINE SOLP
      GOTO 9999
!-------  Erreurs
800    WRITE(*,8000) IS
8000  FORMAT('␣*␣SOLP␣␣-E-RPIVNUL␣Pivot␣nul␣equation',I5)
      GOTO 9999
!-------  Fin
9999  CONTINUE
      RETURN
!========   FIN DU MODULE SOLP      ====
      END SUBROUTINE SOLP
```


Figure 4: Subroutine Résolution par Pivot de Gauss

```
 SUBROUTINE K(K0,u,Ku)
  real(kind=kind(0.d0)) :: K0,u,Ku,p
  p=0.5
  Ku=K0*(u)**p
  RETURN
END SUBROUTINE K
```

Figure 5: Subroutine fonction K

```
 SUBROUTINE norm(U,n,nrm)
  real(kind=kind(0.d0)), dimension(:),allocatable :: U
  real(kind=kind(0.d0)) :: nrm
  integer               :: n
  nrm=0
  do i=1,n
    nrm=nrm+U(i)**2
  enddo
  nrm=sqrt(nrm)
  RETURN
END SUBROUTINE norm
```

Figure 6: Subroutine Calcul de norme

Code morse

— . -.- -.-. .. -.. .——-. .- ...- — .. -.- -.-. ..-