Laura Mendoza
M1 CSSI

Calcul Scientifique

Rapport des travaux pratiques :

- TP1 : Factorisation d'une matrice tridiagonale et application au problème de Laplace en dimension 1
- TP2 : Résolution numérique de l'équation de la chaleur
- TP3 : Résolution de l'équation de transport
- TP5 : Résolution de l'équation des ondes
- TP6 : Résolution de l'équation de Laplace sur un carré
- TP7 : Méthode des éléments finis

FACTORISATION D'UNE MATRICE TRIDIAGONALE

Question 1 :

Ci-dessous un extrait de code qui montre la factorisation LU :

```
      if (ityp.eq.1) then !factorisation LU
      do i=2,n
            l(i)=l(i)/d(i-1) !formule de facto pour le vecteur l (on utilise les
mêmes tableaux)
            d(i)=d(i)-l(i)*s(i-1) !formule de factorisation pour le vecteur d
            ! le vecteur s reste inchangé, donc il n'y a pas besoin de le
modifier
      end do
    end if
```

L'écriture de la matrice A en trois vecteurs nous permet d'épargner beaucoup d'espace mémoire. Lors de la factorisation LU on n'a pas besoin de réserver plus d'espace mémoire puisqu'on peut écraser les données de la matrice A par celles des matrices L et U. Il faut juste faire attention à calculer d'abord 'l' et puis 'd'.

Ci-dessous des calculs de la factorisation LU par notre programme et par Scilab pour n=3.

Notre programme

Matrice A:

low	diag	sup
	3.000	-0.500
-1.000	4.000	-0.500
-1.000	5.000	

Après factolu avec ityp=1

low	diag	sup
	3.000	-0.500
-0.333	3.833	-0.500
-0.260	4.869	

Scilab

```
A  =
    3.  - 0.5    0.
  - 1.    4.   - 0.5
    0.  - 1.     5.
-->[L,U]=lu(A)
U  =
  3.        - 0.5         0.
  0.    3.8333333   - 0.5
  0.        0.          4.8695652
L  =
  1.                  0.          0.
 - 0.3333333    1.          0.
  0.        - 0.2608696    1.
```

On constate qu'on trouve les mêmes résultats.

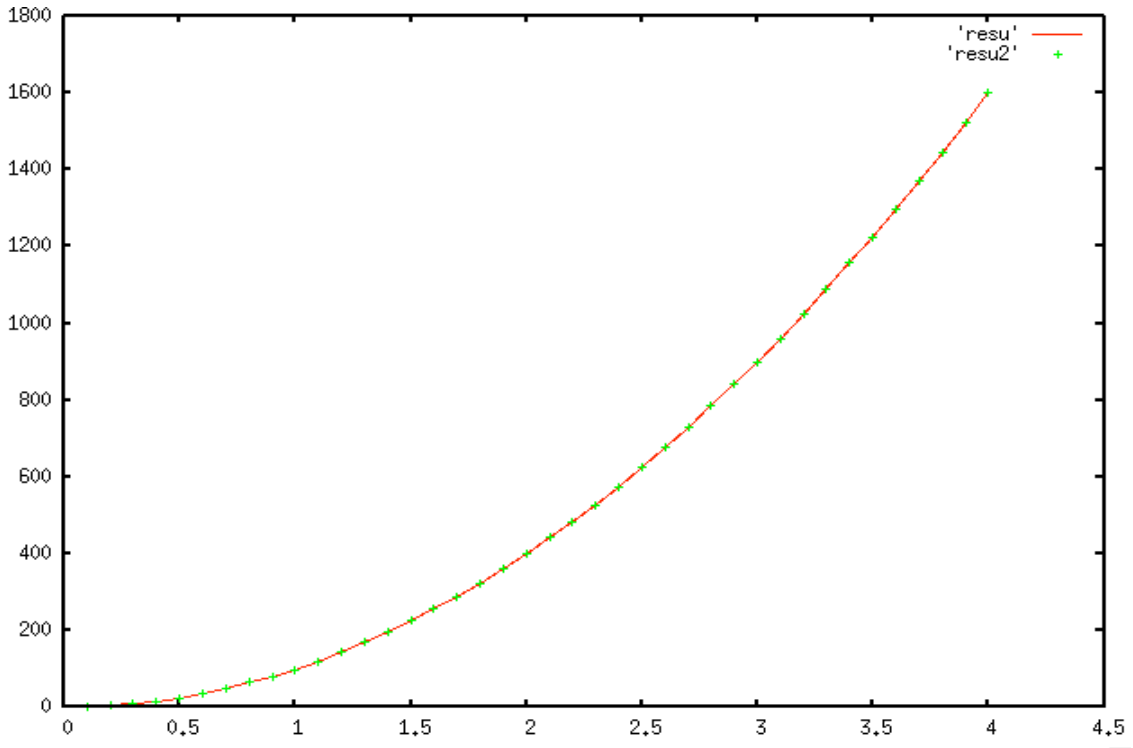
Question 2 :

Pour vérifier si le programme est juste on peut l'exécuter en le testant avec une matrice A de notre choix et une fonction quelconque qui nous donnera la valeur exacte du vecteur x. On calcule alors le vecteur b, en faisant le produit Ax. Il nous faut alors lancer le programme avec la matrice A et le vecteur b calculé (à la fin on cherche à avoir dans un nouveau vecteur xapprox des valeurs approchées à celles qu'on avait calculées en x).

Il faudra alors lancer le programme d'abord avec ityp=1 pour effectuer la factorisation de A sous forme LU, puis avec ityp=2 qui calcule par les algorithmes de remontée et descente le résultat.

On comparera alors le résultat obtenu, qui est stocké dans le vecteur x passé et la solution exacte qui était donnée par la fonction du départ. Pour cela on peut écrire dans un fichier $x_i = i * dx$ avec dx le pas de déplacement (donc la longueur de l'intervalle sur le nombre de divisions de l'intervalle) et le résultat trouvé à ce point là. On peut stocker dans un autre fichier la même chose sauf qu'avec la fonction exacte au lieu de la solution trouvée. On peut alors afficher les courbes grâce aux points dans les fichiers.

Voici un exemple de courbe obtenue avec n=40 et d=0.1:



APPLICATION : PROBLÈME DE LAPLACE EN DIMENSION 1

Données du problème :

$$\begin{cases} -u''(x) = f(x) \\ u(0) = u(L) = 0 \end{cases} \quad \text{et} \quad x \in]0,L[$$

Question 1 :

Discrétisation :

Posons $\Delta x = \frac{L}{N-1}, N \in \mathbb{N}$ et donc $\begin{cases} x_1 = 0 \\ x_i = (i-1) \times \Delta x \\ x_N = L \end{cases}$. De même pour le temps : $\Delta t = \frac{T}{p}, p \in \mathbb{N}$ et $t_n = n * \Delta t$.

Si on note $h = \frac{L}{N-1}$, grâce à un développement limité, on peut estimer $u(x_i + h)$:

$$\begin{cases} u(x_i+h) = u(x_i) + hu'(x_i) + \frac{h^2}{2}u''(x_i) + \frac{h^3}{6}u^{(3)}(x_i) + \frac{h^4}{24}u^{(4)}(\xi_1) \\ u(x_i-h) = u(x_i) - hu'(x_i) + \frac{h^2}{2}u''(x_i) - \frac{h^3}{6}u^{(3)}(x_i) + \frac{h^4}{24}u^{(4)}(\xi_2) \end{cases}$$

En additionnant les 2 équations trouvées, on trouve une approximation pour u"(x), on obtient :

$$u(x_i+h) + u(x_i-h) = u(x_i) + hu'(x_i) + \frac{h^2}{2}u''(x_i) + \frac{h^3}{6}u^{(3)}(x_i) + \frac{h^4}{24}u^{(4)}(\xi_1)$$
$$+ u(x_i) - hu'(x_i) + \frac{h^2}{2}u''(x_i) - \frac{h^3}{6}u^{(3)}(x_i) + \frac{h^4}{24}u^{(4)}(\xi_2)$$
$$u(x_i+h) + u(x_i-h) = u(x_i) + \cancel{hu'(x_i)} + \frac{h^2}{2}u''(x_i) + \cancel{\frac{h^3}{6}u^{(3)}(x_i)} + \frac{h^4}{24}u^{(4)}(\xi_1)$$
$$+ u(x_i) - \cancel{hu'(x_i)} + \frac{h^2}{2}u''(x_i) - \cancel{\frac{h^3}{6}u^{(3)}(x_i)} + \frac{h^4}{24}u^{(4)}(\xi_2)$$

$u(x_i+h)+u(x_i-h)=2*u(x_i)+h^2u''(x_i)+O(h^4)$

Et finalement en isolant u'', on a :

$$(1)\quad u''(x_i)=\frac{u(x_i+h)+u(x_i-h)-2*u(x_i)}{h^2}$$

avec une erreur d'ordre $O(h^4)$ qui tend vers 0 quand h tend vers 0.

D'après la définition des $x_i, i=0..N$ on remarque que $\begin{cases} x_{i+1}=x_i+h \\ x_{i-1}=x_i-h \end{cases}$. On peut alors remplacer la formule (1) par :

$$\boxed{(2)\quad u''(x_i)=\frac{u(x_{i+1})+u(x_{i-1})-2*u(x_i)}{h^2}}\ .$$

On observe donc qu'on a une formule récursive.

Or, on sait que $u(0)=u(x_0)=0$ et $u(L)=u(x_N)=0$, on peut alors traduire l'équation (2) par le système linéaire suivant (écrit sous forme de système matriciel), en prenant comme notation $u_i=u(x_i)$:

$$\begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & 0 & \dots \\ 0 & 0 & \dots & \dots & \dots & 0 \\ 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \dots \\ \dots \\ u_{N-1} \\ u_N \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \dots \\ \dots \\ f(x_{N-1}) \\ f(x_N) \end{pmatrix}$$

<u>Remarque :</u>

On rappelle que les hypothèses initiales nous donnent -u''(x)=f(x). On peut donc changer les signes de la formule et remplacer le membre de gauche par f(x).

Question 2 :

On retrouve alors un problème sous la forme Ax=b et on peut utiliser le problème écrit en I.

En prenant une fonction $u(x)=\sin(\pi*p*x)*\exp(-\lambda*x)$ on peut calculer la dérivée seconde manuellement, on peut alors vérifier si les résultats trouvés par notre algorithme sont justes.

Voici un extrait du programme qui nous permet de tester les résultats :

```
    pi=4.d0*atan(1.d0) !calcul de pi
    p=3 !on met une valeur quelconque pour p
    dx=1.d0/(n+1) !on définit le 'pas'
    !on définit la matrice A qui est tridiagonale
    do i=1,n
      d(i)=2.d0 !on définit le vecteur diagonales
    end do
    do i=2,n
      l(i)=-1.d0!inférieur
    end do
    do i=1,n-1
      s(i)=-1.d0!et supérieur
    end do

    p4=4.d0*p*pi !pour faire moins longue et plus claire l'écriture de f
    do i=1,n !on a manuellement calculé -u''=f et on le garde dans la variable f
      xi=i*dx !on calcule xi=i*delta
      x1=1.d0+xi !variable
      pp=2.d0*p*pi*xi!variable
      f(i)=(p4*p*pi*sin(pp)*log(x1)-p4*cos(pp)/x1+sin(pp)/x1**2)*dx**2 !formule,
f=-u''*dx^2
    end do
    call factolu(n,d,l,s,x(1),f,1) !facto LU
    call factolu(n,d,l,s,x(1),f,2) !calcul de x
```

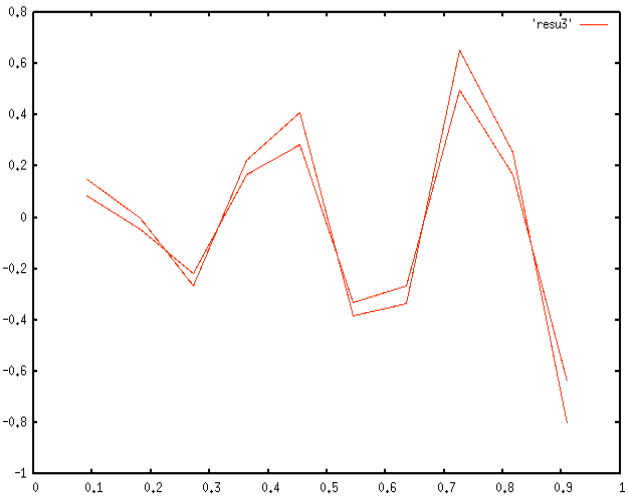
On écrit alors dans un fichier les résultats trouvés (manuellement et par le programme) :

```
open(1,file='resu3') !création de un fichier appelé en machine 'resu3' et dans
le programme '1'
    do i=1,n
      write(1,*) i*dx,x(i) !i*dx=x1, x2, x3,..., et on écrit le résultat
approché f(xi)
    end do
    write(1,*) !retour à la ligne
```

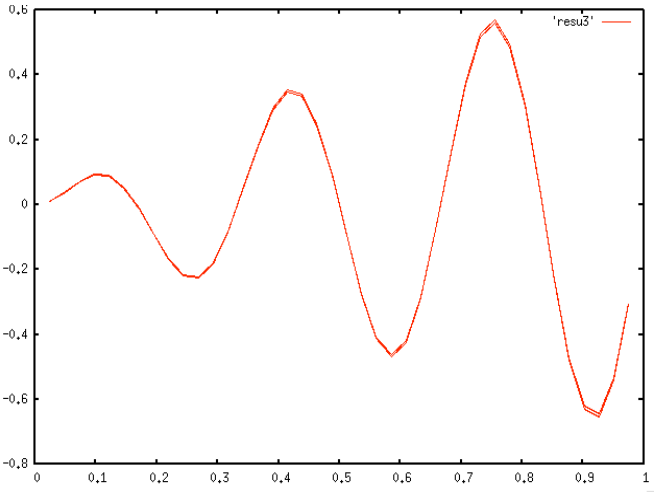
```
    do i=1,n
      xi=i*dx
      write(1,*) i*dx,xexact(xi) !Idem qu'avant mais on écrit le résultat exact
    end do
    close (1) !on a ecrit a chaque fois xi et le résultat calculé et le résultat
exacte pour pouvoir dessiner la courbe grâce a gnuplot.
```

On obtient les courbes :

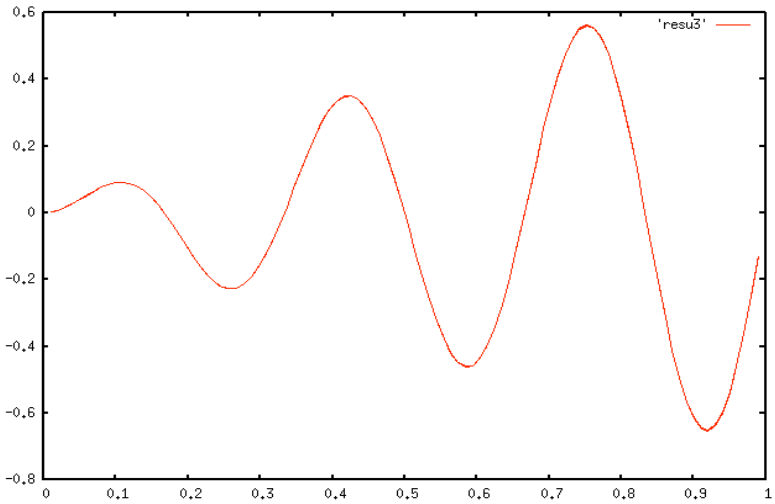
n=10



n=40



n=100



On remarque que plus n est grand, plus le résultat est précis, ce qui est logique car n représente dans ce cas le nombre de points ou on étudie la fonction.

Question 3 :

On peut se demander alors quelle est l'erreur pour chacun de ses n. Pour cela on peut créer une fonction qui calcule le maximum de l'erreur à chaque point. Ce qui donne la fonction :

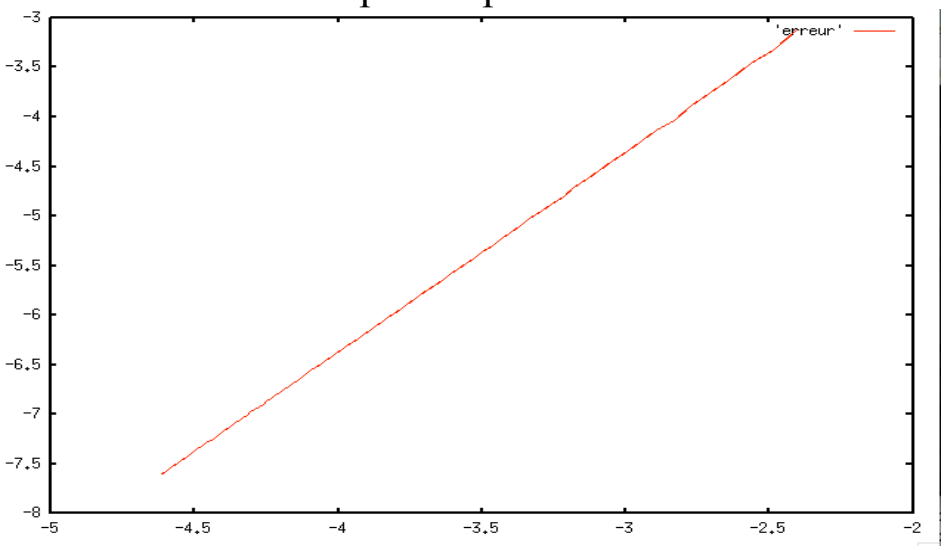
```
function erreur (x, dx, n)
!x est un tableau contenant les valeurs approchées, dx est le pas et n le nombre
de points.
    implicit none
    integer::i,n
    real*8::dx,erreur,xexact !xexact est une fonction qui calcule la valeur
exacte de x
    real*8,dimension(1:n)::x
    erreur=abs(x(1)-xexact(dx)) !On commence par calculer l'erreur au premier
point
    do i=2,n
      if (erreur.lt.abs(x(i)-xexact(i*dx))) then !si on trouve un erreur plus
grand...
            erreur=abs(x(i)-xexact(i*dx)) !on garde cette nouvelle valeur
      end if
    end do
end function erreur
```

Cette fois-ci on s'intéresse à étudier les erreurs par rapport à n, n sera donc une variable et non paramètre. Il faut alors définir les vecteurs comme louables et faire une boucle qui fera varier n.
On obtient alors le programme suivant :

```
program test
    implicit none
    real*8,dimension(:),allocatable::d,f,x,l,s !On ne connait pas encore n
    integer::i,p,n
    real*8::dx,sup,erreur

    open(1,file='erreur') !création d'un fichier appelé 'erreur'
    do n=10,100 !on va prendre n entre 10 et 100
      allocate(d(1:n),f(1:n),x(1:n)) !maintenant que n est défini on peut
allouer les vecteurs
      allocate(l(2:n))
      allocate(s(1:n-1))
    !definition de A
      d(1)=2.d0
      do i=2,n-1
            d(i)=2.d0 !on definit diag
            l(i)=-1.d0!low
```

```
            s(i)=-1.d0!et supp
        end do
        d(n)=2.d0
        l(n)=-1.d0
        dx=1.d0/(n+1) !on defini le 'pas'

        call xapprox(f,dx,n) !fonction qui met les valeurs approchées dans f
        call factolu(n,d,l,s,x(1),f,1) !facto LU
        call factolu(n,d,l,s,x(1),f,2) !calcul de x

        sup=erreur (x(1), dx, n) !on calcul l'erreur max pour n
        write (1,*) log(dx),log(sup) !on écrit dans le fichier
        deallocate(d,f,x,l,s) !on n'a plus besoin des vecteurs
        end do !on a fini
    close (1) !
end program test
```

Affichage de la courbe donnée par les points en 'erreur' :



On obtient approximativement une droite ce qui veut dire que l'erreur dépend proportionnellement de n, et plus n est grand plus l'erreur diminue.

Question 4 :

<u>Problème mixte de Dirichlet-Neumann :</u>

Les nouvelles hypothèses sont : $\begin{cases} -u''(x)=f(x) \\ u'(0)=u(L)=0 \end{cases}$ et $x \in]0,L[$.

On remarque que le changement par rapport au problème précédent est : $u'(0)=0$, ce qui donne : $u(0)=C, C \in \mathbb{N}, constante$.

On n'a pas alors le même système linéaire que précédemment. Notamment on a $u_1=2u_1-u_2-u_0$, la matrice A alors est trop petite et ne doit plus être

de taille N mais de taille N+1, de même pour les x et b. On obtient alors un nouveau système :

$$\begin{pmatrix} 1 & 0 & 0 & \dots & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots & 0 \\ 0 & -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & \dots \\ 0 & \dots & 0 & \dots & \dots & \dots & 0 \\ 0 & \dots & \dots & 0 & -1 & 2 & -1 \\ 0 & \dots & \dots & \dots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ \dots \\ \dots \\ u_{N-1} \\ u_N \end{pmatrix} = \begin{pmatrix} f(x_0) = C \\ f(x_1) \\ f(x_2) \\ \dots \\ \dots \\ f(x_{N-1}) \\ f(x_N) \end{pmatrix}$$

qui équivaut à $\tilde{A}\tilde{x} = \tilde{b}$. Or la matrice Ã est tridiagonale est donc on peut effectuer les mêmes études que sur A.

Concrètement dans notre programme on a quelques changements à faire dû au rajout d'une colonne et une ligne 'au début' de la matrice. Pour simplifier les changements, à faire on peut dire qu'on a ajouté la colonne et la ligne d'indice 0.
On a alors :

```
real*8,dimension(0:n)::d,f,x
real*8,dimension(1:n)::l
real*8,dimension(0:n-1)::s
```

Ce changement est à faire dans le programme principal, mais aussi dans la fonction 'factolu'. On a aussi une nouvelle définition de A :

```
d(0)=1.d0
s(0)=0.d0
do i=1,n-1
  d(i)=2.d0 !on definit diag
  l(i)=-1.d0!low
  s(i)=-1.d0!et supp
end do
d(n)=2.d0
l(n)=-1.d0
```

Pour tester si cette nouvelle marche ou pas il suffit de prendre une fonction telle que C=0. Et on doit trouver la même solution que précédemment.

RÉSOLUTION NUMÉRIQUE DE L'ÉQUATION DE LA CHALEUR

Données du problème :

$$\begin{cases} u_t(x,t) - u_{xx}(x,t) = 0 & (1) \\ u(x,0) = u_0(x) & (2) \\ u_x(0,t) = u_x(L,t) = 0 & (3) \end{cases} \text{ avec } \begin{cases} t > 0 \\ L > x > 0 \end{cases} \text{ et } u_0(x) = \begin{cases} 1 \text{ si } x \in [3,5] \\ 0 \text{ sinon} \end{cases}.$$

Question 1 et 2 :

Cherchons une solution particulière de (1) :
Méthode de séparation des variables :

Cherchons u sous la forme $u(x,t) = f(t)g(x)$.

On a alors $\begin{cases} u_t(x,t) = f'(t)g(x) \\ u_{xx}(x,t) = f(t)g''(x) \end{cases}$.

En remplaçant ces valeurs dans l'équation (1) on a :

$$f'(t)*g(x) = f(t)*g''(x) \Leftrightarrow \frac{f'(t)}{f(t)} = \frac{g''(x)}{g(x)} = C, \quad C \in \mathbb{R}$$

or $f'(t) = \frac{\partial f}{\partial t}$ et donc $\frac{f'(t)}{f(t)} = C \Leftrightarrow \frac{\partial f}{\partial t * f(t)} = C \Leftrightarrow \frac{\partial f}{f} = A * \partial t$

$$\Leftrightarrow \int \frac{\partial f}{f} = \int A * \partial t \Leftrightarrow \ln(f) = C*t + C_1$$

Ce qui impose :

$$|f| = \exp(-C*t + C_1) \Rightarrow f(t) = K*\exp(C*t), \quad C < 0 .$$

De même

$$g''(x) = C*g(x)$$
$$g(x) = \lambda*\cos(\sqrt{C}*x) + \mu*\sin(\sqrt{C}*x)$$

Et en appliquant les conditions initiales de (3), on a :

$$g'(0) = g'(L) = 0 .$$

Or $g'(x) = -\sqrt{C}\lambda\sin(\sqrt{C}x) + \mu\sqrt{C}\cos(\sqrt{C}x)$, donc :

$g'(0) = \mu\sqrt{C} = 0 \rightarrow \mu = 0$ et $g'(L) = -\sqrt{C} * lambda * \sin(\sqrt{C}L) = 0$

on a alors : $\sqrt{C}L = k*\pi, k \in \mathbb{Z}$, on peut conclure que $C = \frac{k^2*\pi^2}{L^2}$.

On a trouvé une famille de solutions :

$$\boxed{u_k(x,t) = \exp(\frac{-k^2\pi^2 t}{L^2})\cos(\frac{k\pi x}{L})}$$

Posons alors :

$$L^2([0,L]) = \{u : [0,L] \rightarrow \mathbb{R}, \int_0^L u^2 < +\infty\}$$

C'est un espace de Hilbert, un espace fermé complet donc normé.

Posons $v_i = \cos(\frac{i\pi x}{L})$.

Calculons :

$$(v_k, v_l) = \int_{x=0}^{L} \cos(\frac{k\pi x}{L})\cos(\frac{l\pi x}{L})dx$$

$$(v_k, v_l) = \frac{1}{2}\int_{x=0}^{L} [\cos(\frac{(k+l)\pi x}{L}) + \cos(\frac{(k-l)\pi x}{L})]dx$$

On peut alors distinguer 3 cas :

a) $k = l = 0$

$$(v_0, v_0) = \frac{1}{2}\int_{x=0}^{L} 1*dx = L$$

b) $k = l \neq 0$

$$(v_k, v_l) = \frac{L}{2}$$

c) $k \neq l$

$$(v_k, v\,l) = 0$$

On remarque donc que $(v_k)_{k\in\mathbb{N}}$ est une base orthogonale de $L^2([0,L])$. On peut alors décomposer n'importe quel élément de $L^2([0,L])$ sur cette base.

Posons $e_k(x) = \frac{1}{\|v_k\|} * v_k$, or d'après calculs : $\|v_0\| = \sqrt{L}$ et $\|v_k\| = \sqrt{\frac{L}{2}}$.

Soit

$$\begin{cases} w_0(x,t) = \dfrac{1}{\sqrt{L}} \\ w_k(x,t) = \sqrt{\dfrac{2}{L}} \exp \dfrac{-k^2 \pi^2 t}{L^2} \cos\left(\dfrac{K\pi}{L} x\right) \end{cases}.$$

On sait que $u(x,t) = \sum_{k=0}^{+\infty} \tilde{c}_k * w_k(x,t)$, ce qui nos donne pour t = 0 :

$u(x,0) = \sum_{k=0}^{+\infty} \tilde{c}_k e_k(x) = u_0$ car $w_k(x,0) = e_k(x)$.

On a alors, si $\underline{k \neq 0}$

$$\tilde{c}_k = (u_0, e_k) = \int_0^L u_0(x) e_k(x) dx$$

$$= \int_0^L u_0(x) \cos(\frac{k\pi}{L} x) dx = \int_3^5 \cos(\frac{k\pi}{L} x) dx = [\frac{L}{k\pi} \sin(\frac{k\pi}{L} x)]_3^5$$

$$= \frac{L}{k\pi} [\sin(5\frac{k\pi}{L}) - \sin(3\frac{k\pi}{L})] = \frac{8}{k\pi} [\sin(\frac{5}{8} k\pi) - \sin(\frac{3}{8} k\pi)] .$$

Finalement on retrouve, en prenant les notations de l'énoncé :

$$\boxed{c_k = \sqrt{\frac{2}{L}} \tilde{c}_k = \sqrt{\frac{1}{4}} \tilde{c}_k = \frac{1}{2} * [\frac{8}{k\pi} (\sin(\frac{5}{8} k\pi) - \sin(\frac{3}{8} k\pi))] = \frac{4}{k\pi} (\sin(\frac{5}{8} k\pi) - \sin(\frac{3}{8} k\pi))}$$

<u>si k=0 :</u>

$$\tilde{c}_0 = (u_0, e_0) = \int_0^L u_0(x) e_0(x) dx = \frac{1}{\sqrt{L}} \int_0^L u_0(x) dx = \frac{1}{\sqrt{L}} \int_3^5 dx = \frac{2}{\sqrt{L}}$$

ce qui donne :

$$\boxed{c_0 = \sqrt{\frac{2}{L}} \tilde{c}_0 = \frac{2\sqrt{2}}{L}}$$

En résumé on a :

$$\boxed{\begin{cases} c_0 = \dfrac{2\sqrt{2}}{L} \\ c_k = \dfrac{4}{k\pi}\left(\sin\left(\dfrac{5}{8}k\pi\right) - \sin\left(\dfrac{3}{8}k\pi\right)\right) \end{cases}}$$

Question 3 :

Sans calcul, la limite de *u(x,t)* quand *t* tend vers l'infini est 0.

Question 4 :

Posons

$$\begin{cases} d_0 = \sqrt{L}\,\tilde{c}_0 \\ d_k = \sqrt{\dfrac{L}{2}}\,\tilde{c}_k \end{cases}$$

On a avec ces notations :

$$u(x,t) = \frac{d_0}{L} + \frac{2}{L}\sum_{k=1}^{+\infty} d_k \exp\left(\frac{-k^2\pi^2 t}{L^2}\right)\cos\left(\frac{k\pi}{L}x\right) = \frac{d_0}{L} + \frac{2}{L}\sum_{k=1}^{+\infty} \lambda_k(x,t)$$

Or si <u>t > 0 :</u>

$$|d_0| \leqslant L \quad \text{et} \quad |d_k| \leqslant \int_0^L |u_0(x)| \left|\cos\left(\frac{k\pi}{L}x\right)\right| dx \leqslant L$$

Avec cette majoration des coefficients d_k on peut alors majorer la fonction λ_k définie auparavant.

$$|\lambda_k(x,t)| \leqslant e^{\frac{-k^2\pi^2 t}{L^2}}, \forall x \in [0,L]$$

$$\Rightarrow \sup_{x\in[0,L]} |\lambda_k(x,t)| \leqslant e^{\frac{-k^2\pi^2 t}{L}}$$

Or $\sum_{k=1}^{+\infty} e^{\frac{-k^2\pi^2 t}{L^2}}$ est une série absolument convergente (quand t > 0) . Donc on peut dire alors que $\sum_{k=1}^{+\infty} \lambda_k(x,t)$ est normalement convergente pour t fixé. Ce qui donne finalement :

$\sum_{k=1}^{N} \lambda_k(x,t) \to \Lambda(x,t)$ uniformément pour $x \in [0, L]$.

Donc la série est uniformément convergente sur [0, L].

Question 5 :

Prolongeons par parité sur [-L, 0], puis par périodicité sur $\mathbb{R}$ la fonction $u(x,t)$. La série u(x,t) est alors une série de Fourier. On peut alors appliquer le théorème de Dirichlet, annoncé ci-dessous :

Théorème de Dirichlet :

Soit u une application K-périodique, dérivable par morceaux sur [0, K]. Pour tout $x_0 \in \mathbb{R}$, la série de Fourier de u converge et a pour somme :

- $u(x_0)$ si x_0 est un point de continuité de u.
- $\frac{1}{2}\lim_{x \to x_0^+} u(x) + \frac{1}{2}\lim_{x \to x_0^-} u(x)$ si x_0 est un point de discontinuité de u.

Appliquons ce théorème à cet exercice :

On a alors

$$\lim_{t \to 0^+} u(x,t) = \frac{1}{2}\lim_{x \to x_0^+} u(x,t) + \frac{1}{2}\lim_{x \to x_0^-} u(x,t) \Leftrightarrow \lim_{t \to 0^+} u(x,t) = \frac{1}{2}(u_0(x^+) + u_0(x^-))$$

Question 6 :

Nouvelles données :

$$u^N = \sum_{i=0}^{N} c_i \exp\left(\frac{-i^2\pi^2}{L^2}t\right)\cos\left(\frac{i\pi}{L}x\right)$$

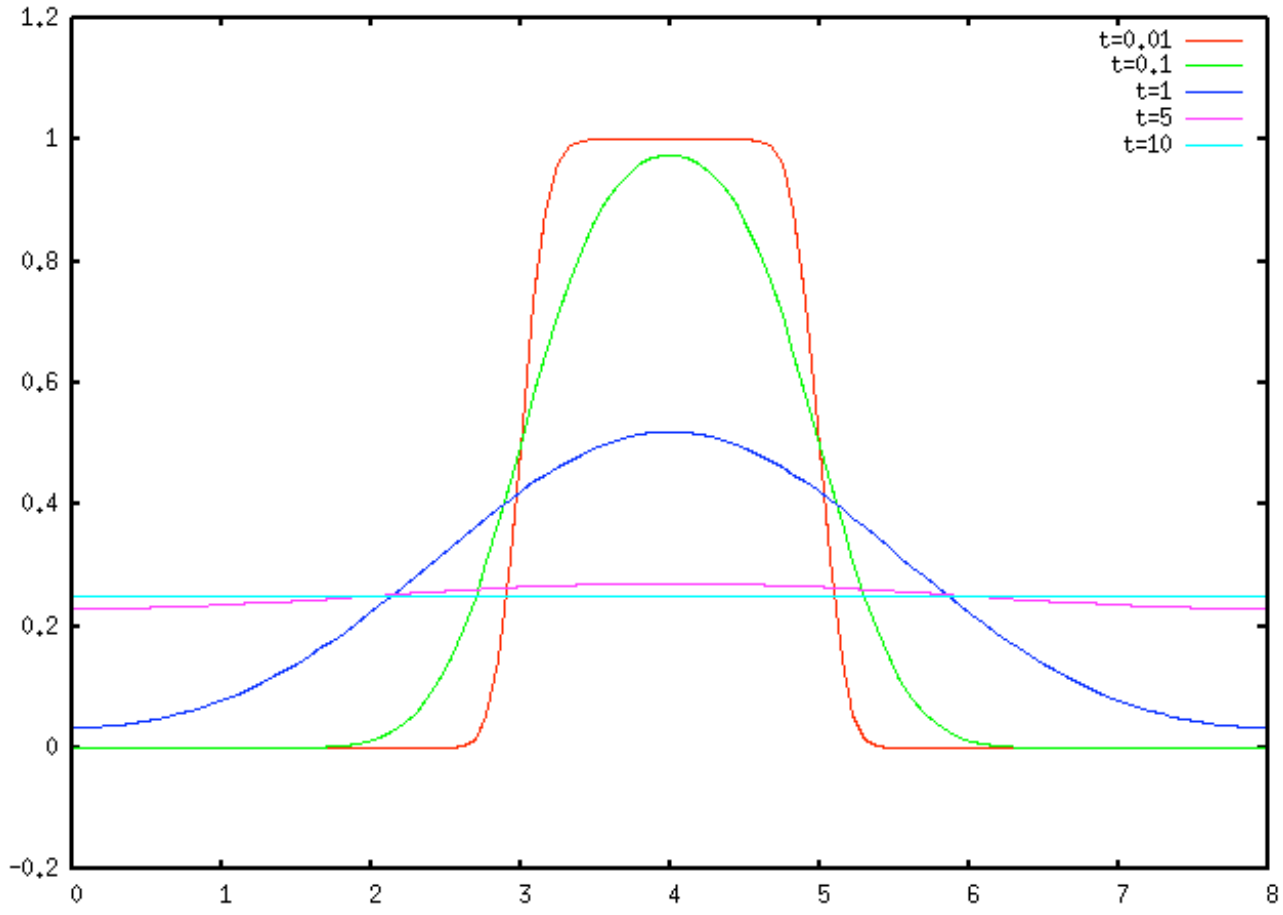
Grâce aux valeurs calculées précédemment de c_i , on peut calculer facilement u_N .

Voici le programme nécessaire pour cela :

```
program sucre
    implicit none
    integer::k,i, M=100
    integer,parameter::nmax=100
    real*8,dimension(0:nmax+1)::u
    real*8::L=8.d0, dx, t=0.01d0, pi, dk, dl, d0
    ! si t->∞ alors u(x,t) tend vers 1/4 car le niveau de sucre va se repartire
equitativement sur [0,8]
    dx=L/(nmax+1) !pas sur l'axe des x = longueur de l'intervalle / nombre de
points
    pi=4.d0*atan(1.d0) !calcul de pi

    d0=2.d0 !initialisation de d0
    u=d0/L !initialisation de u, donc u=u0
    do k=1,M !nombre de fois qu'on somme
      dk=L/k/pi*sin(5*k*pi/L)-L/k/pi*sin(3*k*pi/L) !definition de dk
      do i=0,(nmax+1) !boucle sur le nombre de points
            u(i)=u(i)+d0/L*dk*exp(-k*k*pi*pi*t/L/L)*cos(k*pi*i*dx/L) !on definit
la somme
      end do
    end do

    open(1, file='sucre')
      do i=0,nmax+1
            write(1,*) i*dx,u(i)
      end do
    close(1)
end program
```

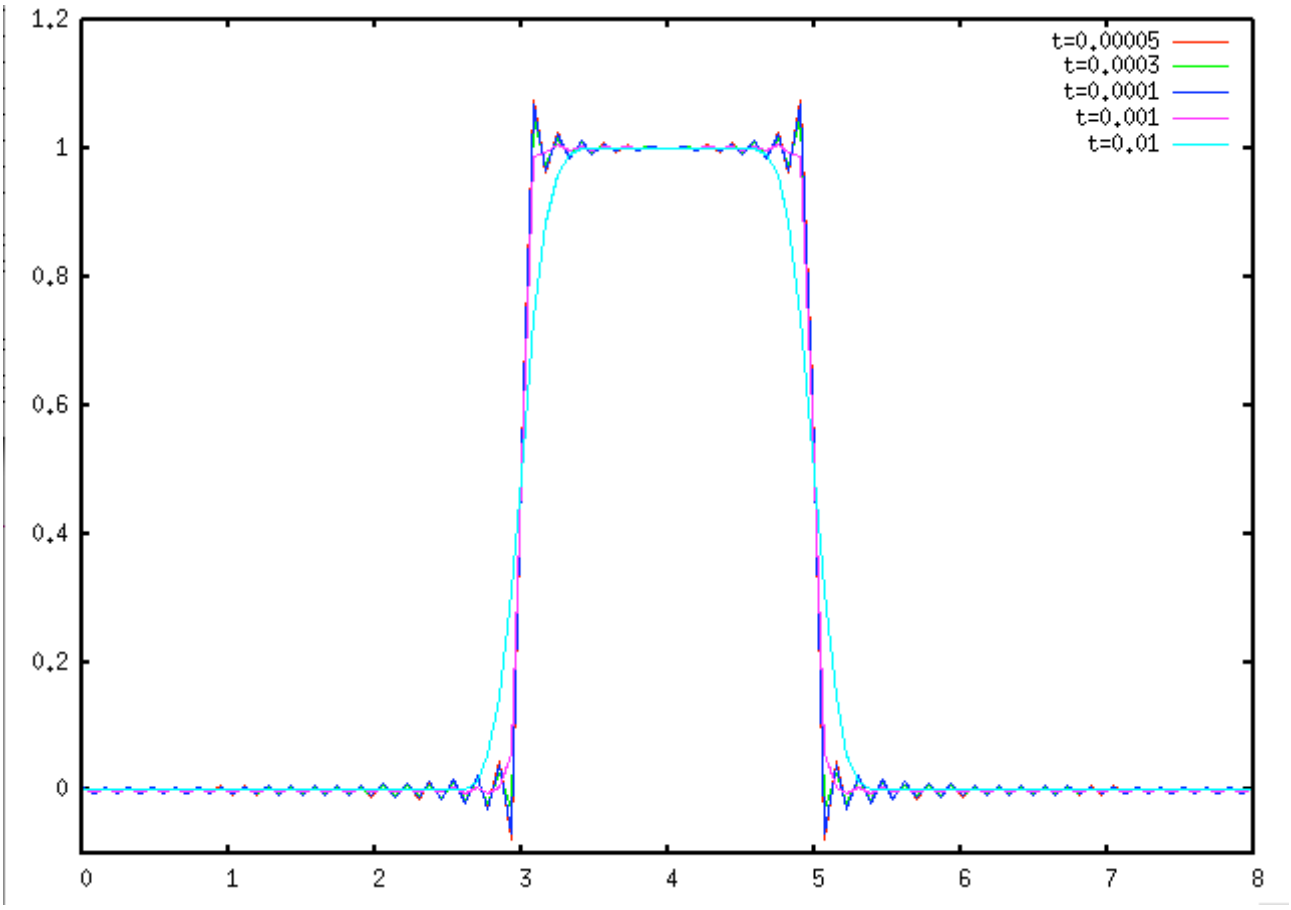
Étudions les résultats pour différentes valeurs de t :

Pour nmax=M=100 :



On remarque que plus t est grand plus la fonction « s'aplatit ». Dans l'analogie d'un cube de sucre dans une tasse de café, les courbes représentent la concentration de sucre aux différents endroits de la tasse. Donc ça semble évident que plus le temps passe, plus le sucre ce dissout dans le café. Après un moment le sucre ce dissout complètement et la concentration en sucre est la même à tout endroit.

De même plus t est petit plus la fonction est proche de $u_0(x)$ tant que t soit dans le domaine de stabilité. Voici des exemples quand t est « trop » petit :



C'est le phénomène de Gibbs.

Remarque :
Faire varier nmax fait incrémenter le nombre de points ce qui donne des courbes précises à chaque intervalle et faire varier M donne plus de précision pour chaque point.

Question 7 :

Résolution du problème par la méthode des différences finies :

Posons $\Delta x = \frac{L}{N-1}$ et donc $x_i = (i-1)\Delta x$ avec $x_1 = 0$ *et* $x_N = L$.

On peut faire de même avec t :

$$\Delta = \frac{T}{P},\ t_i = i\Delta t,\ t_0 = 0 \ \ et \ \ t_P = T \ .$$

On peut alors poser : $u_i^n = u(x_i, t_n)$.

D'après le TP1 on obtient :

$$\frac{\partial^2 u_i^n}{\partial x^2} = \frac{-2\mathrm{u}_i^n + u_{i-1}^n + u_{i+1}^n}{\Delta x^2} \quad \text{et de même} \quad \frac{\partial u_i^n}{\partial t} \simeq \frac{u_i^n - u_i^{n-1}}{\Delta t} \ .$$

D'après le système (1) on a alors :

$$\frac{\partial u_i^n}{\partial t} - \frac{\partial^2 u_i^n}{\partial x^2} = \frac{u_i^n - u_i^{n-1}}{\Delta t} + \frac{-u_{i-1}^n + 2u_i^n - u_{i+1}^n}{\Delta x^2} = 0$$

Que se passe-t-il dans les bornes de l'intervalle ?

D'après le système (1), on sait que : $\frac{\partial u}{\partial x}(0, t_n) = 0$

Or

$$u_0^n \simeq u(-\Delta x, t_n) = u(0, t_n) - \Delta x \underbrace{\frac{\partial u}{\partial x}(0, t_n)}_{=0} + O(\Delta x^2) \simeq u(0, t_n) = u_1^n$$

On peut remplacer alors u_0^n par u_1^n .
En suivant le même raisonnement on peut remplacer u_N par u_{N-1} .

On trouve alors si i=1 :

$$\frac{-u_{i-1}^n + 2u_i^n - u_{i+1}^n}{\Delta x^2} = \frac{-u_0^n + 2u_1^n - u_2^n}{\Delta x^2} = \frac{u_1^n - u_2^n}{\Delta x^2}$$

de même si i=N

$$\frac{-u_{i-1}^n + 2u_i^n - u_{i+1}^n}{\Delta x^2} = \frac{u_{N-1}^n - u_N^n}{\Delta x^2}$$

Introduisons la matrice A et le vecteur U^n :

$$A = \begin{pmatrix} 1 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & 0 & \dots \\ 0 & 0 & \dots & \dots & \dots & 0 \\ 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & \dots & \dots & 0 & -1 & 1 \end{pmatrix} \quad \text{et} \quad U^n = \begin{pmatrix} u_1^n \\ u_2^n \\ \dots \\ \dots \\ u_N^n \end{pmatrix}$$

On obtient alors le système :

$$\frac{U^n - U^{n-1}}{\Delta t} + \frac{1}{\Delta x^2} A U^n = 0$$

Finalement :

$$\boxed{(I + \frac{\Delta t}{\Delta x^2} A) U^n = U^{n-1}}$$

On peut alors faire le même travaille que pour le TP précédent mais avec une différente matrice.

On note que cette matrice est creuse, on peut donc éviter d'écrire la matrice dans un tableau à 2 dimension et on peut écrire la matrice M, sur 3 vecteurs représentant les diagonales.
La matrice est définie en fortran90 comme il suit :

```
  !definition de la matrice de récurrence (I+dt/dx^2*A)
s(1)=-1*dt/dx/dx
d(1)=1*dt/dx/dx+1
do i=2,nmax-1
  d(i)=2*dt/dx/dx+1
  l(i)=-1*dt/dx/dx
  s(i)=-1*dt/dx/dx
end do
l(nmax)=-1*dt/dx/dx
d(nmax)=1*dt/dx/dx+1
```

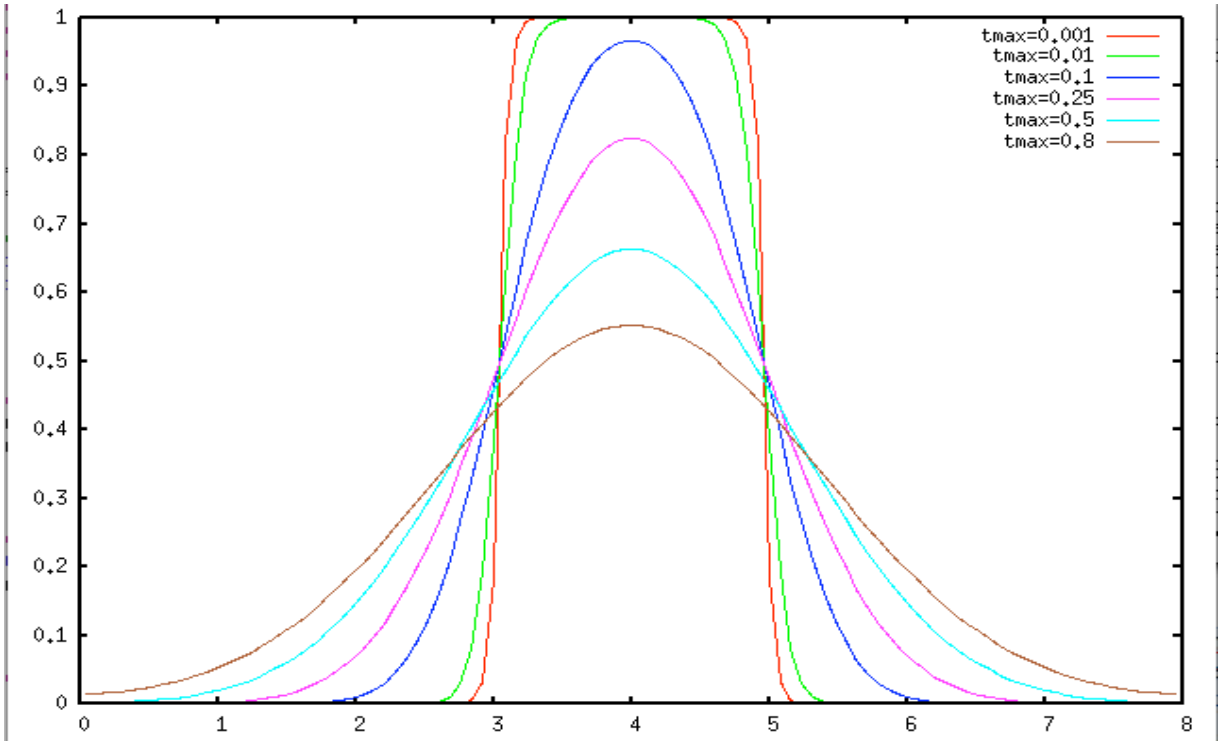
On peut créer une fonction u0 qui nous donnera les valeurs de u_0 dans x donné. De plus, on a déjà écrit la fonction *factolu* qui nous aidera à trouver le terme suivant si on passe la matrice M comme définie ci-dessus.

Or, dans ce problème on cherche U^n et on connait que U^0 . L'algorithme précédent nous permet d'obtenir U^{i+1} grâce à U^i , donc on va devoir itérer N fois l'algorithme pour trouver U^n .

Voici le programme qui nous permettra d'obtenir le résultat :

```
      call factolu(nmax,d, l, s, V, U, 1) !on appelle une première fois factolu
pour factoriser la matrice

    n=0 !compteur
    do while (n*dt.le.tmax) !tant que ne soit pas arrivé a t_n
      call factolu(nmax,d, l, s, V, U, 2) !on fait trouve le resultat et on le
stocke dans V
      U=V !et notre vecteur U pour la prochaine iteration sera le resultat
trouvé à cette étape
      n=n+1 !on incrémente le compteur
    end do
```

Graphes obtenus en affichant le dernier résultat stocké en u par rapport aux x(i) , pour nmax=100



θ-schéma :

Le problème devient :

(7.1) $$\boxed{(I+\theta\frac{\Delta t}{\Delta x^2}A)U^n=(I-(1-\theta)\frac{\Delta t}{\Delta x^2}A)U^{n-1}}$$

On doit alors créer une deuxième matrice (en fortran90, 3 vecteurs) pour pouvoir définir les deux nouvelles matrices, on peut écrire alors :

```
  sa(1)=(1.d0-teta)*dt/dx/dx
s(1)=-dt/dx/dx*teta
da(1)=1.d0-(1.d0-teta)*dt/dx/dx
d(1)=dt/dx/dx*teta+1.d0
do i=2,nmax-1
  da(i)=1.d0-2*(1.d0-teta)*dt/dx/dx
  la(i)=(1.d0-teta)*dt/dx/dx
  sa(i)=(1.d0-teta)*dt/dx/dx
  d(i)=2*dt/dx/dx*teta+1.d0
  l(i)=-1*dt/dx/dx*teta
  s(i)=-1*dt/dx/dx*teta
end do
la(nmax)=1*(1.d0-teta)*dt/dx/dx
da(nmax)=1.d0-1*(1.d0-teta)*dt/dx/dx
l(nmax)=-1*dt/dx/dx*teta
d(nmax)=1*dt/dx/dx*teta+1.d0
```

Les vecteurs d, l, s définissent la matrice à gauche et les vecteurs da, la, sa, la matrice de droite.

Un dernier changement est à faire dans la boucle :

```
  n=0
do while (n*dt.lt.tmax)
!on doit multiplier le vecteur U par la matrice definies par da, la, sa.
  Up(1)=U(1)*da(1)+sa(1)*U(2)
  do i=2,nmax-1
        Up(i)=la(i)*u(i-1)+da(i)*u(i)+sa(i)*u(i+1)
  end do
  Up(nmax)=la(nmax)*u(nmax-1)+da(nmax)*u(nmax)

  call factolu(nmax,d, l, s, V, U, 2)
  U=V
  n=n+1
end do
```

On remarque que si $\theta = 1$ on obtient les mêmes résultats que précédemment.

Question 8 :

On peut écrire le problème (7.1) sous la forme :

$$U^n = K\,U^{n-1} \tag{8.1}$$

avec $K = (I + \delta\,\theta\,A)^{-1}(I - \delta(1-\theta)A)$ et $\delta = \dfrac{\Delta t}{\Delta x^2}$

On sait que A est une matrice symétrique, A multipliée par un coefficient reste symétrique et finalement plus ou moins la matrice la matrice Identité, ça reste une matrice symétrique.

Or, l'inverse d'une matrice symétrique est une matrice symétrique, et de même le produit de deux matrices symétriques est une matrice symétrique. Donc K est symétrique.
Ce qui donne :

$$\|K\|_2 = \rho(K) = \max_i |\mu_i|$$

avec ρ le rayon spectrale de K et les μ_i valeurs propres de K.

Or, les valeurs propres de K sont :

$$\mu_i = \frac{1 - \delta(1-\theta)\lambda_i}{1 + \delta\,\theta\,\lambda_i}$$

avec λ_i valeurs propres de A, qu'on connait :

$$\lambda_i = 4\sin^2\left(\frac{i\pi}{2\mathrm{I}}\right)$$

Or, on a le théorème suivant :

Théorème 8.2 :

Le schéma

$$u^{n+1} = Au^n + \Delta t\, b^{n+1}$$

est stable si A est une matrice symétrique et $\rho(A)$ son rayon spectral tel que :

$\rho(A) \leqslant 1$.

On est dans les hypothèses du théorème (la matrice dite A du théorème est notre matrice K, et le vecteur b est nul), on peut alors l'appliquer.
Pour que le schéma soit stable il nous suffit donc de trouver la condition telle que:

$$\rho(K) \leqslant 1$$

$$\Leftrightarrow \max_i \left| \frac{1-\delta(1-\theta)\lambda_i}{1+\delta\theta\lambda_i} \right| \leqslant 1$$

Or

$$\max_i |\lambda_i| = \max_i \left| 4\sin^2\left(\frac{i\pi}{2I}\right) \right| = 4$$

Ce qui donne :

$$\rho(K) \leqslant 1 \;\Leftrightarrow\; 2\delta(1-2\theta) \leqslant 1$$

Or

$$0 \leqslant \theta < \frac{1}{2} \;\Leftrightarrow\; 1 \geqslant 1-2\theta > 0 \;\Leftrightarrow\; 2\delta \geqslant 2\delta(1-2\theta) > 0$$

On obtient une condition unique :

$2\delta \leqslant 1$

$$\Leftrightarrow\; \delta \leqslant \frac{1}{2} \;\Leftrightarrow\; \frac{\Delta t}{\Delta x^2} \leqslant \frac{1}{2} \;\Leftrightarrow\; \Delta t \leqslant \frac{\Delta x^2}{2}$$

Si $0 \leqslant \theta \leqslant \frac{1}{2}$ le schéma est stable pour $\Delta t \leqslant \frac{\Delta x^2}{2}$.

De même si $\frac{1}{2} \leqslant \theta \leqslant 1$ on peut démontrer que le schéma est stable.

On rappelle que le théorème 8.2 est toujours applicable est dans ce cas on a l'inéquation suivante :

$\rho(K) \leqslant 1 \Leftrightarrow 2\delta(1-2\theta) \leqslant 1$

Or

$\frac{1}{2} \leqslant \theta < 1 \Leftrightarrow 0 \geqslant 1-2\theta > 1 \Leftrightarrow 0 \geqslant 2\delta(1-2\theta) > 2\delta$

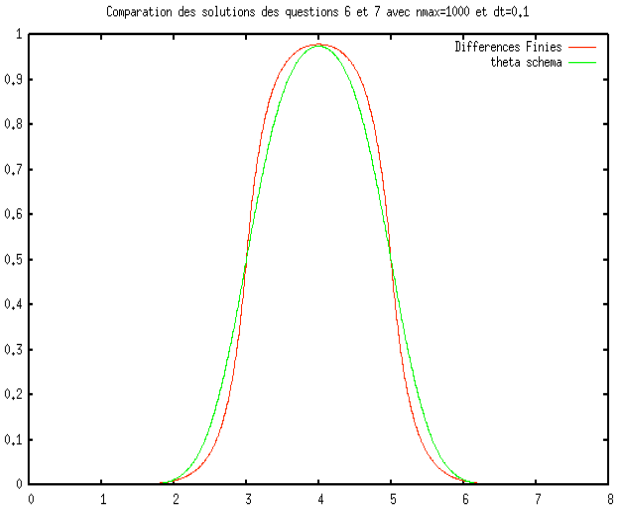
$\Leftrightarrow 1 > 0 \geqslant \rho(K)$

Si $\frac{1}{2} \leqslant \theta \leqslant 1$ le schéma est inconditionnellement stable.

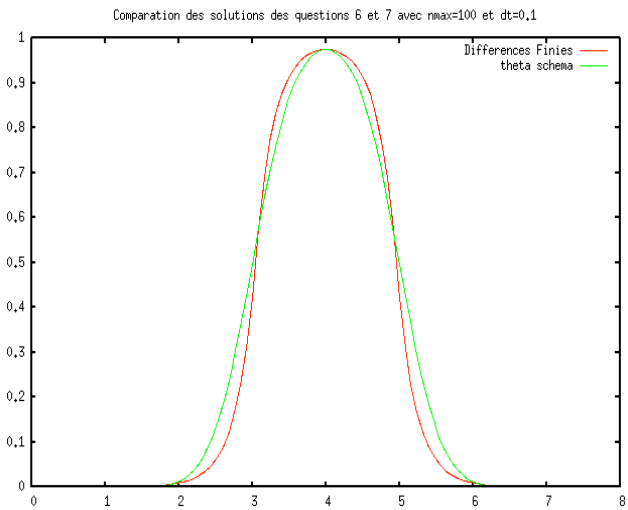
Question 9 :

Prenons $\theta = \frac{2}{3}$ et comparons les résultats obtenus en 6 et en 7 :

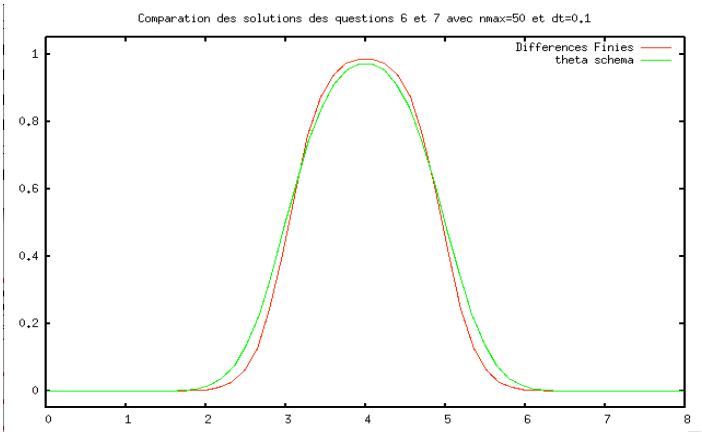
dt=0.1 et nmax=1000



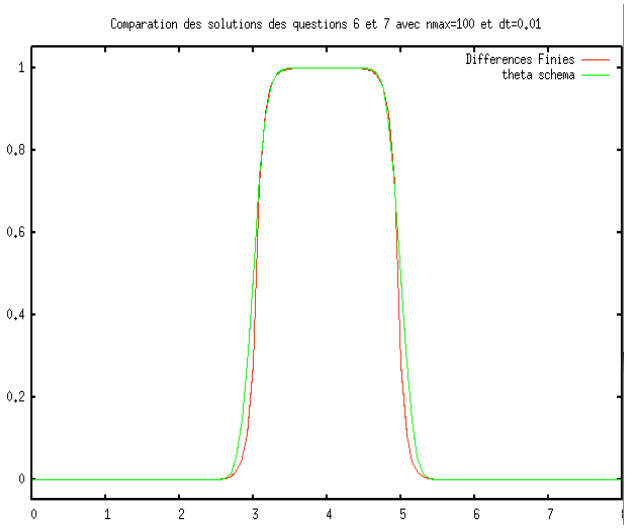
dt=0.1 et nmax=100



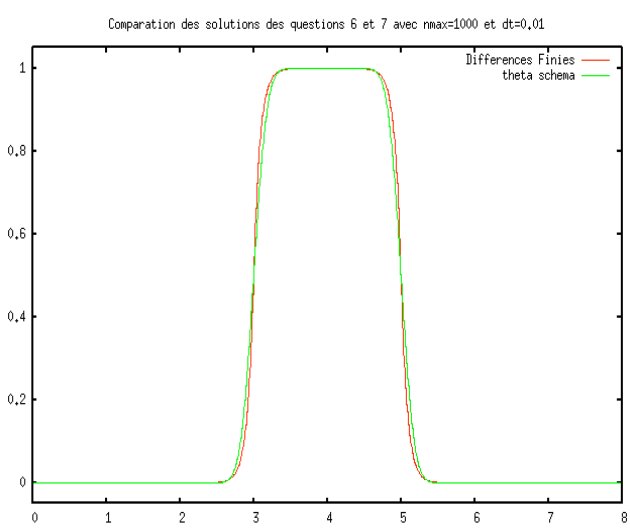
dt=0.1 et nmax=50



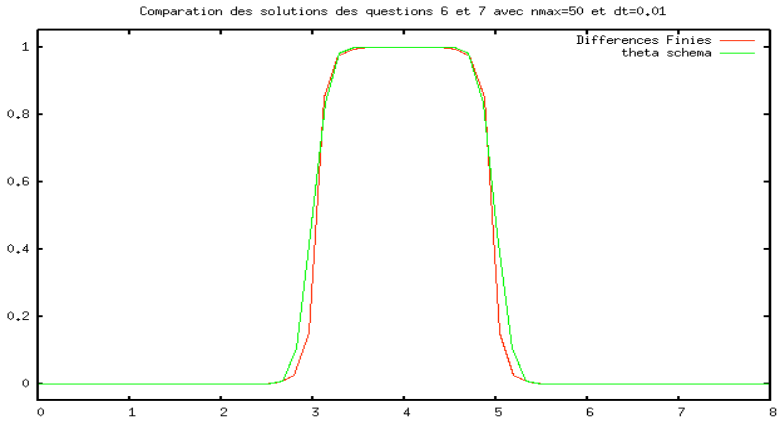
dt=0.01 et nmax=100



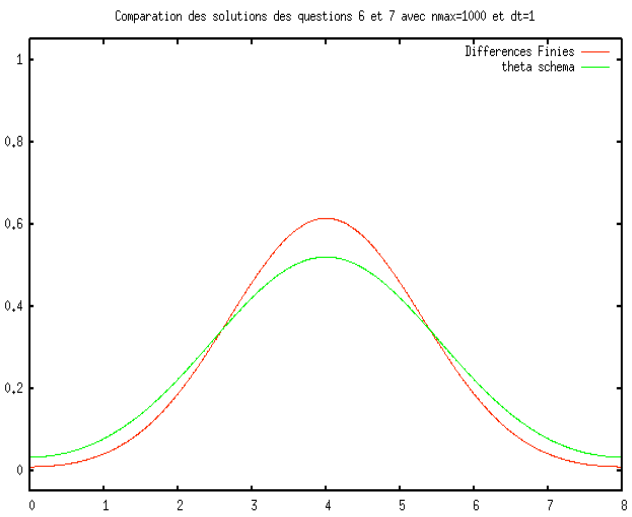
dt=0.01 et nmax=1000



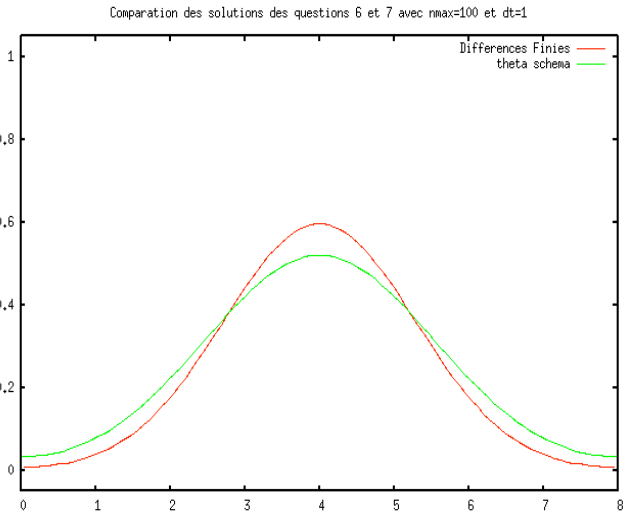
dt=0.01 et nmax=50



dt=1 et nmax=1000



dt=1 et nmax=100



RÉSOLUTION DE L'ÉQUATION DE TRANSPORT

Données du problème :

$$\partial_t u + c\,\partial_x u = 0,\ x \in]0, L[\qquad (0.1)$$
$$u(0,t) = \mathrm{e}^{-t},\ t \in]0, T[\qquad (0.2)$$
$$u(x,0) = 0,\ x \in]0, L[\qquad (0.3)$$

Question 1 :

Notons $t \to \gamma(t) = \begin{pmatrix} x(t) \\ t \end{pmatrix}$ où $x(t)$ est la solution de l'équation (0.1).
La courbe γ dans le plan (x,t) le long de laquelle les solutions de (0.1) sont constantes. De plus la courbe passe par les points (x_0, t_0) .

Or, si k est une constante,

$$u(x(t),t) = k \Leftrightarrow \frac{\partial}{\partial t} u(x(t),t) = 0 \Leftrightarrow \frac{\partial u}{\partial x}(x(t),t)\,x'(t) + \frac{\partial u}{\partial t}(x(t),t) = 0$$

Si $x'(t) = c$ alors γ est la caractéristique de la solution du problème.
Et dans ce cas

$$x(t) = ct + x_0$$

$$u(ct + x_0, t) = k = u(c*0 + x_0, 0) = u(x_0, 0) = u_0(x_0)$$

Les courbes caractéristiques sont des droites parallèles de pente 1/c dans le plan (x,t).

Or $x = t + x_0 \Leftrightarrow x_0 = x - ct$,

La solution de l'équation de transport est donc :

$$\boxed{u(x,t) = u_0(x - ct)} \qquad (1.1)$$

Question 2 :

Pour la discrétisation du problème posons : $\Delta x = \frac{L}{N}$, le pas en espace, Δt le pas en temps, $x_i = i\Delta x$ le maillage en espace et $t_n = n\Delta t$ le maillage en temps, finalement notons $u_i^n \simeq u(x_i, t_n)$.

Par des développements limités d'ordre 3 on a :

$$u(x_{i+1}, t_n) = u(x_i, t_n) + \Delta x \frac{\partial}{\partial x} u(x_i, t_n) + \frac{\Delta x^2}{2} \frac{\partial^2}{\partial x^2} u(x_i, t_n) + O(\Delta x^3) \quad (2.1)$$

$$u(x_{i-1}, t_n) = u(x_i, t_n) - \Delta x \frac{\partial}{\partial x} u(x_i, t_n) + \frac{\Delta x^2}{2} \frac{\partial^2}{\partial x^2} u(x_i, t_n) + O(\Delta x^3) \quad (2.2)$$

Ces deux équations donnent trois façons différentes d'approcher la solution.

On nous demande sur cette question celle qui nous donne le schéma décentré, en plus on sait que *c* est positive alors on a, pour $i = 2,\ldots, nmax$:

$$\frac{u(x_i, t_n) - u(x_{i-1}, t_n)}{\Delta x} = \frac{\partial u}{\partial x}(x_i, t_n) + O(\Delta x) \quad (2.3)$$

et finalement

$$\boxed{\frac{u_i^{n+1} - u_i^n}{\Delta t} + c \frac{u_i^n - u_{i-1}^n}{\Delta x} = 0} \quad (2.4)$$

Pour $i = 1$

$$\frac{u_1^{n+1} - u_1^n}{\Delta t} + c \frac{u_1^n - g(t_n)}{\Delta x} = 0 \quad (2.5)$$

On peut écrire l'équation (2.4) en utilisant les notations introduites au début de la question 2, de la manière suivante :

$$u_i^{n+1} = -c \frac{\Delta t}{\Delta x} (u_i^n - u_{i-1}^n) + u_i^n$$

$$\boxed{\Leftrightarrow U^{n+1} = U^n - c\frac{\Delta t}{\Delta x} A U^n + G^n} \qquad (2.6)$$

avec $A = \begin{pmatrix} 1 & & & \\ -1 & 1 & & \\ & \cdots & \cdots & \\ & & -1 & 1 \end{pmatrix}$ et $G = \begin{pmatrix} c\frac{\Delta t}{\Delta x} g(t_n) \\ 0 \\ \cdots \\ 0 \end{pmatrix}$.

Question 3 :

Montrons la stabilité du schéma dans L^2.

Définissons d'abord la norme suivante :

$$\|u^n\|_{l^2(\mathbb{Z})} = \sqrt{\sum_{i=-\infty}^{+\infty} |u_i^{n2}|}$$

avec cette norme on a alors la condition nécessaire de convergence suivante:

$$\|u^{n+1}\| \leqslant \|u^n\| \quad (l^2 - stabilité)$$

Soit la fonction $\hat{u}^n(\xi), \xi \in [0,1]$, considérons alors les u_i^n comme les coefficients de la série de Fourrier de cette fonction.
On a alors

$$\begin{cases} u_i^n = \int_0^1 \hat{u}^n(\xi) \exp(-2i\pi j\xi) \partial\xi \\ \hat{u}^n(\xi) = \sum_{j=-\infty}^{+\infty} u_j^n \exp(2i\pi j\xi) \end{cases}$$

Où ^ est l'isomorphisme de $l^2(\mathbb{Z}) \to L^2([0,1])$ et $(u_j)_{j\in\mathbb{Z}} \in l^2(\mathbb{Z})$.

On obtient alors l'égalité de Parseval :

$$\sum_j |u_j^{n2}| = \int_0^1 |\hat{u}^n(\xi)^2| \partial\xi$$

Soit T_k l'opérateur de décalage de k, tel que :

$$(T_{ku})_j = u_{j-k}$$

Ce qui donne :

$$(T_k u)\hat{}(\xi) = \sum_j (T_k u)_j \mathrm{e}^{2\mathrm{i}\pi j\xi}$$

$$= \sum_j u_{j-k} \mathrm{e}^{2\mathrm{i}\pi(j-k)\xi} \mathrm{e}^{2\mathrm{i}\pi k\xi}$$

en faisant un changement de variable j'=j-k (puis j=j')

$$(T_k u)\hat{}(\xi) = (\sum_j u_j \mathrm{e}^{2\mathrm{i}\pi j\xi}) \mathrm{e}^{2i\pi k\xi}$$

$$\boxed{(T_k u)\hat{}(\xi) = \hat{u}(\xi)\mathrm{e}^{2\mathrm{i}\pi k\xi}} \qquad (3.1)$$

En notant $u^n=(u_i^n)_{i\in\mathbb{Z}}$, on peut écrire le schéma décentré de la forme suivante :

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} + c\frac{u_i^n-u_{i-1}^n}{\Delta x} = 0 \Leftrightarrow \frac{u^{n+1}-u^n}{\Delta t} + c\frac{u^n-T_1(u^n)}{\Delta x} = 0$$

$$\Rightarrow \left(\frac{u^{n+1}-u^n}{\Delta t} + c\frac{u^n-T_1(u^n)}{\Delta x}\right)\hat{}(\xi) = 0$$

$$\Rightarrow \frac{\hat{u}^{n+1}(\xi)-\hat{u}^n(\xi)}{\Delta t} + c\frac{1-\mathrm{e}^{-2\mathrm{i}\pi\xi}}{\Delta x}\hat{u}^n(\xi) = 0$$

Isolons $\hat{u}^{n+1}$:

$$\hat{u}^{n+1}(\xi)=(1-c\frac{\Delta t}{\Delta x}(1-\mathrm{e}^{-2\mathrm{i}\pi\xi}))\hat{u}^n(\xi) \Rightarrow \left|\hat{u}^{n+1}(\xi)\right|^2 = \left|1+c\frac{\Delta t}{\Delta x}\mathrm{e}^{-2\mathrm{i}\pi\xi} - c\frac{\Delta t}{\Delta x}\right|^2 \left|\hat{u}^n(\xi)\right|^2$$

$$\boxed{\Rightarrow \hat{u}^{n+1}(\xi) = \hat{u}^n(\xi)\lambda(\xi)} \qquad (3.2)$$

où lambda est le coefficient d'amplification de Von Neumann.

Posons $A = \left|1+c\frac{\Delta t}{\Delta x}\mathrm{e}^{-2\mathrm{i}\pi k\xi} - c\frac{\Delta t}{\Delta x}\right|^2$ et $\beta = c\frac{\Delta t}{\Delta x}$.

On a :

$$A = (1-\beta+\beta\cos(\alpha))^2+(\beta\sin(\alpha))^2$$
$$= (1-\beta)^2+2(1-\beta)\beta(\cos\alpha-1+1)+\beta^2(\cos^2\alpha+\sin^2\alpha)$$
$$= (1-\beta+\beta)^2+2(1-\beta)\beta(\cos\alpha+1) \quad \text{(en factorisant)}$$
$$A = 1+2(1-\beta)\beta(1+\cos\alpha) \qquad (3.3)$$

Donc $\lambda(\xi) = A \geqslant 1$ si $\beta \leqslant 1 \Leftrightarrow (1-\beta) \geqslant 0$ car $\beta > 0$ et $1+\cos\alpha \geqslant 0$.

De même si $\beta > 1$ alors $0 \leqslant \lambda(\xi)^2 \leqslant 1$.

Le schéma décentré est l^2 -stable si et seulement si $\beta \leqslant 1 \Leftrightarrow \Delta t \leqslant \frac{\Delta x}{c}$

<u>Démontrons que le schéma est l^∞ - stable :</u>

On rappelle que le schéma est l^∞ - stable si $\|U^{n+1}\|_\infty \leqslant \|U^n\|_\infty$ et que si $(u_i)_{i\in\mathbb{Z}} = u$ alors $\|u\|_{l^\infty(\mathbb{Z})} = \sup_i |u_i|$

Posons $m = \inf_i u_i^n$, $M = \sup_i u_i^n$ et β garde la même définition que précédemment.

On a alors :

$u_i^{n+1} = u_i^n - \beta(u_i^n - u_{i-1}^n)$
$= (1-\beta)u_i^n + \beta u_{i-1}^n \quad \Rightarrow m \leqslant u_i^{n+1} \leqslant M$
$\Rightarrow \|u_i^{n+1}\|_\infty \leqslant \|u_i^n\|_\infty$ si $1 \geqslant \beta \geqslant 0$

Le schéma décentré est l^∞ -stable si et seulement si $\beta \leqslant 1 \Leftrightarrow \Delta t \leqslant \frac{\Delta x}{c}$

Question 4 :

Pour l'implémentation de ce schéma en fortran on va utiliser surtout les équations (2.5) et (2.6).

Les coefficients cfl, dx (Δx), dt (Δt) et L sont définis comme dans les hypothèses du problèmes. De même pour le maillage en espace qui va être stocké dans un vecteur x.

On va créer deux vecteurs de taille n (nmax en fortran) pour stocker les valeurs de u_n et u_{n+1} dans les points x_i . On va initialiser un à 0.

On a besoin également de créer une fonction équivalente à $u_0^n = g(t)$ et d'une fonction qui nous donne les valeurs exactes de u pour la comparaison :

```
function u0t (t)
    implicit none
    real*8:: u0t, t
    u0t=exp(-t)
end function

function uexact (x, t, c)
    implicit none
    real*8::x,t, uexact, c, u0t
    if ((-x+c*t).lt.0) then
      uexact=0.d0
    else
      uexact=u0t(-(x-c*t))
    end if
end function
```

Pour implémenter les équations (2.5) et (2.6) il nous faut une boucle sur le maillage de temps qui remplira unp1 avec les valeurs données par l'algorithme et à la fin on remplacera toujours un par ce nouveau unp1 :

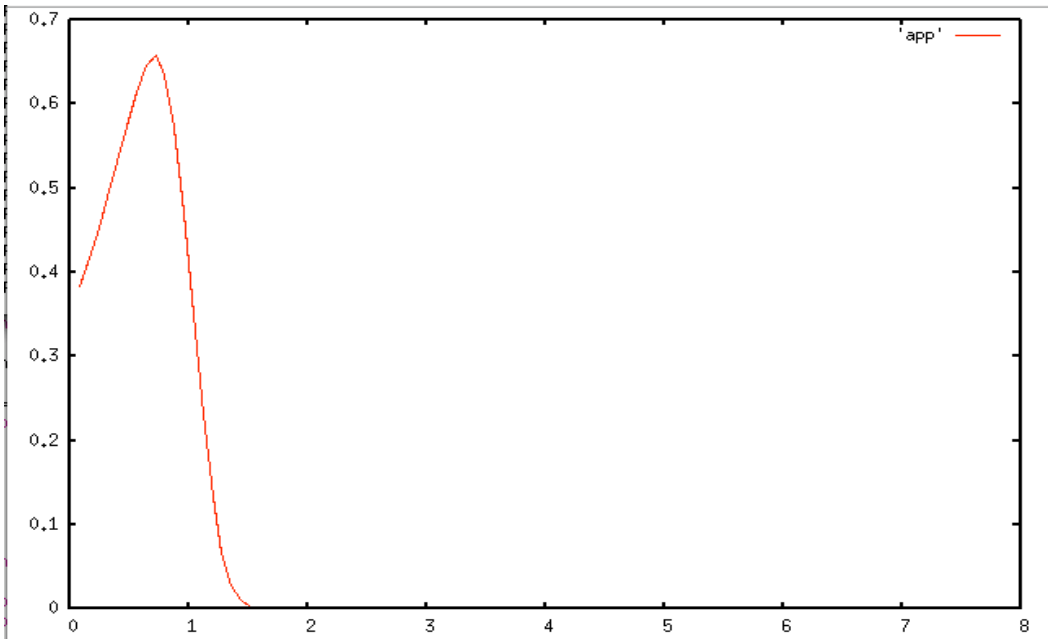
```
    t=dt
    do while (t.lt.tmax)
      t=t+dt
      unp1(1)=un(1)+c*dt/dx*(u0t(t)-un(1))
      do i=2,nmax
            unp1(i)=un(i)-c*dt/dx*(un(i)-un(i-1))
      end do
```

```
    un=unp1
  end do
```

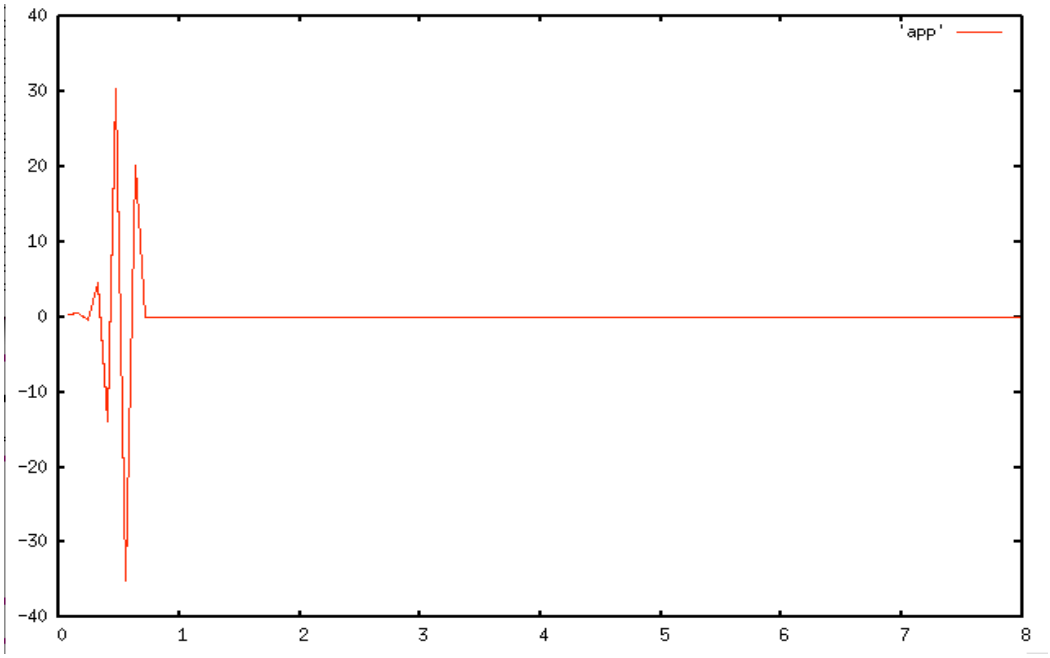
On sait que le schéma est stable si $c\Delta t \leqslant \Delta x$. On a définit $\Delta t = cfl\dfrac{\Delta x}{c}$ donc la condition de stabilité devient : $cfl \leqslant 1$.

Voici alors quelques graphes :

Graphe 4.1 : CFL=0.5

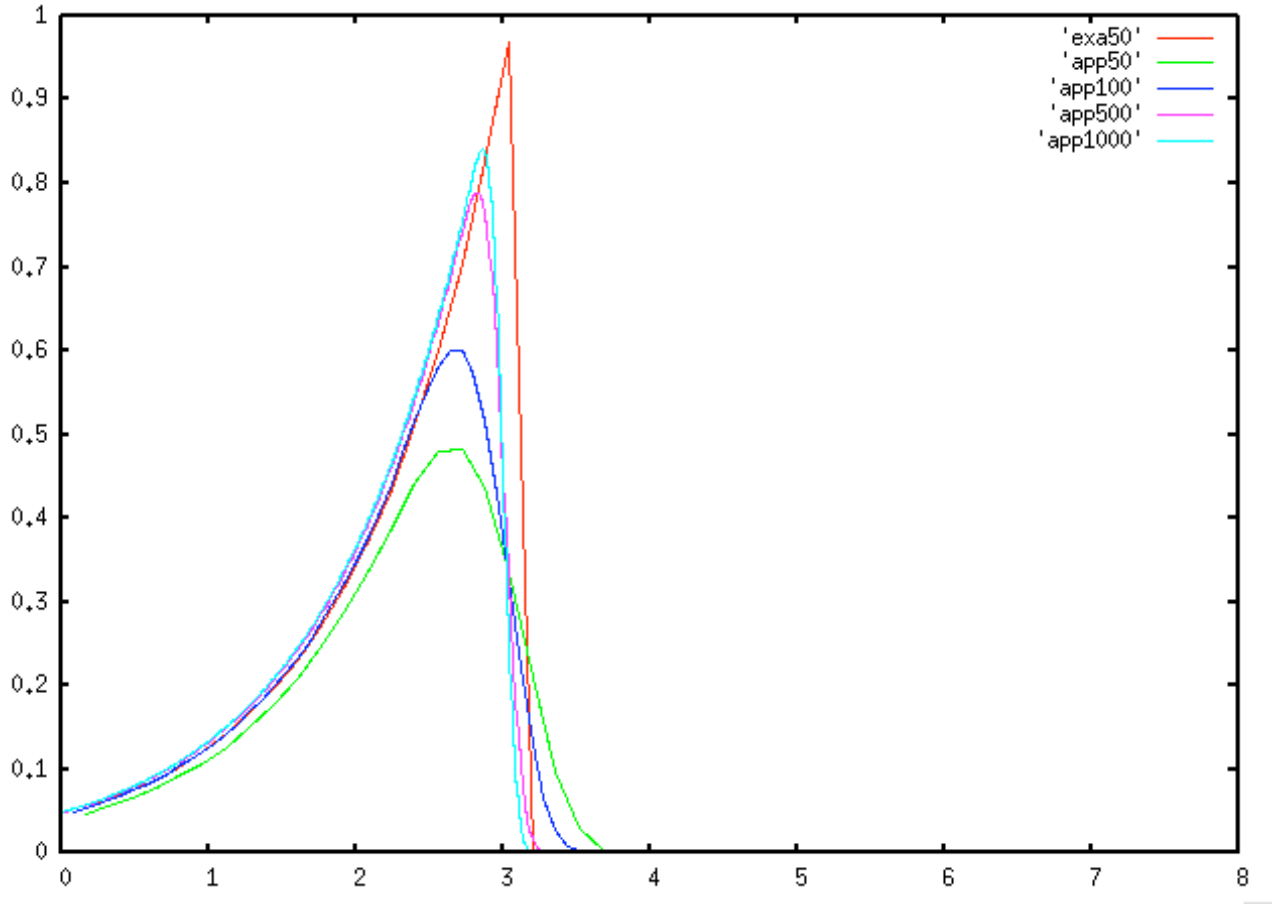


Graphe 4.1 : CFL=1.5



Question 5 :

Pour tmax=3 et cfl=0.8 voici les courbes retrouvées pour différentes valeurs de nmax :



On note que plus n est grand plus les fonctions approchées tendent à la solution exacte.

Question 6 :

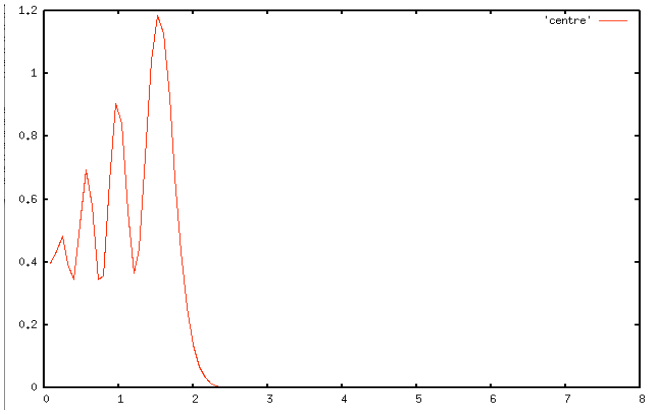
A partir du programme précédent à l'aide de quelques modifications dans la boucle while on a le schéma centré.

Voici les changements effectués dans le programme :

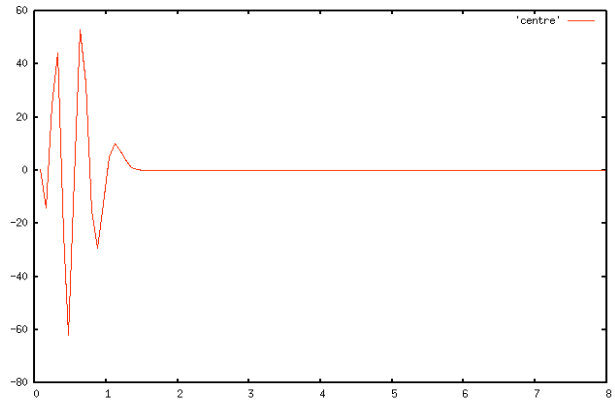
```
t=dt
do while (t.lt.tmax)
  t=t+dt
  unp1(1)=un(1)+c*dt/dx*(u0t(t)-un(1))
```

```
    do i=2,nmax-1
          unp1(i)=un(i)-c*dt/dx*(un(i+1)-un(i-1))
    end do
    unp1(nmax)=un(nmax)-c*dt/dx*(un(nmax))
    un=unp1
  end do
```

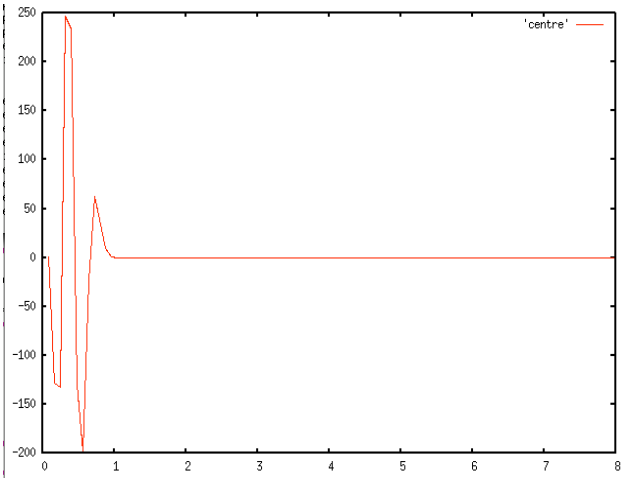
CFL=0.10



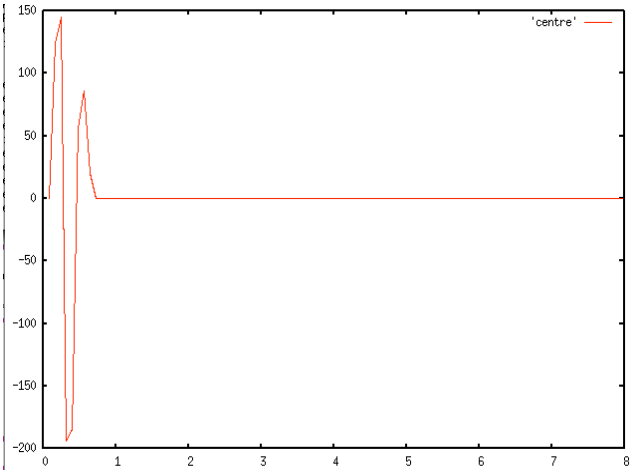
CFL=0.5



CFL=1



CFL=2



On remarque qu'on pourrait mettre CFL à n'importe qu'elle valeur et le schéma sera toujours stable.

Le schéma centré est inconditionnellement instable.

Question 7 :

Le schéma de Lax-Wendroff s'écrit :

$$\frac{u_i^{n+1}-u_i^n}{\Delta t}+c\frac{u_{i+1}^n-u_{i-1}^n}{2\Delta x}-c^2\Delta t\frac{u_{i+1}^n-2u_i^n+u_{i-1}^n}{2\Delta x^2}=0$$

On peut toujours garder le corps du programme du schéma décentré, il suffit de remplacer la boucle principale par :

```
    do while (t.lt.tmax)
      t=t+dt
      unp1(1)=un(1)+c*dt/dx*(u0t(t)-un(2))/2.d0+c*c*dt*dt/dx/dx*(u0t(t)-
2.d0*un(1)+un(2))/2.d0
      do i=2,nmax-1
            unp1(i)=un(i)-c*dt/dx*(un(i+1)-un(i-1))/2.d0
+c*c*dt*dt/dx/dx/2.d0*(un(i-1)-2.d0*un(i)+un(i+1))
      end do
      unp1(nmax)=un(nmax)-c*dt/dx*(-un(nmax-1))/2.d0+c*c*dt*dt/dx/dx*(un(nmax-
1)+2.d0*un(nmax))/2.d0
      un=unp1
```

En suivant le même procédé que pour la question 3, on a :

$$\frac{u_i^{n+1}-u_i^n}{\Delta t}+c\frac{u_{i+1}^n-u_{i-1}^n}{2\Delta x}-c^2\Delta t\frac{u_{i+1}^n-2u_i^n+u_{i-1}^n}{2\Delta x^2}=0$$

$$\Leftrightarrow \frac{u^{n+1}-u^n}{\Delta t}+c\frac{T_{-1}(u^n)-T_1(u^n)}{2\Delta x}-c^2\Delta t\frac{T_{-1}(u^n)-2u^n+T_1(u^n)}{2\Delta x^2}=0$$

$$\Leftrightarrow \frac{\hat{u}^{n+1}(\xi)-\hat{u}^n(\xi)}{\Delta t}+c\frac{e^{-2i\pi\xi}-e^{2i\pi\xi}}{2\Delta x}\hat{u}^n(\xi)-c^2\Delta t\frac{e^{-2i\pi\xi}-2+e^{2i\pi\xi}}{2\Delta x^2}\hat{u}^n(\xi)=0$$

$$\Leftrightarrow \hat{u}^{n+1}(\xi)=\hat{u}^n(\xi)\left(1-c\frac{\Delta t}{2\Delta x}(e^{-2i\pi\xi}-e^{2i\pi\xi})+c^2\frac{\Delta t^2}{2\Delta x^2}(e^{-2i\pi\xi}+e^{2i\pi\xi}-2)\right)$$

$$\Leftrightarrow \left|\hat{u}^{n+1}(\xi)\right|_2=\left|\hat{u}^n(\xi)\right|_2\left|1-c\frac{\Delta t}{2\Delta x}(2i\sin(2\pi\xi))+c^2\frac{\Delta t^2}{2\Delta x^2}(2i\sin(2\pi\xi)-2)\right|_2$$

Posons

$$A = \left| 1 - c\frac{\Delta t}{\Delta x}(i\sin(2\pi\xi)) + c^2\frac{\Delta t^2}{\Delta x^2}(i\sin(2\pi\xi)-1) \right|_2$$

A est donc le coefficient d'amplification de Von Neumann
et posons

$$\beta = c\frac{\Delta t}{\Delta x}$$

On a alors,

$$A = \left|1 - \beta(i\sin(2\pi\xi)) + \beta^2(i\sin(2\pi\xi)-1)\right|_2$$

$$\Leftrightarrow A = \left|1 - \beta(i\sin(2\pi\xi)) + \beta^2 i\sin(2\pi\xi)-\beta^2\right|_2$$

$$\Leftrightarrow A = (1 - \beta^2)^2 + \beta^2\sin^2(2\pi\xi)(\beta^2-1)^2$$

$$\Leftrightarrow A = 1-2\beta^2+\beta^4 + \beta^2\sin^2(2\pi\xi)(\beta^2-1)^2$$

$$\boxed{\Leftrightarrow A = 1 + \beta^2(-2+\beta^2 + \sin^2(2\pi\xi)(\beta^2-1)^2)} \qquad (7.1)$$

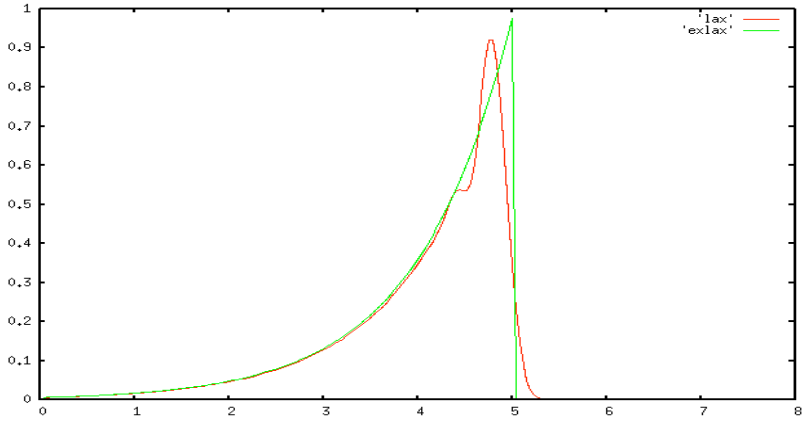
On remarque alors que si $\beta \leqslant 1$ alors $A \geqslant 1$.

Le schéma de Lax-Wendroff est l^2 -stable si $c\frac{\Delta t}{\Delta x} \leqslant 1 \Leftrightarrow \Delta t \leqslant \frac{\Delta x}{c}$

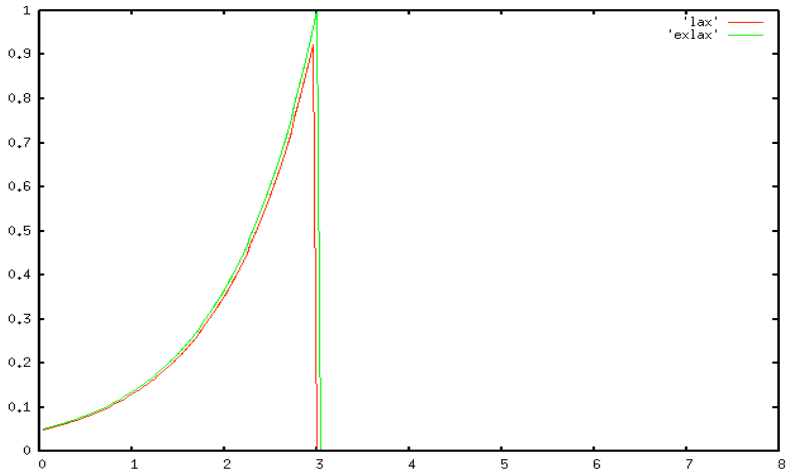
De même si $\beta > 1$ alors $0 \leqslant A \leqslant 1$ .Donc le schéma n'est pas stable.

Démontrons ceci à l'aide de notre programme :

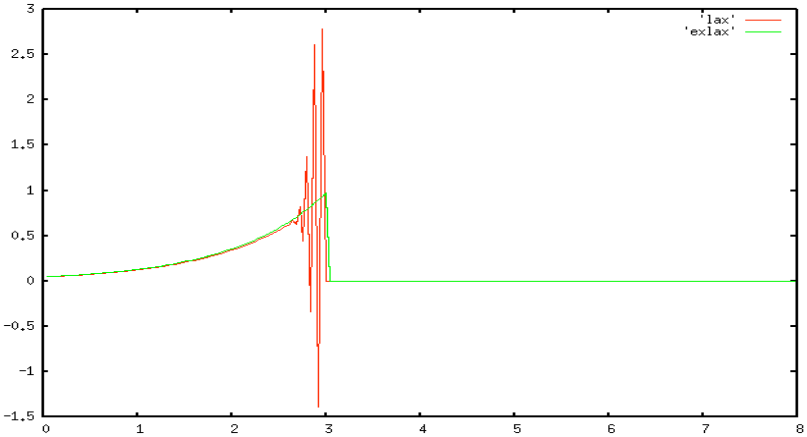
Graphe 7.2 : cfl=0.5



Graphe 7.3 : cfl=1



Graphe 7.4 : cfl=1.01



On voit que dès que cfl est supérieur à 1 on obtient un graphe instable. On remarque aussi que dans le cas contraire on peut voir légèrement le phénomène de Gibbs.

RÉSOLUTION DE L'ÉQUATION DES ONDES

Données du problème :

$$\begin{cases} \partial_{tt} u - c^2 \partial_{xx} u = 0, \quad x \in [-2,2], \quad t \in [0,T] \\ u(x,0) = u_0(x) \\ \partial_t u(x,0) = u_1(x) \\ u(-2,t) = u(2,t) = 0 \end{cases} \tag{1}$$

avec c une constante positive, et u_0 et u_1 a support dans $[-1,1]$

Question 1 :

Calculons la solution exacte.
On va effectuer le changement de variable suivant :

$$\begin{cases} \xi = x - ct \\ \eta = x + ct \end{cases}$$

Soit $\zeta(x,t) = \begin{pmatrix} \xi \\ \eta \end{pmatrix}$ on a alors

$$\zeta' = \begin{pmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial t} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial t} \end{pmatrix} = \begin{pmatrix} 1 & -c \\ 1 & c \end{pmatrix}$$

Posons $v(\xi,\eta) = u(x,t)$.

Cela nous permet de calculer les dérivées premières :

$$\frac{\partial u}{\partial t} = \frac{\partial v}{\partial \xi}\frac{\partial \xi}{\partial t} + \frac{\partial v}{\partial \eta}\frac{\partial \eta}{\partial t}$$
$$\frac{\partial u}{\partial t} = -c\frac{\partial v}{\partial \xi} + c\frac{\partial v}{\partial \eta}$$

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial v}{\partial \eta}\frac{\partial \eta}{\partial x}$$
$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial \xi} + \frac{\partial v}{\partial \eta}$$

On peut alors calculer les dérivées secondes :

$$\begin{aligned}\frac{\partial}{\partial t}\left(\frac{\partial u}{\partial t}\right) &= \frac{\partial}{\partial t}\left(-c\frac{\partial v}{\partial \xi}+c\frac{\partial v}{\partial \eta}\right)\\ &= \frac{\partial}{\partial \xi}\left(-c\frac{\partial v}{\partial \xi}+c\frac{\partial v}{\partial \eta}\right)\times(-c)+\frac{\partial}{\partial \eta}\left(-c\frac{\partial v}{\partial \xi}+c\frac{\partial v}{\partial \eta}\right)\times c\\ &= c^2\frac{\partial^2 v}{\partial \xi^2}-c^2\frac{\partial^2 v}{\partial \xi\,\partial \eta}-c^2\frac{\partial^2 v}{\partial \eta\,\partial \xi}+c^2\frac{\partial^2 v}{\partial \eta^2} \qquad (1.1)\end{aligned}$$

et de même :

$$\begin{aligned}\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right) &= \frac{\partial}{\partial x}\left(\frac{\partial v}{\partial \xi}+\frac{\partial v}{\partial \eta}\right)\\ &= \frac{\partial}{\partial \xi}\left(\frac{\partial v}{\partial \xi}+\frac{\partial v}{\partial \eta}\right)+\frac{\partial}{\partial \eta}\left(\frac{\partial v}{\partial \xi}+\frac{\partial v}{\partial \eta}\right)\\ &= \frac{\partial^2 v}{\partial \xi^2}+2\frac{\partial^2 v}{\partial \xi\,\partial \eta}+\frac{\partial^2 v}{\partial \eta^2} \qquad (1.2)\end{aligned}$$

On peut alors calculer la première différence des hypothèses :

$$\begin{aligned}\frac{\partial^2 u}{\partial t^2}-c^2\frac{\partial^2 u}{\partial x^2} &= c^2\frac{\partial^2 v}{\partial \xi^2}-\boldsymbol{c^2\frac{\partial^2 v}{\partial \xi\,\partial \eta}}-\boldsymbol{c^2\frac{\partial^2 v}{\partial \xi\,\partial \eta}}+c^2\frac{\partial^2 v}{\partial \eta^2}-c^2\frac{\partial^2 v}{\partial \xi^2}-\boldsymbol{c^2\frac{\partial^2 v}{\partial \xi\,\partial \eta}}+c^2\frac{\partial^2 v}{\partial \eta^2}\\ &= -4c^2\frac{\partial^2 v}{\partial \xi\,\partial \eta} = 0\end{aligned}$$

Car on peut remarquer que les seuls termes qui sont en gras vont rester, et de plus, d'après l'hypothèse on sait que cette différence est nulle.

Posons :

$$v = F(\xi) + G(\eta) \quad \text{et} \quad f(\xi) = F'(\xi) = \frac{\partial v}{\partial \xi}$$

Avec ces notations :

$$\boxed{u(x,t) = F(x-ct) + G(x+ct)} \qquad (1.3)$$

On a trouvé alors une solution générale de l'équation des ondes.

On peut utiliser à présent les conditions initiales :

$$u(x,0) = u_0(x) = F(x)+G(x) \qquad (1.4)$$

De même, on a :

$$\frac{\partial u}{\partial t}u(x,0) = u_1(x) = -cF'(x)+cG'(x)$$

On en déduit

$$\int_{-\infty}^{x} u_1(s)\,ds = -cF(x)+cG(x) \qquad (1.5)$$

en supposant que F et G sont nuls à l'infini.

On peut alors calculer :

$$c\times(1.3)-(1.4) = 2\mathrm{cF}(x) = cu_0(x)+\int_{x}^{-\infty} u_1(s)\,ds \qquad (1.6)$$

$$c\times(1.3)+(1.4) = 2\mathrm{cG}(x) = cu_0(x)+\int_{-\infty}^{x} u_1(s)\,ds \qquad (1.7)$$

De (1.6) on en déduit :

$$F(x) = \frac{1}{2}u_0(x)+\frac{1}{2\mathrm{c}}\int_{x}^{-\infty} u_1(s)\,ds$$

et de (1.7) :

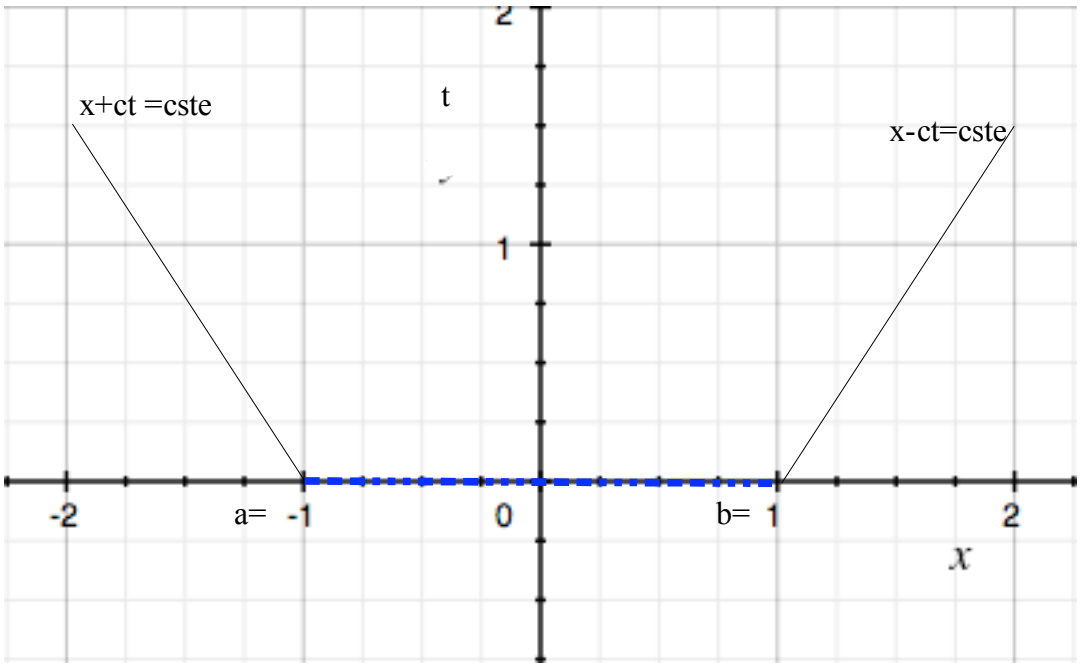
$$G(x) = \frac{1}{2}u_0(x)+\frac{1}{2\mathrm{c}}\int_{-\infty}^{x} u_1(s)\,ds$$

En remplaçant ces dernières valeurs dans l'équation (1.3) :

$$\boxed{u(x,t) = \frac{1}{2}(u_0(x-ct)+u_0(x+ct)) + \frac{1}{2\mathrm{c}}\int_{x-ct}^{x+ct} u_1(s)\,ds} \qquad (1.8)$$

On obtient alors la solution exacte de l'équation des ondes.

Remarque :



La partie bleu représente le support de u_0 et de u_1 .
Or

$$\text{Support}(u_0) \subset [a, b] \quad \text{et} \quad \text{Support}(u_1) \subset [a, b]$$

Donc

cT=(2-b)=1 ou cT=abs(-2-a)=1 donc cT=1.

Or $t \leqslant T \leqslant \frac{1}{c}$.
Donc le résultat est juste sous cette condition.

Pour $t > \frac{1}{c}$ le résultat est généralement faux.

Question 2 :

Posons $\omega = (\partial_t u, \partial_x u)^T = \begin{pmatrix} \partial_t u \\ \partial_x u \end{pmatrix}$ et $A = \begin{pmatrix} 0 & -c \\ -1 & 0 \end{pmatrix}$.

On peut calculer alors :

$$(2.1)\quad \partial_t \omega + A\,\partial_x \omega = \partial_t \begin{pmatrix} \partial_t u \\ \partial_x u \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ -1 & 1 \end{pmatrix} \partial_x \begin{pmatrix} \partial_t u \\ \partial_x u \end{pmatrix}$$
$$= \begin{pmatrix} \partial_{tt} u \\ \partial_t \partial_x u \end{pmatrix} + \begin{pmatrix} 0 & -c \\ -1 & 0 \end{pmatrix} \begin{pmatrix} \partial_x \partial_t u \\ \partial_{xx} u \end{pmatrix}$$
$$= \begin{pmatrix} \partial_{tt} u \\ \partial_t \partial_x u \end{pmatrix} + \begin{pmatrix} -c\,\partial_{xx} u \\ -\partial_x \partial_t u \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

On a trouvé une nouvelle écriture du problème.

Question 3 :

Nouvelles définitions :

$$x_i = -2 + i\,\Delta x,\ i = 0 \ldots N+1,\ \Delta x = \frac{4}{N+1}$$

Le nouveau modèle, trouvé à la question 2, pour le problème ressemble à une équation de transport.

Diagonalisons la matrice A :

Il faut tout d'abord chercher les valeurs propres de la matrice A.

$$\begin{vmatrix} -\lambda & c^2 \\ 1 & -\lambda \end{vmatrix} = \lambda^2 - c^2$$

Ce qui nous donne : $\lambda_1 = -c,\ \lambda_2 = c$.

On peut maintenant calculer les vecteurs propres de A.

Cherchons tout d'abord le vecteur propre sous la forme : $e_1 = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ tel que

$$Ae_1 = -c\,e_1$$
$$\Leftrightarrow \begin{pmatrix} c & c^2 \\ 1 & c \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$
$$\Leftrightarrow \alpha = c,\ \beta = -1$$

Ce qui nous donne $e_1 = \begin{pmatrix} c \\ -1 \end{pmatrix}$. De même on trouve $e_2 = \begin{pmatrix} c \\ 1 \end{pmatrix}$.

On peut écrire la matrice de passage : $P = (e_1, e_2) = \begin{pmatrix} c & c \\ -1 & 1 \end{pmatrix}$

et son inverse $P^{-1} = \frac{1}{2c} \begin{pmatrix} 1 & -c \\ 1 & c \end{pmatrix}$.

Posons $y = P^{-1}\omega$, on remplaçant dans (2.1) on obtient :

$$\frac{\partial}{\partial t}(P\,y) + AP\frac{\partial}{\partial x}(y) = 0$$

Donc :

$$P^{-1}\left(\frac{\partial}{\partial t}(P\,y) + AP\frac{\partial}{\partial x}(y)\right) = 0$$
$$\Leftrightarrow \frac{\partial}{\partial t}y + P^{-1}AP\frac{\partial}{\partial x}y = 0$$

On sait que $P^{-1}AP$ est une matrice diagonale et on connait ses valeurs sur la diagonal. Posons y_1, y_2 les deux valeurs de y. On a le système :

$$\begin{cases} \frac{\partial y_1}{\partial t} - c\frac{\partial y_1}{\partial x} = 0 \\ \frac{\partial y_2}{\partial t} + c\frac{\partial y_2}{\partial x} = 0 \end{cases}$$

Ces deux équations sont des équations de transport découpées.

Utilisons le chemin décentré pour trouver le schéma :
On rappelle le problème (2.1)

$$\partial_t\omega + A\,\partial_x\omega = 0$$

Par discrétisation on obtient :

$$\frac{\omega_i^{n+1} - \omega_i^n}{\Delta t} + \max(\lambda,\,0)\frac{\omega_i^n - \omega_{i-1}^n}{\Delta x} + \min(\lambda,\,0)\frac{\omega_{i+1}^n - \omega_i^n}{\Delta x} = 0$$

posons $\lambda^+ = \max(\lambda,\,0),\ \lambda^- = \min(\lambda,\,0)$, on peut alors écrire pour j=1 ou j=2 :

$$\frac{y_{j,i}^{n+1} - y_{j,i}^n}{\Delta t} + \lambda_j^+\frac{y_{j,i}^n - y_{j,i-1}^n}{\Delta x} + \lambda_j^-\frac{y_{j,i+1}^n - y_{j,i}^n}{\Delta x} = 0$$

On sait que D (la matrice diagonalisée de A) s'écrit :

$$D = \begin{pmatrix} -c & 0 \\ 0 & c \end{pmatrix}$$

et par définition :

$$D^+ = \begin{pmatrix} 0 & 0 \\ 0 & c \end{pmatrix} \quad \text{et} \quad D^- = \begin{pmatrix} -c & 0 \\ 0 & 0 \end{pmatrix}$$

On peut écrire

$$\frac{y_i^{n+1} - y_i^n}{\Delta t} + D^+ \frac{y_i^n - y_{i-1}^n}{\Delta x} + D^- \frac{y_{i+1}^n - y_i^n}{\Delta x} = 0 \quad (3.1)$$

On sait que $\omega = Py$, multiplions (3.1) par P (pour retrouver des ω_i)

$$\frac{\omega_i^{n+1} - \omega_i^n}{\Delta t} + PD^+ P^{-1} \frac{\omega_i^n - \omega_{i-1}^n}{\Delta x} + P D^- P^{-1} \frac{\omega_{i+1}^n - \omega_i^n}{\Delta x} = 0 \quad (3.2)$$

or $PD^+ P^{-1} = A^+$ et $PD^- P^{-1} = A^-$.

On rappelle $\omega = \begin{pmatrix} \frac{\partial u}{\partial t} \\ \frac{\partial u}{\partial x} \end{pmatrix}$, et donc $\omega_i^0 = \begin{pmatrix} u_1(x_i) \\ \frac{u_0(x_{i+1}) - u_0(x_{i-1})}{2\Delta x} \end{pmatrix}$.

Finalement par la méthode des trapèzes on a la formule :

$$\boxed{u(x_i,\ t_n) = \int_2^{x_i} \frac{\partial u}{\partial x}(x, t_n)\, dx \simeq \sum_{j \leqslant i} \omega_{2,j}^n \Delta x} \quad (3.3)$$

<u>Description de la programmation :</u>

On remarque que la formule (1.5) a des termes qui dépendent de la formule en (1.4).

En effet notre formule (1.4) peut s'écrire :

$$\omega_i^{n+1} = -(A^+ \frac{\omega_i^n - \omega_{i-1}^n}{\Delta x} + A^- \frac{\omega_{i+1}^n - \omega_i^n}{\Delta x}) * \Delta t + \omega_i^n \quad (3.4)$$

On a besoin de connaître ω à deux instants on peut créer alors deux vecteurs de taille (1:2, 0:nmax+1) qui stockeront ω^n et ω^{n+1} .
Et on connait une valeur initiale, pour un premier tableau, donnée par :

$$\omega_i^0 = \begin{pmatrix} u_1(x_i) \\ \dfrac{u_0(x_{i+1}) - u_0(x_{i-1})}{2\Delta x} \end{pmatrix}$$

Ceci donne en fortran :

```
      real*8,dimension(1:2,0:nmax+1)::wn,wnp1
      !initialisation de wn = w0
      wn(2,0)=0.d0 !initialisation dans la limite gauche
    wn(1,0)=u1(x(0))
    do i=1, nmax  !initialisation de w0 dans des points non extremats
      wn(1,i)=u1(x(i))
      wn(2,i)=(u0(x(i+1))-u0(x(i-1)))/2.d0/dx
    end do
    wn(2, nmax+1)=0.d0  !initialisation dans la limite droite
    wn(1,nmax+1)=u1(x(nmax+1))
```

Revenons à la formule (3.4), sachant que

$$A^+ = PD^+P^{-1} = \frac{1}{2c}\begin{pmatrix} c & c \\ -1 & 1 \end{pmatrix}\begin{pmatrix} 0 & 0 \\ 0 & c \end{pmatrix}\begin{pmatrix} 1 & -c \\ 1 & c \end{pmatrix} = \frac{1}{2}\begin{pmatrix} c & c^2 \\ 1 & c \end{pmatrix}$$

et $$A^- = PD^-P^{-1} = \frac{1}{2c}\begin{pmatrix} c & c \\ -1 & 1 \end{pmatrix}\begin{pmatrix} -c & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} 1 & -c \\ 1 & c \end{pmatrix} = \frac{1}{2}\begin{pmatrix} -c & c^2 \\ 1 & -c \end{pmatrix} ,$$

on peut calculer un algorithme qui nous donner ω_{n+1} .

En fortran :

```
      t=dt
      do while (t.lt.tmax)
      t=t+dt
      do i=1,nmax
            !Calcul de wnp1(1, :), on a calculé le produit matriciel et on
obtient les formules ci dessous, tmp1 et tmp2 sont des variables temporelles.
```

```
        tmp1=c*(wn(1,i)-wn(1,i-1))+c*c*(wn(2,i)-wn(2,i-1))
        tmp2=-c*(wn(1,i+1)-wn(1,i))+c*c*(wn(2,i+1)-wn(2,i))
        wnp1(1,i)=-(tmp1+tmp2)/2.d0/dx*dt+wn(1,i)
        !De meme on calcul wnp(2,:)
        tmp1=wn(1,i)-wn(1,i-1)+c*(wn(2,i)-wn(2,i-1))
        tmp2=wn(1,i+1)-wn(1,i)-c*(wn(2,i+1)-wn(2,i))
        wnp1(2,i)=-(tmp1+tmp2)/2.d0/dx*dt+wn(2,i)
  end do
  wn=wnp1 !une fois wnp1 on peut recommencer avec cette nouvelle matrice
end do
```

Tout ce qui nous reste est l'implantation de (3.3), pour trouver le résultat final. Cette formule se traduit en fortran par :

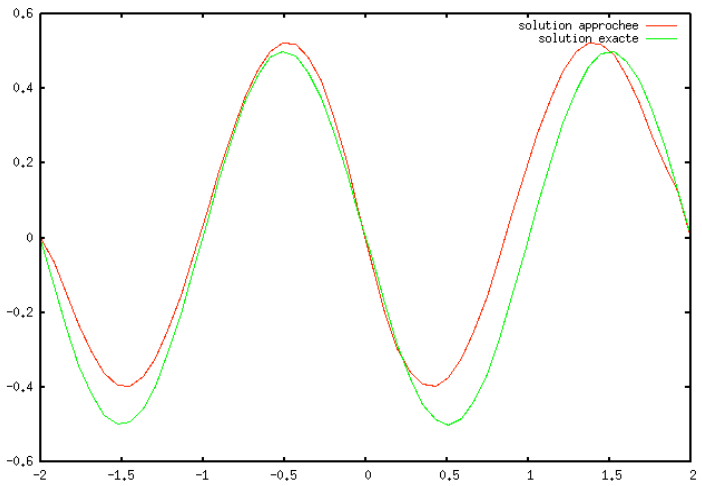
```
  do i=1, nmax
  do j=1,i
        un(i)=un(i)+wn(2, j)*dx
  end do
end do
```

Remarque : Pour faciliter les tests du programme en prend u1 nulle, on évite alors de calculer son intégrale.

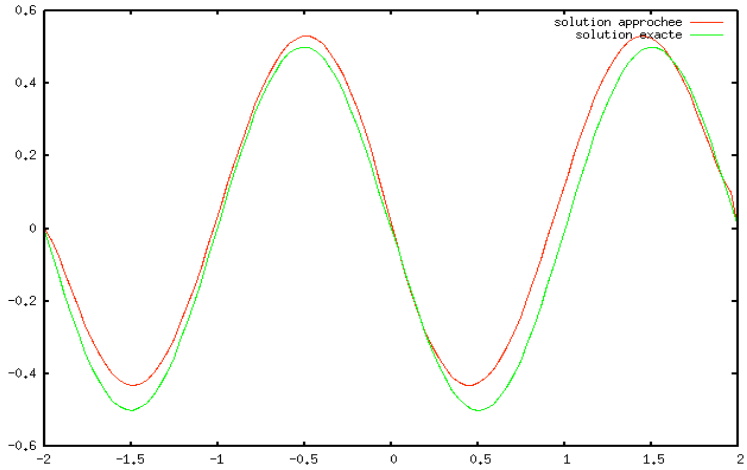
Question 4 :

Ci-dessous les graphes de la valeur exacte et approchée pour différents n :

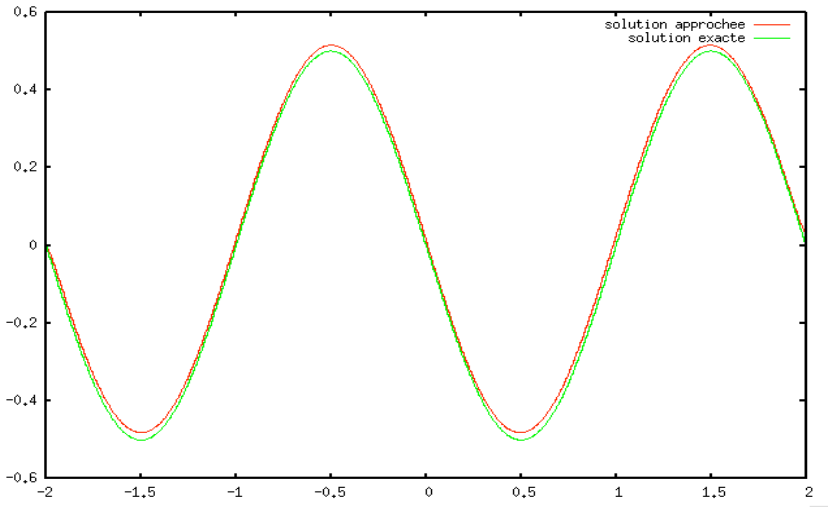
Graphe n=50 (4.1):



Graphe n=100 (4.2) :



Graphe n=1000 (4.3)



Remarque :

On remarque que plus n est grand plus les graphes se confondent, à n=1000 les graphes se superposent à plusieurs points tandis que à n=50 on voit une nette différence entre les deux courbes.

Question 5 :

On va partir de la méthode des différences finies :

$$\frac{\partial^2 u}{\partial t^2}(x_i, t_n) \simeq \frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} \quad (5.1) \text{ et } \quad \frac{\partial^2 u}{\partial x^2}(x_i, t_n) \simeq \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \quad (5.2)$$

En faisant la différence de l'équation (5.1) par c^2 fois l'équation (5.2) (on sait par les hypothèses que le résultat de cette opération sera nulle), on obtient :

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} - c^2 \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} = 0$$

$$\boxed{\Leftrightarrow u_i^{n+1} = -u_i^{n-1} + 2u_i^n + c^2 \frac{\Delta t^2}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)} \quad (5.3)$$

Remarque 5.1 :

C'est un schéma à deux pas en temps : on a besoin de u à l'instant n et n-1. Or,

$$\begin{cases} u_i^0 = u_0(x_i) \\ u_i^1 = u_0(x_i) + \Delta t\, u_1(x_i) \end{cases}$$

Description de la programmation :

D'après la remarque 5.1 on sait qu'on va avoir besoin de 3 vecteurs : un vecteur qui va stocker les valeurs de u à l'instant n, un autre pour l'instant n+1 et un dernier pour l'instant n+2. Notons-les respectivement un, unp1 et unp2.

Pour avoir un point de départ il nous faut définir u_0 et u_1 par des fonctions de notre choix. Prenons par exemple :

$$u_0(x) = \begin{cases} 0 & \text{si } |x| > 1 \\ \sin(\pi x)\exp(x) & \text{sinon} \end{cases} \quad \text{et} \quad u_1(x) = 0$$

Et donc :

$$u(x,t) = \frac{u_0(x-ct) + u_0(x+ct)}{2}$$

On peut sans difficultés définir ces fonctions en gfortran.

Il ne nous reste plus qu'à définir notre division dans l'espace (les xi) qui sera stockée dans un vecteur de taille n+2. Il faut aussi initialiser un, unp1 et unp2 dans les conditions limites.

Pour initialiser les vecteurs un et unp1 en entier :

```
do i=1,nmax
un(i)=u0(x(i))
unp1(i)=un(i)+dt*u1(x(i))
end do
```

Il suffit alors de calculer unp2 par la formule trouvée par le schéma. Pour le schéma saute-moutons :

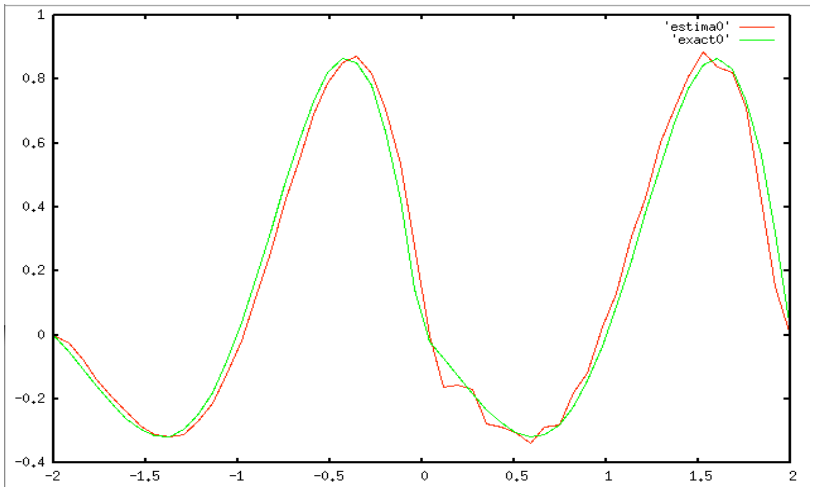
```
t=dt
do while (t.lt.tmax)
  t=t+dt
  do i=1,nmax
        unp2(i)=-un(i)+2*unp1(i)+c*c*dt*dt/dx/dx*(unp1(i+1)-2*unp1(i)
+unp1(i-1))
  end do
  un=unp1 !on va passer à l'étape suivante alors on doit stocker...
  unp1=unp2 !les vecteurs "nouveaux" dans les "anciens"
end do
```

On pourra alors comparer le résultat exact avec celui trouvé en unp2 à la fin.

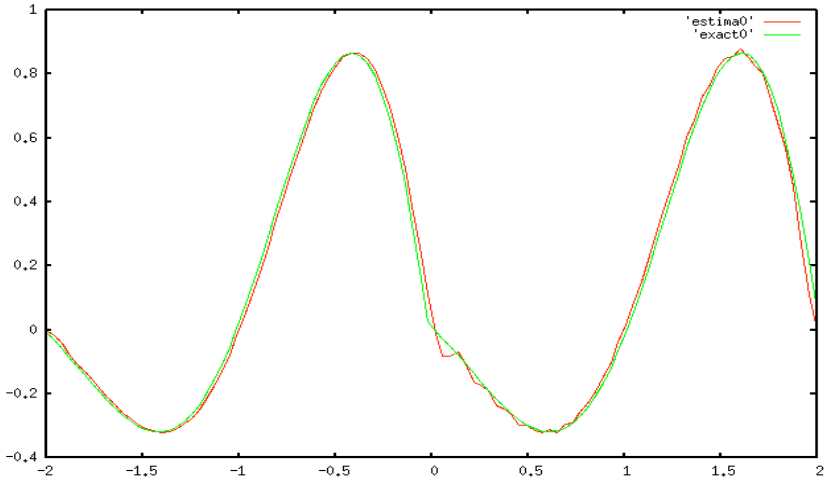
Pour savoir si le schéma est convergent ou pas il suffira de comparer les courbes des fonctions approchée et exacte.

Si le schéma est convergent alors la différence des graphes sera plus petite si n est plus grand.

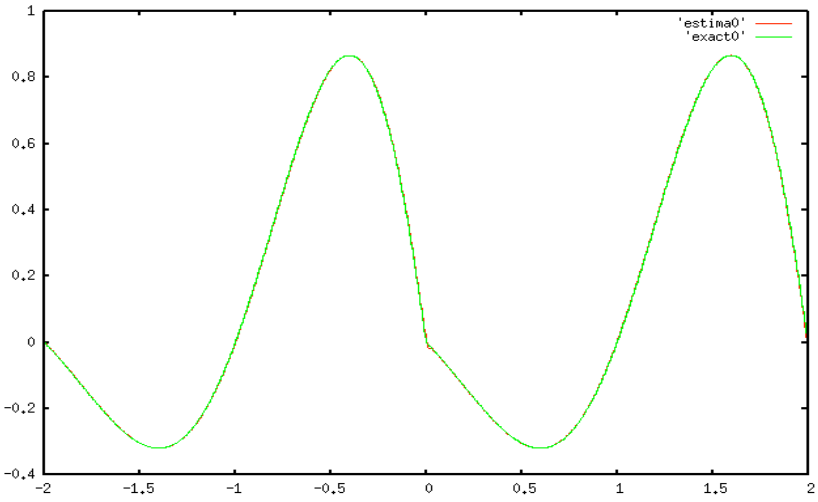
Graphe 5.4 : n=50



Graphe 5.5 : n=100



Graphe 5.6 : n=1000

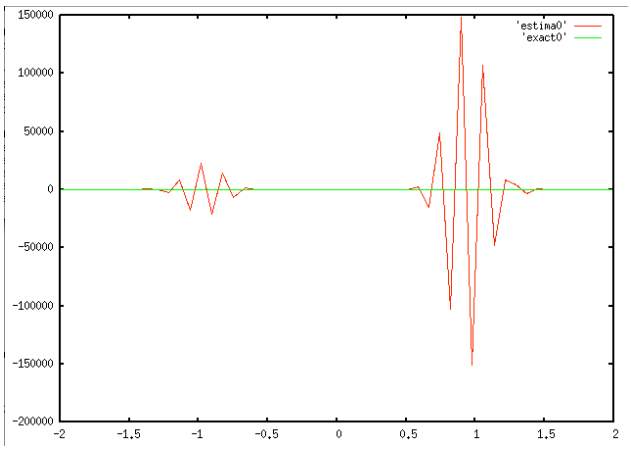


Remarque :

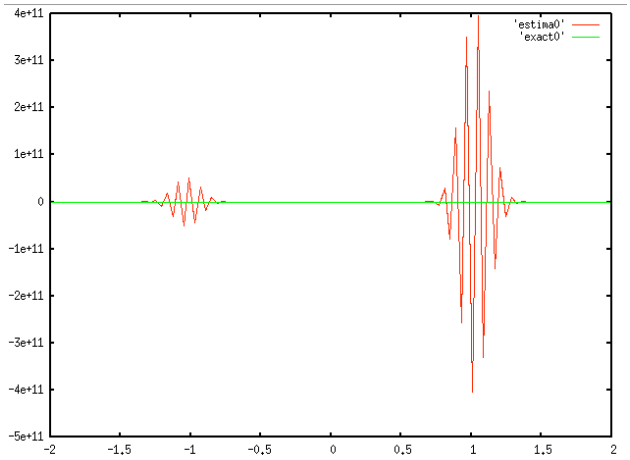
Pour le graphe 5.4 on peut voir qu'il y a plusieurs points où l'on peut observer une nette différence entre les 2 courbes. Dans le graphe 5.5 la différence entre chaque point est plus petite que celle pour le graphe précédent mais on peut encore voir des points de grande différence.
Par contre dans le graphe 5.6 les graphes sont pratiquement confondus.
Le schéma est donc convergent.

Si le schéma n'est pas stable (ie. $c > \frac{\Delta x}{\Delta t}$) , on a les graphes suivants

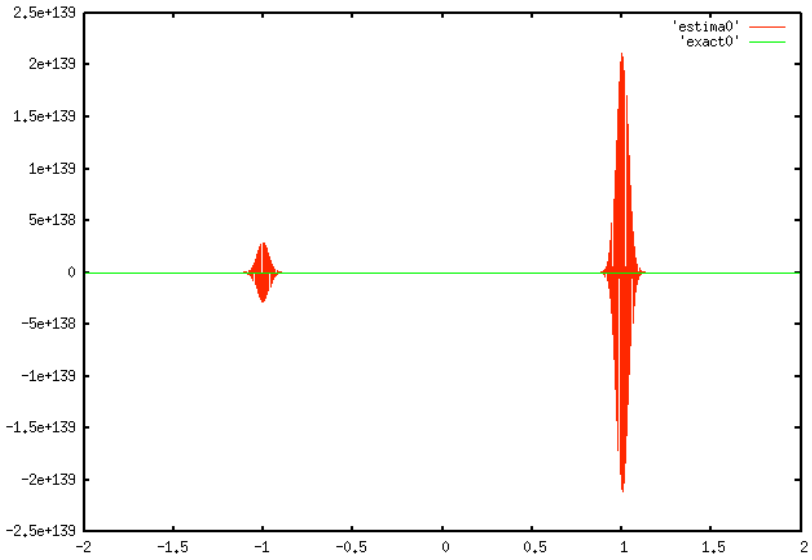
Graphe 5.7 : n=50



Graphe 5.8 : n=100



Graphe 5.9 : n=1000



Question 6 :

Le schéma le plus précis est le schéma saute-moutons. On peut le remarquer en comparant les graphes (4.1) à (5.4), (4.2) à (5.5) et surtout (4.3) à (5.5). Pour chaque couple de graphes on a pris une même valeur de n, néanmoins l'erreur de chaque schéma n'est pas la même.

On voit sur le graphe (5.6) deux courbes quasiment confondues, tandis que sur le graphe (4.3) on peut sans difficultés différencier les deux courbes.

On remarque que sans implémenter les schémas on aurait pu anticiper ce résultat. En effet, le schéma décentré n'utilise que un seul pas, tandis que le schéma saute-moutons a besoin de deux.

Question 7 :

Posons $A = \begin{pmatrix} 2 & -1 & \\ -1 & \ddots & -1 \\ & -1 & 2 \end{pmatrix}$, $B = \begin{pmatrix} 1 & 0 & \dots \\ 0 & 0 & 0 \\ 0 & \dots & 0 \end{pmatrix}$ et $C = c^2 \frac{\Delta t^2}{\Delta x^2}$

On peut alors écrire l'équation (5.3) :

$$u_i^{n+1} = -u_i^{n-1} + 2u_i^n + c^2 \frac{\Delta t^2}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \Leftrightarrow U^{n+1} = -BU^{n-1} + 2BU^n + C^2 AU^n$$

$$\boxed{\Leftrightarrow U^{n+1} = -BU^{n-1} + (2B + c^2 A)U^n}$$

Or les valeurs propres de A sont :

$$\mu_k = 4\sin\left(\frac{2k\pi}{2i}\right)$$

RÉSOLUTION DE L'ÉQUATION DE LA PLACE SUR UN CARRÉ

Données du problème :

$$-\Delta u(x,y) = f(x,y),\ \ (x,y)\in\Omega = [0,1]\times[0,1] \qquad (0.1)$$

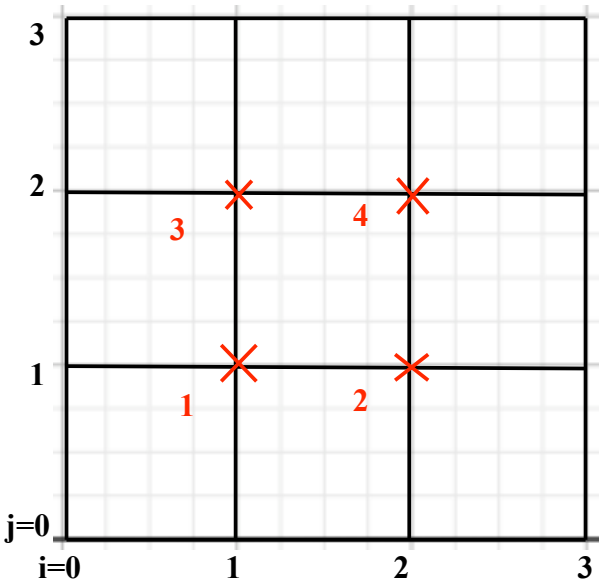
$$u(x,y) = 0,\ \ (x,y)\in\partial\Omega \qquad (0.2)$$

Question 1 :

Posons $h = \frac{1}{N+1}$ et $x_i = hi,\ \ y_j = hj$ (comme sur le graphe 1.0).

On peut alors noter $u_{ij} = u(x_i, y_j)$.

Graphe 1.0 :



Les points marqués en rouge ont pour coordonnées i,j. Mais on peut leur associer également la coordonnée k comme la numérotation en rouge.

Discrétisation par différences finies :

Comme on a vu dans les TP précédents on a :

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) \simeq \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} \qquad (1.1)$$

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) \simeq \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2} \qquad (1.2)$$

Or

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

En utilisant les équations (1.1) et (1.2) on peut écrire :

$$-\Delta u = -\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} - \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2}$$
$$-\Delta u = \frac{4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - ui,j+1}{h^2} = f(x_i, y_j)$$

Question 2 :

Posons N tel que : $1 \leqslant i, j \leqslant N$, et avec la définition de k (cf. Graphe 1.0) on a $1 \leqslant k \leqslant N^2$.

On peut alors introduire la notation :

$$u_{i,j} = u_k, \quad 1 \leqslant i, j \leqslant N, \quad 1 \leqslant k \leqslant n = N^2$$

avec $k = i + N(j-1)$.
Soit

$$F_k = f(x_i, y_j)\Delta x^2 \quad \text{et} \quad U = \begin{pmatrix} u_1 \\ u_2 \\ \dots \\ u_N \end{pmatrix}$$

On peut alors écrire le problème de la façon suivante :

$$AU = F$$

<u>Remarque :</u>

On note que quelque soient i et j les valeurs de A sont soit 0 soit 4 soit -1.

Etudions les valeurs de A plus précisément :

$$\begin{cases} A_{ii} = 4 \\ A_{ij} = -1 & \text{s'il existe une arête de la grille joigneant i et j} \\ A_{ij} = 0 & \text{sinon} \end{cases}$$

Cas particulier : N = 2
Voir le graphe 1.0.
On a alors :

$$U = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} u_{1,1} \\ u_{1,2} \\ u_{2,1} \\ u_{2,2} \end{pmatrix}, \quad F = \Delta x \begin{pmatrix} f(x_{1,} y_1) \\ f(x_{1,} y_2) \\ f(x_{2,} y_1) \\ f(x_{2,} y_2) \end{pmatrix} \quad \text{et} \quad A = \begin{pmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{pmatrix}$$

Question 3 :

Stockage morse :

On sait que la matrice A est creuse, c'est à dire qu'elle va contenir beaucoup de valeurs nulles. Ce qui signifie qu'il est inutile de stocker la matrice en entier (dans un tableau 2D).

On rappelle que la dimension de A est n x n. Posons alors p le nombre de termes non nuls. Il nous suffit d'avoir un vecteur de taille p pour stocker les valeurs non nulles, nommons-le *val*, et deux vecteurs de taille p qui vont respectivement donner l'indice de colonne et l'indice de ligne on peut les appeler indj et indi.

Programme :

Pour créer un programme qui stocke une matrice A de taille n sous le principe du stockage morse, il suffit de suivre les indications précédentes.

```
subroutine morse (a, n, p, val, indi, indj)
    implicit none
    integer::p,n,i,j,k
    real*8,dimension(1:p)::val,indi,indj
    real*8,dimension(1:n,1:n)::a

    k=1 !indique le numéro dans le vecteur val
    do i=1,n !compteur sur les lignes
      do j=1,n !compteur sur les colonnes
            if(a(i,j).ne.0) then !si on trouve une valeur non nulle
                  val(k)=a(i,j) ! on stocke cette valeur dans val à l'indice k
                  indi(k)=i ! on stocke dans indi l'indice de la ligne
                  indj(k)=j ! et dans indj l'indice de la colonne
                  k=k+1 !on incremente le compteur de valeurs non nulles
            end if
      end do
    end do
end subroutine
```

On remarque que l'implémentation a besoin de connaître la valeur de p (nombre de valeurs non nulles) pour pouvoir déclarer les vecteurs. On a créé alors une fonction qui parcourt la matrice A en entier et incrémente un compteur chaque fois qu'elle rencontre une valeur différente de 0.

On obtient la fonction suivante :

```
function calcul_p (a, n)
    integer::n,calcul_p
    real*8,dimension(1:n,1:n)::a
    calcul_p=0
    do i=1,n
      do j=1,n
            if (a(i,j).ne.0) then
                  calcul_p=calcul_p+1
            end if
      end do
    end do
end function
```

Question 4 :

On doit tout d'abord commencer par calculer le profil de la matrice.
Le profil est le nombre de valeurs de la matrice stockées au dessus de la diagonale dans la colonne k.
On remarque d'abord que le profile de 1 est 0.
Sinon on récupère les indices de la k-ième valeur, disons i et j. On sait donc que si $j-i \leqslant prof(j)$ alors la k-ième valeur est dans le profil même si elle est nulle. Dans le cas contraire la valeur n'est pas stockée.
On remarque que le profil est symétrique.

```
    prof=0 !initialisation du vecteur
    do k=2,p !pour toutes les valeurs non nulles, sauf k=1 car prof=0 toujours
      i=indi(k) !on recupere l'indice de ligne dans matrice
      j=indj(k) !idem avec l'indice de colonne
      prof(j)=max(prof(j), j-i) !on applique les formules
      prof(i)=max(prof(i), i-j)
    end do
```

On a aussi besoin d'un vecteur (de taille n+1) pour récupérer les débuts de ligne/colonne. Soit ce vecteur kld, alors kld(i) donne le nombre de valeurs

stockées jusqu'à la (i-1) colonne/ligne.

On peut alors utiliser le profil définit précédemment et on peut écrire :

```
kld(1)=1
do i=2,nn+1
  kld(i)=kld(i-1)+prof(i-1)
end do
```

Il nous reste à définir les vecteur vkgs, vkgi et vkgd qui stockeront respectivement les valeurs intéressantes de A (définies par kld et profil) qui sont au-dessous de la diagonale, au-dessus de la diagonale ou sur la diagonale.

```
do l=1,p !pour toutes les valeurs non nulles
  i=indi(l) !on recupere l'indice de ligne...
  j=indj(l) ! et de colonne
  v=val(l) !et la valeur non nulle
  if (i.eq.j) then !si on est sur la diagonale
        vkgd(i)=v !on stocke la valeur dans vkgd
  end if
  if (i<j) then !si on est au-dessous de la diagonale
        k=kld(j+1)-j+i !on retrouve l'indice
        vkgs(k)=v !et on stocke la valeur
  end if
  if (i>j) then !si on est au-dessus de la diagonale
        k=kld(i+1)-i+j !on retrouve l'indice
        vkgi(k)=v ! et on stocke la valeur
  end if
end do
```

Question 5 :

Pour pouvoir écrire le programme qui permet de résoudre numériquement le problème (0.1)-(0.2) aux différences finies, on doit définir dans le programme la matrice A. Comme le stockage en morse est plus facile on peut la définir premièrement comme cela et puis comme le sous programme *sol.f* utilise le stockage profil, on devra utiliser le programme écrit dans la question précédente pour passer d'un stockage à l'autre.

```
!algorithme pour remplir la matrice A en utilisant le stockage morse
p=0 !on initialise un compteur (pour toutes les valeurs non nulles)
do k=1,n !on sait que sur la diagonale il y a que des 4
```

```
  p=p+1 !on va stocker une valeur on increment donc le compteur
  indi(p)=k !on garde l'indice de ligne des elements sur la diagonale
  indj(p)=k !et l'indice de colonne
  val(p)=4 !puis finalement ça valeur (qui est toujours 4)
end do
do ii=1,nn-1 !on va parcourir les points d'indice k...
  do jj=1, nn ! et on va remplir les liaisons horizontales
        p=p+1 !on va stocker une valeur on incremente donc le compteur
        i=ii+(jj-1)*nn !on recupere l'indice d'un point dans la matrice
        j=ii+1+(jj-1)*nn !et de celui d'à coté
        val(p)=-1 !ils sont reliés donc la valeur dans A vaut -1
        indi(p)=i !on stocke les indices
        indj(p)=j
        p=p+1 !on va stocker une nouvelle valeur
        val(p)=-1 ! A est symetrique
        indi(p)=j !donc on aura un -1 dans le point j,i
        indj(p)=i
  end do
end do
do ii=1,nn !de même que précédemment ...
  do jj=1,nn-1 !on va remplir les liaisons mais cette fois-ci VERTICALEMENT
        i=ii+(jj-1)*nn
        j=ii+jj*nn
        p=p+1
        val(p)=-1
        indi(p)=i
        indj(p)=j
        p=p+1
        val(p)=-1
        indi(p)=j
        indj(p)=i
  end do
end do
call passage (val, indi, indj, nn, n, pmax, kld(1), vkgs, vkgi, vkgd)
```

On peut écrire alors le maillage des x et des y :

```
do i=0,nn+1
  x(i)=i*dx
  y(i)=i*dx
end do
```

Et on doit initialiser le vecteur F :

```
do ii=1,nn
  do jj=1,nn
        i=ii+(jj-1)*nn
        f(i)=pression(x(ii), y(jj))
```

```
      end do
    end do
```

On peut alors appeler la subroutine *sol.f* :

```
    call sol(vkgs,vkgd,vkgi,f,kld,u,n,6,1,1,1,energ,ier)
```

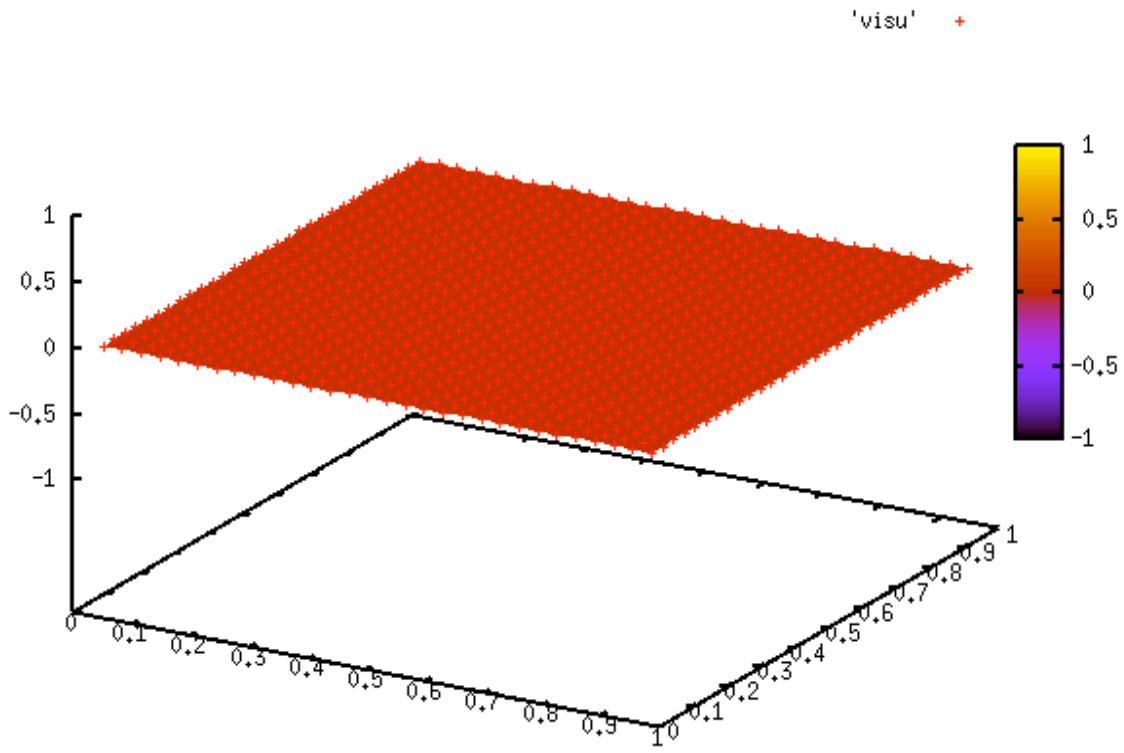
Question 6 :

Choisissons une fonction f (pression dans le programme fortran) :

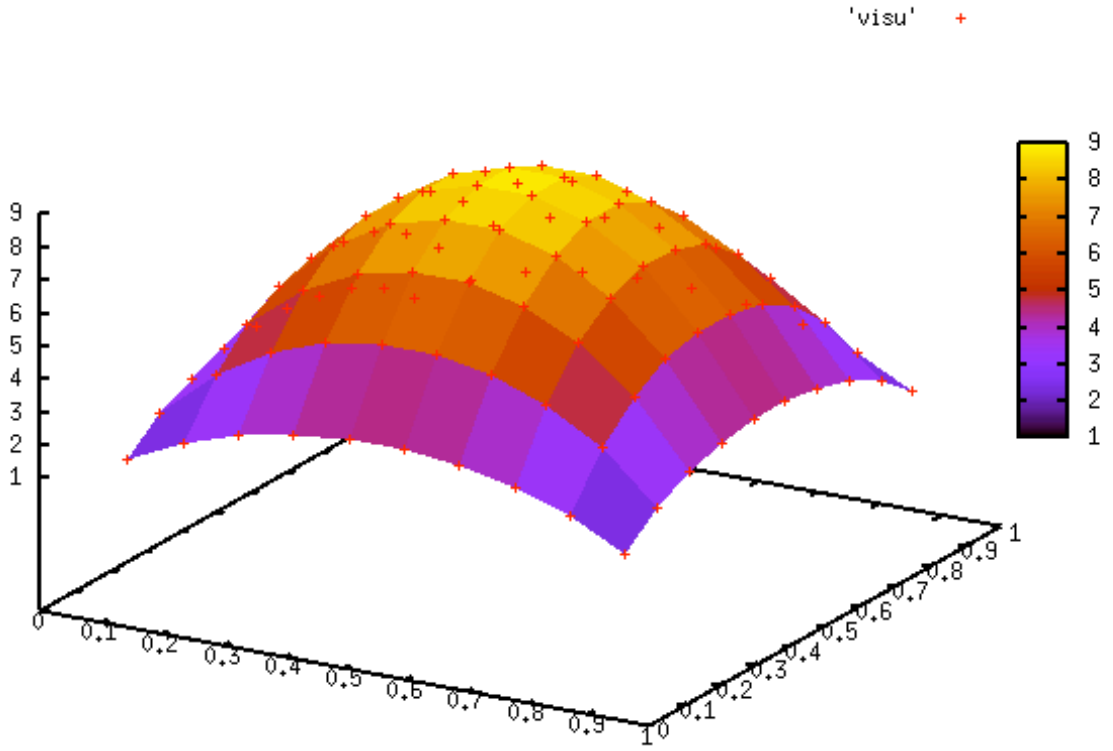
```
function pression (xi, xj)
    implicit none
    real*8::xi,xj,pression
    pression=1.d0
endfunction
```

On obtient alors :

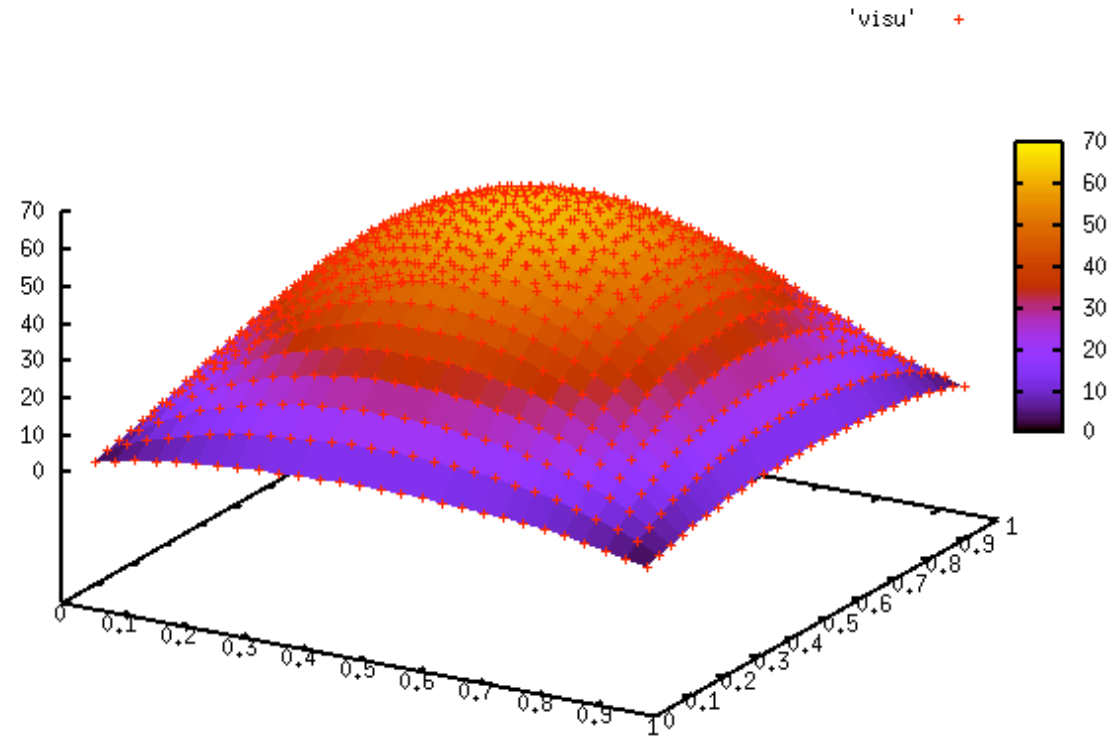
Si f = 0 :



Si f=1 et nn=10



Si f = 1 et nn=28



MÉTHODE DES ÉLÉMENTS FINIS

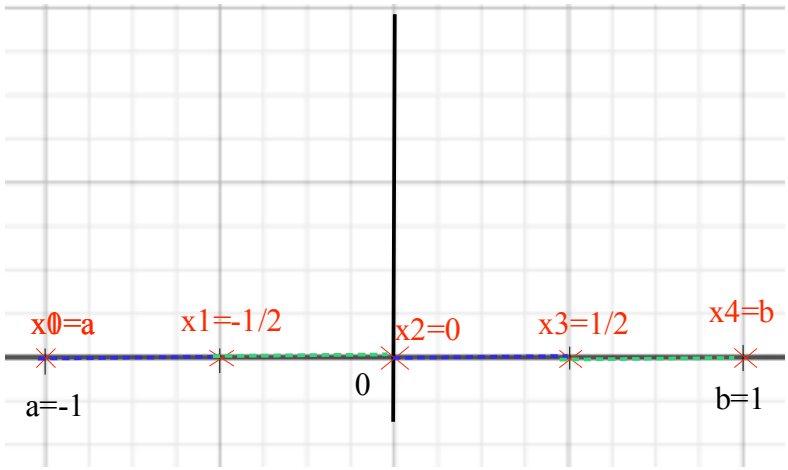
Espace d'interpolation :

Soit $u:[a,b]\to\mathbb{R}$ fonction continue. Soient des points $(N_i)_{i\in I}$ dans [a,b]. On cherche une approximation u_n de u polynomiale par morceaux et continue telle que $u_n(N_i)=u(N_i)$.
Soit N_e le nombre d'éléments finis, et soient

$$h=\frac{b-a}{2},\ x_m=a+mh,\ m=0\ldots N_e$$

Aux bornes on a $x_0=a,\ x_{N_e}=b$

Graphes 1.0 : Ne= 4



L'élément fini $E_m=[x_{m-1},x_m],\quad m=1\ldots N_e$ est marqué alternativement en bleu et en vert sur le graphique.

Soit r le degré des polynômes d'interpolation tel que $r=d-1$. Il nous faut alors *d* points distincts pour définir l'interpolation.
Définissons les noeuds d'interpolation :

$$N_{l,m}=x_{m-1}+(l-1)\frac{h}{d-1}$$

où m est le numéro de l'élément fini et l le numéro local du noeud $N_{l,m}$ dans l'interpolation. On va aussi noter $N_{l,m}=N_i$ où i est le numéro global du noeud et tel que $i=l+(m-1)(d-1)$.

Pour pouvoir passer d'une notation à l'autre, entre (l,m) et i donc entre la numérotation locale et la globale, on va créer un tableau de connectivité.

Ce qui donne en fortran :

```
    !calcul de connec
     do l=1,d !boucle sur les points locales
       do m=1,Ne !boucle sur les elements
             connec(l,m)=l+(m-1)*(d-1) !formule pour le passage de notation
       end do
     end do
```

On va interpoler les polynômes sur l'élément fini m, en utilisant l'interpolation de Lagrange.
Soit

$$L_{l,m}(x) = \prod_{\substack{k=1\ldots d \\ k\neq l}} \frac{x - N_{k,m}}{N_{l,m} - N_{k,m}}$$

le polynôme de Lagrange de degré d-1. On constate que

$$\begin{cases} L_{l,m} = 1 & \text{si } l = k \\ L_{l,m} = 0 & \text{sinon} \end{cases}$$

En fortran cela donne :

```
function lagrange (k,d,h,x)
    implicit none
    real*8::lagrange,x,h
    integer::k,d,i,j
    real*8,dimension(d)::N1
    do i=1,d
      N1(i)=(i-1)*h/(d-1) !definition des noeuds d'interpolation du premier
element
    end do
    lagrange=1.d0 !initialisation du resultat
      do j=1,d !boucle sur le numéro local du noeud
            if (j.ne.k) then
                  lagrange=lagrange*(x-N1(j))/(N1(k)-N1(j)) !definition du
polynome de Lagrange
            end if
      end do
end function lagrange
```

On doit alors définir des fonctions d'interpolation sur [a,b] tout entier. Soit $N_n = N_l(d-1)+1$ le nombre noeuds du maillage et $(\varphi_i)_{i=1..N_n}$ les fonctions d'interpolation, on veut alors $\varphi_i(N_j) = \delta_{ij}$. On cherche alors la valeur de $\varphi_i(x)$ on a deux cas possibles :

> Soit i n'est pas un nœud de l'élément m alors la fonction est nulle.
> Soit i est une nœ alors ud l'élément m alors la fonction est le polynôme de Lagrange.

(φ_i) est une base d'interpolation, on peut alors écrire :

$$\boxed{u_h(x) = \sum_{i=1}^{N_n} u_i \varphi_i(x) = \sum_{i=1}^{N_n} U_h(N_i)\varphi_i(x)} \quad (1.1)$$

<u>Application à la résolution d'une équation aux dérivées partielles :</u>

<u>Données du problème :</u>

$$(2.0)\begin{cases} -\mathrm{u}'' = \mathrm{f} \\ \mathrm{u}'(\mathrm{b}) + \mu\,\mathrm{u}(\mathrm{b}) = 0 \\ -\mathrm{u}'(\mathrm{a}) + \lambda\,\mathrm{u}(\mathrm{a}) = 0 \end{cases}$$

avec $\lambda, \mu > 0$ contantes.

Au lieu de dériver comme pour le schéma des différences finies, on intègre :

Soit v une fonction quelconque dérivable telle que :

$$\int_a^{b-u} ''v = \int_a^b f\,v \;\Leftrightarrow\; \int_a^b u'v' - [u'v]_a^b = \int_a^b fv$$

$$\Leftrightarrow \int_a^b u'v' + \mu u(b)v(b) + \lambda u(a)v(a) = \int_a^b fv \quad (2.1)$$

On peut remplacer le système (2.0) par l'équation (2.1) .

Or, on avait déjà défini une approximation de la fonction u dans la formule écrite en 1.1.

On a alors

$$\int_a^b u_n' v_n' + \lambda u_n(a) v_n(a) + \mu u_n(b) v_n(b) = \int_a^b f v_n \quad (2.2)$$

Où

$$U_n(x) = \sum_{j=1}^{N_n} u_j \varphi_j(x) \quad \text{et} \quad V_n = \varphi_i$$

En remplaçant en (2.2) on obtient :

$$\boxed{\int_a^b \sum_j u_j \varphi_j' \varphi_i + \lambda u_1 \delta_{i1} + \mu u_{Nn} \delta_{i\,Nn} = \int f \varphi_i} \quad (2.3)$$

Soit

$$\begin{cases} A_{ij} = \int_a^b \varphi_i' \varphi_j' \quad \text{si } 1 < i < N_n \\ A_{1j} = \int_a^b \varphi_1' \varphi_j' + \lambda \delta_{j,1} \\ A_{Nn,j} = \int_a^b \varphi_{Nn}' \varphi_j + \mu \delta_{j,Nn} \end{cases}, \quad U = \begin{pmatrix} u_1 \\ u_2 \\ \dots \\ u_{Nn} \end{pmatrix} \quad \text{et} \quad F = \begin{pmatrix} \int f \varphi_1 \\ \int f \varphi_2 \\ \dots \\ \int f \varphi_{Nn} \end{pmatrix}$$

Finalement on peut alors écrire le système (0.1) comme un système linéaire AU=F.

<u>Procédure d'assemblage :</u>

Soient i et j respectivement, les numérotations globales des nœuds d'indices locales k et m, et d'indices locales l et m, pour tout m numéro d'élément fini et l et k numéros des nœuds locaux de l'élément fini m.

On a alors :

$$A_{i,j} = A_{i,j} + \underbrace{\int_{Em} L'_{k,m}(x) L'_{l,m}(x)\, dx}_{\text{matrice locale } M_{k,l}}$$

où les polynômes L sont les polynômes de Lagrange.

En fortran voici l'exemple de définition de la matrice locale pour d=2 et d=3.

```
function mloc(d, h, k, l)
    integer::d
    real*8::h, mloc
    real*8,dimension(d,d)::ml
    if (d.eq.2) then
      ml(1,1)=1.d0
      ml(1,2)=-1.d0
      ml(2,1)=-1.d0
      ml(2,2)=1.d0
    end if
    if (d.eq.3) then
      ml(1,1)=7.d0/3.d0
      ml(1,2)=-8.d0/3.d0
      ml(1,3)=1.d0/3.d0
      ml(2,1)=-8.d0/3.d0
      ml(2,2)=16.d0/3.d0
      ml(2,3)=-8.d0/3.d0
      ml(3,1)=1.d0/3.d0
      ml(3,2)=-8.d0/3.d0
      ml(3,3)=7.d0/3.d0
    end if
    ml=ml/h
    mloc=ml(k,l)
end function
```

On peut alors définir la matrice A de la manière qui suit :

```
    !on va calculer p = nombre de valeurs non nulles
    p=0 !initialisation
    do i=1,nn !sur tous les noeuds (et la taille du profil)
      p=p+prof(i) !on incremente p
    end do

    allocate(vkgs(p), vkgi(p)) !une fois p calculé on peut allouer les vecteurs
nécessaires
    !init de A
    vkgd=0.d0
```

```
    vkgs=0.d0
    vkgi=0.d0
    !fin init de A

    do m=1,Ne !boucle sur les élements
      do k=1, d !pour un noeud local
            do l=1,d !idem
                  i=connec(k,m) !on retrouve les numérotations globales pour un
noeud
                  j=connec(l,m) !idem pour l'autre noeud
                  if (i.eq.j) then ! si on est sur la diagonale
                        vkgd(i)=vkgd(i)+mloc(d, h, k, l)
                  end if
                  if (i.lt.j) then ! si on est au dessous de la diagonale
                        vkgs(kld(j+1)-j+i)=vkgs(kld(j+1)-j+i)+mloc(d, h, k,l)
                  end if
                  if (i.gt.j) then !au cas contraire
                        vkgi(kld(i+1)-i+j)=vkgi(kld(i+1)-i+j)+mloc(d, h, k,l)
                  end if
            end do
      end do
    end do
    !definition des extremums de la diagonale de A
    vkgd(1)=vkgd(1)+lambda
    vkgd(nn)=vkgd(nn)+mu
```

Il nous reste à calculer le membre de droit.

<u>Intégration numérique :</u>

Utilisons la formule de Simpson :

$$\int_x^y f(t)dt \simeq \frac{1}{6}(y-x)(f(x)+4\mathrm{f}(\frac{x+y}{2})+f(y)) = h\sum_{i=1}^{3}\omega_i f(x+\xi_i h)$$

où oméga sont les poids d'intégration et $x+\xi_i h$ les points d'intégration, définis en fortran ci-dessous :

```
    real*8,dimension(3)::xi=(/0.d0, 1.d0/2.d0, 1.d0/), omega=(/ 1.d0/6.d0,
2.d0/3.d0, 1.d0/6.d0 /)
```

On peut alors écrire :

```
    vf=0.d0 !Assemblage des forces
    do m=1,Ne !boucle sur les elements
      do k=1,d !boucle sur les noeuds
            i=connec(k,m)
            do ii=1,3 !pt d'integration
                  xg=a+(m-1)*h+xi(ii)*h !point de gauss
                  xref=xi(ii)*h
                  vf(i)=vf(i)+omega(ii)*h*f(xg)*lagrange(k,d,h,xref) !calcul du
membre de gauche
            end do
      end do
    end do
```

Où f est une fonction quelconque, dans l'exemple de ce TP on a choisit :

```
function f(xi)
    implicit none
    real*8::f,xi,pi,p4,pp,h, x1
    integer::p=3
    pi=4.d0*atan(1.d0) !calcul de pi

    x1=1.d0+xi !variable
    pp=2.d0*p*pi*xi!variable

    f=(-p*p*(1-pi**2)*exp(-p*xi)*sin(p*pi*xi)+2*p**2*pi*exp(-p*xi)*cos(p*pi*xi))
!fonction f, utilisée au TP1
end function
```

Cette fonction est la dérivée seconde de la fonction qui suit :

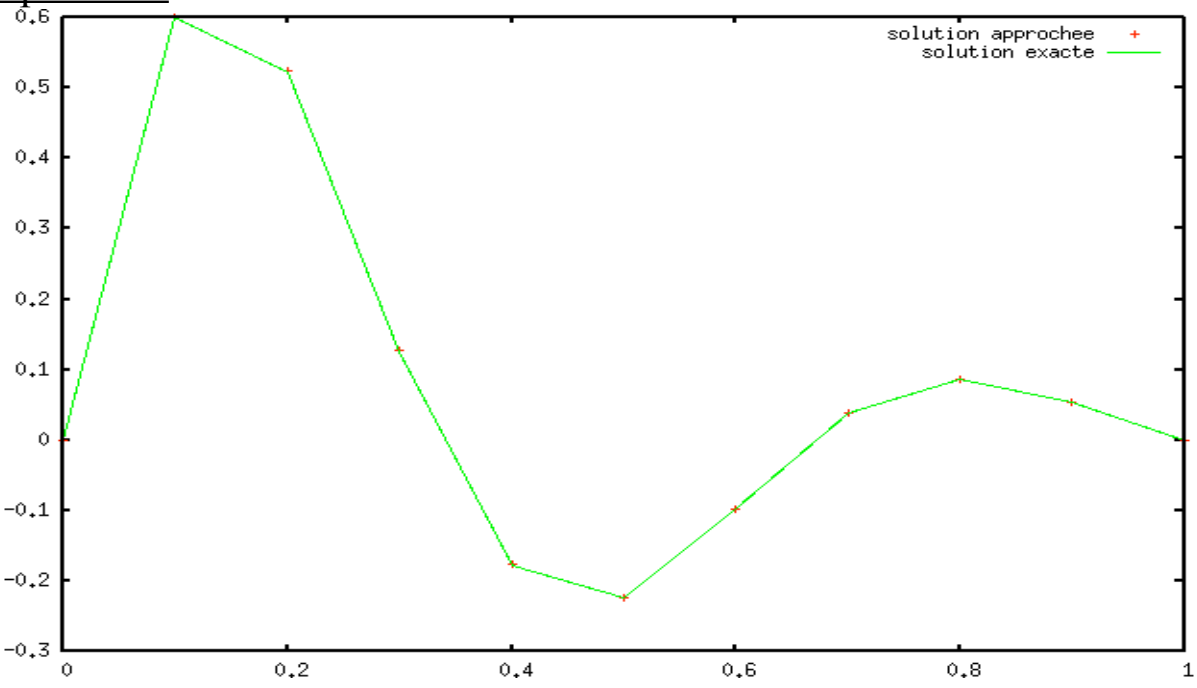
```
function xexact (x) !fonction qui calcule la valeur exacte
    implicit none
    integer::p
    real*8::x,xexact,pi
    p=3
    pi=4.d0*atan(1.d0)
    xexact=sin(pi*p*x)*exp(-p*x)
end function xexact
```

On a alors toutes les données nécessaires pour pouvoir appeler la subroutine de *sol.f* de la manière suivante :

```
    call sol(vkgs, vkgd, vkgi, vf, kld, u, nn, 6, 1, 1, 1, energ, ier)
```

Voici alors quelques exemples de courbes pour différentes valeurs de départ :

Graphe 3.0 : Avec 10 éléments et d = 2



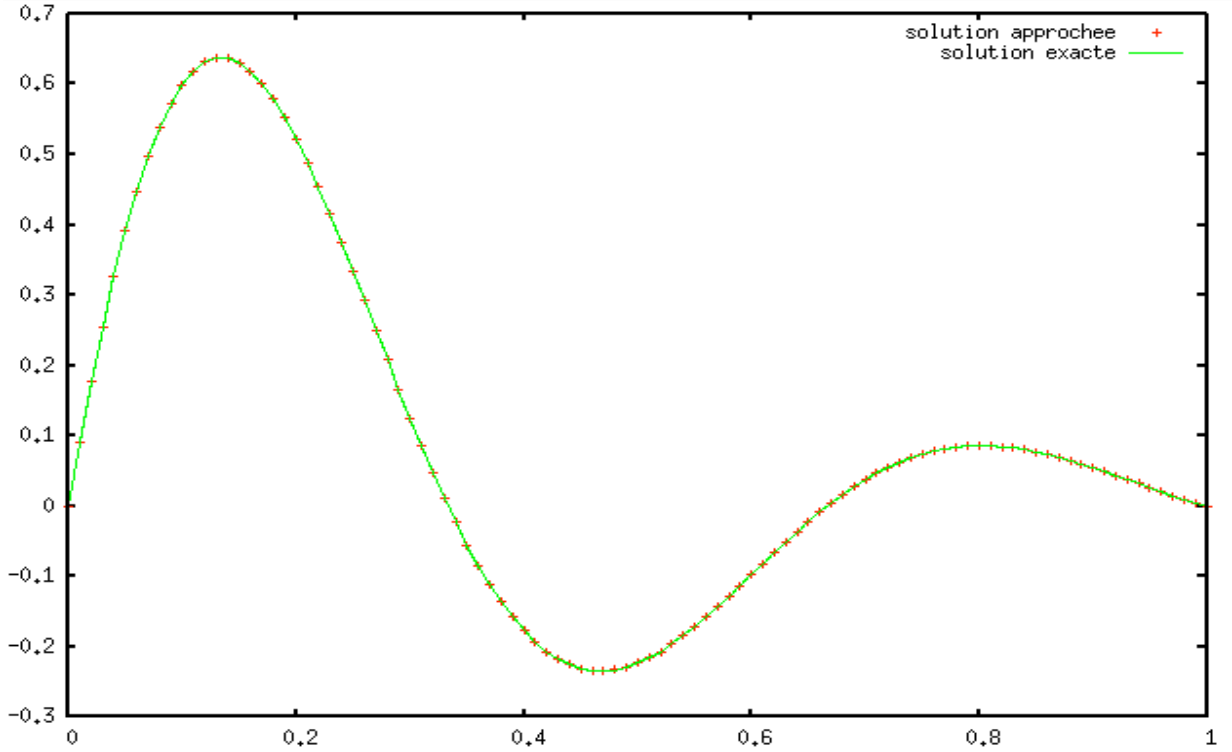
Même si les solutions semblent très approchées, on peut voir qu'il y a quand même une erreur de l'ordre de 10^{-4} dans l'affichage des fichiers :

Abscisses	Solution Approchée	Solution Exacte
0.00000000000000	9.423694317558300E-005	0.00000000000000
0.100000000000000	0.599684518213775	0.599334530274120
0.200000000000000	0.522810442325973	0.521950882725828
0.300000000000000	0.126723526258792	0.125636934257085
0.400000000000000	-0.176109250102420	-0.177037515837846
0.500000000000000	-0.222540762563491	-0.223130160148430
0.600000000000000	-9.685546100877712E-002	-9.716024871699044E-002
0.700000000000000	3.800763138414215E-002	3.784111740062794E-002
0.800000000000000	8.640380512833977E-002	8.627790062085508E-002
0.900000000000000	5.446044020695313E-002	5.437040192213960E-002
1.00000000000000	4.703609734813537E-006	1.829147217469050E-017

Les résultats étant tellement approchés à la solution exacte on peut alors

supposer que plus on a des éléments plus le résultat sera exacte :

Graphe 3.1 : Avec 100 éléments et d = 2



Dans la même logique, si on incrémente le degré des polynômes on va obtenir des résultats plus précis.

Graphe 3.2 : Avec 10 éléments et d = 3