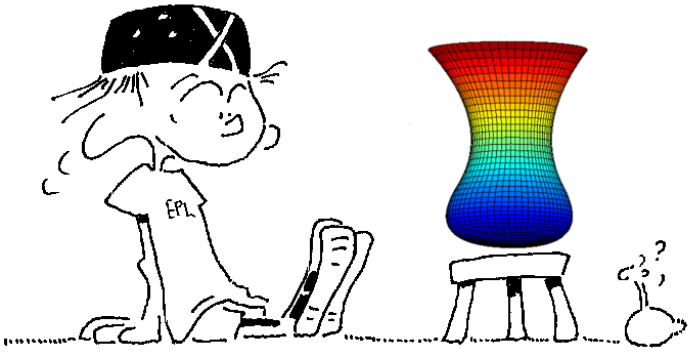
Ecole Polytechnique de Louvain



INFORMATIQUE
ET
METHODES NUMERIQUES
...ou les aspects facétieux du calcul sur un ordinateur

V. Legat

Solutions des exercices pour le cours FSAB1104
Année académique 2019-2020 (version 4.7 : 19-09-2019)

python™

Les ordinateurs sont comme les Dieux de l'Ancien Testament : beaucoup de règles et aucune pitié.
(Joseph Campbell)

Interpolation : solutions

1

1. $a = \dfrac{U_0 X_1 - U_1 X_0}{X_1 - X_0}, \qquad b = \dfrac{U_1 - U_0}{X_1 - X_0}$
2. $u^h(x) = 2x$
3. *L'erreur d'interpolation est définie comme* $u(x) - u^h(x)$.
   On obtient donc très aisément $e^h(1/9) = 1/3 - 2/9 = 1/9$.
   Il s'agit d'une erreur relative de 33% *!*

2

1. $$\begin{bmatrix} 1 & 3 & 9 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$
2. *Ce système possède une solution unique : si vous ne me croyez pas sur ma bonne foi, vous pouvez vérifier que la matrice de Vandermonde pour des abscisses distinctes a toujours un déterminant non-nul. Ce n'est évidemment plus le cas si deux abscisses sont identiques !*

3

Une implémentation possible est donnée par :

```
from numpy import *

def compute(n,U,t):

#
# -1- Calcul du noeud qui precede Xestim
#     Gestion des cas critiques (extrapolation à droite et à gauche)
#
  k = int(floor(t))
  k =  n-2 if (k+2 >  n)  else k
  k = -n+1 if (k-1 < -n)  else k
#
# -2- Estimation avec des polynomes de Lagrange
#     xi = coordonnee locale avec Xlocal = [-1 0 1 2]
#
  Ulocal = U[n+k-1:n+k+3]
  xi =  t - k
  phi = array([ -xi*(xi-1)*(xi-2)   ,3*(xi+1)*(xi-1)*(xi-2),
                -3*(xi+1)*xi*(xi-2),(xi+1)*xi*(xi-1)]) / 6;
  return Ulocal @ phi
```

Sur la Figure **??**, *on peut appréhender intuitivement le comportement de cette procédure avec la fonction* $u(x) = \sqrt{|x|}$.

4

Cette affirmation est vraie dans la plupart des cas. Toutefois, en choisissant avec soin les abscisses, il est possible d'obtenir un tel polynôme. Par exemple, si l'on choisit trois points sur une droite, on peut construire un polynôme de degré un (la droite ! :-) passant par les trois points.
Et, donc l'affirmation est fausse.

5

Comme $x \in [X_0, X_n]$, *il existe un intervalle* $I = [X_{i-1}, X_i]$ *tel que* $x \in I$. *Il est aisé d'observer*[1] *que :*

[1] Bon, bon si vraiment, vous ne le voyez pas, voilà quelques indications complémentaires. Il suffit de calculer la dérivée première de $f(x) = (x - X_{i-1})(x - X_i)$ et de l'annuler. On obtient ainsi $x = (X_i + X_{i-1})/2$ et le maximum correspondant est donc bien $(h/2)^2$.

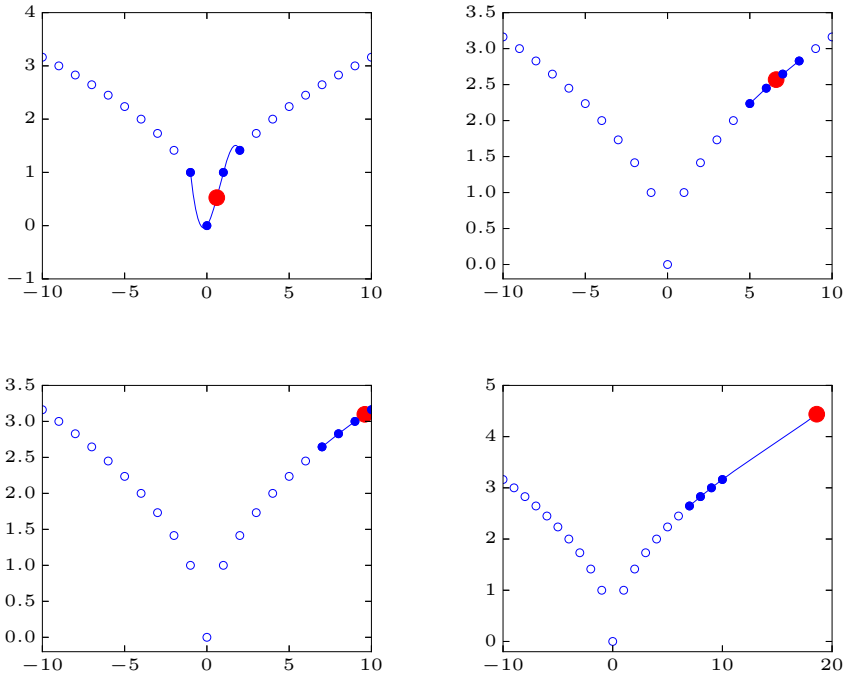


Figure 1: En fonction des cas ($x = 0.6, 6.6, 9.6, 18.6$), on effectue une interpolation et une extrapolation. La robustesse de l'implémentation nécessite un soin particulier pour le traitement des cas limites nécessitant une extrapolation

$$\max_{x\in[X_{i-1},X_i]} |(x - X_{i-1})(x - X_i)| = \frac{h^2}{4}.$$

Ensuite, on peut borner les autres facteurs possibles comme suit :

$$\begin{array}{rcl} |(x - X_0)| & \leq & ih \\ & \cdots & \\ |(x - X_{i-3})| & \leq & 3h \\ |(x - X_{i-2})| & \leq & 2h \\ |(x - X_{i+1})| & \leq & 2h \\ |(x - X_{i+2})| & \leq & 3h \\ & \cdots & \\ |(x - X_n)| & \leq & (n - i + 1)h \end{array}$$

Le produit donne $i!(n - i + 1)!$ *et le cas possible le plus défavorable est obtenu avec* $i = 1$ *ou* $i = n$. *En d'autres mots, cela se produit lorsque* x *se trouve sur le premier ou le dernier intervalle. Le produit des termes à considérer sera* $n!\ h^{n-1}$.

On obtient donc finalement le résultat :

$$\boxed{|\prod_{i=0}^{n}(x - X_i)| \quad \leq \quad n!\frac{h^{n+1}}{4}} \tag{1}$$

On peut ensuite utiliser le théorème sur l'erreur d'interpolation pour obtenir le résultat demandé. Nous savons que si la fonction $u(x)$ *est définie sur l'intervalle* $[X_0, X_n]$ *et qu'elle est* $(n + 1)$ *fois dérivable sur* $]X_0, X_n[$, *alors pour tout* $x \in]X_0, X_n[$, *il existe* $\xi(x) \in]X_0, X_n[$ *tel que :*

$$\boxed{e^h(x) \quad = \quad \frac{u^{(n+1)}(\xi(x))}{(n+1)!}\ (x - X_0)(x - X_1)(x - X_2)\cdots(x - X_n).} \tag{2}$$

*On déduit immédiatement le résultat demandé de la combinaison de (**??**) et de (**??**).*

□

6

Voici un tableau des différentes valeurs des polynômes en $x = 1.05$ *en fonction du degré, ainsi qu'une approximation de l'erreur commise.*

n	$u^h(1.05) =$	$e^h(1.05) \approx$
1	0.852839300	0.00029913
2	0.853138425	0.00000410

Il existe de multiples manières d'estimer l'erreur d'interpolation. Si votre ordre de grandeur correspond approximativement, votre méthode est correcte. Si vous obtenez une moins bonne estimation de l'erreur, essayez de l'améliorer.

On constate sur le tableau que l'approximation de l'erreur absolue est inférieure à $0.5\ 10^{-5}$ *pour le polynôme de degré deux : on a donc arrêté le calcul à cette étape...*

7

1. *La réponse se trouve dans le livre de physique :-)*
2. *On obtient un modèle plausible en calculant les coefficients d'un polynôme de degré trois* $f(t)$ *tel que* $f(0) = 0.2$, $f(3) = 2$ *et vérifiant de plus que* $f'(0) = f'(3) = 0$.

Interpolation par morceaux : solutions

8 *Il vaut mieux utiliser la représentation paramétrique pour l'interpolation par splines cubiques... On remarque que l'interpolation cubique par morceaux du quart de cercle en utilisant quatre noeuds est particulièrement décevante en utilisant une fonction du type $y = y(x)$.*

Observons aussi l'excellente représentation du cercle complet en utilisant une courbe spline paramétrée.

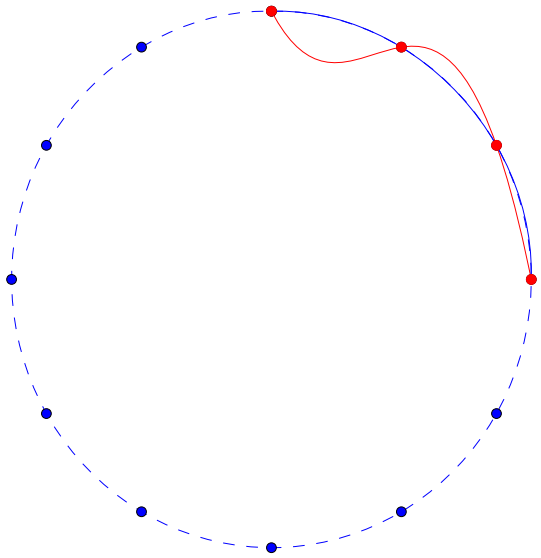


Figure 2.1: Approximation du quart de cercle par une courbe spline cubique et une courbe spline cubique paramétrée

9 *Dans tous les cas, nous avons $n = 4$ et nous devons donc estimer la dérivée cinquième de chaque fonction sur l'intervalle donné pour pouvoir utiliser le résultat que l'on a obtenu dans l'exercice 5.*

$$\boxed{\left|e^h(x)\right| \quad \leq \quad \frac{\max_{x\in[X_0,X_4]}\left|u^{(5)}(x)\right|}{20}h^5}$$

En particulier, nous obtenons :

$$\max_{x\in[-1,1]}\Big|\sinh(x)\Big| \quad \leq \quad 1.1752 \quad \Rightarrow \quad |e^h(x)| \quad \leq \quad \frac{1.1752\ (0.5)^5}{20} \quad = \quad 0.0019$$

$$\max_{x\in[-1,1]}\Big|\cosh(x)\Big| \quad \leq \quad 1.5431 \quad \Rightarrow \quad |e^h(x)| \quad \leq \quad \frac{1.5431\ (0.5)^5}{20} \quad = \quad 0.0025$$

$$\max_{x\in[-\pi/2,\pi/2]}\Big|-\sin(x)+\cos(x)\Big| \quad \leq \quad \sqrt{2} \quad \Rightarrow \quad |e^h(x)| \quad \leq \quad \frac{\sqrt{(2)}(\pi/4)^5}{20} \quad = \quad 0.0212$$

10 *Il suffit d'écrire quelques lignes de commande* `python` *telles que*

```
 >>> from numpy import *
>>> T = [1975,1980,1985,1990]
>>> U = [70.2,70.2,70.3,71.2]
>>> a = polyfit(T,U,3)
>>> print(polyval(a,[1970,1995]))
[ 69.59999999  73.59999999]
```

On pourrait être tenté d'estimer l'erreur pour 1995 *par la différence entre la prédiction pour* 1970 *et la donnée disponible. On constate que l'estimation obtenue pour* 1970 *coïncide parfaitement avec la valeur réelle. Mais, ne croyez pas que l'estimation pour* 1995 *est aussi précise : elle est largement surestimée. En réalité, la durée de vie moyenne est de* 71.2*.*

Pour quatre points, la fonction `CubicSpline` *produit exactement le même résultat puisqu'on utilise comme conditions aux limites supplémentaires pour définir la courbe spline que le polynôme sur le premier et le dernier intervalle doit être identique à celui de l'intervalle qui suit ou qui précède. Pour trois intervalles, cela revient à dire qu'il n'y a qu'un polynôme de degré trois qui doit automatiquement coïncider avec le polynôme d'interpolation passant par les quatre points !*

```
>>> from scipy.interpolate import CubicSpline as spline
>>> help(spline)
    ...
 |      If ‘bc_type‘ is a string, then the specified condition will be applied
 |      at both ends of a spline. Available conditions are:
 |
 |      * ’not-a-knot’ (default): The first and second segment at a curve end
 |        are the same polynomial. It is a good default when there is no
 |        information on boundary conditions.
    ...
```

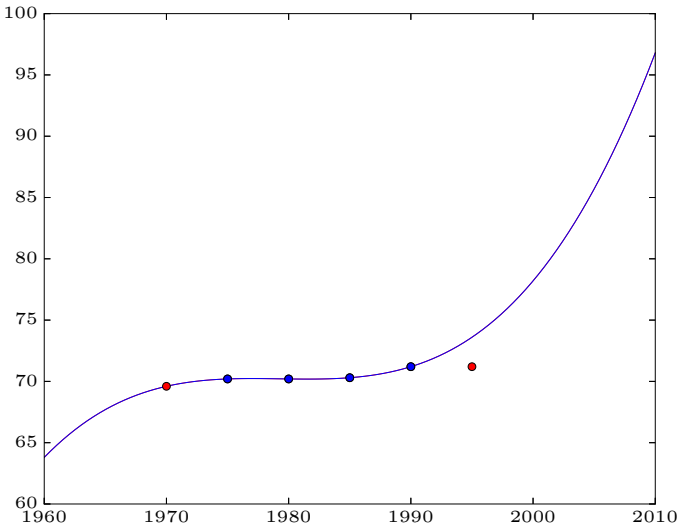


Figure 2.2: Toujours, être attentifs aux prédictions obtenues par extrapolation ! Les courbes du polynôme de Lagrange et de la spline cubique avec des conditions aux limites par défaut sont superposées sur la figure.

11 *Un polynôme d'interpolation ou une courbe spline est obtenu par le biais des instructions suivantes.*

```
from numpy import *
from scipy.interpolate import CubicSpline as spline

X = linspace(-1,1,21)
U = sin(2.0*pi*X)
pert = 1e-4
Upert = U + (-1)**arange(1,22)*pert

x = linspace(-1,1,1000)
uLag = polyval(polyfit(X,U,20),x)
uLagPert = polyval(polyfit(X,Upert,20),x)

uSpl = spline(X,U)(x)
uSplPert = spline(X,Upert)(x)
```

Lorsque nous utilisons des données non-perturbées, les graphes de `uLag` *et de* `uSpl` *coïncident parfaitement avec celui de la fonction sinus. Toutefois, la situation change totalement pour des données légèrement perturbées. On voit que le polynôme de Lagrange présente de très fortes oscillations aux extrémités de l'intervalle, tandis que l'interpolation au moyen des splines cubiques est pratiquement inchangée. En d'autres mots, on dira que l'interpolation par splines cubiques est, en général, nettement plus stable par rapport aux erreurs de perturbation des données.*

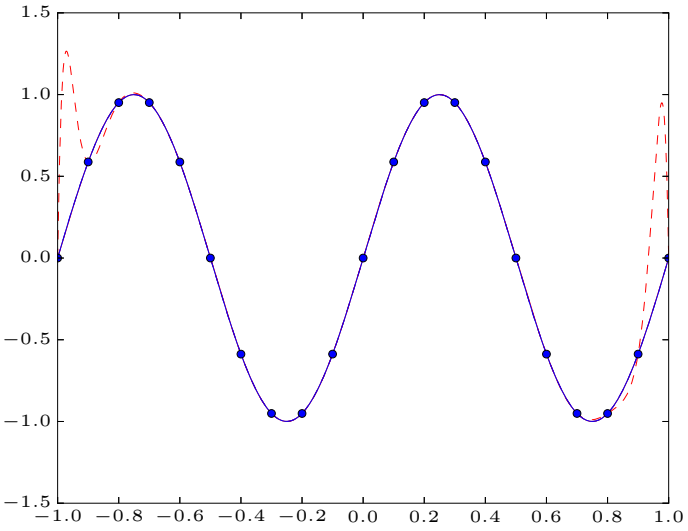


Figure 2.3: Le polynôme d'interpolation de Lagrange (en traits discontinus) passant par les données perturbées présente des oscillations parasites. Par contre, le polynôme de Lagrange pour les données non perturbées, ainsi que les courbes splines calculées pour les données perturbées et non perturbées sont semblables à l'échelle du dessin

Une façon de comprendre ce résultat est de voir les données perturbées comme les valeurs de la fonction

$$u_\epsilon(x) = \sin(2\pi x) + 10^{-4}\ \cos(10\pi x)$$

Les erreurs pour l'interpolation par splines cubiques et par un polynôme de Lagrange seront proportionnelles à la dérivée quatrième $u_\epsilon^{(4)}$ *ou la dérivée* $u_\epsilon^{(21)}$ *de cette fonction. Ces dérivées peuvent*

s'écrire sous la forme

$$u_\epsilon(x)^{(4)} = 16\ \sin(2\pi x)\ \pi^4 + \cos(10\pi x)\ \pi^4$$

$$u_\epsilon(x)^{(21)} = 2^{21}\ \cos(2\pi x)\pi^{21} - 10^{17}\ \sin(10\pi x)\ \pi^{21}$$

On comprend mieux ainsi que l'erreur de l'interpolation de Lagrange risque d'être très élevée en raison de l'introduction d'une perturbation de fréquence élevée...

12

1. *Une interpolation sur base des quatre sommets nécessite l'introduction des 4 paramètres. Le polynôme de degré le plus bas permettant une telle interpolation comprendra donc -par exemple- un terme constant et des termes en x, y et xy. La manière la plus simple de procéder est d'utiliser l'approche des fonctions de base de Lagrange associées à chaque sommet.*

   En imposant que la fonction de base associée à un sommet s'annule aux trois autres et vaille l'unité au sommet correspondant, on obtient les fonctions de Lagrange bilinéaires. Pour trouver les quatre coefficients de $\phi_1(x, y) = a + bx + cy + dxy$, on a ainsi utilisé les contraintes :

$$\phi_1(-1, -1) = 1$$

et

$$\phi_1(-1, 1) = \phi_1(1, 1) = \phi_1(1, -1) = 0.$$

Ces quatre fonctions valent bien l'unité à un sommet et s'annulent sur les 3 autres sommets comme on peut l'observer sur la Figure. Il s'agit de la simple généralisation de la démarche unidimensionnelle suivie pour les polynômes de Lagrange. Notons que ces fonctions bilinéaires sont obtenues par le produit des fonctions unidimensionnelles linéaires dans les deux directions : c'est pourquoi, on parle de fonctions bilinéaires.

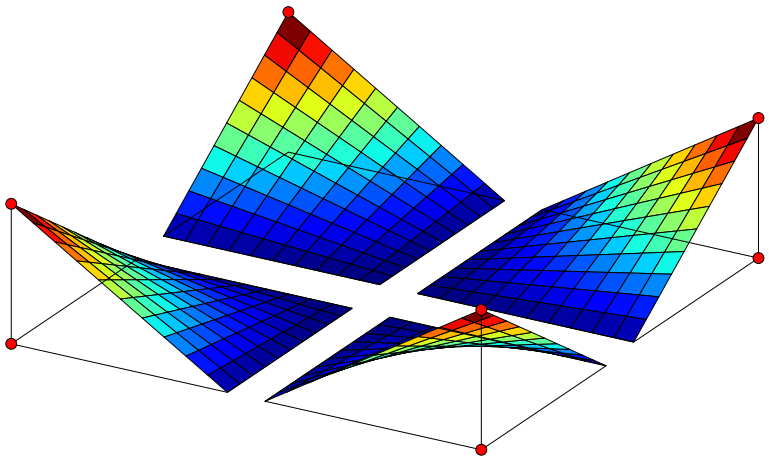


Figure 2.4: Fonctions de base de Lagrange bilinéaires sur le carré.

Les fonctions de base bilinéaires sont :

$$\begin{aligned}
\phi_1(x,y) &= \frac{(1-x)(1-y)}{4} \\
\phi_2(x,y) &= \frac{(1-x)(1+y)}{4} \\
\phi_3(x,y) &= \frac{(1+x)(1+y)}{4} \\
\phi_4(x,y) &= \frac{(1+x)(1-y)}{4}
\end{aligned}$$

L'interpolation de la température en un point quelconque est obtenue par :

$$t^h(x,y) = \sum_{i=1}^{4} T_i \phi_i(x,y)$$

Pour l'anecdote, c'est de cette manière que les graphiques bidimensionnels de `python` *sont réalisés :-) Cette approche est aussi utilisée dans la fonction* `interp2d` *de* `scipy.interpolate`.

2. *Une implémentation simple*[2] *est donnée par :*

```
from numpy import *
def interp(T,x,y):
  phi = array([ (1.0-x)*(1.0-y)/4.0,
                (1.0-x)*(1.0+y)/4.0,
                (1.0+x)*(1.0+y)/4.0,
                (1.0+x)*(1.0-y)/4.0 ])
  return T @ phi
```

3. *Il suffit d'ajouter au début de la fonction le test suivant.*

```
if ((x < -1.0) or (x > 1.0) or (y < -1.0) or (y > 1.0)):
  raise Exception('Extrapolation outside the square')
```

Pour les petits futés, il est possible de généraliser cette fonction afin qu'elle accepte des tableaux de même taille pour les données `x` *et* `y`. *La fonction retournera alors un tableau* `t` *de la même taille.*

13 *Il suffit de regarder les sources de la fonction* `CubicSplines` *de* `scipy.interpolate`...

```
>>> from scipy.interpolate import CubicSpline as spline
>>> import inspect
>>> lines = inspect.getsource(spline)
>>> print(lines)
```

[2] Observer toutefois l'usage de l'instruction `array` pour être certain que le produit scalaire entre les deux vecteurs puisse bien être effectué au cas où l'argument `T` serait une liste !

Approximation : solutions

14 *Il faut résoudre le système suivant :*

$$\begin{bmatrix} 3 & \sum_{i=0}^{2} X_i \\ \sum_{i=0}^{2} X_i & \sum_{i=0}^{2} X_i^2 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{2} U_i \\ \sum_{i=0}^{2} X_i U_i \end{bmatrix}.$$

1. *La solution du système est donnée par*

$$a = \frac{\left(\sum_{i=0}^{2} U_i\right)\left(\sum_{i=0}^{2} X_i^2\right) - \left(\sum_{i=0}^{2} X_i U_i\right)\left(\sum_{i=0}^{2} X_i\right)}{3\left(\sum_{i=0}^{2} X_i^2\right) - \left(\sum_{i=0}^{2} X_i\right)\left(\sum_{i=0}^{2} X_i\right)},$$

$$b = \frac{3\left(\sum_{i=0}^{2} X_i U_i\right) - \left(\sum_{i=0}^{2} X_i\right)\left(\sum_{i=0}^{2} U_i\right)}{3\left(\sum_{i=0}^{2} X_i^2\right) - \left(\sum_{i=0}^{2} X_i\right)\left(\sum_{i=0}^{2} X_i\right)}$$

2. $u^h(x) = \dfrac{22\ x - 2}{21}$

3. *L'erreur d'approximation est définie comme* $u(x) - u^h(x)$.
   On obtient donc très aisément $e^h(1/3) = 1/9 - 16/63 = -9/63 = -1/7$.
   Il s'agit d'une erreur relative de plus de 100% *alors que la valeur était fournie !*

15

1. $J(a,b) = \sum_{i=0}^{2} (U_i - a - bX_i)^2$

2. $$\begin{bmatrix} \dfrac{\partial J(a,b)}{\partial a} \\ \dfrac{\partial J(a,b)}{\partial b} \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{2} -2\,(U_i - a - bX_i) \\ \sum_{i=0}^{2} -2X_i\,(U_i - a - bX_i) \end{bmatrix}$$

3. $$\begin{bmatrix} \dfrac{\partial^2 J(a,b)}{\partial a^2} & \dfrac{\partial^2 J(a,b)}{\partial a \partial b} \\ \dfrac{\partial^2 J(a,b)}{\partial b \partial a} & \dfrac{\partial^2 J(a,b)}{\partial b^2} \end{bmatrix} = \begin{bmatrix} 6 & \sum_{i=0}^{2} 2X_i \\ \sum_{i=0}^{2} 2X_i & \sum_{i=0}^{2} 2X_i^2 \end{bmatrix}$$

   Cette matrice est une constante.
   C'est logique puisque $J(a,b)$ *est une fonction du second degré !*

4. *Le gradient doit s'annuler : ce sont les équations normales.*

5. *Le déterminant de la matrice hessienne doit être positif, ainsi que* J_{aa}. *L'information se trouve dans le livre de Thomas' Calculus, page 940... C'est une question classique en analyse ! On peut observer que les conditions du second ordre seront satisfaites, sauf si les trois abscisses sont identiques.*

6. *Ce système possède une solution unique sauf si les trois abscisses sont identiques !*

16 *Il suffit de considérer la première équation normale...*

$$\begin{aligned} (m+1)\ a + b\ \left(\sum_{i=0}^{m} X_i\right) &= \left(\sum_{i=0}^{m} U_i\right) \\ \Big\downarrow & \\ a + b\ \overline{X} &= \overline{U} \end{aligned}$$

17 *Les équations normales pour $n = m$ sont données par :*

$$\sum_{j=0}^{n} \left(\sum_{i=0}^{n} \phi_k(X_i)\ \phi_j(X_i) \right)\ a_j = \sum_{i=0}^{n} \phi_k(X_i) U_i \qquad k = 0, 1, \ldots, n$$

tandis que les équations de l'interpolation polynomiale sont données par :

$$\sum_{j=0}^{n} \phi_j(X_i)\ a_j = U_i \qquad i = 0, 1, \ldots, n$$

La manière la plus élégante de démontrer l'énoncé est d'écrire ces deux systèmes sous forme matricielle en définissant les éléments de la matrice $\mathbf{A}$ *comme étant* $A_{ij} = \phi_j(X_i)$*. On définit aussi deux vecteurs* $\mathbf{a}$ *et* $\mathbf{u}$ *dont les composantes sont respectivement* a_i *et* U_i*. Les deux systèmes s'écrivent respectivement :*

$$\mathbf{A}^T \cdot \mathbf{A} \cdot \mathbf{a} = \mathbf{A}^T \cdot \mathbf{u}$$

$$\mathbf{A} \cdot \mathbf{a} = \mathbf{u}$$

Si la matrice est régulière, les deux systèmes admettent la même solution !

18 *Une implémentation possible est donnée par :*

```
from numpy import *
from scipy.sparse import csr_matrix
from scipy.sparse import spdiags
from scipy.sparse.linalg import spsolve
from scipy.integrate import quad

import matplotlib.pyplot as plt
import matplotlib


def u(x):
  return (1/((x-0.3)**2 + 0.01) + 1/((x-0.9)**2 + 0.04) - 6)

def alphonse(X):

  """ Calcul de la meilleure approximation au sens de la norme L2
  de la fonction humps au moyen d'une fonction linéaire par morceaux
  sur l'intervalle ]X(0),X(-1)[ """

  n = len(X)       # nombre de points
  e = n-1          # nombre d'éléments
  h = diff(X)      # taille des éléments (c'est un vecteur !)

#
# -1- Assemblage de la matrice et du membre de droite
#     La matrice est calculée analytiquement et est tridiagonale
```

```
#     Le membre de droite est intégré numériquement via quad
#     -- Important : utilisation des matrices creuses !
#

  hlow = array([*h,0])/6
  hup  = array([0,*h])/6
  A = spdiags([hlow,2*(hup+hlow),hup],[-1,0,1],n,n)
  B = zeros((n,1))
  for elem in range(e):
    Xleft,Xright = X[elem],X[elem+1]
    B[elem]   += quad(lambda x: u(x)*(Xright-x)/(Xright-Xleft),Xleft,Xright)[0]
    B[elem+1] += quad(lambda x: u(x)*(x- Xleft)/(Xright-Xleft),Xleft,Xright)[0]

#
# -2- Résolution du système
#

  return spsolve(csr_matrix(A),B)
```

Sur la Figure **??**, *on peut observer le résultat obtenu avec* `X = linspace(0,3,10)`. *L'utilisation des matrices creuses et d'un solveur linéaire creux du package* `scipy.sparse` *rend le programme nettement plus rapide et tire profit du caractère tridiagonal du système à résoudre. Il est toutefois également possible de résoudre ce problème de petite taille avec un solveur linéaire usuel : la réalisation de cette implémentation nettement plus simple est laissée à votre sagacité : c'est une excellente manière d'améliorer vos talents de programmateur* `python`.

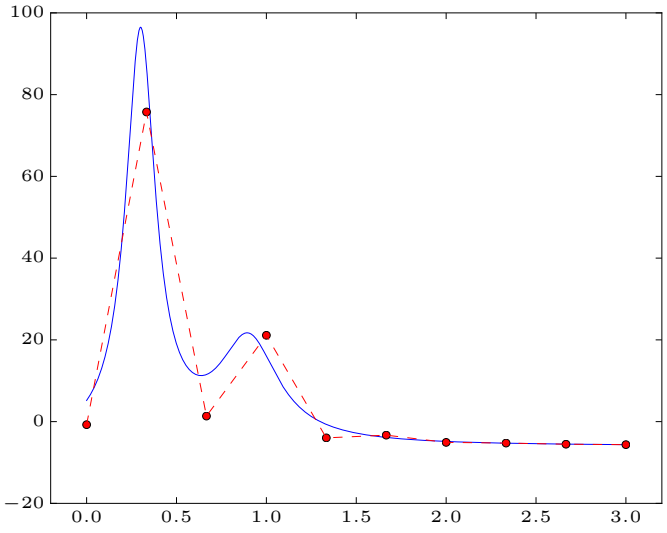


Figure 3.1: Approximation linéaire par morceaux au sens intégral des moindres carrés de $u(x) = \dfrac{1}{(x-0.3)^2+0.01} + \dfrac{1}{(x-0.9)^2+0.04} - 6$ sur l'intervalle $]0,3[$.

19 *Sur la Figure* **??**, *on voit que l'approximation qui minimise l'erreur globale n'est pas l'interpolation...*

1. *Théoriquement, il est logique que l'erreur globale de Paul soit supérieure à celle d'Alphonse. Alphonse va rechercher parmi toutes les fonctions linéaires par morceaux dont les noeuds se situent en X_i celle qui va minimiser E^h. Par contre, Paul se contente de sélectionner arbitrairement celle qui interpole la fonction aux abscisses : l'interpolation ne minimise pas E^h, mais minimise l'expression suivante*

$$\widetilde{E}^h = \sum_{i=0}^{n} \Big(u(X_i) - u^h(X_i) \Big)^2$$

2. *Lorsque $n \to \infty$, l'approximation d'Alphonse et l'interpolation de Paul vont toutes deux devenir de plus en plus proches de $u(x)$. L'écart entre elles va également diminuer puisque $\widetilde{E}^h$ va tendre vers E^h, du moins si on considère des abscisses uniformément réparties.*

20

Voir les notes du cours d'algèbre ! Il suffit de montrer que $< f, g >$ satisfait bien les conditions de la définition d'un produit scalaire.

1. *L'opérateur est bilinéaire : évident car l'intégrale d'une somme est la somme des intégrales.*
2. *L'opérateur est symétrique : à l'oeil nu... tu le vois, mon frère.*
3. *L'opérateur est défini positif : il n'y a que la fonction identiquement nulle sur l'intervalle dont l'intégrale du carré vaudra zéro.*

Courbes de Bézier, B-splines et NURBS : solutions

21

1. *Il faut montrer que les fonctions B-splines sont de classe $\mathcal{C}^2$. Lorsque $p = 3$, les fonctions $B_i^p(t)$ sont composées de quatre polynômes de degré trois définis sur les quatre intervalles existant entre $[T_i, T_{i+4}[$. Pour obtenir l'expression analytique, il suffit d'appliquer la définition, en considérant sans perte de généralité $\{T_0, T_1, T_2, T_3, T_4\} = \{0, 1, 2, 3, 4\}$.*

	$t \in [0,1]$	$t \in [1,2]$	$t \in [2,3]$	$t \in [3,4]$
$B_0^1(t) =$	t	$(2-t)$	0	0
$B_1^1(t) =$	0	$(t-1)$	$(3-t)$	0
$B_2^1(t) =$	0	0	$(t-2)$	$(4-t)$
$2B_0^2(t) =$	t^2	$(2-t)t$ $+(t-1)(3-t)$	$(3-t)^2$	0
$2B_1^2(t) =$	0	$(t-1)^2$	$(3-t)(t-1)$ $+(t-2)(4-t)$	$(4-t)^2$
$6B_0^3(t) =$	t^3	$(2-t)t^2$ $+(t-1)(3-t)t$ $+(t-1)^2(4-t)$	$(3-t)^2t$ $+(3-t)(t-1)(4-t)$ $+(t-2)(4-t)^2$	$(4-t)^3$

En développant ces expressions[3]*, on peut obtenir ensuite :*

	$t \in]-\infty, 0]$	$t \in [0,1]$	$t \in [1,2]$	$t \in [2,3]$	$t \in [3,4]$	$t \in [4, \infty[$
$6B_0^3(t) =$	0	t^3	$-3t^3 + 12t^2 - 12t + 4$	$\ldots$	$\ldots$	0
$6(B_0^3)'(t) =$	0	$3t^2$	$-9t^2 + 24t - 12$	$\ldots$	$\ldots$	0
$6(B_0^3)''(t) =$	0	$6t$	$-18t + 24$	$\ldots$	$\ldots$	0
$6(B_0^3)'''(t) =$	0	6	-18	18	-6	0

En calculant les expressions à droite et à gauche des quatre noeuds, on observe immédiatement que les fonctions B-splines de degré 3 sont de classe $\mathcal{C}^2$, mais pas de classe $\mathcal{C}^3$. Démontrer ce résultat dans le cas général est évidemment nettement plus compliqué et sort largement du cadre d'un cours d'introduction aux méthodes numériques.

2. *Pour démontrer que la somme des quatre fonctions qui existent sur un intervalle $[T_i, T_{i+1}]$ vaut l'unité, on utilise la propriété suivante lorsque l'écart entre deux noeuds contigus vaut l'unité.*

[3] En utilisant un argument de symétrie, on peut se contenter de calculer la première moitié du tableau, uniquement !

$$B_{i-1}^p(t) = B_i^p(t+1)$$

Cela permet d'obtenir directement les expressions des quatre fonctions B-splines de degré 3 *non nulles sur l'intervalle* $[0,1]$ *:*

$$\begin{aligned}
6B_0^3(t) &= t^3 \\
6B_{-1}^3(t) &= (1-t)(t+1)^2 + t(2-t)(t+1) + t^2(3-t) \\
6B_{-2}^3(t) &= (1-t)^2(t+2) + (1-t)(t+1)(2-t) + t(2-t)^2 \\
6B_{-3}^3(t) &= (1-t)^3
\end{aligned}$$

Et en développant les expressions,

$$\begin{aligned}
6B_0^3(t) &= t^3 \\
6B_{-1}^3(t) &= -3t^3 + 3t^2 + 3t + 1 \\
6B_{-2}^3(t) &= 3t^3 - 6t^2 + 4 \\
6B_{-3}^3(t) &= -t^3 + 3t^2 - 3t + 1
\end{aligned}$$

on vérifie bien que $\displaystyle\sum_{i=-3}^{0} B_i^3(t) = 1$.

L'exercice est un peu calculatoire, c'est vrai... Il vous fournit toutefois une expression analytique des fonctions de base pour des abscisses équidistantes. Si vous souhaitez obtenir des B-splines avec des noeuds équidistants, utiliser ces expressions sera la manière la plus efficace de travailler.

22

En utilisant adéquatement la définition des fonctions B-splines pour $\mathbf{T} = \{0,0,0,0,1,1,1,1\}$*. D'une part, on observe qu'il n'y aura toujours qu'un seul des deux termes de la somme qui n'aura pas un dénominateur nul dans l'expression de la récurrence. D'autre part, les coefficients de mélange seront toujours* $(1-x)$ *ou* x *! Ces observations permettent d'écrire finalement :*

$$\begin{aligned}
B_0^3(t) &= (1-t)^3 \\
B_1^3(t) &= 3(1-t)^2 t \\
B_2^3(t) &= 3(1-t)t^2 \\
B_3^3(t) &= t^3
\end{aligned}$$

Pour démontrer que la somme des quatre polynômes de Bernstein de degré trois vaut l'unité, il suffit d'écrire que :

$$1 = \Big((1-t)+t\Big)^3 = (1-t)^3 + 3(1-t)^2 t + 3(1-t)t^2 + t^3$$

$$\Big\downarrow$$

$$= \sum_{i=0}^{3} B_i^p(t)$$

□

Pour conclure, les polynômes de Bernstein de degré quelconque sont définis par la relation suivante.

$$\boxed{B_i^p(t) \quad = \quad C_i^p t^i (1-t)^{p-i} \qquad\qquad i = 0, \ldots, p}$$

où $C_i^p = p!/(i!(p-i)!)$ est le nombre binomial représentant le nombre de combinaisons possibles lorsqu'on retire i billes sans remise dans un lot de p billes différentes.

23

1. *La courbe*

$$\frac{x^2}{16} + \frac{y^2}{4} + \frac{xy}{8} = 1$$

peut s'écrire sous la forme matricielle suivante :

$$\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 16.$$

La courbe sera une ellipse si les valeurs propres de la matrice sont positives (ou plus exactement si le produit de celles-ci est positif) : ce qui est bien le cas.

2. *Les trois fonctions B-splines sont :*

$$\begin{aligned} B_0^2(t) &= (1-t)^2 \\ B_1^2(t) &= 2t(1-t) \\ B_2^2(t) &= t^2 \end{aligned}$$

3. *Une façon de résoudre le problème est de choisir $\mathbf{P}_0 = (0,2)$, $\mathbf{P}_1 = (X,Y)$, $\mathbf{P}_2 = (4,0)$ et $\mathbf{W} = (1,W,1)$. Il n'y a que trois inconnues (X, Y et W) mais cela laissera assez de liberté pour que les deux courbes coïncident. La courbe B-spline ainsi définie s'écrira :*

$$\begin{aligned} x(t) &= \frac{2WXt(1-t) + 4t^2}{(1-t)^2 + 2Wt(1-t) + t^2} \\ y(t) &= \frac{2(1-t)^2 + 2WYt(1-t)}{(1-t)^2 + 2Wt(1-t) + t^2} \end{aligned}$$

Il faut ensuite trouver X, Y et W afin que $x^2(t) + 2x(t)y(t) + 4y^2(t) = 16$, pour toute valeur de t comprise entre zéro et un. Ce qui revient à exiger que :

$$\begin{aligned} &\Big(2WXt(1-t) + 4t^2 \Big)^2 \\ &\quad + 2\Big(2WXt(1-t) + 4t^2 \Big)\Big(2(1-t)^2 + 2WYt(1-t) \Big) \\ &\quad + 4\Big(2(1-t)^2 + 2WYt(1-t) \Big)^2 \quad = \quad 16\Big((1-t)^2 + 2Wt(1-t) + t^2 \Big)^2 \end{aligned}$$

$$\downarrow$$

$$\begin{aligned} &4W^2X^2t^2(1-t)^2 + 16t^4 + 16WXt^3(1-t) \\ &\quad + 8WXt(1-t)^3 + 8W^2XYt^2(1-t)^2 \\ &\quad + 16t^2(1-t)^2 + 16WYt^3(1-t) \\ &\quad + 16(1-t)^4 + 16W^2Y^2t^2(1-t)^2 \\ &\quad + 32WYt(1-t)^3 \quad = \quad 16(1-t)^4 + 64W^2t^2(1-t)^2 \\ &\qquad\qquad\qquad\qquad + 16t^4 + 64Wt(1-t)^3 \\ &\qquad\qquad\qquad\qquad + 64Wt^3(1-t) + 32t^2(1-t)^2 \end{aligned}$$

Pour que l'égalité soit vraie pour tout t, tous les coefficients des polynômes doivent être identiques.

t^4	$16 = 16$
$(1-t)^4$	$16 = 16$
$t^2(1-t)^2$	$4W^2X^2 + 8W^2XY + 16 + 16W^2Y^2 = 64W^2 + 32$
$t^3(1-t)$	$16WX + 16WY = 64W$
$t(1-t)^3$	$8WX + 32WY = 64W$

Les deux dernières équations deviennent alors

$$\begin{aligned} 2X + 2Y &= 8 \\ X + 4Y &= 8 \\ &\downarrow \\ X &= 8/3 \\ Y &= 4/3 \end{aligned}$$

De la troisième équation, on peut ensuite déduire que $W^2 = 3/4$. La réponse finale est donc :

$$\boxed{X = \frac{8}{3} \qquad Y = \frac{4}{3} \qquad W = \sqrt{\frac{3}{4}}}$$

4. *Le calcul précédent constitue la démonstration demandée... Notons que le point (X, Y) pouvait être très facilement obtenu comme l'intersection des deux tangentes de l'ellipse en A et en B. L'observation attentive et préliminaire de la figure du syllabus aurait dû vous mettre sur la bonne piste....*

24 *L'application successive des étapes de l'algorithme de Paul de Casteljau s'écrit comme suit :*

$$\begin{aligned} \boldsymbol{P}_{3,3} &= (1-\xi)\boldsymbol{P}_{2,2} &&+ \xi\boldsymbol{P}_{3,2} \\ &= (1-\xi)\left[(1-\xi)\boldsymbol{P}_{1,1} + \xi\boldsymbol{P}_{2,1}\right] &&+ \xi\left[(1-\xi)\boldsymbol{P}_{2,1} + \xi\boldsymbol{P}_{3,1}\right] \\ &= (1-\xi)[(1-\xi)[(1-\xi)\boldsymbol{P}_0 + \xi\boldsymbol{P}_1] &&+ \xi[(1-\xi)\boldsymbol{P}_1 + \xi\boldsymbol{P}_2]] \\ &\quad + \xi[(1-\xi)[(1-\xi)\boldsymbol{P}_1 + \xi\boldsymbol{P}_2] &&+ \xi[(1-\xi)\boldsymbol{P}_2 + \xi\boldsymbol{P}_3]] \end{aligned}$$

En remettant un peu d'ordre dans cette expression, et en posant $\boldsymbol{P}_{3,3} = \mathbf{u}^h(\xi)$, on obtient la forme voulue

$$\begin{aligned} \mathbf{u}^h(\xi) &= (1-\xi)^3\boldsymbol{P}_0 + 3(1-\xi)^2\xi\boldsymbol{P}_1 + 3(1-\xi)\xi^2\boldsymbol{P}_2 + \xi^3\boldsymbol{P}_3 \\ &= \sum_i \phi_i(\xi)\boldsymbol{P}_i \end{aligned}$$

où $\phi_0(\xi) = (1-\xi)^3$, $\phi_1(\xi) = 3(1-\xi)^2\xi$, $\phi_2(\xi) = 3(1-\xi)\xi^2$ et $\phi_3(\xi) = \xi^3$ sont les différentes fonctions de forme et correspondent bien aux quatre polynômes de Bernstein.

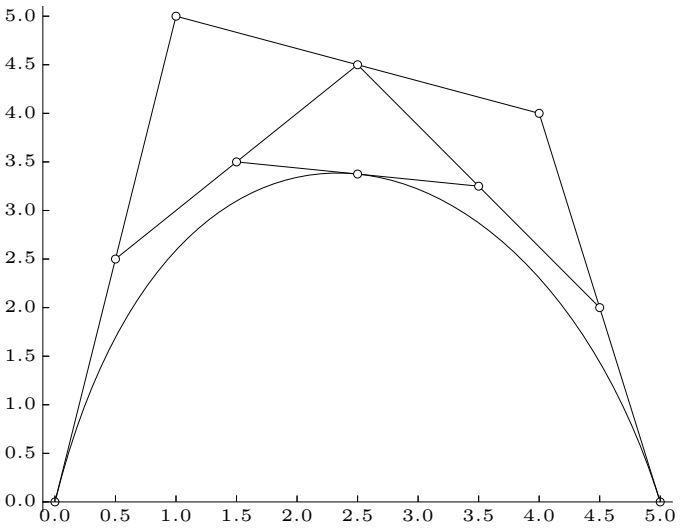


Figure 4.1: Interprétation graphique de l'algorithme de Paul de Casteljau ($\xi = 0.5$)

25

Pour que les deux cubiques coïncident, il faut leur imposer quatre conditions disctinctes. Il faut toutefois tenir compte que les représentations paramétriques sont distinctes.

Un choix astucieux (car fournissant des relations simples) est d'écrire :

$$\begin{aligned}
\boldsymbol{u}(0) &= \boldsymbol{v}(0), \\
\boldsymbol{u}(1/2) &= \boldsymbol{v}(1), \\
\left.\frac{\partial \boldsymbol{u}}{\partial t}\right|_{t=0} &= \left.\frac{\partial \boldsymbol{v}}{\partial \eta}\right|_{\eta=0} \frac{\partial \eta}{\partial t}, \\
\left.\frac{\partial^2 \boldsymbol{u}}{\partial t^2}\right|_{t=0} &= \left.\frac{\partial^2 \boldsymbol{v}}{\partial \eta^2}\right|_{\eta=0} \left(\frac{\partial \eta}{\partial t}\right)^2 .
\end{aligned}$$

Il faut ensuite écrire les dérivées des polynômes de Bernstein. Ensuite en développant les expressions en termes de points de contrôle, on obtient

$$\begin{aligned}
\mathbf{P}_0 &= \mathbf{V}_0 \\
\mathbf{P}_0 + 3\mathbf{P}_1 + 3\mathbf{P}_2 + \mathbf{P}_3 &= 8\mathbf{V}_3 \\
-3\mathbf{P}_0 + 3\mathbf{P}_1 &= 2(-3\mathbf{V}_0 + 3\mathbf{V}_1) \\
6\mathbf{P}_0 - 12\mathbf{P}_1 + 6\mathbf{P}_2 &= 4(6\mathbf{V}_0 - 12\mathbf{V}_1 + 6\mathbf{V}_2) \\
&\downarrow \\
\mathbf{V}_0 &= \mathbf{P}_0 \\
\mathbf{V}_1 &= (\mathbf{P}_0 + \mathbf{P}_1)/2 \\
\mathbf{V}_2 &= (\mathbf{P}_0 + 2\mathbf{P}_1 + \mathbf{P}_2)/4 \\
\mathbf{V}_3 &= (\mathbf{P}_0 + 3\mathbf{P}_1 + 3\mathbf{P}_2 + \mathbf{P}_3)/8
\end{aligned}$$

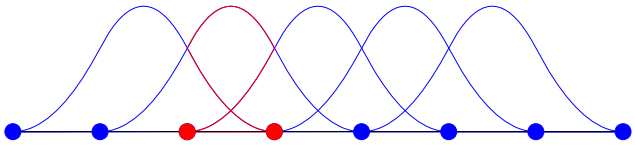
26

1. *L'interpolation polynomiale paramétrique :*

$$\begin{array}{rcl} x_L(t) &=& \sum_{i=0}^{3} X_i \phi_i(t) \\ \downarrow & & \\ &=& \phi_1(t) \\ &=& \dfrac{(t-0)(t-2)(t-3)}{(1-0)(1-2)(1-3)} \\ &=& \dfrac{t(t-2)(t-3)}{2} \end{array} \qquad \begin{array}{rcl} y_L(t) &=& \sum_{i=0}^{3} Y_i \phi_i(t) \\ \downarrow & & \\ &=& \phi_2(t) \\ &=& \dfrac{(t-0)(t-1)(t-3)}{(2-0)(2-1)(2-3)} \\ &=& \dfrac{t(t-1)(t-3)}{-2} \end{array}$$

   En $t = \frac{1}{2}$, on en déduit la position : $(\frac{15}{16}, -\frac{5}{16})$.
   Un nombre non négligeable d'étudiants vont obtenir ce résultat en faisant usage d'une algèbre surdimensionnée et complexe et s'arrêtent péniblement ici... Et pourtant, l'utilisation de polynômes de Lagrange simplifie largement les calculs : il n'est ni demandé, ni nécessaire, ni utile de développer les expressions. En général, c'est même une source d'erreurs de calcul.

2. *Il suffit d'esquisser l'allure des 5 fonctions de base qui interviennent :*



3. *En définissant* $(X_4, Y_4) = (X_1, Y_1)$*, l'approximation B-spline de degré deux s'écrit :*

$$\begin{array}{rcl} x_B(t) &=& \sum_{i=0}^{4} X_i B_i^2(t) \\ \downarrow & & \\ &=& B_1^2(t) + B_4^2(t) \end{array} \qquad \begin{array}{rcl} y_B(t) &=& \sum_{i=0}^{4} Y_i B_i^2(t) \\ \downarrow & & \\ &=& B_2^2(t) \end{array}$$

   Pour développer cette expression sur l'intervalle $[0, 1[$*, il faut uniquement évaluer les deux fonctions B-splines :* $B_1^2(t)$ *et* $B_2^2(t)$*, car la fonction* $B_4^2(t)$ *s'annule sur l'intervalle* $[0, 1[$*, comme on le voit clairement sur la représentation des fonctions de base.*

$$\begin{array}{rcl} B_1^2(t) &=& \dfrac{(t+1)}{2} B_1^1(t) + \dfrac{(2-t)}{2} B_2^1(t) \\ & & \\ & & \textit{Comme } B_1^1(t) = (1-t) \textit{ et } B_2^1(t) = t. \\ & & \\ &=& \dfrac{1+2t-2t^2}{2} \end{array} \qquad \begin{array}{rcl} B_2^2(t) &=& \dfrac{t}{2} B_2^1(t) \\ & & \\ & & \textit{Comme } B_2^1(t) = t. \\ & & \\ &=& \dfrac{t^2}{2} \end{array}$$

   On obtient donc :

$$x_B(t) = \frac{1+2t-2t^2}{2} \qquad y_B(t) = \frac{t^2}{2}$$

En $t = \frac{1}{2}$, on en déduit la position : $(\frac{3}{4}, \frac{1}{8})$. En parcourant les points de contrôle dans un autre ordre, deux autres positions peuvent être déduites et sont également correctes... (et donc, acceptées par le correcteur)

4. *Pour assurer au mieux une continuité C^2, une façon de procéder serait d'ajouter au début et à la fin du vecteur des données, les coordonnées des deux derniers et des deux premiers points... Notons qu'il s'agirait d'une solution très similaire à celle que l'on a adoptée pour l'approximation B-splines ! C'est une option assez simple et efficace, mais elle n'assure pas une vraie égalité entre les dérivées secondes du point de départ et du point d'arrivée. En effet, même si on repousse dans le passé la condition initiale, elle continuera à avoir un impact (marginal, mais réel) sur la trajectoire....*

```
X = [1,0,0,1,0,0,1,0]
Y = [0,1,0,0,1,0,0,1]
T = [0,1,2,3,4,5,6,7]
t = linspace(2,5,100)
plt.plot(spline(T,X)(t),spline(T,Y)(t),'-g')
```

Obtenir une vraie solution périodique avec les splines usuelles nécessite de modifier le système linéaire à résoudre en imposer une égalité entre les dérivées secondes entre les deux extrémités... Obtenir cette implémentation est laissé à votre sagacité.

Intégration : solutions

27

On observe que l'intégrale de u est approchée par l'intégrale du polynôme qui l'interpole en trois points. Pour répondre à la question, il suffit d'écrire le polynôme en utilisant les fonctions de Lagrange.

$$
\begin{aligned}
\int_{-h}^{h} u(x)dx \quad &\approx \quad \overbrace{\int_{-h}^{h} u^h(x)dx}^{I^h} \\
&\downarrow \\
&\approx \quad \sum_{i=0}^{2} u(X_i) \underbrace{\int_{-h}^{h} \phi_i(x)dx}_{w_i} \\
&\downarrow \qquad \textit{En fixant } X_0 = -h,\ X_1 = 0 \textit{ et } X_2 = h \\
&\approx \quad u(-h) \underbrace{\int_{-h}^{h} \frac{(x-h)x}{2h^2}\, dx}_{w_0} \\
&\qquad\qquad + u(0) \underbrace{\int_{-h}^{h} \frac{(x-h)(x+h)}{-h^2}\, dx}_{w_1} \\
&\qquad\qquad + u(h) \underbrace{\int_{-h}^{h} \frac{(x+h)x}{2h^2}\, dx}_{w_2}
\end{aligned}
$$

Les trois poids sont obtenus immédiatement en intégrant les trois fonctions de base de Lagrange :

$$
\begin{aligned}
w_0 &= \int_{-h}^{h} \frac{(x-h)x}{2h^2}\, dx = \left[\frac{x^3}{6h^2} - \frac{x^2}{4h}\right]_{-h}^{h} = \frac{h}{3} \\
w_1 &= \int_{-h}^{h} \frac{(h^2-x^2)}{h^2}\, dx = \left[x - \frac{x^3}{3h^2}\right]_{-h}^{h} = 2h - \frac{2h}{3} = \frac{4h}{3} \\
w_2 &= \int_{-h}^{h} \frac{(x+h)x}{2h^2}\, dx = \left[\frac{x^3}{6h^2} + \frac{x^2}{4h}\right]_{-h}^{h} = \frac{h}{3}
\end{aligned}
$$

28

L'exercice consiste à découvrir et à analyser une nouvelle méthode numérique relativement simple : la méthode d'intégration des rectangles.

1. *L'erreur locale d'intégration sur un intervalle $[0, h]$ (qui représente n'importe quel intervalle $[X_i, X_{i+1}]$, sans aucune perte de généralité !) est la différence entre l'intégration de $u(x)$ et son approximation constante $u^h = u(0)$. En notant $|f'(\xi)|$, le maximum de la valeur absolue de f' sur l'intervalle $[a, b]$, on peut écrire que l'erreur locale d'intégration E^h_{loc} est bornée comme suit :*

$$E^h_{loc} \leq \int_0^h \underbrace{|u(x) - u^h|}_{\leq |f'(\xi)|x} dx$$

$\downarrow$ *Car dans* $[2,7]$, *la fonction* $|f'(x)| = \frac{1}{x^2}$ *a son maximum en* $x = 2$

$$\leq \frac{1}{4} \underbrace{\int_0^h x \; dx}_{\frac{h^2}{2}}$$

$\downarrow$ *Car* $nh = (b-a) = 5$ *pour une méthode composite*

$$\leq \frac{25}{8n^2}$$

La réponse est donc : $\boxed{B_n = \frac{25}{8n^2}}$

Il est possible de trouver ce résultat immédiatement en réalisant un graphique approximatif du problème. On voit que l'erreur locale est maximale sur le premier intervalle et peut être bornée par la superficie du triangle de hauteur $-hf'(2) = h/4$ *et de base* h. *Et on en déduit très facilement que* $B_n = \frac{h^2}{8} = \frac{25}{8n^2}$.

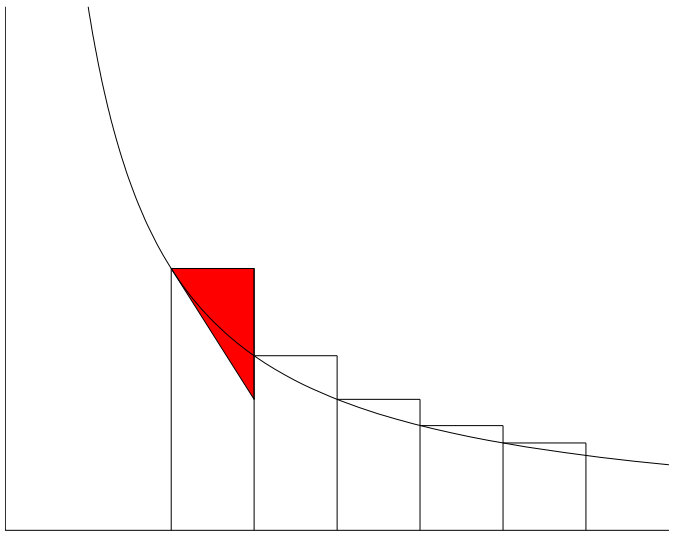


Figure 5.1: Représentation graphique de B_n pour $n = 5$ et $h = 1$

2. *Du résultat précédent, on déduit immédiatement que :* $\boxed{E^h \leq nB_n = \frac{25}{8n} = \frac{5h}{8}}$

Comme cette erreur est en $\mathcal{O}(h)$, *il s'agit donc d'une méthode d'ordre 1.*
On peut aussi déduire l'ordre directement en observant qu'une fonction linéaire non constante

ne sera pas intégrée de manière exacte : c'est une bonne idée. Ne pas confondre E^h et B_n ! C'est une erreur difficile à pardonner, car c'est un des messages essentiels du cours de méthodes numériques !

3. *Il suffit d'écrire :*

$$\begin{array}{rcl} E^h & \leq & 10^{-3} \\ & \downarrow & \\ \underbrace{\frac{25000}{8}}_{3125} & \leq & n \end{array}$$

La réponse est donc : $\boxed{n \geq 3125}$.

Ce résultat est sûr car il est basé sur une borne supérieure. En d'autres mots, avec 3125 *points, on est vraiment (si, si !) certain d'obtenir une précision meilleure que* 10^{-3}.
Le pas $h = \frac{5}{3125} = 0,0016$ *est petit mais encore suffisamment grand pour n'avoir aucun problème avec le calcul en virgule flottante. Votre ordinateur ou votre calculatrice n'auront donc pas de comportement facétieux... et il ne fallait donc pas me parler du danger des erreurs d'arrondi. Eh oui, on peut parfois avoir confiance aux résultats théoriques de méthodes numériques.*

4. *Il suffit d'effectuer une extrapolation de Richardson pour une méthode d'intégration linéaire (attention, pas exactement comme Romberg !!!).*

$$\begin{array}{cccc} h & I_{0,0} = 1,2617 & \searrow & \\ \frac{h}{2} & I_{1,0} = 1,2572 & \longrightarrow & I_{1,1} = 2I_{1,0} - I_{0,0} = 1,2527 \\ & \mathcal{O}(h) & & \mathcal{O}(h^2) \end{array}$$

On obtient ainsi un résultat très précis. L'intégrale exacte vaut $1,2528$. *Il est possible de montrer que l'erreur de l'extrapolation de Richardson est bornée par la fraction*

$$\frac{5h^2}{16} = 7,8125\ 10^{-4}$$

On observe ainsi que la borne sur l'erreur de l'approximation fournie par cette simple combinaison linéaire est bien meilleure que celle obtenue en appliquant la méthode des rectangles avec 3125 points. A méditer !

29

Les résultats numériques sont :

1. *Rectangles :* 1.512437 , 1.633799 , 1.709705 *;*
2. *Trapèzes :* 1.727222 , 1.719713 , 1.718296 *;*
3. *Simpson :* 1.718319 , 1.718283 , 1.718282 *;*
4. *Gauss-Legendre :* 1.717896 , 1.718282 *;*
5. *Romberg :* 1.718282.

30

La méthode de Romberg permet d'obtenir 0.987435.
Les données correspondent à la fonction

$$u(x) = arcos(x)$$

et l'intégrale exacte vaut exactement un.

31

$J_1(2) = 0.576725$.

32

On écrit que

$$y^h(x) \;=\; Y_0 \frac{(x - X_1)}{(X_0 - X_1)} + Y_1 \frac{(x - X_0)}{(X_1 - X_0)}.$$

On trouve alors

$$I \;=\; \frac{(X_1 - X_0)}{2} (Y_0 + Y_1).$$

C'est la méthode des trapèzes. C'est normal puisque cette méthode consiste à remplacer la fonction à intégrer par le segment de droite $y^h(x)$.

33

Pour que les pas soient en progression géométrique, il faut considérer les abscisses 0.2, 0.6 et 1.8, soit k=3. On trouve alors $f(0) = 9.25$. Ce n'est toutefois pas la meilleure idée. On peut également calculer une extrapolation polynomiale en utilisant toutes les données et comparer la valeur obtenue avec la prédiction basée sur seulement trois données. Evidemment, cela nécessite davantage de calculs mais avec un outil tel que `python`*, le résultat sera immédiat.*

34

On procède comme suit

$$\frac{Erreur(h = 0.2)}{Erreur(h = 0.1)} = \frac{0.009872}{0.001235} = 7.99 \approx 2^3$$

La méthode est donc d'ordre trois.

35

Comme la fonction n'est pas définie en $x = 0$, il faut utiliser une méthode ouverte.

Par exemple, la méthode de Gauss-Legendre, on trouve 1.750863 *(3 points) et* 1.841600 *(5 points). La valeur exacte est 2. L'erreur de troncature est encore importante, il faudrait prendre plus de points en considérant plusieurs sous-intervalles.*

Dérivation : solutions

36

1. *On écrit les développements en série de Taylor comme suit :*

$$\begin{aligned} u(-h) &= U_0 - hU_0' + \frac{h^2}{2}U_0'' - \frac{h^3}{6}U_0^{(3)} + \frac{h^4}{24}U_0^{(4)} + \dots \\ u(-2h) &= U_0 - 2hU_0' + \frac{4h^2}{2}U_0'' - \frac{8h^3}{6}U_0^{(3)} + \frac{16h^4}{24}U_0^{(4)} + \dots \\ u(-3h) &= U_0 - 3hU_0' + \frac{9h^2}{2}U_0'' - \frac{27h^3}{6}U_0^{(3)} + \frac{81h^4}{24}U_0^{(4)} + \dots \end{aligned}$$

En substituant ces développements dans la formule d'Yves, on obtient alors :

$$\begin{aligned} &\frac{(2+a+4+b)}{ch^2}U_0 - \frac{h(a+8+3b)}{ch^2}U_0' + \frac{h^2(a+16+9b)}{2ch^2}U_0'' \\ &\quad - \frac{h^3(a+32+27b)}{6ch^2}U_0^{(3)} + \frac{h^4(a+64+81b)}{24ch^2}U_0^{(4)} + \dots \end{aligned}$$

Pour annuler les termes en U_0 et U_0', il faut que $a+b+6=0$ et $a+3b+8=0$.
Ce qui est obtenu en prenant $a=-5$ et $b=-1$.
Pour ensuite obtenir un coefficient unitaire devant U_0'', il convient de sélectionner $c=1$.
Avec ces valeurs, le coefficient de U_0''' s'annule également et on obtient :

$$\boxed{u''(0) = \frac{(2U_0 - 5U_{-h} + 4U_{-2h} - U_{-3h})}{h^2} - \frac{22h^2}{24}u^{(4)}(\zeta)}$$

2. *On vient d'obtenir le terme d'erreur : l'ordre de cette formule est 2.*
3. *Il suffit de minimiser* $f(h) = \dfrac{12\epsilon}{h^2} + \dfrac{22}{24}\,4h^2$.

$$\begin{aligned} \text{On exige donc que } f'(h) = \frac{-24\epsilon}{h^3} + \frac{44h}{6} &= 0 \\ \downarrow & \\ h^4 &= \frac{36}{11}\,\epsilon \end{aligned}$$

On conclut que le pas optimal est donné par : $\boxed{h = \sqrt[4]{\dfrac{36}{11}\,\epsilon}}$

37

1. *On peut prendre une différence centrée d'ordre deux avec $h=0.1$ et l'on trouve $P'(X \leq 1.2) \approx 0.194328$. En prenant ensuite $h=0.2$, on trouve $P'(X \leq 1.2) \approx 0.1947465$.*
On effectue ensuite une extrapolation de Richardson (en tenant compte qu'il n'y a pas de termes d'erreur de degré impair !) pour obtenir l'estimation suivante :

$$P'(X \leq 1.2) \approx 0.1941885$$

qui est une approximation d'ordre quatre. La valeur exacte est donnée par

$$P'(X \leq 1.2) = \frac{1}{\sqrt{2\pi}} \exp \frac{-(1.2)^2}{2} = 0.194186055$$

L'erreur commise est de $2.4450\ 10^{-6}$.

2. *Une différence centrée d'ordre deux fournit directement :*

$$P''(X \leq 1.2) \approx -0.232720$$

38 *Il suffit d'écrire les développements en série de Taylor suivants :*

$$u(x \pm h) = u(x) \pm hu'(x) + \frac{h^2}{2}u''(x) \pm \frac{h^3}{6}u'''(x) + \frac{h^4}{24}u^{(4)}(x) \pm \frac{h^5}{120}u^{(5)}(x) + \mathcal{O}(h^6)$$

$$u(x \pm 2h) = u(x) \pm 2hu'(x) + \frac{4h^2}{2}u''(x) \pm \frac{8h^3}{6}u'''(x) + \frac{16h^4}{24}u^{(4)}(x) \pm \frac{32h^5}{120}u^{(5)}(x) + \mathcal{O}(h^6)$$

On effectue ensuite la combinaison linéaire de la relation à démontrer. On peut observer immédiatement que tous les termes impairs disparaissent par symétrie et on obtient finalement

$$\begin{aligned}
&\frac{u(x-2h) - 4u(x-h) + 6u(x) - 4u(x+h) + u(x+2h)}{h^4} \\
&= \left(\underbrace{1 - 4 + 6 - 4 + 1}_{0}\right)\frac{u(x)}{h^4} + \left(\underbrace{8h^2 - 8h^2}_{0}\right)\frac{u''(x)}{2h^4} + \left(\underbrace{32h^4 - 8h^4}_{24h^4}\right)\frac{u''''(x)}{24h^4} + \frac{\mathcal{O}(h^6)}{h^4} \\
&= u''''(x) + \mathcal{O}(h^2)
\end{aligned}$$

Pour obtenir la meilleure estimation de l'ordre de l'erreur, il faut considérer des développements jusqu'au terme d'ordre six ! La justification adéquate de l'ordre de l'erreur fait évidemment partie de la réponse à fournir.

Il est également possible d'obtenir cette relation en utilisant le fait qu'une dérivée quatrième est une dérivée seconde d'une dérivée seconde et d'appliquer récursivement la relation d'une différence centrée du second ordre.

39

1. *La différence centrée s'écrit en prenant respectivement* h *et* $2h$.

$$\begin{aligned}
D_h &= \frac{u(x+h) - u(x-h)}{2h} \\
D_{2h} &= \frac{u(x+2h) - u(x-2h)}{4h}
\end{aligned}$$

Puisqu'il s'agit d'une approximation d'ordre deux, l'extrapolation de Richardson s'écrit :

$$\begin{aligned}
u'(x) = \frac{4D_h - D_{2h}}{3} &= \frac{4u(x+h) - 4u(x-h)}{6h} - \frac{u(x+2h) - u(x-2h)}{12h} \\
&= \frac{-u(x+2h) + 8u(x+h) - 8u(x-h) + u(x-2h)}{12h} \qquad \square
\end{aligned}$$

2. *Il s'agit d'une différence centrée : il n'y a pas de termes d'erreur de degré impair. On peut donc prédire que cette estimation sera d'ordre quatre.*

3. *En utilisant* $h = 0.1$ *et* $h = 0.3$*, on trouve respectivement* 0.999996666 *et* 0.999727092. *On peut donc écrire que :*

$$\frac{Erreur(h = 0.3)}{Erreur(h = 0.1)} = \frac{0.000272908}{0.000003333} = 81.8 \approx 3^4$$

L'expérience numérique est donc en accord avec la théorie...

40 *La borne de troncature théorique* E^h *est donnée par*

$$E^h = \frac{h^2}{6} u'''(\xi)$$

avec ξ *sur l'intervalle* $[2 - h, 2 + h]$*. Dans le cas de l'exponentielle, obtenir la dérivée troisième ne pose aucune difficulté. Et on peut donc borner* $u'''(\xi)$ *par* $\exp(2 + h)$*.*

La comparaison entre la borne d'erreur théorique et l'erreur observée est -ici- remarquable. C'est évidemment dû au fait que l'on connaît la dérivée troisième de la fonction et qu'il est aisé de la borner.

h_k	D^{h_k}	$\exp(2) - D^{h_k}$	E^{h_k}
$h_1 = 0.1$	*7.401377351441907*	*0.012321252511256*	*0.013610283187613*
$h_2 = 0.01$	*7.389179250481304*	*0.000123151550653*	*0.000124388622455*
$h_3 = 0.001$	*7.389057330439375*	*0.000001231508724*	*0.000001232741475*

Pour les pas considérés, les erreurs d'arrondi sont encore totalement négligeables.

41

1. *En vertu de la définition de* $u^h(x)$*, il est immédiat que* $u^h(X_0) = u(X_0)$*. Ensuite, nous avons que :*

$$\begin{aligned} u^h(X_1) &= u(X_0) + \left(\frac{u(X_0 + h) - u(X_0)}{h}\right)(X_1 - X_0) \\ &= u(X_0) + \left(\frac{u(X_1) - u(X_0)}{h}\right) h \;=\; u(X_1) \end{aligned}$$

Finalement, on procède de la même manière pour :

$$\begin{aligned} u^h(X_2) &= u(X_0) + \left(\frac{u(X_1) - u(X_0)}{h}\right) 2h \\ &\quad + \left(\frac{u(X_2) - 2u(X_1) + u(X_0)}{2h^2}\right) 2h^2 \\ &= u(X_0) + 2u(X_1) - 2u(X_0) + u(X_2) - 2u(X_1) + u(X_0) \;=\; u(X_2) \end{aligned}$$

Donc, la fonction u^h *est le polynôme du second degré qui interpole* u *en* X_0*,* X_1 *et* X_2*.*

2. *Définissons la fonction* $g(x) = u(x) - u^h(x)$*. Comme* $g(X_0) = g(X_1) = g(X_2) = 0$ *et comme la fonction* g *est continûment dérivable, le théorème de Rolle permet de déduire que :*

$$\begin{aligned} &\exists \xi_1 \in [X_0, X_1] \text{ tel que } g'(\xi_1) = 0 \text{ ce qui revient à : } u'(\xi_1) = (u^h)'(\xi_1) \\ &\exists \xi_2 \in [X_1, X_2] \text{ tel que } g'(\xi_2) = 0 \text{ ce qui revient à : } u'(\xi_2) = (u^h)'(\xi_2) \end{aligned}$$

3. *Comme $g'(\xi_1) = g'(\xi_2) = 0$ et comme la fonction g' est continûment dérivable, le théorème de Rolle permet à nouveau de déduire que :*

$$\exists \xi_3 \in [\xi_1, \xi_2] \text{ tel que } g''(\xi_3) = 0 \text{ ce qui revient à : } u''(\xi_3) = (u^h)''(\xi_3)$$

4. *Considérons d'abord le cas où $x \in [X_0, X_1]$. L'autre cas se traite de manière analogue.*

$$\begin{aligned}
u(x) - u^h(x) &= g(x) \\
&\downarrow \quad \textit{Comme } g(X_0) = 0, \\
u(x) - u^h(x) &= g(x) - g(X_0) \;=\; \int_{X_0}^{x} g'(s)ds \\
&\downarrow \\
|u(x) - u^h(x)| &\leq \int_{X_0}^{x} |g'(s)|ds \;\leq\; h \max_{\zeta \in [X_0, X_1]} |g'(\zeta)|. \\
&\downarrow \quad \textit{Comme } g'(\xi_1) = 0, \\
|u(x) - u^h(x)| &\leq h \max_{\zeta \in [X_0, X_1]} |g'(\zeta) - g'(\xi_1)| \;\leq\; h \left| \int_{\xi_1}^{\zeta} |g''(s)|\, ds \right| \\
&\leq h \int_{X_0}^{X_1} |g''(s)|\, ds \;\leq\; h^2 \max_{\eta \in [X_0, X_1]} |g''(\eta)| \\
&\downarrow \quad \textit{Comme } g''(\xi_3) = 0, \\
|u(x) - u^h(x)| &\leq h^2 \max_{\eta \in [X_0, X_1]} |g''(\eta) - g''(\xi_3)| \;\leq\; h^2 \left| \int_{\xi_3}^{\eta} |g'''(s)|\, ds \right| \\
&\leq h^2 \int_{X_0}^{X_2} |g'''(s)|\, ds \;\leq\; 2h^3 \max_{\xi \in [X_0, X_2]} |g'''(\xi)| \\
&\downarrow \quad \textit{Comme la dérivée troisième d'un polynôme du second degré vaut zéro,} \\
|u(x) - u^h(x)| &\leq 2h^3 \max_{\xi \in [X_0, X_2]} |u'''(\xi)|
\end{aligned}$$

□

5. *Le polynôme de Taylor est défini par :*

$$u^t(x) \;=\; u(X_0) + u'(X_0)(x - X_0) + u''(X_0)\frac{(x - X_0)^2}{2}$$

Le théorème du reste de Taylor sur l'intervalle $[X_0, X_2]$ nous permet d'obtenir une borne tout-à-fait semblable sur l'erreur.

$$\begin{aligned} |u(x) - u^t(x)| &\leq \left| \frac{(x - X_0)^3}{6} u'''(\alpha(x)) \right| \\ &= \frac{(2h)^3}{6} \max_{\xi \in [X_0, X_2]} |u'''(\xi)| \\ &= \frac{4h^3}{3} \max_{\xi \in [X_0, X_2]} |u'''(\xi)| \end{aligned}$$

En conclusion, nous avons donc deux polynômes de degré deux $u^h(x)$ et $u^t(x)$ permettant d'approcher une fonction u trois fois continûment dérivable, au voisinage d'un point X_0. Pour être un tout petit peu plus précis, on considère l'intervalle $[X_0, X_0 + 2h]$. Sur cet intervalle, l'erreur maximale ponctuelle est d'ordre trois pour u^h et u^t. Le polynôme de Taylor u^t fait appel aux dérivées première et seconde de u au point X_0. Par contre, le polynôme d'interpolation u^h fait appel à des approximations numériques de ces dérivées. On voit donc qu'il existe un lien réel entre polynôme de Taylor et interpolation.

42

La solution est $sin(0.8) = 0.717356$.

On s'en rapproche en prenant par exemple comme pas $h = 0.1, 0.2, 0.4, 0.8...$ et on peut ensuite faire une extrapolation de Richardson.

On peut vérifier que des pas trop grands donnent une erreur de troncature tandis que des pas trop petits donnent une erreur d'arrondi. On peut également calculer le pas optimal pour minimiser l'erreur totale se composant des erreurs de troncature et d'arrondi. Ce calcul est fait dans les notes de cours.

Calcul en virgule flottante : solutions

43 *Il suffit d'écrire la relation définissant la propagation des erreurs d'arrondi.*

$$f(x,y) = f(\tilde{x},\tilde{y}) \pm \underbrace{\left(\left|\frac{\partial f(\tilde{x},\tilde{y})}{\partial x}\right| \Delta x + \left|\frac{\partial f(\tilde{x},\tilde{y})}{\partial y}\right| \Delta y\right)}_{\Delta f}$$

Et on en déduit immédiatement que les erreurs d'arrondi pour la soustraction et l'addition s'additionnent. On peut aussi noter la similitude entre la propagation d'erreurs et les règles pour le calcul de la dérivée d'une somme, d'une différence, d'un produit ou d'un quotient de deux fonctions.

	Δf	$\frac{\Delta f}{\lvert f\rvert}$
$x+y$	$\Delta x + \Delta y$	$\dfrac{\Delta x + \Delta y}{\lvert x+y\rvert}$
$x-y$	$\Delta x + \Delta y$	$\dfrac{\Delta x + \Delta y}{\lvert x-y\rvert}$
$x \times y$	$\lvert y\rvert\Delta x + \lvert x\rvert\Delta y$	$\dfrac{\Delta x}{\lvert x\rvert} + \dfrac{\Delta y}{\lvert y\rvert}$
$x \div y$	$\dfrac{\lvert y\rvert\Delta x + \lvert x\rvert\Delta y}{y^2}$	$\dfrac{\Delta x}{\lvert x\rvert} + \dfrac{\Delta y}{\lvert y\rvert}$

44

1. *On constate facilement que $a_{n-1} = 0$ est vérifié pour tous les entiers positifs car 2^{n-1} est forcément plus grand que toutes les combinaisons de puissances de 2 inférieures à $n-1$. Un nombre entier positif est donc représenté par 0 suivi de son expression binaire habituelle de $n-1$ bits. Pour obtenir la représentation du nombre négatif k, il suffit de lui ajouter 2^{n-1}, de transformer le résultat en forme binaire et de faire précéder ce résultat d'un bit unitaire.*
2.

125	01111101
−125	10000011
0	00000000
−175	*(pas représentable en 8 bits)*
−100	10011100

45

1. *Dans* `python`*, nous avons $\mathbb{F}(2, 53, -1021, 1024)$. Plus précisément, l"espace des nombres réels représentés par* `python` *correspond à F(2, 53, -1021, 1024) avec les nombres dénormalisés. La notion de nombre dénormalisé intervient dans la représentation des nombres en informatique par la méthode de la virgule flottante, telle que normalisé par la norme IEEE 754. C'est une manière de représenter des nombres ayant une valeur absolue très petite. Cela permet de représenter par exemple 2^{-1050}. Pour que la définition de F puisse correspondre avec celle des nombres réels représentés par IEEE 754, il faudrait autoriser $a_1 = 0$ si $e = L$.*

2. *Nous n'avons que des nombres de la forme* $\pm 0.1a\ 2^e$ *avec* $a = 0, 1$ *et* $e = \pm 2, \pm 1, 0$. *Pour chaque exposant, on peut représenter seulement deux nombres et leurs opposés. La dimension de l'espace est donc* 20 *et son epsilon machine est* $1/2$.

3. *Pour chaque exposant, chacun des chiffres (digits)* $a_2, \ldots a_n$ *peut avoir* β *valeurs possibles, tandis que* a_1 *n'a que* $\beta - 1$ *possibilités. On peut donc en tenant compte des nombres négatifs, dire que* $2(\beta - 1)(\beta)^{n-1}$ *peuvent être représentés pour chaque exposant. Finalement, nous avons* $U - L + 1$ *exposants possibles et la démonstration est terminée.*

4. *L'erreur relative introduite par l'inexactitude de la représentation en virgule flottante (roundoff error) est heureusement liée à l'epsilon machine par la relation*

$$2\frac{|x - \widetilde{x}|}{|x|} \leq \epsilon.$$

46

1. *Le premier bit témoigne du signe du nombre. On utilise ensuite 3 bits pour le codage des exposants* $(-2, -1, 0, 1, 2, 3)$ *qui sont représentés par* (001 010 011 100 101 110). *Il s'agit d'un codage en représentation binaire en excès de trois pour les puristes. La mantisse est normalisée de la manière suivante : le premier chiffre à gauche de la virgule est toujours un et ne doit pas être stocké en mémoire. Il s'ensuit que l'on ne peut se contenter de quatre bits pour une mantisse de taille cinq (mantisses allant de* 1.0000 *à* 1.1111*).*

   Notons qu'il existe des exceptions et des cas particuliers...

2.

	s	e_i	a_i
$+0$	0	000	0000
-0	1	000	0000
$+\infty$	0	111	1000
$-\infty$	1	111	1000
$+NaN$	0	111	1...
$-NaN$	1	111	1...
x_{min}	0	000	0000
x_{max}	0	110	1111

   L'abréviation `NaN` = *signifie simplement Not a Number. En utilisant le plus petit exposant (-2) et la plus petite mantisse (0000), on obtient* x_{min} *:*

$$(-1)^0\ 2^{-2}1.0000 = 2^{-2}$$

3. *L'epsilon machine de cet ordinateur est tout simplement* 2^{-4}. *La valeur absolue (4) de cet exposant (-4) est le nombre de digits de la mantisse (5, même si on ne stocke que quatre digits en mémoire, puisqu'on utilise une représentation normalisée) moins une unité.*

4. *On a* $(3.25)_{10} = (11.01)_2 = 1.1010 \times 2^1$. *L'exposant est donc* 1 *et sera codé par* 100, *tandis que la mantisse est* 1010. *Le nombre sera donc représenté par* 01001010.

47

1. *Cette question correspond à une très légère variation du développement effectué dans les notes pour l'obtention d'une formule unilatérale pour une dérivée première.*

   En considérant $X_0 = 0.8$, $X_1 = 0.9$, $X_2 = 1.0$, $U_0 = 0.8237$, $U_1 = 1.1341$, $U_2 = 1.5574$ *et* $h = 0.1$, *on obtient le polynôme d'interpolation comme :*

$$
\begin{array}{rcl}
p(x) & = & U_0 \ \dfrac{(x-X_1)(x-X_2)}{2h^2} - U_1 \ \dfrac{(x-X_0)(x-X_2)}{h^2} + U_2 \ \dfrac{(x-X_0)(x-X_1)}{2h^2} \\
& \Big\downarrow & \\
p''(x) & = & \underbrace{\dfrac{U_0 - 2U_1 + U_2}{h^2}}_{\textit{constant !}} \ = \ \dfrac{0.8237 - 2.2682 + 1.5574}{0.01} \ = \ 11.29
\end{array}
$$

On obtient ainsi la différence finie centrée d'ordre deux avec $h = 0.1$ *pour* $x = 0.9$.

La réponse est donc $\boxed{f''(0.9) \approx 11.29}$

2. *Nous disposons déjà du résultat pour* $h = 0.1$ *et il est possible de calculer la même différence centrée en utilisant les points* 0.85, 0.90 *et* 0.95 *et d'effectuer une extrapolation de Richardson sur ces deux résultats.*

La réponse est donc

$$
\boxed{
\begin{array}{rcl}
D_{0,0} & = & 11.29 \\
D_{1,0} & = & \dfrac{0.9676 - 2.2682 + 1.3285}{\frac{1}{4}\ 0.01} \ = \ 11.16 \\
D_{1,1} & = & \dfrac{4\ 11.16 - 11.29}{3} \ = \ 11.12
\end{array}
}
$$

On divise le pas par un facteur deux et on tire profit du fait que le terme d'erreur ne contient que des puissances paires dont le premier terme est en h^2. *Le dénominateur est donc* $2^2 - 1 = 3$.

3. *L'erreur théorique de* $D_{0,0}$ *et celle de* $D_{1,0}$ *sont en* $\mathcal{O}(h^2)$.
L'erreur théorique de $D_{1,1}$ *est en* $\mathcal{O}(h^4)$.
L'impact des erreurs d'arrondi des données est fourni par l'expression suivante du formulaire (à ne pas re-déduire donc !)

$$
|\tilde{E}^h| \leq \frac{4\epsilon}{h^2} + \frac{C_4 h^2}{12}
$$

avec $\epsilon = 10^{-4}$. *On peut directement y observer que le premier terme qui contient les erreurs d'arrondi est de l'ordre de* 10^{-2}. *Le second terme qui contient l'erreur théorique est du même ordre de grandeur ! A priori, il est donc possible et même très probable que les estimations théoriques ne soient pas fiables. A titre d'illustration, les mêmes calculs effectués avec des données en double précision sur* `python` *fournissent des résultats nettement meilleurs :*

i	$D_{i,0}$	$D_{i,1}$
0	11.2834	
1	11.1046	11.0450

On observe maintenant que l'on double bien les chiffres exacts (la valeur exacte est égale à 11.0463*) en effectuant l'extrapolation de Richardson. Ce n'était pas le cas dans le cadre du*

calcul effectué avec des données à quatre chiffres après la virgule. Les erreurs d'arrondi sur les données ont donc bien un impact très important ! Dernière question laissée à votre sagacité : est-il possible maintenant d'estimer C_4 et comment ?

Il n'était réellement pas nécessaire de disposer d'une calculatrice pour répondre à cette question. Toutefois, l'expression analytique de la dérivée seconde est $f''(x) = (2 + 2x\tan(x))(1 + \tan(x)^2)$ et avec une calculatrice,il est possible de calculer les erreurs réelles et de les comparer aux estimations... Et c'est évidemment plus facile !

48

La boucle se termine lorsque le test `s+t == s` *est satisfait. En d'autres mots, il faut que* `t` *soit très petit par rapport à* `s`*, afin que les représentations en virgule flottante de* `s+t` *et de* `s` *soient identiques. Une manière de résoudre l'exercice est de modifier la fonction comme suit :*

```
from math import sin,pi

def powersin2(x):
  s=0; t=x; n=1; i=0; tmax=abs(t);
  while ((s + t) != s):
    s = s+t
    t = - x*x*t/((n+1)*(n+2))
    n = n+2
    i = i+1;
    tmax = max(tmax,abs(t));
  return s,tmax,i

x = [pi/2, 13*pi/2, 23*pi/2, 33*pi/2]
for j in range(len(x)):
  sinEst,tmax,nmax = powersin2(x[j])
  sinRef = sin(x[j])
  print(' ==== Computed value of sin(%6.3f) = %11.4e (expected = %11.4e)' % (x[j],sinEst,sinRe
  print('      Error = %11.4e : maximum term = %10.4e : number of terms : %d' % (sinEst-sinRef
```

On peut ensuite calculer...

```
bash-3.2$ python powersin.py
 ==== Computed value of sin( 1.571) =  1.0000e+00 (expected =  1.0000e+00)
      Error =  2.2204e-16 : maximum term = 1.5708e+00 : number of terms : 11
 ==== Computed value of sin(20.420) =  1.0000e+00 (expected =  1.0000e+00)
      Error =  4.8522e-09 : maximum term = 6.3989e+07 : number of terms : 42
 ==== Computed value of sin(36.128) = -1.0154e+00 (expected = -1.0000e+00)
      Error = -1.5410e-02 : maximum term = 3.2392e+14 : number of terms : 64
 ==== Computed value of sin(51.836) = -1.2748e+05 (expected =  1.0000e+00)
      Error = -1.2748e+05 : maximum term = 1.8017e+21 : number of terms : 81
```

Nous observons que lorsque le plus grand terme est de l'ordre de 10^p, la précision chute de p digits. La série en puissance fournit une technique de calcul satisfaisante lorsque $x \leq \pi/2$, mais pour de plus grandes valeurs, le coût calcul augmente et la précision chute dramatiquement... Toutefois, il est possible de ramener le calcul d'un sinus d'un angle quelconque comme celui d'un angle inférieur à $pi/2$ en utilisant judicieusement quelques relations trigonométriques et donc c'est bien une technique possible pour obtenir une approximation fiable d'un sinus !

49 *Le coefficient binomial peut être calculé par le programme suivant.*

```
from numpy import *
from scipy.misc import comb

def binomial(n,k):
  n = fix(n); k = fix(k)
  if k > n:
    raise Exception("Tes valeurs sont loufoques...  : %d < %d :-(" % (n,k) )
  if k > n/2:
    k = n - k
  numerator   = arange(n-k+1,n+1)
  denominator = arange(1,k+1)
  coeff = prod(numerator/denominator)
  return coeff

print(" === Result obtained with binomial(45,8)        : %d " % binomial(45,8))
print(" === Result obtained with scipy.misc.comb(45,8) : %d " % comb(45,8))
```

La fonction `numpy.prod(v)` *permet d'effectuer le produit de tous les éléments d'un vecteur* `v`*, tandis que la fonction* `numpy.fix(n)` *donne l'entier inférieur le plus proche de* `n`*. Il est encore possible de directement utiliser la fonction* `scipy.misc.comb(n,k)` *et de ne strictement rien faire ! A nouveau, on constate que la boîte à outils de* `python` *est bien fournie.*

```
bash-3.2$ python binomial.py
 === Result obtained with binomial(45,8)        : 215553195
 === Result obtained with scipy.misc.comb(45,8) : 215553195
```

50 *L'algorithme proposé est une application directe de la méthode de Monte-Carlo. Il est implémenté par le programme :*

```
from numpy.random import rand

def montecarlo(n):
  x = rand(n)
  y = rand(n)
  r = x*x + y*y
  return 4 * sum(r <= 1)/n

print(montecarlo(500))
for i in range(7):
  print(" === %d : %14.7e " % (i,montecarlo(pow(10,i))))
```

La commande `rand` *génère une séquence de nombres pseudo-aléatoires. La génération de nombres aléatoires est une tâche délicate et est étonnamment loin d'être simple. Il faut observer l'instruction* `sum(r <= 1)` *permettant d'obtenir le nombre de fois où l'inégalité est satisfaite. En effet, le résultat d'un test vaudra* 0 *ou* 1 *en fonction de la réussite ou non et la fonction* `sum` *effectue simplement la somme de tous les éléments du vecteur solution.*

En exécutant le programme pour plusieurs valeurs de `n`*, on obtient une approximation de plus en plus précise de* π*...*

```
bash-3.2$ python montecarlo.py
2.968
```

```
=== 0 :  4.0000000e+00
=== 1 :  3.2000000e+00
=== 2 :  3.0000000e+00
=== 3 :  3.1640000e+00
=== 4 :  3.1256000e+00
=== 5 :  3.1389200e+00
=== 6 :  3.1393320e+00
```

Problèmes différentiels aux conditions initiales 1 : solutions

51 *La solution exacte du problème est :*

$$u(x) = \frac{1}{2}\left(e^x - \sin(x) - cos(x) \right).$$

On peut appliquer Euler explicite et implicite comme suit :

$$\begin{aligned} U_{i+1} &= U_i + h\ (\ \sin(X_i) + U_i\ ) && \textit{(Euler explicite)} \\ U_{i+1} &= U_i + h\ (\ \sin(X_{i+1}) + U_{i+1}\ ) && \textit{(Euler implicite)} \\ &\downarrow \\ U_{i+1} &= \frac{(\ U_i + h\sin(X_{i+1})\ )}{(1-h)} \end{aligned}$$

Comme l'équation différentielle est linéaire, il est possible de résoudre très facilement de manière analytique le problème algébrique de la méthode d'Euler implicite à chaque pas. Il est donc possible (et conseillé !) d'éviter de faire appel à `fsolve`*. N'utilisons pas l'artillerie lourde pour tuer une mouche ! On remarque également que le choix* $h = 1$ *n'est pas judicieux pour le cas de la méthode d'Euler implicite...*

Un programme `python` *pour le calcul pourrait alors être :*

```
from numpy import *

Xstart = 0; Xend = 1
Ustart = 0; Uend = 0.5*(exp(Xend) - sin(Xend) - cos(Xend))
Eexpl = zeros(8)
Eimpl = zeros(8)

for j in range(8):
  n = pow(2,j+1)
  h = (Xend-Xstart)/n
  X = linspace(Xstart,Xend,n+1)
  Uexpl = zeros(n+1); Uexpl[0] = Ustart
  Uimpl = zeros(n+1); Uimpl[0] = Ustart
  for i in range(n):
    Uexpl[i+1] =  Uexpl[i] + h*(sin(X[i]) + Uexpl[i])
    Uimpl[i+1] = (Uimpl[i] + h*sin(X[i+1]))/(1 - h)
  Eexpl[j] = abs(Uexpl[-1] - Uend)
  Eimpl[j] = abs(Uimpl[-1] - Uend)

Oexpl = log(abs(Eexpl[:-1]/Eexpl[1:]))/log(2)
Oimpl = log(abs(Eimpl[:-1]/Eimpl[1:]))/log(2)
print("orderExpl ", *['%.4f ' % val for val in Oexpl])
print("orderImpl ", *['%.4f ' % val for val in Oimpl])
```

Les estimations de la convergence obtenues par ce programme seront :

```
orderExpl  0.7696  0.8662  0.9273  0.9620  0.9806  0.9902  0.9951
orderImpl  1.5199  1.1970  1.0881  1.0418  1.0204  1.0101  1.0050
```

52

1. *L'équation est linéaire.*
2. *L'équation n'est pas homogène.*
3. *La solution analytique est* $u(x) = e^{-5x} + x$.
4. *Le jacobien* $J = -5$ *est négatif et le problème différentiel est stable. Les conditions de stabilité pour les trois méthodes sont données dans le tableau ci-dessous.*

Euler explicite	$\lvert(1-5h)\rvert < 1$	$h < \frac{2}{5} = 0.4$
Euler implicite	$\left\lvert\frac{1}{(1+5h)}\right\rvert < 1$	*Inconditionnellement stable*
Runge Kutta 4	$\left\lvert 1-5h+\frac{25h^2}{2}-\frac{125h^3}{6}+\frac{625h^4}{24}\right\rvert < 1$	$h < \frac{2.7853}{5} = 0.5571$

La méthode d'Euler explicite n'est stable que si $h < 0.4$. *Il faut donc au moins 10 pas. La méthode d'Euler implicite est inconditionnellement stable. La méthode de Runge-Kutta est conditionnellement stable. Visiblement, le pas* $h = 1.0$ *est trop grand alors que le pas* $h = 0.5$ *est suffisant.*

5. *Une implémentation* `python` *de la méthode de Runge Kutta est donnée par :*

```
from numpy import *
from matplotlib import pyplot as plt

Xstart = 0; Xend = 4; Ustart = 1;
x = linspace(Xstart,Xend,100); u = exp(-5*x)+x
f = lambda x,u : 5*(x-u) + 1

print(' ==================== Explicit 4th-order Runge Kutta ==========');
for j in range(1,5):
  n = pow(2,j+1)
  X = linspace(Xstart,Xend,n+1)
  h = (Xend - Xstart)/n
  U = zeros(n+1); U[0] = Ustart
  for i in range(n):
    K1 = f(X[i]    ,U[i]       )
    K2 = f(X[i]+h/2,U[i]+K1*h/2)
    K3 = f(X[i]+h/2,U[i]+K2*h/2)
    K4 = f(X[i]+h  ,U[i]+K3*h  )
    U[i+1] = U[i] + h*(K1+2*K2+2*K3+K4)/6
  plt.subplot(2,2,j)
  plt.xlim((-0.1,4.1)); plt.ylim((-2.0,6.0)); plt.yticks(arange(-2,7,2))
  plt.plot(x,u,'-k',X,U,'.-b',markersize='5.0')
  print(' u(4) = %21.14e (Explicit 4th-order Runge Kutta with %2d steps ) '
        % (U[-1],n))
```

Pour la méthode de Runge-Kutta, il devient indispensable de définir la fonction f *de manière distincte. On voit aussi qu'il serait intéressant de la faire passer en argument de ce programme pour qu'il puisse intégrer n'importe quelle fonction, tout comme la fonction* `solve_ivp`. *L'implémentation des méthodes d'Euler explicite et implicite se fait exactement comme dans l'exercice précédent !*

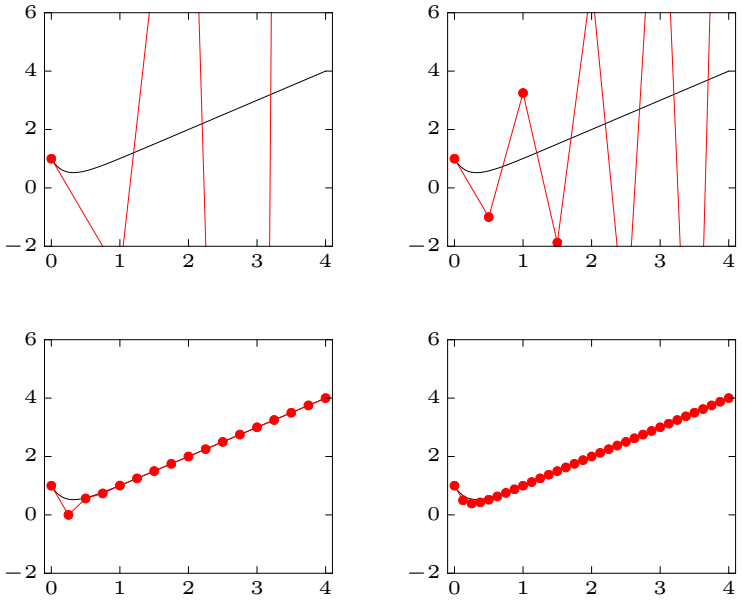


Figure 8.1: Résolution par la méthode d'Euler explicite de $u'(x) = 5\,(x - u(x)) + 1$ avec $4, 8, 16$ et 32 pas

Les résultats obtenus sont :

```
==================== Explicit 1st-order Euler ===============
u(4) =  2.60000000000000e+02 (Explicit 1th-order Euler with  4 steps )
u(4) =  2.96289062500000e+01 (Explicit 1th-order Euler with  8 steps )
u(4) =  4.00000000023283e+00 (Explicit 1th-order Euler with 16 steps )
u(4) =  4.00000000000002e+00 (Explicit 1th-order Euler with 32 steps )
==================== Implicit 1st-order Euler ===============
u(4) =  4.00077160493827e+00 (Implicit 1th-order Euler with  4 steps )
u(4) =  4.00004440743054e+00 (Implicit 1th-order Euler with  8 steps )
u(4) =  4.00000231782002e+00 (Implicit 1th-order Euler with 16 steps )
u(4) =  4.00000017893377e+00 (Implicit 1th-order Euler with 32 steps )
==================== Explicit 4th-order Runge Kutta ==========
u(4) =  3.53173261025511e+04 (Explicit 4th-order Runge Kutta with  4 steps )
u(4) =  4.03125683367881e+00 (Explicit 4th-order Runge Kutta with  8 steps )
u(4) =  4.00000000637517e+00 (Explicit 4th-order Runge Kutta with 16 steps )
u(4) =  4.00000000215160e+00 (Explicit 4th-order Runge Kutta with 32 steps )
```

53

Dans notre cas, il est possible de calculer la solution analytique $u(x) = \log(1 - x^2/2)$. *Il s'agit d'une fonction qui décroît de manière monotone sur l'intervalle* $[0, 1]$. *On peut alors calculer le jacobien sur la trajectoire de la solution et obtenir*

$$J(x) = x \exp\left(-\log\left(1 - \frac{x^2}{2}\right)\right) = \frac{x}{\left(1 - \frac{x^2}{2}\right)}$$

Le jacobien est toujours positif. Il croît de manière monotone de 0 *à la valeur* 2. *Le problème différentiel est donc instable et les erreurs propagées par la méthode d'Euler explicite vont donc s'amplifier progressivement. Toutefois, sur l'intervalle considéré, l'amplification de ces erreurs reste encore limitée.*

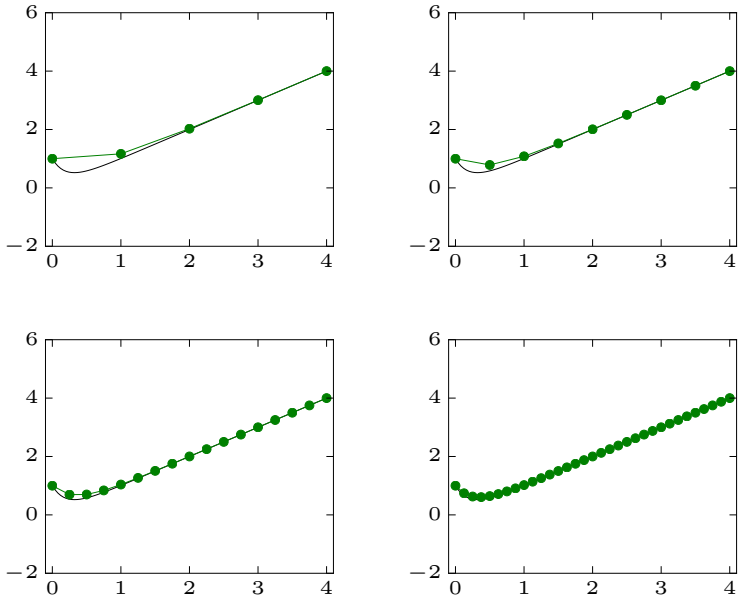


Figure 8.2: Résolution par la méthode d'Euler implicite de $u'(x) = 5\,(x - u(x)) + 1$ avec $4, 8, 16$ et 32 pas

En définissant $J = \max_i |J_i|$ *et* $e = \max_i |e_i|$, *on peut écrire, après* n *pas, une borne pour l'erreur comme suit.*

$$
\begin{aligned}
|e^h(X_n)| &\leq \left(1 + (1+hJ) + (1+hJ)^2 + \cdots + (1+hJ)^{n-1}\right) e \\
&\downarrow \quad \textit{En utilisant l'identité } \sum_{i=0}^{n-1} a(1+a)^i = (1+a)^n - 1 \\
&\leq \frac{(1+hJ)^n - 1}{hJ}\, e \\
&\downarrow \quad \textit{En sachant que l'erreur locale } e \leq Mh^2/2 \\
&\leq \frac{(1+J/n)^n - 1}{J}\, \frac{M}{2}\, h \\
&\downarrow \quad \textit{En sachant également que } (1+J/n)^n \leq e^{J/n} \\
&\leq \frac{e^J - 1}{J}\, \frac{M}{2}\, h
\end{aligned}
$$

On voit donc qu'il est possible de borner l'erreur commise par le produit d'une constante (dépendant de J*) et du facteur* h. *Ce résultat général peut se mettre sous la forme d'un théorème. Le terme* M *représente le maximum de la dérivée seconde de la solution.*

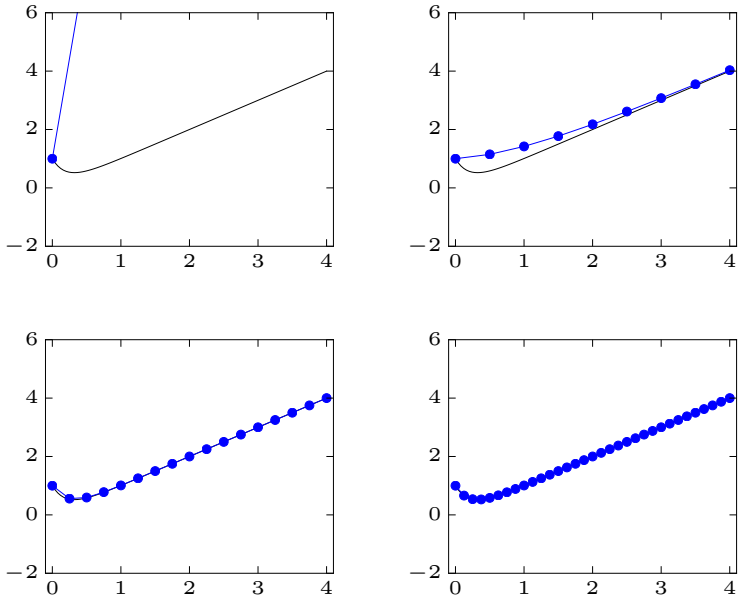


Figure 8.3: Résolution par la méthode de Runge-Kutta explicite d'ordre 4 de $u'(x) = 5\,(x - u(x)) + 1$ avec $4, 8, 16$ et 32 pas

Théorème 8.1.

Sur un intervalle de temps de taille finie $[a, b]$, l'erreur obtenue avec la méthode d'Euler explicite appliquée à un problème stable ou un problème instable de raideur finie ($-\infty < J < \infty$) peut être bornée comme suit :

$$\exists C, \quad \forall i = 0, \ldots, n \leq (b-a)/h, \qquad |u(X_i) - U_i| \leq Ch$$

On dit que la méthode d'Euler explicite converge avec un ordre linéaire.

Est-il possible maintenant de faire une estimation ? Considérons que les valeurs du couple (x, u) restent dans l'intervalle $[0, 1] \times [-1, 0]$: on suppose donc implicitement que l'on ne s'écartera pas trop de la valeur de la solution pendant le calcul. Ce qui est légitime, si l'on considère que l'erreur tend vers zéro lorsque l'on diminue le pas h.

$$\begin{aligned} J &= \max_{x,u} |\partial f/\partial u|, & M &= \max_{x,u} |u''| \\ &= \max_{x,u} |xe^{-u}| & &= \max_{x,u} |-e^{-u} - x^2 e^{-2u}| \\ &= e & &= e + e^2 \end{aligned}$$

Il est alors possible de conclure en écrivant

$$|e^h(X_{100})| \leq \frac{(1 + e/100)^{100} - 1}{1} \frac{(1+e)}{200} = 0.2351$$

Cette estimation est une borne supérieure à l'erreur exacte. Le calcul par la méthode d'Euler explicite devrait fournir `U(end) = -0.6785`. *Il est évidemment possible de calculer la valeur exacte qui est $u(1) = -0.6931$ et d'en déduire la vraie erreur exacte qui vaut 0.0146.*

54 *On obtient cette méthode comme la résolution numérique d'une intégrale en écrivant*

$$\int_{X_i}^{X_{i+1}} f(x,u(x))dx \quad = \quad u(X_{i+1}) - u(X_i)$$

$$\Big\downarrow \quad \textit{En approximant l'intégrale par un trapèze,}$$

$$\frac{h}{2}\ (f(X_i,U_i) + f(X_{i+1},U_{i+1})\ ) \quad \approx \quad U_{i+1} - U_i \qquad \square$$

55 *Interdire que les erreurs propagées s'amplifient, revient à imposer que*

$$|1-h+ih| \quad < \quad 1$$

$$\Big\downarrow$$

$$0 \ < \quad h \quad < \ 1$$

Problèmes différentiels aux conditions initiales 2 : solutions

56

1. *La réponse se trouve chez Paul Fisette.*
2. *La solution analytique est donnée par* $x(t) = 3e^{-2t} - 2e^{-3t}$.

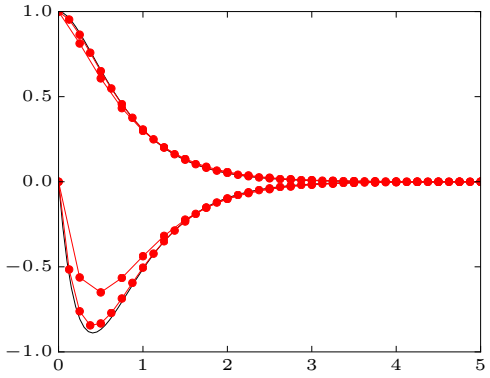


Figure 9.1: Approximations de $x(t)$ et de $x'(t)$ obtenues par une méthode de Heun avec 20 et 40 pas. La solution exacte est donnée en trait continu

3. *L'équation du second ordre est équivalente au système des deux équations du premier ordre :*

$$\begin{aligned} x'(t) &= u(t) \\ u'(t) &= -5u(t) - 6x(t) \end{aligned}$$

avec les conditions initiales $x(0) = 1$ *et* $u(0) = 0$.
On peut effectuer une implémentation de la méthode de Heun comme suit :

```
from numpy import *
from matplotlib import pyplot as plt

def f(x,u):
  C = 5.0; k = 6
  dudx = u[1]
  dvdx = - C*u[1] - k*u[0]
  return array([dudx, dvdx])

Xstart = 0; Xend = 5;
Ustart = [1,0]

x = linspace(0,5,100)
u =  3.0*exp(-2*x) - 2*exp(-3*x)
v = -6.0*exp(-2*x) + 6*exp(-3*x)
plt.plot(x,u,'-k',x,v,'-k')

for n in [20,40]:
  X = linspace(Xstart,Xend,n+1)
  U = zeros((n+1,2)); U[0] = Ustart
  h = (Xend - Xstart)/n
  for i in range(n):
    P        = U[i,:] + h*f(X[i],U[i,:])
    U[i+1,:] = U[i,:] + h*(f(X[i],U[i,:]) + f(X[i+1],P))/2
  plt.plot(X, U[:,0],'.-r',X, U[:,1], '.-r',markersize='5.0')
```

57

1. *L'équation est linéaire.*
2. *L'équation est homogène.*
3. *La solution analytique est* $u(x) = Ce^{-5(x-1)^2}$ *avec* $C = 0.05/e^{-5}$.
4. *Le Jacobien* $J = -10(x-1)$ *est positif si* $x < 1$ *: le problème est donc d'abord instable... Puis, ce jacobien est négatif lorsque* $x > 1$ *et le problème différentiel est stable. Il devient de plus en plus raide lorsque* x *grandit. On constatera, toutefois, que malgré le fait que l'on ne soit pas dans la région de stabilité des méthodes au début du calcul, la plupart des méthodes produiront des résultats tout à fait acceptables. Par contre, si on continue le calcul dans la zone raide du problème, les méthodes deviendront inappropriées.*

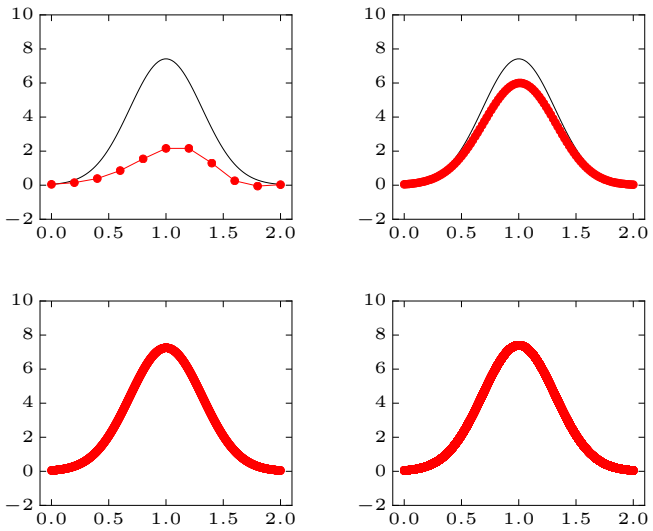


Figure 9.2: Résolution par la méthode d'Euler explicite de $u'(x) = -10(x-1)u(x)$ avec $10, 100, 1000$ et 10000 pas

5. *L'implémentation est tout à fait semblable à ce qui a été réalisé auparavant !*

```
==================== Explicit 1st-order Euler ===============
u(2) =  3.11351040000000e-02 (Explicit 1th-order Euler with 10 steps )
u(2) =  3.11805334393736e-02 (Explicit 1th-order Euler with 100 steps )
u(2) =  4.77200130718345e-02 (Explicit 1th-order Euler with 1000 steps )
u(2) =  4.97672099931233e-02 (Explicit 1th-order Euler with 10000 steps )
==================== Implicit 1st-order Euler ===============
u(2) =  8.02952191841080e-02 (Implicit 1th-order Euler with 10 steps )
u(2) =  8.01782305893166e-02 (Implicit 1th-order Euler with 100 steps )
u(2) =  5.23889211060498e-02 (Implicit 1th-order Euler with 1000 steps )
u(2) =  5.02338789002930e-02 (Implicit 1th-order Euler with 10000 steps )
==================== Explicit 4th-order Runge Kutta ==========
u(2) =  2.71270102454453e+00 (Explicit 4th-order Runge Kutta with  4 steps )
u(2) =  1.98037474236935e-01 (Explicit 4th-order Runge Kutta with  8 steps )
u(2) =  5.32801013336645e-02 (Explicit 4th-order Runge Kutta with 16 steps )
u(2) =  5.00938793371536e-02 (Explicit 4th-order Runge Kutta with 32 steps )
```

58 *La méthode de Heun est définie par :*

$$\begin{cases} U_{i+1} &= U_i + \dfrac{h}{2}(K_1 + K_2) \\ \\ K_1 &= f(X_i, U_i) \\ K_2 &= f(X_i + h, U_i + hK_1) \end{cases}$$

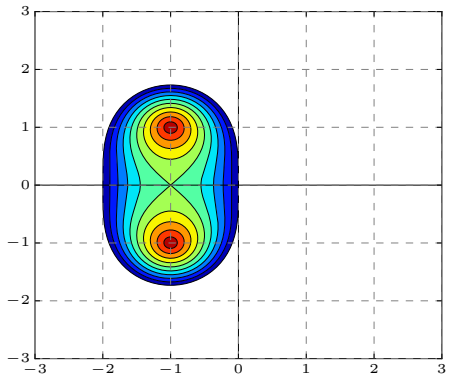


Figure 9.3: Région de stabilité de la méthode de Heun dans le plan complexe $h\lambda$

En appliquant ceci à notre problème modèle, on obtient

$$K_1 = \lambda U_i \qquad K_2 = \lambda U_i + \lambda^2 h U_i \qquad U_{i+1} = U_i + \lambda h U_i + \lambda^2 \frac{h^2}{2} U_i$$

Le facteur d'amplification de l'erreur est donc $\left|1 + h\lambda + \dfrac{h^2\lambda^2}{2}\right|$

La région de stabilité est obtenue avec les trois (!) instructions suivantes.

```
import matplotlib.pyplot as plt
from numpy import *

x,y = meshgrid(linspace(-3,3,1000),linspace(-3,3,1000))
z = x + 1j*y
plt.contourf(x,y,abs(1+ z + z*z/2),arange(0,1.1,0.1),cmap=plt.cm.jet_r)
```

Remarquez que `python` *peut utiliser des variables complexes et que leur usage permet d'obtenir le graphe en reprenant seulement la formule du cours.*

L'interprétation des couleurs correspond à la valeur du facteur d'amortissement de l'erreur propagée.

59 *Le système des deux équations du second ordre est équivalent au système des quatre équations du premier ordre :*

$$\begin{aligned} x'(t) &= u(t) \\ y'(t) &= v(t) \\ u'(t) &= 2\omega \sin(\phi)\ v(t) - k^2\ x(t) \\ v'(t) &= -2\omega \sin(\phi)\ u(t) - k^2\ y(t) \end{aligned}$$

avec les conditions initiales $x(0) = 1$, $y(0) = 0$, $u(0) = 0$ *et* $v(0) = 0$.

Nous pouvons alors implémenter la méthode d'Euler comme suit :

```
from numpy import *
from matplotlib import pyplot as plt

def f(x,u):
  L  = 20; k2 = 9.81/L; phi = pi/2;  omega = 7.29e-05
  dudx = zeros(4)
  dudx[0] = u[2]
  dudx[1] = u[3]
  dudx[2] =  2*omega*sin(phi)*u[3] - k2*u[0]
  dudx[3] = -2*omega*sin(phi)*u[2] - k2*u[1]
  return dudx

Xstart = 0; Xend = 300
Ustart = [1,0,0,0]

print("==== Using explicit Euler integrator :-(")
n = 30000
X = linspace(Xstart,Xend,n+1)
U = zeros((n+1,4)); U[0] = Ustart
h = (Xend - Xstart)/n
for i in range(n):
    U[i+1,:] = U[i,:] + h*f(X[i],U[i,:])
print(" number of steps used : %d " % n)

plt.figure(); plt.xlim((-2.00, 2.00)); plt.ylim((-0.05, 0.05))
plt.plot(U[:,0],U[:,1],'-r')
```

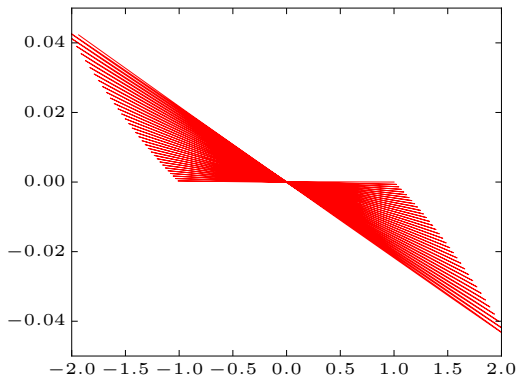


Figure 9.4: Trajectoires du pendule dans le plan de phase avec la méthode d'Euler explicite ($h = 0.01$ sec)

Même en utilisant un grand nombre d'intervalles ($n = 30000$) et un tout petit pas ($h = 0.01$ sec), nous obtenons une solution inacceptable d'un point de vue physique. Comme espéré, le plan de rotation du pendule change avec le temps, mais ce qui est surprenant, c'est que l'amplitude des oscillations augmente avec le temps. Des résultats semblables seront obtenus pour des pas encore plus petits ou pour la méthode du second ordre de Heun.

Ces mauvaises performances numériques sont dues au fait que le problème modèle caractéristique de ce système a un coefficient λ purement imaginaire. La solution correspondante est une sinusoïde qui reste bornée lorsque x grandit mais ne tend pas vers zéro. Malheureusement, les régions de

stabilité d'Euler explicite et de Heun ne contiennent aucun point de l'axe imaginaire à l'exception de l'origine. Elles ne sont donc stables que lorsque $h = 0$: ce qui n'est pas un pas acceptable !

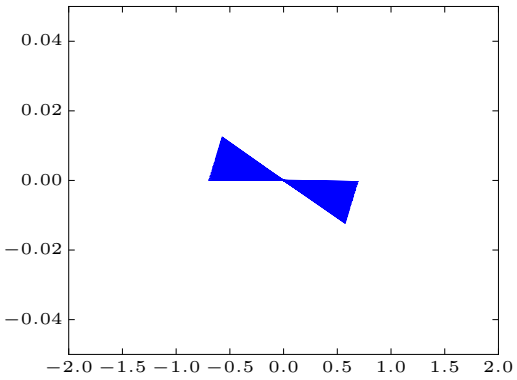


Figure 9.5: Trajectoires du pendule dans le plan de phase avec une méthode adaptative de Runge-Kutta d'ordre trois

Pour obtenir une solution numérique acceptable, il faut donc utiliser une méthode dont la région de stabilité englobe une partie de l'axe imaginaire. C'est le cas, par exemple, de la méthode adaptative de Runge-Kutta d'ordre trois implémentée dans la fonction `scipy : solve_ivp`. *En ajoutant ces quelques lignes au programme précédent, on obtient une solution numérique acceptable !*

```
print("==== Using RK23 integrator from scipy :-)")
from scipy.integrate import solve_ivp
sol = solve_ivp(f,[Xstart,Xend],Ustart, method='RK23')
print(" number of steps used : %d " % len(sol.t))

plt.figure(); plt.xlim((-2.00, 2.00)); plt.ylim((-0.05, 0.05))
plt.plot(sol.y[2], sol.y[3], '-b')
```

```
>bash-3.2$ python pendulum.py
==== Using explicit Euler integrator :-(
 number of steps used : 30000
==== Using RK23 integrator from scipy :-)
 number of steps used : 783
```

Avec seulement 783 pas, on obtient ainsi la solution de la Figure **??** *qui est en bon accord avec la solution analytique.*

60 *La région de stabilité est obtenue avec le tout petit programme...*

```
import matplotlib.pyplot as plt
from numpy import *

#
# Région de stabilité de la méthode suivante
#    P = U_i + h (-F_(i-1) + 3F_i)/2
#    U_(i+1) = U_i + h (F_i + f(X_(i+1),P))/2
#
# Problème modèle u? = lambda u
# a^(i+1) = a^i + h/2( lambda a^i + lambda a^i
```

```
#                                   - h lambda^2/2 a^(i-1)
#                                   + 3 h lambda^2/2 a^i )
#
# polynôme du second degré en a ! =>
# =>  domaine de stabilité est la region
#     où le module des deux racines <= 1
#

x,y = meshgrid(linspace(-3,3,1000),linspace(-3,3,1000))
z = x + 1j*y
b = 0.5 + z/2 + 3*z*z/8
c = z*z/4.0
f1 = abs(b - sqrt(b*b - c))
f2 = abs(b + sqrt(b*b - c))
gain = maximum(f1,f2)

plt.contourf(x,y,gain,arange(0,1.1,0.1),cmap=plt.cm.jet_r)
plt.contour(x,y,gain,arange(0,1.1,0.1),colors='black')

ax = plt.gca()
ax.axhline(y=0,color='k')
ax.axvline(x=0,color='k')
ax.yaxis.grid(color='gray',linestyle='dashed')
ax.xaxis.grid(color='gray',linestyle='dashed')
plt.xticks(arange(-3,4,1))
plt.yticks(arange(-3,4,1))
```

La partie la plus compliquée de ce code est finalement la gestion graphique des axes pour exactement obtenir la figure souhaitée... Eh oui, obtenir exactement une figure précise avec `matplotlib` *est parfois un peu fastidieux, car les options par défaut sont rarement satisfaisantes. Il faut donc faire office de patience et de persévérance pour obtenir exactement le résultat souhaité.*

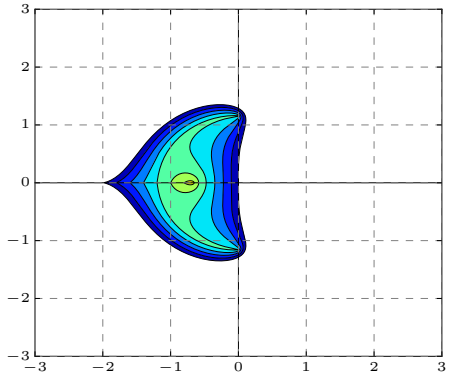


Figure 9.6: Zone de stabilité d'une méthode de type Adams

Equations non linéaires (... et linéaires) : solutions

61

1. $x = 3.693441$
2. $x = 1.412391$ *et* $x = 3.057104$
3. $x = 0.567143$
4. $x = 1.365230$

62

Dans le premier cas, la méthode converge vers la racine $x = 0$, tandis que dans le second elle devrait diverger... toutefois, le hasard fait converger la méthode sur la solution après 29 itérations.

Une implémentation possible est donnée par

```
from math import exp

f    = lambda x : x*exp(x)
dfdx = lambda x : (x + 1.0)*exp(x)

def newton(x,tol,nmax):
  n = 0; delta = float("inf")
  while ((abs(delta) > tol) and (n < nmax)) :
    delta = -f(x) / dfdx(x)
    x = x + delta
    n = n + 1
    print(" x = %14.7e (Estimated error %13.7e at iteration %d)"
          % (x,abs(delta),n))
  return x

print("Found x = %14.7e" % newton( 0.2,1e-13,50))
print("Found x = %14.7e" % newton(20.0,1e-13,50))
```

L'instruction `newton(0.2,10e-14,50)` *engendre une jolie convergence quadratique (on observe aussi que l'erreur estimée est une fort bonne estimation de l'erreur exacte qui est évidemment x !)*

```
 x =  3.3333333e-02 (Estimated error 1.6666667e-01 at iteration 1)
 x =  1.0752688e-03 (Estimated error 3.2258065e-02 at iteration 2)
 x =  1.1549611e-06 (Estimated error 1.0741139e-03 at iteration 3)
 x =  1.3339337e-12 (Estimated error 1.1549598e-06 at iteration 4)
 x =  1.7795692e-24 (Estimated error 1.3339337e-12 at iteration 5)
 x =  0.0000000e+00 (Estimated error 1.7795692e-24 at iteration 6)
Found x =  0.0000000e+00
```

L'instruction `newton(20,10e-14,50)` *génère*

```
 x =  1.9047619e+01 (Estimated error 9.5238095e-01 at iteration 1)
 x =  1.8097500e+01 (Estimated error 9.5011876e-01 at iteration 2)
 x =  1.7149863e+01 (Estimated error 9.4763713e-01 at iteration 3)
 x =  1.6204960e+01 (Estimated error 9.4490317e-01 at iteration 4)
 x =  1.5263083e+01 (Estimated error 9.4187723e-01 at iteration 5)
 x =  1.4324572e+01 (Estimated error 9.3851104e-01 at iteration 6)
 x =  1.3389826e+01 (Estimated error 9.3474532e-01 at iteration 7)
 x =  1.2459320e+01 (Estimated error 9.3050646e-01 at iteration 8)
 x =  1.1533618e+01 (Estimated error 9.2570204e-01 at iteration 9)
 x =  1.0613403e+01 (Estimated error 9.2021458e-01 at iteration 10)
```

```
 x =  9.6995107e+00 (Estimated error 9.1389260e-01 at iteration 11)
 x =  8.7929729e+00 (Estimated error 9.0653778e-01 at iteration 12)
 x =  7.8950870e+00 (Estimated error 8.9788596e-01 at iteration 13)
 x =  7.0075086e+00 (Estimated error 8.8757839e-01 at iteration 14)
 x =  6.1323914e+00 (Estimated error 8.7511721e-01 at iteration 15)
 x =  5.2725968e+00 (Estimated error 8.5979457e-01 at iteration 16)
 x =  4.4320204e+00 (Estimated error 8.4057639e-01 at iteration 17)
 x =  3.6161140e+00 (Estimated error 8.1590644e-01 at iteration 18)
 x =  2.8327464e+00 (Estimated error 7.8336757e-01 at iteration 19)
 x =  2.0936559e+00 (Estimated error 7.3909049e-01 at iteration 20)
 x =  1.4168981e+00 (Estimated error 6.7675785e-01 at iteration 21)
 x =  8.3065156e-01 (Estimated error 5.8624652e-01 at iteration 22)
 x =  3.7690516e-01 (Estimated error 4.5374640e-01 at iteration 23)
 x =  1.0317159e-01 (Estimated error 2.7373357e-01 at iteration 24)
 x =  9.6488866e-03 (Estimated error 9.3522706e-02 at iteration 25)
 x =  9.2211276e-05 (Estimated error 9.5566753e-03 at iteration 26)
 x =  8.5021354e-09 (Estimated error 9.2202774e-05 at iteration 27)
 x =  7.2286306e-17 (Estimated error 8.5021353e-09 at iteration 28)
 x =  0.0000000e+00 (Estimated error 7.2286306e-17 at iteration 29)
Found x =  0.0000000e+00
```

63 *Une implémentation possible est donnée par :*

```
from cmath import sqrt

f    = lambda x : 2*sqrt(x-1)

def iter(x,tol,nmax):
  n = 0; delta = float("inf")
  while ((abs(delta) > tol) and (n < nmax)) :
    xold = x
    x = f(x)
    delta = x - xold
    n = n + 1
    print(" x = %05f%+05fi (Estimated error %13.7e at iteration %d)"
          % (x.real,x.imag,abs(delta),n))
  return x

print("Found x = ", iter(1.5,1e-2,50))
print("Found x = ", iter(2.5,1e-2,50))
```

L'instruction `iter(1.5,10e-3,50)` *nous fait découvrir les joies des variables complexes...*

```
 x = 1.414214+0.000000i (Estimated error 8.5786438e-02 at iteration 1)
 x = 1.287189+0.000000i (Estimated error 1.2702506e-01 at iteration 2)
 x = 1.071799+0.000000i (Estimated error 2.1538907e-01 at iteration 3)
 x = 0.535908+0.000000i (Estimated error 5.3589111e-01 at iteration 4)
 x = 0.000000+1.362485i (Estimated error 1.4640917e+00 at iteration 5)
 x = 1.174801+2.319517i (Estimated error 1.5152777e+00 at iteration 6)
 x = 2.236468+2.074267i (Estimated error 1.0896257e+00 at iteration 7)
 x = 2.702334+1.535167i (Estimated error 7.1250294e-01 at iteration 8)
 x = 2.826533+1.086255i (Estimated error 4.6577636e-01 at iteration 9)
 x = 2.811285+0.772782i (Estimated error 3.1384362e-01 at iteration 10)
 x = 2.749740+0.562076i (Estimated error 2.1950985e-01 at iteration 11)
 x = 2.678635+0.419674i (Estimated error 1.5916783e-01 at iteration 12)
```

```
 x = 2.611105+0.321453i (Estimated error 1.1919545e-01 at iteration 13)
 x = 2.551065+0.252015i (Estimated error 9.1796025e-02 at iteration 14)
 x = 2.498988+0.201693i (Estimated error 7.2417285e-02 at iteration 15)
 x = 2.454174+0.164368i (Estimated error 5.8322607e-02 at iteration 16)
 x = 2.415619+0.136087i (Estimated error 4.7814700e-02 at iteration 17)
 x = 2.382337+0.114247i (Estimated error 3.9808415e-02 at iteration 18)
 x = 2.353460+0.097089i (Estimated error 3.3589989e-02 at iteration 19)
 x = 2.328260+0.083400i (Estimated error 2.8677544e-02 at iteration 20)
 x = 2.306138+0.072329i (Estimated error 2.4737944e-02 at iteration 21)
 x = 2.286603+0.063263i (Estimated error 2.1535873e-02 at iteration 22)
 x = 2.269256+0.055757i (Estimated error 1.8901840e-02 at iteration 23)
 x = 2.253768+0.049479i (Estimated error 1.6711587e-02 at iteration 24)
 x = 2.239871+0.044180i (Estimated error 1.4872541e-02 at iteration 25)
 x = 2.227344+0.039670i (Estimated error 1.3314713e-02 at iteration 26)
 x = 2.216000+0.035804i (Estimated error 1.1984482e-02 at iteration 27)
 x = 2.205687+0.032465i (Estimated error 1.0840255e-02 at iteration 28)
 x = 2.196274+0.029564i (Estimated error 9.8494092e-03 at iteration 29)
Found x =  (2.1962743753966536+0.029563598820072076j)
```

Tandis que l'instruction `iter(2.5,10e-3,50)` *génère une convergence guère rapide. La convergence devient de plus en plus lente car*

$$lim_{i\to\infty} g'(x_i) = g'(2) = 1$$

avec $g(x) = 2\sqrt{x-1}$. *En d'autres mots, plus on s'approche de la solution, moins vite on converge.*

```
 x = 2.449490+0.000000i (Estimated error 5.0510257e-02 at iteration 1)
 x = 2.407895+0.000000i (Estimated error 4.1594610e-02 at iteration 2)
 x = 2.373095+0.000000i (Estimated error 3.4799995e-02 at iteration 3)
 x = 2.343583+0.000000i (Estimated error 2.9512293e-02 at iteration 4)
 x = 2.318260+0.000000i (Estimated error 2.5322424e-02 at iteration 5)
 x = 2.296310+0.000000i (Estimated error 2.1949971e-02 at iteration 6)
 x = 2.277113+0.000000i (Estimated error 1.9197848e-02 at iteration 7)
 x = 2.260188+0.000000i (Estimated error 1.6924466e-02 at iteration 8)
 x = 2.245162+0.000000i (Estimated error 1.5026105e-02 at iteration 9)
 x = 2.231737+0.000000i (Estimated error 1.3425460e-02 at iteration 10)
 x = 2.219673+0.000000i (Estimated error 1.2064009e-02 at iteration 11)
 x = 2.208776+0.000000i (Estimated error 1.0896827e-02 at iteration 12)
 x = 2.198887+0.000000i (Estimated error 9.8889848e-03 at iteration 13)
Found x =  (2.1988867501240392+0j)
```

64

Il y a deux solutions : $x = 2$ *et* $x = 4$.
Un programme possible pour réaliser les itérations est :

```
def solve(x,tol,nmax):
  n = 0; delta = float("inf")
  while ((abs(delta) > tol) and (n < nmax)) :
    xold = x
    x = 4*x - x*x/2 - 4
    delta = x-xold
    n = n+1
    print(" x = %14.7e (Estimated error %13.6e at iteration %d)"
          % (x,abs(delta),n))
  return x
```

```
print("Found x = %14.7e" % solve(1.9,1e-5,50))
print("Found x = %14.7e" % solve(3.8,1e-5,50))
print("Found x = %14.7e" % solve(6.0,1e-5,50))
```

L'itération en partant de 1.9 *diverge tandis que la méthode converge vers* $x = 4$*, en partant de* 3.8*. Dans le premier cas, la condition de Lipschitz n'est pas vérifiée tandis qu'elle l'est dans le second cas. On observera une convergence quadratique dans le second cas. En partant de* $x = 6$*, on obtient directement :-)*

```
 x =   1.7950000000000e+00 (Estimated error  1.050000e-01 at iteration 1)
 x =   1.5689875000000e+00 (Estimated error  2.260125e-01 at iteration 2)
 x =   1.0450891124219e+00 (Estimated error  5.238984e-01 at iteration 3)
 x =  -3.6574917676388e-01 (Estimated error  1.410838e+00 at iteration 4)
 x =  -5.5298829372072e+00 (Estimated error  5.164134e+00 at iteration 5)
 x =  -4.1409334398437e+01 (Estimated error  3.587945e+01 at iteration 6)
 x =  -1.0270038252545e+03 (Estimated error  9.855945e+02 at iteration 7)
 x =  -5.3148044384473e+05 (Estimated error  5.304534e+05 at iteration 8)
 x =  -1.4123785702047e+11 (Estimated error  1.412373e+11 at iteration 9)
 x =  -9.9740661284326e+21 (Estimated error  9.974066e+21 at iteration 10)
 x =  -4.9740997567173e+43 (Estimated error  4.974100e+43 at iteration 11)
 x =  -1.2370834194888e+87 (Estimated error  1.237083e+87 at iteration 12)
 x = -7.6518769338700e+173 (Estimated error 7.651877e+173 at iteration 13)
 x =                 -inf (Estimated error           inf at iteration 14)
 x =                 -inf (Estimated error           nan at iteration 15)
Found x =           -inf
 x =   3.9800000000000e+00 (Estimated error  1.800000e-01 at iteration 1)
 x =   3.9998000000000e+00 (Estimated error  1.980000e-02 at iteration 2)
 x =   3.9999999800000e+00 (Estimated error  1.999800e-04 at iteration 3)
 x =   4.0000000000000e+00 (Estimated error  2.000000e-08 at iteration 4)
Found x =  4.0000000e+00
 x =   2.0000000000000e+00 (Estimated error  4.000000e+00 at iteration 1)
 x =   2.0000000000000e+00 (Estimated error  0.000000e+00 at iteration 2)
Found x =  2.0000000e+00
```

On remarquera que dans le premier cas, le programme a formellement enregistré une convergence puisque l'estimation de l'erreur commise est basée sur la différence entre deux itérées (ici, on a obtenu deux fois `-inf`*) et a donc estimé que nous avions obtenu la convergence du processus de manière erronée.*

65

Le carré de la distance[4] *du point* P *à la parabole est donné par*

$$d(x) = (x-3)^2 + (y-1)^2 = x^4 - x^2 - 6x + 10$$

Comme il faut minimiser cette fonction, il faut donc annuler sa dérivée

$$d'(x) = 4x^3 - 2x - 6$$

Ce qui fournit $x = 1.289624$*. Le point demandé est donc le point* $(1.289624,\ 1.663130)$*.*

[4]Minimiser le carré de la distance ou minimiser la distance produira le même résultat. Mais, c'est vachement plus simple de dériver le carré de la distance, que la distance elle-même et comme je suis fondamentalement paresseux...

66

La boîte obtenue a comme dimensions : x, $10-2x$ *et* $16-2x$.

On a donc $x(10-2x)(16-2x)=100$.

Il y a trois solutions 8.759232, 3.401748 *et* 0.839018. *La première valeur doit être rejetée, car la largeur et la longueur de la boîte correspondante sont négatives !*

67

En élevant cette relation au cube, on obtient $x^3=3+x$.
La solution de cette équation est $x=1.671700$.

68

La solution du système est $x=1.407640$, $y=1.981451$.

Si l'itération converge on obtiendra :

$$\begin{cases} 2x = 2x - x^2 + y \\ 36y = 8x - 4x^2 + 32 + 36y - 9y^2 \end{cases}$$

En simplifiant les termes à droite et à gauche qui s'éliminent, on obtient le système d'équations original. On voit donc que la construction d'une itération simple s'effectue en ajoutant un terme $2x$ *et un terme* $36y$ *de chaque côté de l'égalité. Tout l'art pour obtenir une méthode itérative efficace est de choisir judicieusement ces termes... On le voit, il reste ici un peu de magie que l'on peut toutefois contrôler avec un peu de raisonnement.*

Une implémentation possible de Newton-Raphson est donnée par

```
from scipy.linalg import norm,solve

def f(x):
  return [x[0]*x[0] - x[1],
          4*x[0]*x[0] + 9*x[1]*x[1] - 8*x[0] - 32]

def dfdx(x):
  return [[2*x[0]    ,-1     ],
          [8*x[0] - 8,18*x[1]]]

def newton(x,tol,nmax):
  n = 0; delta = tol+1
  while (norm(delta) > tol and n < nmax):
    delta = - solve(dfdx(x),f(x))
    x = x + delta
    n = n+1
    print(" Estimated error %14.7e at iteration %d" % (norm(delta),n))
  return x

print(newton([1.4,2.0],1e-13,50))
```

On obtient le résultat suivant en partant du point indiqué.

```
 Estimated error  2.0000000e-02 at iteration 1
 Estimated error  1.0221072e-04 at iteration 2
 Estimated error  2.7083017e-09 at iteration 3
 Estimated error  4.0952993e-16 at iteration 4
[1.40764008 1.9814506 ]
```

On observe bien une convergence quadratique de l'erreur estimée comme prédit par la théorie !

69 *Il est possible de trouver quatre solutions :*

x	y
1.781541	2.064179
−1.192873	1.278444
2.923492	−3.510016
−1.912160	−2.232608

70 *Le système n'a pas de solution.*

71

1. *La solution* $\mathbf{x}$ *est* $x = 5$ *et* $y = 0.2$.
2. *Les deux résidus sont* $\mathbf{r}_1 = (-0.6, -0.60001)$, $\mathbf{r}_2 = (0, 0.0004)$ *et leurs normes respectives sont données par* $||\mathbf{r}_1||_2 = 0.8485$, $||\mathbf{r}_2||_2 = 0.0004$. *La deuxième solution semble donc "moins mauvaise".*

   Pourtant $||\mathbf{x} - \mathbf{s}_1||_2 = 0.1414$ *tandis que* $||\mathbf{x} - \mathbf{s}_2||_2 = 4.0792$. *Dans ce cas, les résidus ne donnent pas une bonne idée de l'erreur commise : la matrice est mal conditionnée. En effet, son nombre de condition dépasse* 10^5.
3. *La nouvelle solution est* $x = 0$ *et* $y = 1.2$. *Une toute petite perturbation du second membre cause de grandes variations de la solution : c'est le comportement caractéristique d'un système linéaire dont la matrice est mal conditionnée.*

72 *La méthode ne converge pas. Elle ne converge pas non plus si on permute les deux équations. Par contre, si on remplace la deuxième équation par* $-6x_1 - 8x_2 = -4$, *on converge vers la solution* $(x_1, x_2) = (1.636363, -0.727272)$. *Une autre option consiste à choisir comme seconde équation* $x_1 + 5x_2 = -2$, *ce qui permet de converger encore plus rapidement vers la solution : il y a donc beaucoup de choix possibles permettant d'obtenir des itérations qui vont converger !*

Une implémentation possible est donnée par :

```
from numpy import *
from scipy.linalg import norm

def g(x):
  x[0] =  (-3*x[1]+6)/5
  x[1] =  -(6*x[0]-4)/8
  return x

def gauss(x,tol,nmax):
  n = 0; delta = tol+1;
  x = array(x,dtype=float)
  while (norm(delta) > tol and n < nmax):
    n = n+1
    xold = x.copy()
    x = g(x)
    delta = x - xold
    print(" Estimated error %14.7e at iteration %d" % (norm(delta),n))
  return x

print(gauss([0,0],10e-6,50))
```

Comme annoncé dans les notes, l'implémentation de la méthode de Gauss-Seidel peu apparaître plus simple que celle de Jacobi. Il est possible d'utiliser le même vecteur pour l'input et l'output du calcul de la fonction `g`. *On obtient le résultat suivant en partant de l'origine.*

```
 Estimated error  1.2649111e+00 at iteration 1
 Estimated error  3.0000000e-01 at iteration 2
 Estimated error  1.3500000e-01 at iteration 3
 Estimated error  6.0750000e-02 at iteration 4
 Estimated error  2.7337500e-02 at iteration 5
 Estimated error  1.2301875e-02 at iteration 6
 Estimated error  5.5358437e-03 at iteration 7
 Estimated error  2.4911297e-03 at iteration 8
 Estimated error  1.1210084e-03 at iteration 9
 Estimated error  5.0445376e-04 at iteration 10
 Estimated error  2.2700419e-04 at iteration 11
 Estimated error  1.0215189e-04 at iteration 12
 Estimated error  4.5968349e-05 at iteration 13
 Estimated error  2.0685757e-05 at iteration 14
 Estimated error  9.3085907e-06 at iteration 15
[ 1.63635754 -0.72726816]
```

Il y a toutefois deux points particulier à observer dans ce programme. Tout d'abord, l'instruction `xold = x.copy()` *pour effectuer une copie du tableau* `x`*. Ecrire* `xold = x` *ne créerait qu'une autre vue vers le même tableau et en conséquence,* `delta` *serait toujours nul... puisqu'on calculerait la différence entre deux vues identiques du même espace mémoire. Faites l'essai pour vous en convaincre :-)*

Ensuite, il faut aussi noter l'instruction `x = array(x,dtype=float)` *qui assure que le vecteur* `x` *soit bien un tableau de réels ! Ecrire simplement* `x=array(x)` *avec un argument* `[0,0]` *créerait un tableau d'entiers et cela aurait un comportement indésirable. On peut aussi ici observer qu'écrire* `gauss([0.0,0.0],10e-6,50)` *permettrait d'éviter tout souci. C'est une des difficultés du langage* `python`*, on peut avoir une liste ou un tableau d'entiers ou encore un tableau de réels avec des possibilités de transformer à la volée le type des variables en fonction des opérations effectuées. Il convient donc d'être particulièrement attentif au type des variables utilisées : l'implémentation des méthodes de Jacobi et de Gauss-Seidel est l'occasion d'observer cet aspect un peu particulier de la syntaxe des tableaux* `numpy`*.*

73 Juin 2003

On dispose de n mesures (X_i, Y_i) d'une fonction inconnue $y = u(x)$ et on souhaite approximer $u(x)$ par une expression

$$u^h(x) = \frac{a}{x+b}$$

en ajustant les deux paramètres réels a et b afin de minimiser la somme du carré des n écarts $Y_i - u^h(X_i)$.

1. *Les deux équations que doivent satisfaire a et b en termes des n données X_i et Y_i sont obtenues en minimisant la fonction :*

$$f(a,b) = \sum_{i=1}^{n} \left(Y_i - \frac{a}{(X_i+b)} \right)^2$$

Deux équations (non-linéaires) que doivent donc satisfaire a et b sont donc :

$$0 = \frac{\partial f}{\partial a} = \sum_{i=1}^{n} -2 \left(Y_i - \frac{a}{(X_i+b)} \right) \frac{1}{(X_i+b)} = -2 \sum_{i=1}^{n} \frac{Y_i}{(X_i+b)} + 2a \sum_{i=1}^{n} \frac{1}{(X_i+b)^2}$$

$$0 = \frac{\partial f}{\partial b} = \sum_{i=1}^{n} -2 \left(Y_i - \frac{a}{(X_i+b)} \right) \frac{-a}{(X_i+b)^2} = 2a \sum_{i=1}^{n} \frac{Y_i}{(X_i+b)^2} - 2a^2 \sum_{i=1}^{n} \frac{1}{(X_i+b)^3}$$

2. *Le schéma de Newton-Raphson pour les deux équations s'écrit :*

On calcule (a_{k+1}, b_{k+1}) à partir de (a_k, b_k) avec

$$\begin{bmatrix} \frac{\partial^2 f}{\partial a^2}(a_k,b_k) & \frac{\partial^2 f}{\partial b \partial a}(a_k,b_k) \\ \frac{\partial^2 f}{\partial a \partial b}(a_k,b_k) & \frac{\partial^2 f}{\partial b^2}(a_k,b_k) \end{bmatrix} \begin{bmatrix} \Delta a \\ \Delta b \end{bmatrix} = - \begin{bmatrix} \frac{\partial f}{\partial a}(a_k,b_k) \\ \frac{\partial f}{\partial b}(a_k,b_k) \end{bmatrix}$$

$$\begin{aligned} a_{k+1} &= a_k + \Delta a \\ b_{k+1} &= b_k + \Delta b \end{aligned}$$

avec les dérivées partielles secondes données par les expressions suivantes :

$$\frac{\partial^2 f}{\partial a^2} = 2\sum_{i=1}^{n} \frac{1}{(X_i+b)^2}$$

$$\frac{\partial^2 f}{\partial b^2} = -4a\sum_{i=1}^{n} \frac{Y_i}{(X_i+b)^3} + 6a^2 \sum_{i=1}^{n} \frac{1}{(X_i+b)^4}$$

$$\frac{\partial^2 f}{\partial a \partial b} = \frac{\partial^2 f}{\partial b \partial a} = 2\sum_{i=1}^{n} \frac{Y_i}{(X_i+b)^2} - 4a\sum_{i=1}^{n} \frac{1}{(X_i+b)^3}$$

Il ne suffit pas de recopier le formulaire pour répondre correctement. Il faut expliquer comment calculer tous les termes requis dans le schéma de Newton-Raphson à partir des données, comme demandé dans l'énoncé !

3. *On peut facilement obtenir a en sachant que $b = 1$ pour les trois mesures (X_i, Y_i) :*

X_i	0	1	2
Y_i	$\frac{49}{8}$	2	$\frac{25}{8}$

Il suffit de calculer :

$$\sum_{i=0}^{2} \frac{Y_i}{(X_i+b)} = \frac{49}{8} + \frac{2}{2} + \frac{25}{24} = \frac{49\times 3 + 24 + 25}{24} = \frac{49}{6}$$

$$\sum_{i=0}^{2} \frac{1}{(X_i+b)^2} = 1 + \frac{1}{4} + \frac{1}{9} = \frac{36+9+4}{36} = \frac{49}{36}$$

d'en déduire que $\frac{49}{36}\, a = \frac{49}{6}$. *La réponse est donc :* $\boxed{a = 6}$

4. *Comme $X_1 = 0$, choisir comme valeur initiale $b = 0$ impliquera l'apparition de dénominateurs nuls dans toutes les expressions du schéma de Newton-Raphson et va donc poser quelques difficultés. Typiquement, on observera...*

```
Warning: Divide by zero.
ans = Inf
```

74 Septembre 2003

On dispose toujours de n mesures (X_i, Y_i) d'une fonction inconnue $y = u(x)$ et on souhaite approximer $u(x)$ par une expression

$$u^h(x) = \frac{6}{x+b}$$

en ajustant le paramètre réel b afin de minimiser la somme des carrés des n écarts $Y_i - u^h(X_i)$. Cette question est une simple variation de la question de juin !

1. *Il s'agit de minimiser la fonction suivante :*

$$f(b) = \sum_{i=1}^{n} \left(Y_i - \frac{6}{(X_i + b)} \right)^2$$

Une équation que doit satisfaire b est :

$$0 \quad = \quad f'(b) \quad = \quad \sum_{i=1}^{n} 2 \left(Y_i - \frac{6}{(X_i + b)} \right) \frac{6}{(X_i + b)^2}$$

La réponse est donc $\boxed{\displaystyle\sum_{i=1}^{n} \frac{Y_i}{(X_i + b)^2} \quad = \quad 6 \sum_{i=1}^{n} \frac{1}{(X_i + b)^3}}$

2. *Le schéma de Newton-Raphson s'écrit :*

On calcule b_{k+1} à partir de b_k avec

$$f''(b_k)\ \Delta b \quad = \quad -f'(b_k)$$

$$b_{k+1} \quad = \quad b_k + \Delta b$$

avec les dérivées données par les expressions suivantes :

$$f'(b) \quad = \quad 12 \sum_{i=1}^{n} \frac{Y_i}{(X_i + b)^2} - 72 \sum_{i=1}^{n} \frac{1}{(X_i + b)^3}$$

$$f''(b) \quad = \quad -24 \sum_{i=1}^{n} \frac{Y_i}{(X_i + b)^3} + 216 \sum_{i=1}^{n} \frac{1}{(X_i + b)^4}$$

Il est évidemment possible de simplifier toutes ces expressions par un facteur 12.

3. *En remplaçant $b = 1$ dans l'équation obtenue au premier point, on constate que c'est une solution du problème !*

$$\sum_{i=1}^{2} \frac{Y_i}{(X_i + b)^2} \quad = \quad \frac{25}{4} + \frac{2}{4} \quad = \quad \frac{25+2}{4} \quad = \quad \frac{27}{4}$$

$$\sum_{i=1}^{2} \frac{6}{(X_i + b)^3} \quad = \quad 6 + \frac{6}{8} \quad = \quad \frac{24+3}{4} \quad = \quad \frac{27}{4}$$

Comme la dérivée seconde de f est non-nulle en ce point, le schéma de Newton-Raphson fournira directement comme résultat final $b_1 = b_0 = 1$ avec un diagnostic de convergence.
Représenter intuitivement les données permet d'imaginer aisément une valeur probable de b et de mieux cerner le problème !

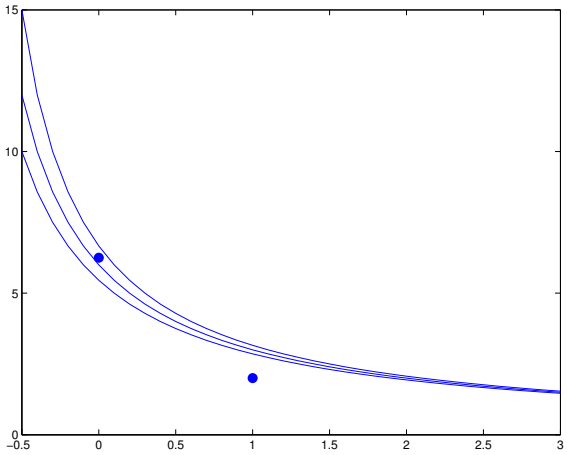


Figure 10.1: Trois approximations possibles avec $b = 0.9,\ 1.0$ et 1.1 respectivement. On observe qu'imposer le passage par le second point est nettement plus délicat que par le premier point et que la solution n'est pas une courbe dont l'erreur en $x = 0$ et $x = 1$ est identique : ce que pourrait faire croire trompeusement une mauvaise intuition...

Transfert de chaleur : solutions

75

1. En tirant profit de la symétrie du problème (fonction source et conditions aux limites), il est possible de pressentir que la solution analytique s'écrira sous la forme suivante :

$$u(x,y) = \sum_{i,j \text{ impairs}} C_{ij} \sin\Big(\frac{i\pi(x+1)}{2}\Big) \sin\Big(\frac{j\pi(y+1)}{2}\Big)$$

Cette série satisfait les conditions aux limites. Pour obtenir les coefficients C_{ij}, il reste à satisfaire l'équation de Poisson, ce qui revient à écrire...

$$\begin{aligned}
\boldsymbol{\nabla}^2 u(x,y) &= -1, \\
\sum_{i,j \text{ impairs}} \frac{\pi^2(i^2+j^2)}{4} C_{ij} \sin\Big(\frac{i\pi(x+1)}{2}\Big) \sin\Big(\frac{j\pi(y+1)}{2}\Big) &= 1, \\
&\downarrow \quad \text{En vertu de l'orthogonalité des sinus,} \\
\frac{\pi^2(i^2+j^2)}{4} C_{ij} &= \underbrace{\int_\Omega \sin\Big(\frac{i\pi(x+1)}{2}\Big) \sin\Big(\frac{j\pi(y+1)}{2}\Big) d\Omega}_{16/(ij\pi^2)},
\end{aligned}$$

On déduit finalement que la solution analytique s'écrit sous la forme suivante.

$$\boxed{u(x,y) = \sum_{i,j \text{ impairs}} \frac{64}{\pi^4(i^2+j^2)ij} \sin\Big(\frac{i\pi(x+1)}{2}\Big) \sin\Big(\frac{j\pi(y+1)}{2}\Big)}$$

2. Une implémentation simple est donnée par :

```
from numpy import *
from numpy.linalg import solve

def poissonSolve(nx,ny):
  n = nx*ny; h = 2/(ny-1)
  A = zeros((n,n))
  B = zeros(n)
  for i in range(n):
    A[i,i] = 1.0
  for i in range(1,nx-1):
    for j in range(1,ny-1):
      index = i + j*nx
      A[index,index]    =  4.0
      A[index,index-1]  = -1.0
      A[index,index+1]  = -1.0
      A[index,index-nx] = -1.0
      A[index,index+nx] = -1.0
      B[index] = 1
  A = A / (h*h)
  return solve(A,B).reshape(ny,nx)
```

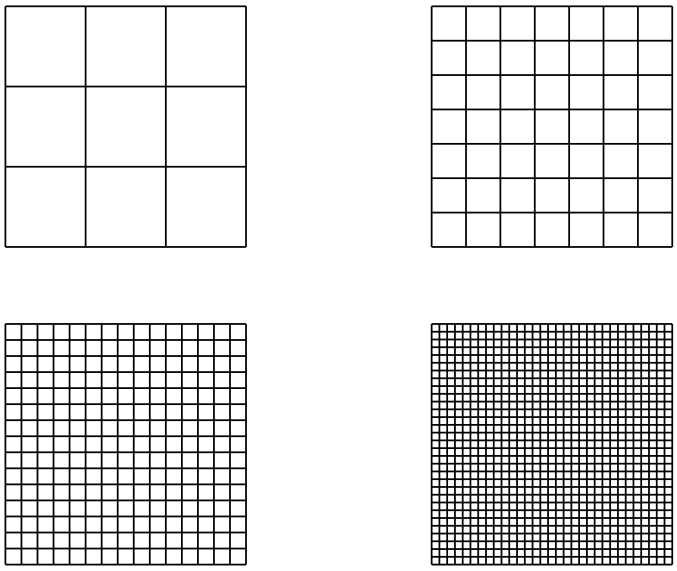


Figure 11.1: Grilles de 4^2, 8^2, 16^2 et 32^2 noeuds.

Il serait astucieux de tirer profit de la symétrie du problème et de ne le résoudre que sur un quart du carré. Il serait aussi possible de ne pas inclure les noeuds de la frontière dans le système à résoudre comme cela a été montré au cours :-)

3. Pour estimer l'erreur numérique, le plus simple est de raffiner successivement la grille et de comparer les erreurs obtenues en comparant avec la solution analytique :

```
nx = ny = 2; error = zeros(4)
for iRef in range(4):
  nx = 2*nx; ny = 2*ny

  ...

  error[iRef] = sqrt(amax((Uref - U)**2))
  print(' === Discretization nx = %4d : error  = %15.7e ' % (nx,error[iRef]))
order = log(abs(error[:-1]/error[1:]))/log(2)
print('\n ===== Estimated order of convergence  = %15.7e ' % mean(order));
```

```
 === Discretization nx =    4 : error  =   1.9161507e-02
 === Discretization nx =    8 : error  =   4.4728014e-03
 === Discretization nx =   16 : error  =   1.0188850e-03
 === Discretization nx =   32 : error  =   2.4094496e-04

 ===== Estimated order of convergence  =   2.1044545e+00
```

On observe un ordre de convergence quadratique : ce qui correspond bien à la théorie. On pouvait s'en douter puisque les différences finies ont une erreur de troncature en $\mathcal{O}(h^2)$. Toutefois, la déduction formelle d'un ordre de précision ponctuelle de la méthode des différences finies n'est pas immédiate et sort du cadre de ce cours d'introduction.

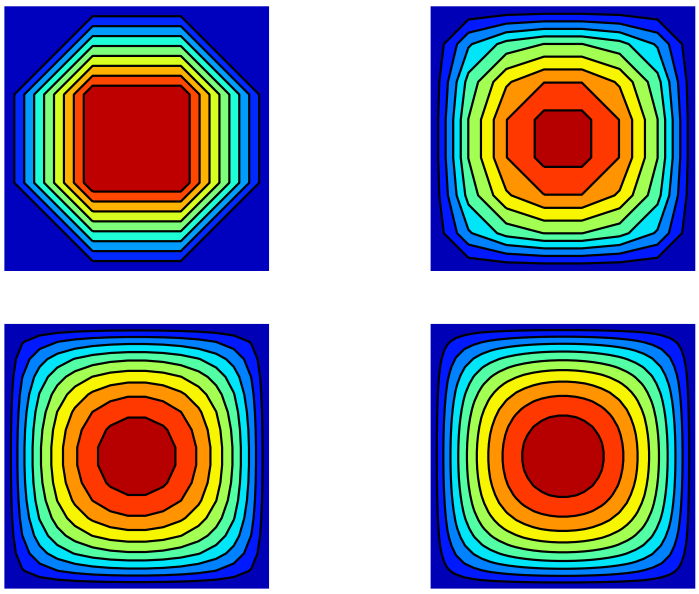


Figure 11.2: Solution par différences finies avec 4^2, 8^2, 16^2 et 32^2 noeuds.

4. En modifiant légèrement le code pour tirer profit du caractère creux de la méthode, on peut augmenter de manière spectaculaire les performances[5]

```
from numpy import *
from scipy.sparse import dok_matrix
from scipy.sparse.linalg import spsolve

def poissonSolveSparse(nx,ny):
  n = nx*ny; h = 2/(ny-1)
  A = dok_matrix((n,n),dtype=float32)
  B = zeros(n)
  for i in range(n):
    A[i,i] = 1.0
  for i in range(1,nx-1):
    for j in range(1,ny-1):
      index = i + j*nx
      A[index,index]    =  4.0
      A[index,index-1]  = -1.0
      A[index,index+1]  = -1.0
      A[index,index-nx] = -1.0
      A[index,index+nx] = -1.0
      B[index] = 1
  A = A / (h*h)
  return spsolve(A.tocsr(),B).reshape(ny,nx)
```

Typiquement, on a remplacé la création d'une matrice plein par une matrice creuse avec

[5]Notons toutefois que tirer profit de tout le potentiel des solveurs linéaires creux requiert en certain soin en `python`

```
A = dok_matrix((n,n),dtype=float3)
```

et on a fait appel à un solveur linéaire creux avec

```
spsolve(A.tocsr(),B)
```

Pour effectuer une comparaison entre les deux implémentations, on peut considérer par exemple une matrice de taille $100 \times 100 = 1000$ et observer que le solveur creux devient plus efficace pour des matrices de grande taille.

```
from timeit import default_timer as timer
nx = ny = 100;
print(' === Considering n = %d' % (nx*ny))
tic = timer(); U = poissonSolve(nx,ny); toc = timer() - tic
print(' === Full solver   : elapsed time is %f seconds.' % toc)
tic = timer(); U = poissonSolveSparse(nx,ny); toc = timer() - tic
print(' === Sparse solver : elapsed time is %f seconds.' % toc)

 === Considering n = 10000
 === Full solver   : elapsed time is 5.416771 seconds.
 === Sparse solver : elapsed time is 0.635170 seconds.
```

76

1. L'analyse de stabilité consiste à obtenir une estimation du facteur d'amplification d'une perturbation quelconque de la forme suivante :

$$U_{i,j}^n = U^n e^{ik_x X_i} e^{ik_y Y_j}$$

Cette perturbation évoluera comme suit lors d'un pas de temps :

$$\begin{aligned} U_{i,j}^{n+1} &= U_{i,j}^n + \beta\Big(U_{i+1,j}^n + U_{i-1,j}^n + U_{i,j+1}^n + U_{i,j-1}^n - 4U_{i,j}^n\Big) \\ &= U_{i,j}^n\Big(1 + \beta\Big(e^{ik_x\Delta x} + e^{-ik_x\Delta x} + e^{ik_y\Delta x} + e^{-ik_y\Delta x} - 4\Big)\Big) \\ &= U_{i,j}^n\Big(1 + \beta\Big(2\cos(k_x\Delta x) + 2\cos(k_y\Delta x) - 4\Big)\Big) \end{aligned}$$

Pour que la perturbation ne s'accroisse pas, il faut que pour toutes valeurs de k_x et k_y, la valeur absolue du facteur d'amplification reste inférieure à l'unité.

$$\begin{aligned} |1 + \beta\Big(2\cos(k_x\Delta x) + 2\cos(k_y\Delta x) - 4\Big)| &\leq 1 \\ &\downarrow \quad \text{En prenant le cas le plus défavorable} \\ &\quad\ \text{où } k_x\Delta x = k_y\Delta x = \pi, \\ -1 &\leq 1 - 8\beta \\ 4\beta &\leq 1 \end{aligned}$$

contrairement à MATLAB : c'est ici qu'on atteint aux limites de notre nouveau langage qui est moins performant pour

On obtient donc finalement comme condition de stabilité, l'inégalité suivante sur les pas de discrétisation spatiale et temporelle.

$$\boxed{\beta \quad = \quad \frac{k\Delta t}{\rho c(\Delta x)^2} \quad \leq \quad \frac{1}{4}}$$

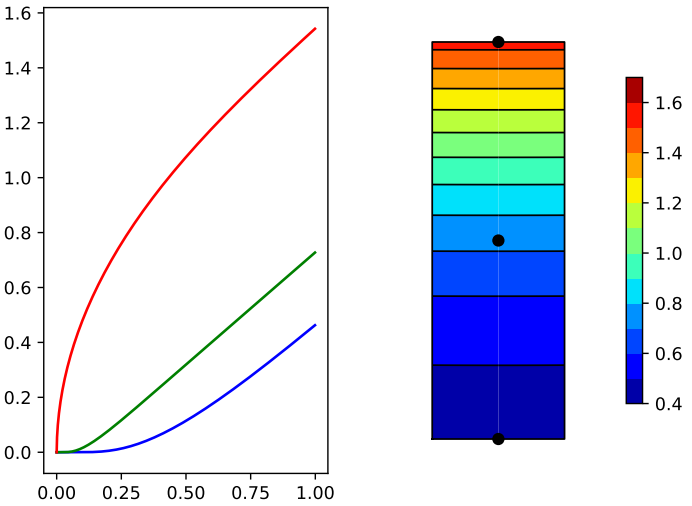


Figure 11.3: Isolignes de $T - T_f$ après $1s$ ($\beta = 0.25$) et évolution temporelle jusqu'à cet instant pour les points suivants : $(0, 0)$ (rouge), $(0, L_y/2)$ (bleu) et $(0, L_y)$ (vert). .

certaines applications spécifiques...

2. Une implémentation possible est donnée par :

```
def diffusionSolve(betaRequested,n):
  rho = 2707; c = 896; k = 204; alpha = k / (rho * c)
  Ly = 0.015; Lx = Ly/6; Lt = 60
  qin = 30000; Tf = 300; hx = 60; hy = 100

  dx = Lx/n; nx = n + 1; ny = 6*n + 1
  dt = (dx * dx) * betaRequested / alpha
  nt = int(ceil(Lt/dt)); dt = Lt/nt
  beta = alpha * dt / (dx * dx)

  T = Tf * ones((nx+2,ny+2))
  for it in range(nt):
    T[1:-1,0]  = T[1:-1,2]  - 2*dx*hy*(T[1:-1,1]-Tf) /k
    T[1:-1,-1] = T[1:-1,-3] + 2*dx*qin /k
    T[0,1:-1]  = T[2,1:-1]
    T[-1,1:-1] = T[-3,1:-1] - 2*dx*hx*(T[-2,1:-1]-Tf) /k
    T[1:-1,1:-1] += beta*(T[2:,1:-1] + T[:-2,1:-1]
                        + T[1:-1,2:] + T[1:-1,:-2] - 4*T[1:-1,1:-1])
   return Tr
```

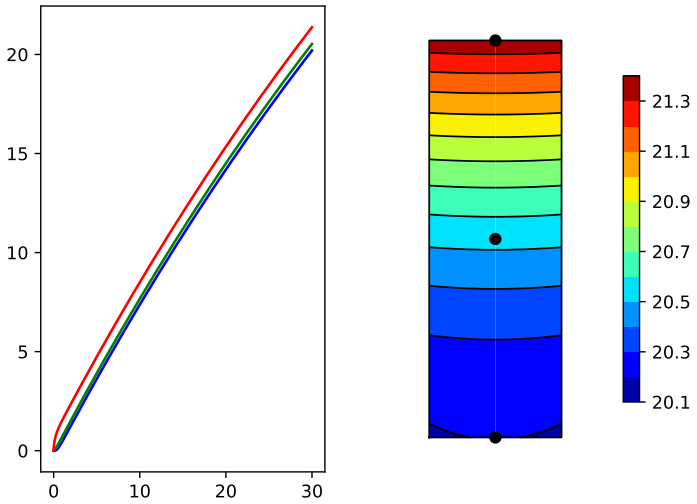


Figure 11.4: Isolignes de $T - T_f$ après $30s$ ($\beta = 0.25$) et évolution temporelle jusqu'à cet instant pour les points suivants : $(0, 0)$ (rouge), $(0, L_y/2)$ (bleu) et $(0, L_y)$ (vert). .

La simulation jusqu'en $t = 60$ s pour $n = 5$ pour $\beta = 0.25$:

```
dx  =  5.000e-04 [m]          (nx = 6, ny = 31)
dt  =  7.431e-04 [m]          (nt = 80744)
=> beta requested  =  2.5000000e-01 [m]
=> beta selected   =  2.4999705e-01 [m]
Results at t = 60.0 [s] :
```

```
 T(0,0   )-Tf  =   3.427956e+01 [K]
 T(0,Ly/2)-Tf  =   3.464845e+01 [K]
 T(0,Ly  )-Tf  =   3.550562e+01 [K]
```

Utiliser une valeur plus faible de $\beta = 0.125$ va nécessiter deux fois plus de pas de temps pour un résultat quasiment identique : c'est donc peu utile !

```
 dx  =  5.000e-04 [m]          (nx = 6, ny = 31)
 dt  =  3.715e-04 [m]          (nt = 161487)
 => beta requested  =  1.2500000e-01 [m]
 => beta selected   =  1.2499930e-01 [m]
Results at t = 60.0 [s] :
 T(0,0   )-Tf  =   3.427950e+01 [K]
 T(0,Ly/2)-Tf  =   3.464840e+01 [K]
 T(0,Ly  )-Tf  =   3.550557e+01 [K]
```

Par contre, utiliser une valeur de $\beta = 0.3$ va créer une instabilité....

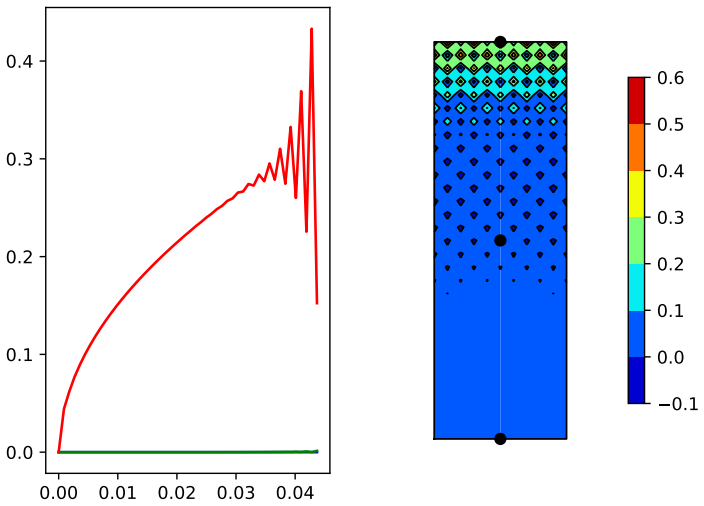


Figure 11.5: Développement de l'instabilité numérique après $0.44s$ ($\beta = 0.3$) .

3. Pour conserver l'évolution temporelle de quelques valeurs ponctuelles, il suffit de modifier le code précédent comme suit.

```
T = Tf * ones((nx+2,ny+2))
Tcurve = Tf * ones((3,nt+1))

for it in range(nt):
  T[1:-1,0]  = T[1:-1,2]  - 2*dx*hy*(T[1:-1,1]-Tf) /k
  T[1:-1,-1] = T[1:-1,-3] + 2*dx*qin /k
  T[0,1:-1]  = T[2,1:-1]
  T[-1,1:-1] = T[-3,1:-1] - 2*dx*hx*(T[-2,1:-1]-Tf) /k
  T[1:-1,1:-1] += beta*(T[2:,1:-1] + T[:-2,1:-1]
```

```
                    + T[1:-1,2:] + T[1:-1,:-2] - 4*T[1:-1,1:-1])

  Tcurve[0,it+1] = T[1,1]
  Tcurve[1,it+1] = T[1,(ny+1)//2]
  Tcurve[2,it+1] = T[1,-2]
```

Par contre, il n'est vraiment pas conseillé de stocker l'entièreté de toutes les valeurs nodales pour tous les pas de temps : il y a plus de 16000 dans le cas de la simulation numérique avec $\beta = 0.125$ et on n'a pas encore atteint le point d'équilibre du système !

4. Comparons notre code vectoriel

```
tic = timer()
T = Tf * ones((nx+2,ny+2))
for it in range(nt):
  T[1:-1,0]  = T[1:-1,2]  - 2*dx*hy*(T[1:-1,1]-Tf) /k
  T[1:-1,-1] = T[1:-1,-3] + 2*dx*qin /k
  T[0,1:-1]  = T[2,1:-1]
  T[-1,1:-1] = T[-3,1:-1] - 2*dx*hx*(T[-2,1:-1]-Tf) /k
  T[1:-1,1:-1] += beta*(T[2:,1:-1] + T[:-2,1:-1]
                      + T[1:-1,2:] + T[1:-1,:-2] - 4*T[1:-1,1:-1])
toc = timer() - tic
print(' === Vectorized version : elapsed time is %f seconds.' % toc)
print('   T(0,Ly) = %10.6f [K] after %d steps ' % (T[1,-2],nt))
```

avec une version écrite avec de boucles plus explicites.

```
tic = timer()
T = Tf * ones((nx+2,ny+2))
Tnew = zeros((nx+2,ny+2))
for it in range(nt):
  for i in range(1,nx+1):
    T[i,0]  = T[i,2]  - 2*dx*hy*(T[i,1]-Tf) /k
    T[i,-1] = T[i,-3] + 2*dx*qin /k
  for j in range(1,ny+1):
    T[0,j]  = T[2,j]
    T[-1,j] = T[-3,j] - 2*dx*hx*(T[-2,j]-Tf) /k
  for i in range(1,nx+1):
    for j in range(1,ny+1):
      Tnew[i,j] = T[i,j] + beta*(T[i+1,j] + T[i-1,j]
                         + T[i,j+1] + T[i,j-1] - 4*T[i,j])
  T[:] = Tnew[:]
toc = timer() - tic
print(' === Loops version : elapsed time is %f seconds.' % toc)
print('   T(0,Ly) = %10.6f [K] after %d steps ' % (T[1,-2],nt))
```

Cette version est peut-être plus simple à écrire et à comprendre, mais elle est nettement moins efficace ! Pour obtenir un code efficace, il est indispensable de vectoriser les boucles ! Dans le cas présent, la différence de performance est vraiment spectaculaire... On verra dans l'exercice suivant que cela peut avoir un impact fondamental sur le choix d'une méthode numérique pour résoudre un problème précis.

```
=== Vectorized version : elapsed time is 2.231116 seconds.
  T(0,Ly) = 335.505622 [K] after 80744 steps
=== Loops version : elapsed time is 26.108858 seconds.
  T(0,Ly) = 335.505622 [K] after 80744 steps
```

5. La convection est plus importante sur la paroi supérieure car l'écoulement de l'air se fait plus librement qu'entre les ailettes où la géométrie confine davantage l'écoulement et ne permet pas un renouvellement aussi efficace de l'air le long du convecteur.

77

1. Pour obtenir le flux sur une frontière, il suffit de calculer la densité de flux de chaleur en estimant la dérivée première par une différence finie en chaque neoud de la frontière et ensuite d'intégrer cette densité sur cette frontière.

   En utilisant, la méthode des trapèzes, cela revient à additionner toutes les dérivées premières estimées en mettant toutefois un poids deux fois plus petit sur les deux extrémités.

   Typiquement, le flux de chaleur sur la frontière droite du domaine est obtenu comme suit :

```
sum(T[2:-2,0] - T[2:-2,2])*k/2 + sum(T[[1,-2],0] - T[[1,-2],2])*k/4
```

2. Une implémentation possible est donnée par :

```
def diffusionSolveSteady(omega,tol,n,method):
 rho = 2707; c = 896; k = 204; alpha = k / (rho * c)
 Ly = 0.015; Lx = Ly/6; Lt = 60; qin = 30000; Tf = 300; hx = 60; hy = 100
 dx = Lx/n; nx = n + 1; ny = 6*n + 1
 beta = 0.25; ntmax = 10000000

 tic = timer()
 T = Tf * ones((nx+2,ny+2))
 E = zeros(ntmax); Echeck = 2*tol; it = 0

 while (abs(Echeck) >= tol and abs(Echeck) <= 2.0 and it < ntmax) :
   T[1:-1,0]  = T[1:-1,2]  - 2*dx*hy*(T[1:-1,1]-Tf) /k
   T[1:-1,-1] = T[1:-1,-3] + 2*dx*qin /k
   T[0,1:-1]  = T[2,1:-1]
   T[-1,1:-1] = T[-3,1:-1] - 2*dx*hx*(T[-2,1:-1]-Tf) /k

   Qin  =  qin * Lx
   Qout = (sum(T[2:-2,0] - T[2:-2,2])*k/2 +
           sum(T[[1,-2],0] - T[[1,-2],2])*k/4 +
           sum(T[-1,2:-2] - T[-3,2:-2])*k/2 +
           sum(T[-1,[1,-2]] - T[-3,[1,-2]])*k/4 )
   Echeck = E[it] = abs(Qin + Qout) / Qin

   if (method == "Gauss-Seidel") :
     for i in range(1,nx+1):
       for j in range(1,ny+1):
         T[i,j] += omega * beta * (T[i+1,j] + T[i-1,j]
                                 + T[i,j+1] + T[i,j-1] - 4*T[i,j])
   if (method == "Jacobi") :
     T[1:-1,1:-1] += omega * beta*(T[2:,1:-1] + T[:-2,1:-1]
                       + T[1:-1,2:] + T[1:-1,:-2] - 4*T[1:-1,1:-1])
```

```
    it = it+1

  toc = timer() - tic
  print(' === %s : omega = %4.2f :  elapsed time is %f seconds.' % (method,omega,toc))
  print('   T(0,Ly) = %10.6f [K] after %d steps ' % (T[1,-2],it))
  print('   |Qin-Qout|/Qin = %14.6e    ' % E[it-1])

  return E[0:it]
```

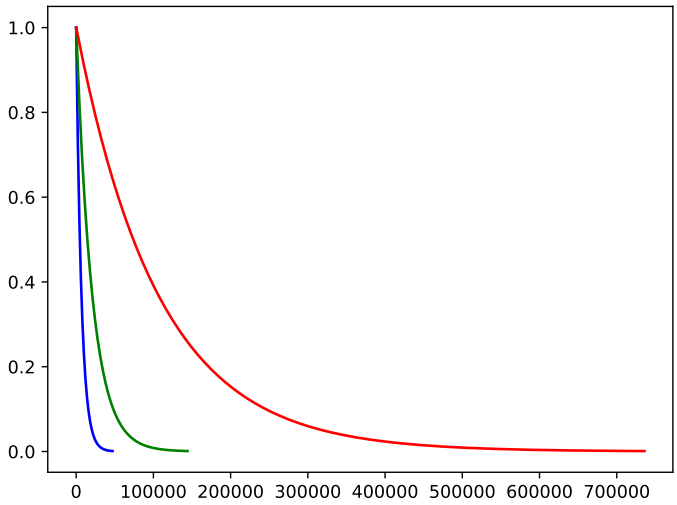


Figure 11.6: Evolution de $e^n = |Q_{in}^n - Q_{out}^n|/Q_{in}^n$ en fonction de n : Gauss-Seidel, $\omega = 1.87$ (bleu) , Gauss-Seidel, $\omega = 1.50$ (vert), Jacobi, $omega = 1.00$ (rouge)

3. La méthode de Gauss-Seidel ne peut pas être vectorisée mais avec un facteur de relaxation optimal $\omega = 1.87$, le gain en nombre d'itérations est tel que cela reste un meilleur choix que la méthode de Jacobi qui est vectorisée mais qui converge plus lentement. Toutefois, lorsque la taille de problème devient encore plus importante, le constat s'inverse.

   Les résultats obtenus en exécutant notre programme sont :

```
=== Gauss-Seidel : omega = 1.87 :  elapsed time is 14.489338 seconds.
  T(0,Ly) = 366.099943 [K] after 47046 steps
  |Qin-Qout|/Qin =   9.999992e-04
=== Gauss-Seidel : omega = 1.50 :  elapsed time is 45.547707 seconds.
  T(0,Ly) = 366.099921 [K] after 144120 steps
  |Qin-Qout|/Qin =   9.999960e-04
=== Jacobi : omega = 1.00 :  elapsed time is 45.425531 seconds.
  T(0,Ly) = 366.099909 [K] after 735885 steps
  |Qin-Qout|/Qin =   9.999907e-04
```

   Il est encore possible de faire mieux en modifiant subtilement la valeur initiale de la température...

4. Une étude de sensibilité permet d'observer qu'une valeur de $\omega = 1.87$ semble être optimale. Pour un domaine carré, le calcul théorique de la valeur optimale du paramètre de surrelaxation est déduite dans le livre d'Haberman qui sert de référence dans le cours LEPL1103 (pages 264-265).

Equation d'ondes : solutions

78

1. L'énergie potentielle accumulée sous la forme d'énergie de déformation élastique de la corde est donnée par :

$$E_p(t) \;=\; \frac{T_0}{2}\int_0^L \left(\frac{\partial u}{\partial x}(x,t)\right)^2 \, dx$$

2. L'énergie cinétique est donnée :

$$E_c(t) \;=\; \frac{\rho}{2}\int_0^L \left(\frac{\partial u}{\partial t}(x,t)\right)^2 \, dx$$

3. L'énergie totale et sa variation dans le temps peuvent s'écrire aisément.

$$\begin{aligned}
\overbrace{E_c(t)+E_p(t)}^{E} &= \frac{\rho}{2}\int_0^L \left(\frac{\partial u}{\partial t}\right)^2 dx + \frac{\rho c^2}{2}\int_0^L \left(\frac{\partial u}{\partial x}\right)^2 dx \\
\Big\downarrow & \\
\frac{dE}{dt} &= \rho\int_0^L \left(\frac{\partial^2 u}{\partial t^2}\frac{\partial u}{\partial t} + c^2\frac{\partial u}{\partial x}\frac{\partial^2 u}{\partial x \partial t}\right) dx \\
&= \rho\int_0^L \left(c^2\frac{\partial^2 u}{\partial x^2}\frac{\partial u}{\partial t} + c^2\frac{\partial u}{\partial x}\frac{\partial^2 u}{\partial x \partial t}\right) dx \\
&= \rho c^2\int_0^L \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\frac{\partial u}{\partial t}\right) dx \;=\; \rho c^2\left[\frac{\partial u}{\partial x}\frac{\partial u}{\partial t}\right]_0^L
\end{aligned}$$

Lorsque les deux extrémités de la corde sont fixées, on en déduit bien que l'énergie totale est conservée.

79

1. En tirant profit des conditions aux limites et de la condition initiale en vitesse, on déduit aisément que la solution analytique s'écrira sous la forme suivante :

$$u(x,t) = \sum_{n=1}^{\infty} C_n \sin\left(\frac{n\pi x}{L}\right)\cos\left(\frac{n\pi c t}{L}\right) \qquad \text{avec } c = \sqrt{\frac{T_0}{\rho}}.$$

Pour obtenir les coefficients, il faut ensuite imposer la condition initiale...
ce qui revient à écrire...

$$\begin{aligned}
u(x,0) &= u_0\left(\frac{x^2}{L^2} - \frac{x^3}{L^3}\right), \\
\sum_{n=1}^{\infty} C_n \sin\left(\frac{n\pi x}{L}\right) &= u_0\left(\frac{x^2}{L^2} - \frac{x^3}{L^3}\right), \\
\Big\downarrow & \quad \text{En vertu de l'orthogonalité des sinus,} \\
\frac{L\,C_n}{2} &= u_0\int_0^L \left(\frac{x^2}{L^2} - \frac{x^3}{L^3}\right)\sin\left(\frac{n\pi x}{L}\right) dx,
\end{aligned}$$

En effectuant ensuite l'intégrale sur l'intervalle unité et aussi en observant que les termes en sinus des primitives ont une contribution nulle (et peuvent donc être omis...), la partie calculatoire se réduit énormément

$$\begin{aligned} C_n &= 2u_0 \int_0^1 \left(t^2 - t^3\right) \sin\left(n\pi t\right) dt \\ &= 2u_0 \left[\left(\frac{2-(n\pi)^2 t^2}{(n\pi)^3} - \frac{6t-(n\pi)^2 t^3}{(n\pi)^3}\right) \cos\left(n\pi t\right)\right]_0^1 \\ &= 2u_0 \left(\frac{2-(n\pi)^2-6+(n\pi)^2}{(n\pi)^3}\ (-1)^n - \frac{2}{(n\pi)^3}\right) = \frac{4u_0}{n^3\pi^3}\Big(2(-1)^{n+1}-1\Big) \end{aligned}$$

On a bien ainsi obtenu l'expression demandée.

2. Un programme possible est :

```
def waveAnalytic(x,t,L,c,u0):
 tol = 1e-8; nmax = 2000; n = 1; errorloc = 2*tol
 u = zeros(size(x)); delta = zeros(size(x))
 error = zeros(nmax+1)
 error[0] = errorloc
 while errorloc > tol and n < nmax:
   delta = ( sin(n*pi * x/L)
           * cos(n*pi * c*t/L)
           * (2*(-1)**(n+1) - 1)/(n**3) )
   u = u + delta
   error[n] = max(abs(delta))
   errorloc = max(error[n],error[n-1])
   n = n+1
 u = u * 4 * u0/(pi**3)
 if (n == nmax) :
   print("Error, not converged yek, yek :-(")
 return u,error,n
```

Estimer le nombre de termes requis pour obtenir une précision donnée n'est pas aussi simple qu'on pourrait le croire. Comme certains termes de la série sont parfois nuls, il est donc nécessaire d'au moins tenir compte de deux termes pour construire un critère d'arrêt un brin fiable... Le nombre de termes requis est de 595 pour une estimation de 101 points à mi-période. L'apparition des pics est directement liée aux termes qui s'annulent en tous les points de calcul. Augmenter d'un facteur deux le nombre de points d'évaluation diminuera d'un même facteur le nombre de pics !

3. L'énergie totale est obtenue en intégrant l'énergie de déformation de la solution initiale.

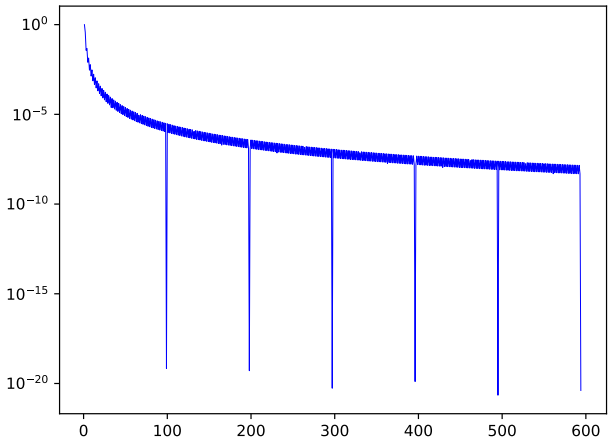


Figure 12.1: Evolution du maximum de la valeur absolue des termes de la solution analytique : on observe à intervalles réguliers la présence de termes nuls aux erreurs machine près. $(n = 100, t = 0.1s)$.

$$
\begin{aligned}
E &= \frac{\rho c^2}{2}\int_0^L \left(\frac{\partial u}{\partial x}(x,0)\right)^2 dx \\
&= \frac{\rho c^2 u_0^2}{2}\int_0^L \left(\frac{2x}{L^2} - \frac{3x^2}{L^3}\right)^2 dx \\
&= \frac{\rho c^2 u_0^2}{2}\int_0^L \left(\frac{4x^2}{L^4} - \frac{12x^3}{L^5} + \frac{9x^4}{L^6}\right) dx \\
&= \frac{\rho c^2 u_0^2}{2}\left[\frac{4x^3}{3L^4} - \frac{12x^4}{4L^5} + \frac{9x^5}{5L^6}\right]_0^L \\
&= \frac{\rho c^2 u_0^2}{2}\left(\frac{4}{3L} - \frac{3}{L} + \frac{9}{5L}\right) = \frac{\rho c^2 u_0^2}{2L}\left(\frac{20-45+27}{15}\right) = \frac{\rho c^2 u_0^2}{15L}
\end{aligned}
$$

Pour les valeurs numériques fournies, l'énergie totale vaut

$$E = \frac{10^{-3}10^{2}10^{-6}}{15}[J] = 6.67\ 10^{-9}[J].$$

80

1. Une implémentation possible est donnée par :

```
from scipy.sparse import spdiags
from scipy.integrate import solve_ivp

def waveRungeKutta(n,Lt,c,Uo):
  m = n+1
  e = array([0,*ones(m-2),0])
  D = spdiags([e,-2*e,e],[-1,0,1],m,m)
  Dt = (D.T)*(c*c)/(dx*dx)

  def dfdt(t,y):
    u = y[:m]; v = y[m:]
```

```
    dudt = v; dvdt = Dt * u
    return [*dudt,*dvdt]

  y0 = zeros(2*m); y0[0:m] = Uo
  return solve_ivp(dfdt,[0,Lt],y0,method="RK45",rtol=1e-6,atol=1e-9)
```

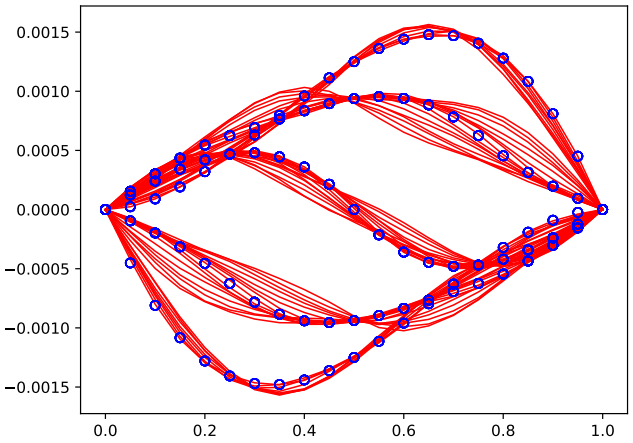


Figure 12.2: Méthode de Runge-Kutta adaptative : superposition de la solution numérique calculée sur une période (lignes continues $m = 21$) et de la solution exacte (cercles)..

2. L'obtention des solutions se fait très aisément de la manière suivante :

```
from scipy.interpolate import interp1d

sol = waveRungeKutta(m-1,Lt,c,Uo)

plt.figure()
t = sol.t; tplot = linspace(0,Lt,8*nP+1)
y = sol.y; yplot = interp1d(t,y)(tplot)
plt.plot(x,yplot[:m,:],'-r',linewidth=1)
plt.plot(x,analytic(x,tplot),'ob',markerfacecolor='none')
```

3. L'évolution de l'énergie (cinétique, potentielle et totale) est donnée sur la Figure **??**. L'évolution de l'énergie (cinétique, potentielle et totale) s'obtient de la manière suivante :

```
u=y[:m,:]; v=y[m:,:]
Ec = zeros(len(t)); Ep = zeros(len(t))
for iter in range(len(t)):
  Ec[iter] =     rho*dx * sum(v[:,iter]**2) / 2;
  Ep[iter] = rho*c*c*dx * sum((u[1:,iter]-u[0:-1,iter])**2)/(2*dx*dx)
t = t*c/2*L
plt.plot(t,Ec,'-b',t,Ep,'-r',t,Ep+Ec,'-g',[0,nP],[E,E],'-k',linewidth=1)
```

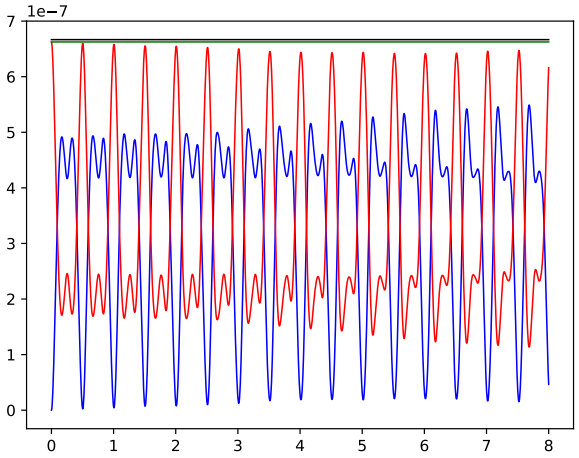


Figure 12.3: Méthode de Runge-Kutta adaptative : évolution de E (vert), E_c (bleu) et E_p (rouge) en fonction du temps adimensionnalisé par la période. La valeur analytique est indiquée en noir..

4. Le pas de temps oscille entre 10^{-3} et 10^{-4}. Typiquement, le pas de temps est principalement limité par l'apparition d'instabilité lorsque celui devient trop important : cette limite est proche de celle qu'on obtient avec l'analyse de stabilité théorique faite pour la méthode des différences finies centrées !

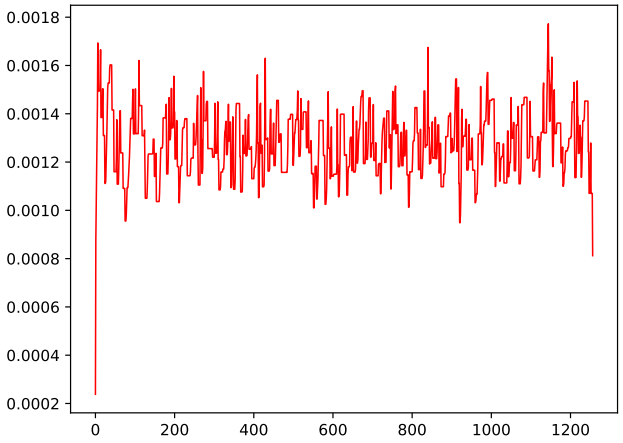


Figure 12.4: Méthode de Runge-Kutta adaptative : évolution du pas de temps en fonction du nombre d'itérations sur une période ($m = 21$)..

81

1. Pour effectuer l'analyse de stabilité, on cherche à obtenir une estimation du facteur d'amplification d'une perturbation quelconque de la forme suivante :

$$U_i^n = U^n e^{ikX_i}$$

Cette perturbation évoluera comme suit lors d'un pas de temps :

$$\begin{aligned}
U_i^{n+1} &= 2U_i^n + \beta^2\Big(U_{i+1}^n - 2U_i^n + U_{i-1}^n\Big) - U_i^{n-1} \\
&= U_i^n\Big(2 + \beta^2\Big(e^{ik\Delta x} + e^{-ik\Delta x} - 2\Big)\Big) - U_i^{n-1} \\
&= U_i^n\Big(2 + \beta^2\Big(2\cos(k\Delta x) - 2\Big)\Big) - U_i^{n-1} \\
&= U_i^n\Big(2 - 4\beta^2 \sin^2\left(\tfrac{k\Delta x}{2}\right)\Big) - U_i^{n-1}
\end{aligned}$$

De cette dernière relation, on voit que le facteur d'amplification U est une solution de l'équation du second degré :

$$U^2 - 2aU + 1 = 0$$

$$\downarrow$$

$$U = a \pm \sqrt{a^2 - 1}$$

avec $a = 1 - 2\beta^2 \sin^2(\frac{k\Delta x}{2})$. Pour que la perturbation ne s'accroisse pas, il faut que pour toutes valeurs de k, le module du facteur d'amplification U (qui peut être éventuellement complexe !) reste inférieur à l'unité.

Dans le cas de racines complexes conjuguées, le module de celles-ci vaudra toujours l'unité ($|U| = \sqrt{a^2 + 1 - a^2} = 1$). Dans le cas de racines réelles, on observe que le produit des racines vaut l'unité. Imposer que la valeur absolue de ces racines soit inférieure à l'unité, revient à se restreindre au cas unique de la racine double de valeur unitaire. La condition de stabilité revient simplement à exiger que $a^2 \leq 1$ et à écrire :

$$\begin{aligned}
\left(1 - 2\beta^2 \sin^2(\frac{k\Delta x}{2})\right)^2 &\leq 1 \\
1 - 4\beta^2 \sin^2(\frac{k\Delta x}{2}) + 4\beta^4 \sin^4(\frac{k\Delta x}{2}) &\leq 1 \\
-1 + \beta^2 \sin^2(\frac{k\Delta x}{2}) &\leq 0
\end{aligned}$$

$$\downarrow \quad \text{En prenant le cas le plus défavorable}$$

$$\beta^2 \leq 1$$

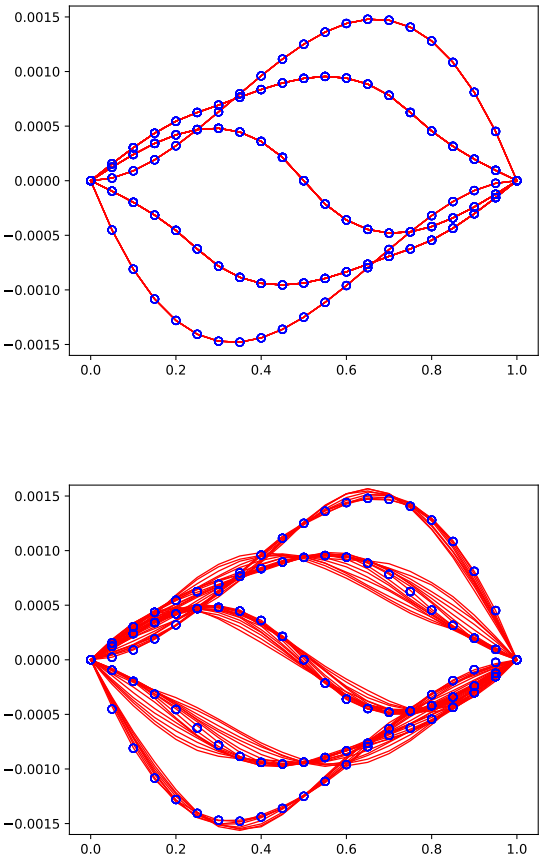


Figure 12.5: Différences finies centrées avec $\beta = 1.0$ (haut) et $\beta = 0.5$ (bas): superposition de la solution numérique calculée sur huit périodes (lignes continues $m = 21$) et de la solution exacte (cercles)..

On obtient donc finalement comme condition de stabilité, l'inégalité suivante sur les pas de discrétisation spatiale et temporelle.

$$\boxed{\beta \quad = \quad \frac{c\Delta t}{\Delta x} \quad \leq \quad 1}$$

Dans un tel cas, les perturbations introduiront un comportement oscillant de la solution et ne seront pas dissipées dans le temps : ce qui correspond exactement au comportement physique de l'équation d'onde. Notre schéma sera donc numériquement stable, sans être "trop stable" et n'introduira pas de dissipation numérique.

2. Une implémentation possible est donnée par :

```
from numpy import *
from scipy.sparse import spdiags


def waveSolve(beta,nx,nt,c,L,Uo):
```

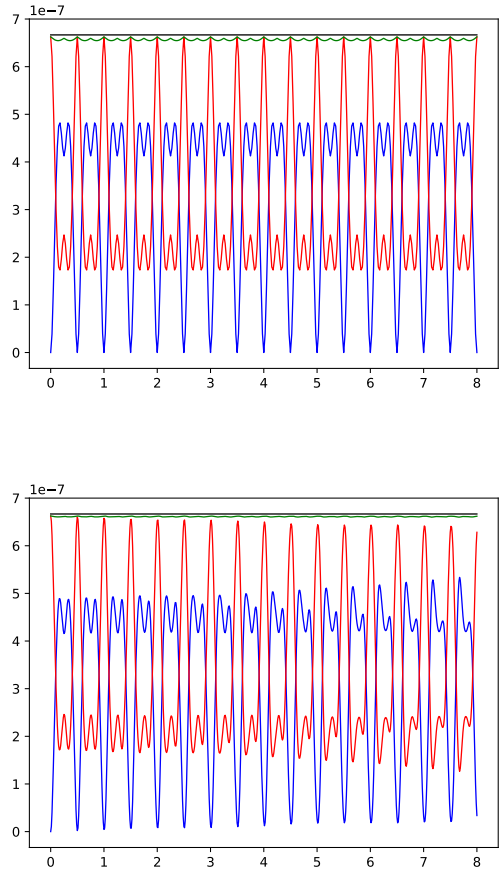


Figure 12.6: Différences finies centrées avec $\beta = 1.0$ (haut) et $\beta = 0.5$ (bas): évolution de E (vert), E_c (bleu) et E_p (rouge) en fonction du temps adimensionnalisé par la période. La valeur analytique est indiquée en noir..

```
dx = L/nx; dt = dx * beta/c

e = array([0,*ones(nx-1),0])
D = spdiags([e,-2*e,e],[-1,0,1],nx+1,nx+1)
D = D.T / (dx*dx)

Uoo = (2*Uo + ((c*dt)**2)*D @ Uo)/2
for t in range(nt):
  U = 2*Uo + ((c*dt)**2 * D @ Uo ) - Uoo
  Uoo = copy(Uo)
  Uo = copy(U)
return U
```

3. Avec un nombre de noeuds $m = 21$, la comparaison des résultats pour $\beta = 1.0$ et $\beta = 0.5$ avec la solution analytique est donnée ci-dessous. ci-dessous.

```
beta =  1.00e+00
```

```
Results in x = L/2
 Numerical and analytical values  1.250000e-03  1.250000e-03 at time = 0.000
 Numerical and analytical values  9.375000e-04  9.375000e-04 at time = 0.025
 Numerical and analytical values  0.000000e+00  1.799515e-19 at time = 0.050
 Numerical and analytical values -9.375000e-04 -9.375000e-04 at time = 0.075
 Numerical and analytical values -1.250000e-03 -1.250000e-03 at time = 0.100
 Numerical and analytical values -9.375000e-04 -9.375000e-04 at time = 0.125
 Numerical and analytical values  1.101549e-18  1.152539e-18 at time = 0.150
 Numerical and analytical values  9.375000e-04  9.375000e-04 at time = 0.175
 Numerical and analytical values  1.250000e-03  1.250000e-03 at time = 0.200

beta =  5.00e-01
Results in x = L/2
 Numerical and analytical values  1.250000e-03  1.250000e-03 at time = 0.000
 Numerical and analytical values  9.375000e-04  9.375000e-04 at time = 0.025
 Numerical and analytical values  8.447985e-06  1.799515e-19 at time = 0.050
 Numerical and analytical values -9.352734e-04 -9.375000e-04 at time = 0.075
 Numerical and analytical values -1.250979e-03 -1.250000e-03 at time = 0.100
 Numerical and analytical values -9.369128e-04 -9.375000e-04 at time = 0.125
 Numerical and analytical values -1.802991e-05  1.152539e-18 at time = 0.150
 Numerical and analytical values  9.397669e-04  9.375000e-04 at time = 0.175
 Numerical and analytical values  1.249430e-03  1.250000e-03 at time = 0.200
```

La comparaison de la conservation de l'énergie pour les deux cas $\beta = 1.0$ et $\beta = 0.5$ est fournie sur la Figure **??**. On observe à nouveau une meilleure précision pour $\beta = 1.0$ lorsqu'on est à la limite de la stabilité numérique... Diminuer le pas de temps rend paradoxalement la solution numérique moins bonne !

4. L'instabilité est toujours un peu délicate à visualiser, il faut se limiter à un quart de période pour pouvoir observer un joli développement...

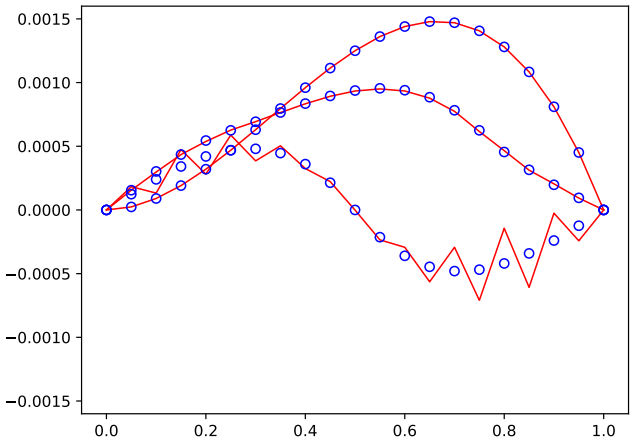


Figure 12.7: Différences finies centrées avec $\beta = 1.1$: développement de l'instabilité (lignes continues $n = 21$) et de la solution exacte (cercles)..

Question d'examens : solutions

Remarque générale : le contenu et le cahier de charges du cours ayant été largement modifiés par les réformes pédagogiques successives, il faut parfois regarder avec un oeil critique certaines questions d'examen des années précédentes : elles sont nettement inférieures ou supérieures aux exigences de compétences de cette année-ci.

82 **Juin 2001**

1. L'erreur de discrétisation est la différence entre la solution d'un problème mathématique et la solution numérique (calculable avec un ordinateur) d'une approximation de ce problème. Afin de limiter cette erreur qui est souvent écrite sous la forme $\mathcal{O}(h^n)$, on peut
   - diminuer le pas de discrétisation h. (♠ erreur d'arrondis !)
   - augmenter l'ordre de la méthode n. (♠ phénomène de Runge !)

   Peu d'étudiants respectent la contrainte de longueur. Nombreux sont ceux qui tombent dans les remarques anecdotiques au détriment d'une réponse de synthèse.

2. Démontrons la relation suivante :

$$\boxed{u''''(x) \approx \frac{u(x-2h) - 4u(x-h) + 6u(x) - 4u(x+h) + u(x+2h)}{h^4} + \mathcal{O}(h^2)}$$

   Il suffit d'écrire les développements en série de Taylor suivants :

$$u(x \pm h) = u(x) \pm hu'(x) + \frac{h^2}{2}u''(x) \pm \frac{h^3}{6}u'''(x) + \frac{h^4}{24}u^{(4)}(x) \pm \frac{h^5}{120}u^{(5)}(x) + \mathcal{O}(h^6)$$

$$u(x \pm 2h) = u(x) \pm 2hu'(x) + \frac{4h^2}{2}u''(x) \pm \frac{8h^3}{6}u'''(x) + \frac{16h^4}{24}u^{(4)}(x) \pm \frac{32h^5}{120}u^{(5)}(x) + \mathcal{O}(h^6)$$

   On effectue ensuite la combinaison linéaire de la relation à démontrer. On peut observer immédiatement que tous les termes impairs disparaissent par symétrie et on obtient finalement

$$\begin{aligned}
&\frac{u(x-2h) - 4u(x-h) + 6u(x) - 4u(x+h) + u(x+2h)}{h^4} \\
&= \left(\underbrace{1 - 4 + 6 - 4 + 1}_{0}\right)\frac{u(x)}{h^4} + \left(\underbrace{8h^2 - 8h^2}_{0}\right)\frac{u''(x)}{2h^4} + \left(\underbrace{32h^4 - 8h^4}_{24h^4}\right)\frac{u''''(x)}{24h^4} + \frac{\mathcal{O}(h^6)}{h^4} \\
&= u''''(x) + \mathcal{O}(h^2)
\end{aligned}$$

   Pour obtenir la meilleure estimation de l'ordre de l'erreur, il faut considérer des développements jusqu'au terme d'ordre six ! La justification adéquate de l'ordre de l'erreur fait évidemment partie de la réponse à fournir.

   Il est également possible d'obtenir cette relation en utilisant le fait qu'une dérivée quatrième est une dérivée seconde d'une dérivée seconde et d'appliquer récursivement la relation de la dérivée seconde des notes de cours et de référer à la démonstration y correspondant. Peu d'étudiants ont choisi cette voie qui est pourtant plus rapide et moins calculatoire.

   Cet exercice est une extrapolation directe du développement effectué dans les notes de cours. Il s'agit donc d'une question d'un niveau relativement élémentaire.

83 **Septembre 2001**

Une approximation de la dérivée première d'une fonction u en un point X_i peut être effectuée au moyen d'une différence centrée écrite sous la forme

$$u'(X_i) \approx \frac{U_{i+1} - U_{i-1}}{2h}$$

où h est l'écart entre deux abscisses voisines et U_i est la valeur de la fonction u en un point X_i. Une telle approximation permet de construire une méthode connue sous le nom de *leapfrog method*, définie par la relation :

$$U_{i+1} = U_{i-1} + 2hf(X_i, U_i),$$

1. Il s'agit d'une méthode à pas liés car pour calculer U_{i+1}, il est nécessaire de connaître U_i et U_{i-1}.
2. Cette méthode peut être utilisée seulement lorsqu'on possède U_0 et U_1.
   Il n'est donc pas possible de commencer directement le calcul.
3. Pour déterminer l'ordre de précision de la méthode, il suffit d'écrire

$$\begin{aligned} u'(X_i) &= f(X_i, U_i) \\ &\downarrow \quad \text{En approximant } u'(X_i) \text{ par la différence centrée d'ordre deux} \\ \frac{U_{i+1} - U_{i-1}}{2h} + \mathcal{O}(h^2) &= f(X_i, U_i) \\ U_{i+1} &= U_{i-1} + 2hf(X_i, U_i) + \mathcal{O}(h^3) \end{aligned}$$

   Il s'agit donc d'une méthode d'ordre deux comme la méthode de Heun. On gagne un ordre de précision par l'utilisation d'une différence centrée pour estimer la dérivée, contrairement aux méthodes d'Euler d'ordre un basées sur des différences décentrées.
4. Pour effectuer l'analyse de stabilité, on demande de considérer le problème modèle

$$\begin{cases} u' = \lambda u \\ u(0) = U_0 \end{cases}$$

   où λ est un réel négatif. Le problème modèle est donc stable, mais cela ne signifie évidemment pas que la méthode numérique sera numériquement stable pour la résolution de ce problème.
   Pour déterminer la stabilité numérique d'une méthode, il faut montrer que les erreurs numériques propagées ne s'accroissent pas. Dans cette optique, on utilise le changement de variable $U_i = a^i U_0$ (où a peut être interprété comme le facteur d'amplification des erreurs propagées) dans la formule de la méthode et on obtient :

$$\begin{aligned} 2h\lambda a^i U_0 &= a^{i+1}U_0 - a^{i-1}U_0 \\ &\downarrow \quad \text{En simplifiant par } a^{i-1}U_0 \text{ ,} \\ a^2 - 2h\lambda a - 1 &= 0 \\ a &= h\lambda \pm \sqrt{h^2\lambda^2 + 1} \end{aligned}$$

La méthode sera numériquement stable pour les valeurs de $h\lambda$ telles que le facteur d'amplification soit de norme inférieure à 1

$$|h\lambda \pm \sqrt{h^2\lambda^2 + 1}| \leq 1$$

Comme $h\lambda$ est un réel négatif, le cas le plus critique est atteint pour le signe négatif de la racine carrée. On peut donc écrire comme condition de stabilité numérique pour $h > 0$ et $\lambda < 0$

$$\begin{aligned} h\lambda - \sqrt{h^2\lambda^2 + 1} &\geq -1 \\ -\sqrt{h^2\lambda^2 + 1} &\geq -h\lambda - 1 \\ h^2\lambda^2 + 1 &\leq h^2\lambda^2 + 1 + 2h\lambda \\ 0 &\leq 2h\lambda \end{aligned}$$

Cette relation n'est jamais satisfaite. Au mieux, l'égalité est obtenue pour $h\lambda = 0$: ce qui signifie que dans un tel cas, les erreurs propagées ne s'accroissent pas, mais ne décroissent pas non plus.

Dans le cas $h\lambda$ complexe (non-demandé), on peut montrer que l'égalité est obtenue pour le segment de droite allant de $-i$ à i. Un tel résultat peut permettre de comprendre intuitivement la stabilité numérique des méthodes de différences finies centrées pour une équation différentielle du second ordre.

84 Septembre 2002

On souhaite calculer la dérivée seconde de la fonction $f(x) = x\ \tan(x)$ en $x = 0.9$, en utilisant uniquement les valeurs (arrondies à quatre chiffres après la virgule) de la table ci-dessous.

x	0.8000	0.8500	0.9000	0.9500	1.0000
$x\tan(x)$	0.8237	0.9676	1.1341	1.3285	1.5574

1. Cette question correspond à une très légère variation du développement effectué dans les notes de cours pour l'obtention d'une formule unilatérale pour une dérivée première. Il s'agit donc d'une question plutôt simple.

   En considérant $X_0 = 0.8$, $X_1 = 0.9$, $X_2 = 1.0$, $U_0 = 0.8237$, $U_1 = 1.1341$, $U_2 = 1.5574$ et $h = 0.1$, on obtient le polynôme d'interpolation comme :

$$p(x) = U_0\ \frac{(x - X_1)(x - X_2)}{2h^2} - U_1\ \frac{(x - X_0)(x - X_2)}{h^2} + U_2\ \frac{(x - X_0)(x - X_1)}{2h^2}$$

$$\downarrow$$

$$p''(x) = \underbrace{\frac{U_0 - 2U_1 + U_2}{h^2}}_{\text{constant !}} = \frac{0.8237 - 2.2682 + 1.5574}{0.01} = 11.29$$

   On obtient ainsi la différence finie centrée d'ordre deux avec $h = 0.1$ pour $x = 0.9$.

   La réponse est donc $\boxed{f''(0.9) \approx 11.29}$

2. Nous disposons déjà du résultat pour $h = 0.1$ et il est possible de calculer la même différence centrée en utilisant les points 0.85, 0.90 et 0.95 et d'effectuer une extrapolation de Richardson sur ces deux résultats.

   La réponse est donc
   $$\begin{array}{rcl} D_{0,0} & = & 11.29 \\ D_{1,0} & = & \dfrac{0.9676 - 2.2682 + 1.3285}{\frac{1}{4}\ 0.01} \ = \ 11.16 \\ D_{1,1} & = & \dfrac{4\ 11.16 - 11.29}{3} \ = \ 11.12 \end{array}$$

   Comme on divise le pas par un facteur deux et que l'on tire profit du fait que le terme d'erreur ne contient que des puissances paires dont le premier terme est en h^2, le dénominateur est donc $2^2 - 1 = 3$.

3. L'erreur théorique de $D_{0,0}$ et $D_{1,0}$ est en $\mathcal{O}(h^2)$.
   L'erreur théorique de $D_{1,1}$ est en $\mathcal{O}(h^4)$.
   L'impact des erreurs d'arrondi des données est fourni par l'expression suivante du formulaire (à ne pas redéduire donc !)
   $$|\tilde{E}^h| \leq \frac{4\epsilon}{h^2} + \frac{C_4 h^2}{12}$$
   avec $\epsilon = 10^{-4}$. On peut directement y observer que le premier terme qui contient les erreurs d'arrondi est de l'ordre de 10^{-2}. Le second terme qui contient l'erreur théorique est du même ordre de grandeur ! A priori, il est donc possible et même très probable que les estimations théoriques ne soient pas fiables[6].

Il n'est réellement pas nécessaire de disposer d'une calculatrice pour répondre à cette question. L'expression analytique de la dérivée seconde valait évidemment $f''(x) = (2 + 2x\tan(x))(1 + \tan(x)^2)$. Comme vous disposez maintenant de votre calculatrice, cela devrait vous permettre de calculer les erreurs réelles et de les comparer aux estimations... Et c'est évidemment plus facile !

85 Juin 2004 : Modèle de Maxwell

Le modèle de Maxwell se réduit au problème de Cauchy

$$\lambda u'(t) + u(t) = \mu\dot{\gamma} \qquad u(0) = 0$$

où $\lambda > 0$ et $\mu > 0$ représentent respectivement un temps de relaxation et une viscosité du matériau.

[6] Pour illustrer ce propos, les mêmes calculs effectués avec des données en double précision sur MATLAB fournissent des résultats nettement meilleurs :

i	$D_{i,0}$	$D_{i,1}$
0	11.2834	
1	11.1046	11.0450

On observe maintenant que l'on double bien le nombre des chiffres exacts (la valeur exacte est égale à 11.0463) en effectuant l'extrapolation de Richardson. Ce n'était pas le cas dans le cadre du calcul effectué avec des données à quatre chiffres après la virgule. Les erreurs d'arrondi sur les données ont donc bien un impact très important ! Dernière question laissée à votre sagacité : est-il possible maintenant d'estimer C_4 et comment ?

1. Il s'agit vraiment d'une question élémentaire...

La solution du problème de Cauchy est : $\boxed{u(t) \quad = \quad \mu\dot{\gamma}\left(1 - e^{-t/\lambda}\right)}$

2. Le jacobien de ce problème ($J = -1/\lambda$) est toujours négatif. Une des difficultés de l'exercice consiste à remarquer que le temps de relaxation vaut l'opposé de l'inverse de λ^* du problème modèle $u' = \lambda^* u$. En d'autres mots, λ n'est pas ce qu'on pourrait croire ! La méthode d'Euler sera stable si $|1 - h/\lambda| \; < \; 1$.

La réponse est donc : $\boxed{0 \; < \quad h \quad < \; 2\lambda}$

3. Pour obtenir les dérivées du membre de droite afin de faire apparaître une relation algébrique ne faisant intervenir que U_i, h, λ, μ et $\dot{\gamma}$, il suffit d'effectuer les développements suivants :

$$f(t) = \frac{\mu\dot{\gamma} - u(t)}{\lambda} \qquad f'(t) = \frac{-u'(t)}{\lambda} = \frac{-\mu\dot{\gamma} + u(t)}{\lambda^2} \qquad f''(t) = \frac{u'(t)}{\lambda^2} = \frac{\mu\dot{\gamma} - u(t)}{\lambda^3}$$

et de conclure que : $\boxed{U_{i+1} \quad = \quad U_i + h\left[\left(\frac{\mu\dot{\gamma} - U_i}{\lambda}\right) + \frac{h}{2!}\left(\frac{-\mu\dot{\gamma} + U_i}{\lambda^2}\right) + \frac{h^2}{3!}\left(\frac{\mu\dot{\gamma} - U_i}{\lambda^3}\right)\right]}$

4. Pour estimer l'ordre de précision de l'erreur locale de la méthode (fictive) de Petrov-Smacholowski-Birnov, il suffit d'écrire que :

$$U_{i+1} = -4\ U_i + 5U_{i-1} + h\left(4F_i + 2F_{i-1}\right)$$

$$U_{i+1} = -4\ U_i + 5U_i - 5hU_i' + 5\frac{h^2}{2}U_i'' - 5\frac{h^3}{6}U_i''' + \mathcal{O}(h^4) + h\left(4U_i' + 2U_i' - 2hU_i'' + h^2U_i''' + \mathcal{O}(h^3)\right)$$

$$U_{i+1} = (5-4)U_i + (6-5)hU_i' + (5-4)\frac{h^2}{2}U_i'' + (6-5)\frac{h^3}{6}U_i''' + \mathcal{O}(h^4)$$

$$U_{i+1} = U_i + hU_i' + \frac{h^2}{2}U_i'' + \frac{h^3}{6}U_i''' + \mathcal{O}(h^4)$$

et de conclure que l'ordre de l'erreur locale est quatre et la méthode est donc d'ordre trois.

Pour effectuer l'analyse de stabilité, il faut étudier la valeur absolue des deux racines du polynôme
$a^2 = -4a + 5 - 4\frac{h}{\lambda}a - 2\frac{h}{\lambda}$ où λ est le temps de relaxation (toujours à ne pas confondre avec λ^* !)

Les deux racines sont : $\boxed{a \quad = \quad -\left(2 + \frac{2h}{\lambda}\right) \pm \sqrt{\left(2 + \frac{2h}{\lambda}\right)^2 + \left(5 - \frac{2h}{\lambda}\right)}}$

Il n'existe aucune valeur de h qui fournisse un schéma stable.
A titre d'exemple, pour $h = 0$, on observe deux racines $a_1 = -5$ et $a_2 = 1$.
Ce schéma est donc inconditionnellement instable.

86 Septembre 2004 : Quadrature de l'été

La méthode d'intégration numérique sur l'intervalle $[0, h]$:

$$\underbrace{\int_0^h u(x)\ dx}_{I} \approx \underbrace{\frac{h}{4}\left(u(0) + \alpha\ u(\frac{2h}{3})\right)}_{I^h}$$

1. Il faut choisir $\alpha = 3$ pour obtenir le résultat demandé :

$$\begin{aligned}\int_0^h (bx + a)dx &= \frac{h}{4}(a + 2bh + 3a)\\ \frac{bh^2}{2} + ah &= \frac{bh^2}{2} + ah\end{aligned}$$

2. Le développement de Taylor d'ordre cinq autour de l'origine pour estimer $u(a)$ à partir des valeurs de la fonction et de ses dérivées à l'origine est :

$$u(a) = u(0) + a\ u'(0) + \frac{a^2\ u''(0)}{2} + \frac{a^3\ u'''(0)}{6} + \frac{a^4\ u''''(0)}{24} + \frac{a^5\ u'''''(0)}{120} + \mathcal{O}(a^6)$$

3. Le membre de droite s'écrit :

$$\begin{aligned}I^h &= hu(0) + \frac{3h}{4}\left(\frac{2h}{3}\ u'(0) + \left(\frac{2h}{3}\right)^2 \frac{u''(0)}{2} + \left(\frac{2h}{3}\right)^3 \frac{u'''(0)}{6} \dots\right)\\ &= hu(0) + \frac{h^2\ u'(0)}{2} + \frac{h^3\ u''(0)}{6} + \frac{h^4\ u'''(0)}{27} + \dots\end{aligned}$$

Et le membre de gauche est obtenu par l'intégration du développement en série :

$$I = hu(0) + \frac{h^2\ u'(0)}{2} + \frac{h^3\ u''(0)}{6} + \frac{h^4\ u'''(0)}{24} + \dots$$

4. Le premier terme d'erreur est donc $h^4 u'''(0)\left(\frac{1}{27} - \frac{1}{24}\right)$ et l'ordre de précision est donc 4.

5. Le degré de précision est 2, car

$$\begin{aligned}\int_0^h (cx^2 + bx + a)dx &= \frac{h}{4}\left(a + \frac{4ch^2}{3} + 2bh + 3a\right)\\ \frac{ch^3}{3} + \frac{bh^2}{2} + ah &= \frac{ch^3}{3} + \frac{bh^2}{2} + ah\end{aligned}$$

87 Janvier 2006 : Dérivation numérique

1. Pour obtenir le résultat à partir de l'extrapolation de Richardson, il faut écrire que les différences centrées d'ordre deux s'écrivent en prenant respectivement h et $2h$.

$$\begin{aligned}D_h &= \frac{u(x+h) - u(x-h)}{2h}\\ D_{2h} &= \frac{u(x+2h) - u(x-2h)}{4h}\end{aligned}$$

Puisqu'il s'agit d'une approximation d'ordre deux, l'extrapolation de Richardson s'écrit :

$$\begin{aligned} u'(x) &= \frac{4D_h - D_{2h}}{3} = \frac{4u(x+h) - 4u(x-h)}{6h} - \frac{u(x+2h) - u(x-2h)}{12h} \\ &= \frac{-u(x+2h) + 8u(x+h) - 8u(x-h) + u(x-2h)}{12h} \end{aligned}$$ □

2. Il s'agit d'une différence centrée : il n'y a pas de termes d'erreur de degré impair. On peut donc prédire que cette estimation sera d'ordre quatre.

88 **Janvier 2006 : Equation de transport**

1. Les deux schémas sont explicites.
2. Les expressions de G pour les deux schémas s'obtiennent aisément en injectant la forme de la perturbation dans les relations de récurrence des deux schémas.
   - Pour le schéma de Lajos, on obtient :

$$\begin{aligned} U_j^{n+1} &= U_j^n - \frac{c\Delta t}{2\Delta x}\Big(U_{j+1}^n - U_{j-1}^n\Big) \\ &= U_j^n\Big(1 - \frac{c\Delta t}{2\Delta x}\Big(e^{ik\Delta x} - e^{-ik\Delta x}\Big)\Big) \\ &= U_j^n\Big(1 - \frac{i\ c\Delta t}{\Delta x}\Big(\sin(k\Delta x)\Big)\Big) \end{aligned}$$

   On en déduit donc que : $\boxed{G_{Lajos} = 1 - \frac{i\ c\Delta t}{\Delta x}\Big(\sin(k\Delta x)\Big)}$.

   - Pour le schéma de Lax, on obtient :

$$\begin{aligned} U_j^{n+1} &= \frac{1}{2}\Big(U_{j+1}^n + U_{j-1}^n\Big) - \frac{c\Delta t}{2\Delta x}\Big(U_{j+1}^n - U_{j-1}^n\Big) \\ &= U_j^n\Big(\frac{1}{2}\Big(e^{ik\Delta x} + e^{-ik\Delta x}\Big) - \frac{c\Delta t}{2\Delta x}\Big(e^{ik\Delta x} - e^{-ik\Delta x}\Big)\Big) \\ &= U_j^n\Big(\cos(k\Delta x) - \frac{i\ c\Delta t}{\Delta x}\Big(\sin(k\Delta x)\Big)\Big) \end{aligned}$$

   On en déduit donc que : $\boxed{G_{Lax} = \cos(k\Delta x) - \frac{i\ c\Delta t}{\Delta x}\Big(\sin(k\Delta x)\Big)}$.

3. Pour obtenir la condition de stabilité en termes de Δx et Δt pour les deux schémas, il faut effectuer le calcul du module de G_{Lajos} et G_{Lax} et vérifier que ce module soit inférieur à l'unité.
   - Pour le schéma de Lajos, on obtient

$$|G_{Lajos}|^2 = \underbrace{1 + \left(\frac{c\Delta t}{\Delta x}\right)^2 \sin^2(k\Delta x)}_{>1}$$

   On en conclut que le schéma est inconditionnellement instable.
   Lajos n'a jamais existé... et ce schéma est donc inutilisable.

– Pour le schéma de Lax, on obtient

$$\begin{aligned}
|G_{Lax}|^2 &= \cos^2(k\Delta x) + \left(\frac{c\Delta t}{\Delta x}\right)^2 \sin^2(k\Delta x) \\
&= 1 - \sin^2(k\Delta x) + \left(\frac{c\Delta t}{\Delta x}\right)^2 \sin^2(k\Delta x) \\
&= 1 + \sin^2(k\Delta x)\left(\left(\frac{c\Delta t}{\Delta x}\right)^2 - 1\right)
\end{aligned}$$

Maintenant, il est possible de déduire une condition de stabilité sur le pas de temps...

$$\begin{aligned}
|G_{Lax}| &\leq 1 \\
\sin^2(k\Delta x)\left(\left(\frac{c\Delta t}{\Delta x}\right)^2 - 1\right) &\leq 0 \\
\left(\frac{c\Delta t}{\Delta x}\right)^2 - 1 &\leq 0 \\
c\Delta t &\leq \Delta x
\end{aligned}$$

Le schéma de Lax est conditionnellement stable (Monsieur Lax existe bien...). Ce schéma introduit un terme additionnel qui stabilise le schéma en y incorporant de la dissipation numérique : ce qui a évidemment un impact non négligeable sur sa précision.

La condition de stabilité sur le pas de temps s'écrit donc $\boxed{\Delta t \leq \frac{\Delta x}{c}}$.

89 Janvier 2007 : Dérivation numérique

1. On effectue simplement les calculs...

$$\begin{aligned}
D^{2h} &= \frac{u(-2h) - 2u(0) + u(2h)}{4h^2} &&= \frac{-0.0025}{h^2} \\
D^{h} &= \frac{u(-h) - 2u(0) + u(h)}{h^2} &&= \frac{-0.003}{h^2} \\
D_{Richardson} &= \frac{4D^h - D^{2h}}{3} &&= \frac{-0.0120 + 0.0025}{3h^2} \approx \frac{-0.0032}{h^2}
\end{aligned}$$

La combinaison linéaire doit évidemment être effectuée pour éliminer le premier terme de l'erreur de discrétisation. Il est opportun d'observer que le calcul est très largement simplifié en observant qu'une dérivée seconde est insensible à une translation verticale de la fonction et en remplaçant la seconde ligne des données par [-0.008 -0.003 0 0 -0.002]...

2. On sait que l'erreur d'une différence centrée d'ordre deux satisfait :

$$|\tilde{E}^h| \leq \frac{4\epsilon}{h^2} + \frac{C_4 h^2}{12} + \frac{C_6 h^4}{360},$$

– D'une part, on obtient l'erreur de discrétisation en effectuant la combinaison linéaire correspondante de ces erreurs pour D^{2h} et D^h.

$$\frac{1}{3}\left[4\left(\frac{C_4h^2}{12}+\frac{C_6h^4}{360}\right)-\left(\frac{C_4 4h^2}{12}+\frac{C_6 16h^4}{360}\right)\right] \quad = \quad -\frac{C_6h^4}{90}$$

– D'autre part, l'erreur d'arrondi s'obtient en propageant les erreurs d'arrondis de D^{2h} et D^h lors de la combinaison linéaire de Richardson : ces erreurs doivent s'additionner (et non se soustraire comme la majorité des étudiants le pensent) !

$$\frac{1}{3}\left[4\frac{4\epsilon}{h^2}+\frac{\epsilon}{h^2}\right] \quad = \quad \frac{17\epsilon}{3h^2}$$

On conclut que la borne de l'erreur du calcul de Paul-Emile satisfait : $\boxed{|\tilde{E}_{Richardson}| \quad \leq \quad \frac{17\epsilon}{3h^2}+\frac{C_6h^4}{90}}$

3. Il suffit de minimiser $f(h)=\frac{17\epsilon}{3h^2}+\frac{h^4}{90}$, et d'écrire que

$$\begin{aligned} f'(h) &= \frac{-34\epsilon}{3h^3}+\frac{4h^3}{90} = 0 \\ &\downarrow \\ h^6 &= \frac{45}{2}\frac{34}{3}\,\epsilon = 255\,\epsilon \end{aligned}$$

On conclut que Paul-Emile a sélectionné le pas comme suit : $\boxed{h \quad = \quad \sqrt[6]{255\ \epsilon}}$

90 Janvier 2007 : Méthode de Gear

1. L'erreur locale d'une méthode d'ordre trois s'écrit en $\mathcal{O}(h^4)$. En d'autres mots, $p=4$.
2. Comme $F_i = U_i'$, il suffit d'écrire que :

$$\begin{aligned} 11U_{i+1} &= 2U_{i-2}-9U_{i-1}+18U_i+6hU_{i+1}' \\ &\downarrow \\ &\text{En sachant que } U_{i-2}=U_i-2hU_i'+\tfrac{4h^2}{2}U_i''-\tfrac{8h^3}{6}U_i'''+\mathcal{O}(h^4) \\ &\text{En tenant compte que } U_{i-1}=U_i-hU_i'+\tfrac{h^2}{2}U_i''-\tfrac{h^3}{6}U_i'''+\mathcal{O}(h^4) \\ &\text{En n'oubliant pas que } U_{i+1}'=U_i'+hU_i''+\tfrac{h^2}{2}U_i'''+\mathcal{O}(h^3) \\ &\downarrow \\ 11U_{i+1} &= \underbrace{(2-9+18)}_{11}U_i+h\underbrace{(-4+9+6)}_{11}U_i'+\frac{h^2}{2}\underbrace{(8-9+12)}_{11}U_i''+\frac{h^3}{6}\underbrace{(-16+9+18)}_{11}U_i'''+\mathcal{O}(h^4) \\ U_{i+1} &= U_i+hU_i'+\frac{h^2}{2}U_i''+\frac{h^3}{6}U_i'''+\mathcal{O}(h^4) \end{aligned}$$

La formule de Gear est équivalente à un développement de Taylor d'ordre trois.
L'erreur locale commise est en $\mathcal{O}(h^4)$ $\square$.

91 Janvier 2007 : Problème de Blasius

1. Il suffit de poser que $v = u'$ et $w = u''$ et d'écrire

$$\begin{cases} u'(x) &= v(x) \\ v'(x) &= w(x) \\ w'(x) &= -u(x)w(x) \end{cases}$$

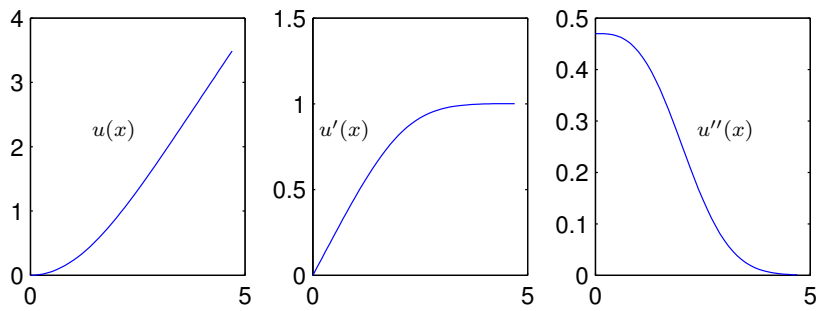


Figure 12.8: Solution du problème de Blasius

2. Il faut combiner le programme de la méthode de Heun et celui de la technique de la bissection. (donnés dans les notes ou les transparents !) La méthode du tir consiste à trouver de manière itérative (avec par exemple, la méthode de la bissection : d'autres choix sont possibles :-), la condition initiale manquante $w(0)$ afin de satisfaire la condition à l'infini. Il n'est évidemment pas possible de vérifier les bornes de départ de l'intervalle de la méthode de bissection sans ordinateur et toute valeur proposée (et plus ou moins sensée) est donc admise par le correcteur. Une implémentation possible est donnée ci-dessous :

```
function u = blasius(h)
n = 1; nmax = 10;
a = 0; fa = heun(a,h);
b = 1; fb = heun(b,h);
delta = (b-a)/2;
if (fa*fb > 0) error(); end;
while (abs(delta) >= 0.01 && n <= nmax)
   x = a + delta; fx = heun(x,h);
   if (fx*fa > 0)   a = x;  fa = fx;
   else             b = x;  fb = fx;
   end
```

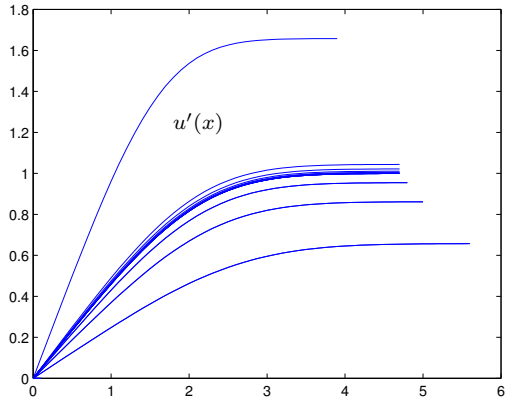


Figure 12.9: Convergence de la méthode du tir appliquée au problème de Blasius $a = 0$ et $b = 1$

```
   delta = (b-a)/2; n = n+1;
end
if (n > nmax)  error(); end
[b u] = heun(x,h);  u = u(:,1);
end

function [b u] = heun(a,h)
i = 1; imax = 100; test = 1;
X = [0:imax]*h; U = [[0 0 a] ; zeros(imax,3)];
while (test > 0.01 && i < imax)
   P        = U(i,:) + h * f(X(i),U(i,:));
   U(i+1,:) = U(i,:) + h * ( f(X(i),U(i,:)) + f(X(i+1),P) ) /2;
   test = (U(i+1,2) - U(i,2)); i = i+1;
end
if (i == imax)  error(); end
b = 1 - U(i,2); u = U(1:i,:);
end


function dudx = f(x,u)
dudx =[u(2) u(3) -u(1)*u(3)];
end
```

92 Janvier 2008 : Différences finies

1. On écrit les développements en série de Taylor comme suit :

$$u(\pm 2h) = U_0 \pm 2hU_0' + \frac{4h^2}{2}U"_0 \pm \frac{8h^3}{6}U_0^{(3)} + \frac{16h^4}{24}U_0^{(4)} \pm \frac{32h^5}{120}U_0^{(5)} + \dots$$

$$u(\pm h) = U_0 \pm hU_0' + \frac{h^2}{2}U"_0 \pm \frac{h^3}{6}U_0^{(3)} + \frac{h^4}{24}U_0^{(4)} \pm \frac{h^5}{120}U_0^{(5)} + \dots$$

En substituant ces développements dans la formule de Jonathan, on obtient alors :

$$\frac{2h(-a-2)}{bh}U_0' + \frac{2h^3(-a-8)}{6bh}U_0^{(3)} + \frac{2h^5(-a-32)}{120bh}U_0^{(5)} + \dots$$

Pour annuler le terme en $U_0^{(3)}$, il suffit de prendre $a = -8$ (et pas $a = 8$:-)
Pour ensuite obtenir un coefficient unitaire devant U_0', il convient de sélectionner $b = 12$.

$$\boxed{u'(0) \simeq \frac{u(-2h) - 8u(-h) + 8u(h) - u(2h)}{12h} + \frac{h^4}{30}u^{(5)}(\zeta)}$$

L'ordre de cette formule est 4.
On obtient exactement le même résultat en observant que la formule de Jonathan peut être construite comme une extrapolation de Richardson des formules de dérivation centrée d'ordre 2 pour h et $2h$.

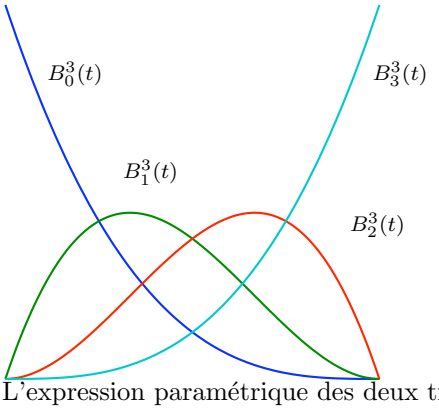
2. Il suffit de minimiser $f(h) = \frac{18\epsilon}{12h} + \frac{h^4}{30}$.

$$\text{On exige donc que } f'(h) = \frac{-3\epsilon}{2h^2} + \frac{4h^3}{30} = 0$$
$$\downarrow$$
$$h^5 = \frac{90}{8}\,\epsilon$$

On conclut que le pas optimal est donné par : $\boxed{h = \sqrt[5]{\frac{45}{4}\,\epsilon}}$

93 Janvier 2008 : Courbes de Bézier

1. Les quatre fonctions sont données par :



$$\boxed{\begin{array}{rcl} B_0^3(t) &=& (1-t)^3 \\ B_1^3(t) &=& 3(1-t)^2 t \\ B_2^3(t) &=& 3(1-t)t^2 \\ B_3^3(t) &=& t^3 \end{array}}$$

2. L'expression paramétrique des deux trajectoires définies par les expressions

$$\underbrace{(x_p(t), y_p(t))}_{\mathbf{p}(t)} = \sum_{i=0}^{3} B_i^3(t)\ \mathbf{P}_i \quad \text{et} \quad \underbrace{(x_q(t), y_q(t))}_{\mathbf{q}(t)} = \sum_{i=0}^{3} B_i^3(t)\ \mathbf{Q}_i \qquad 0 \le t \le 1$$

est donnée par :

$$\boxed{\begin{array}{rcl} x_p(t) = y_p(t) &=& 3(1-t)^3 + 6(1-t)^2 t + 3\alpha(1-t)t^2 \\ x_q(t) &=& -3(1-t)^2 t + t^3 \\ y_q(t) &=& 6(1-t)t^2 \end{array}}$$

3. Les endroits où se trouvent Patrick et Quentin au temps $t = 0.5$ sont :

$$\boxed{\begin{array}{rcl}\Big(x_p(\frac{1}{2}), y_p(\frac{1}{2})\Big) & = & \Big(\frac{9+3\alpha}{8}, \frac{9+3\alpha}{8}\Big) \\ \Big(x_q(\frac{1}{2}), y_q(\frac{1}{2})\Big) & = & \Big(\frac{-1}{4}, \frac{3}{4}\Big)\end{array}}$$

4. Il suffit de rechercher le temps t_* où Quentin croise la droite que parcourt Patrick : $x_q(t_*) = y_q(t_*)$.

$$\begin{array}{rcl} -3(1-t_*)^2 t_* + t_*^3 & = & 6(1-t_*)t_*^2 \\ \downarrow & & \\ -3(1-t_*)^2 + t_*^2 & = & 6(1-t_*)t_* \\ -3 - 3t_*^2 + 6t_* + t_*^2 & = & 6t_* - 6t_*^2 \\ 4t_*^2 & = & 3 \end{array}$$

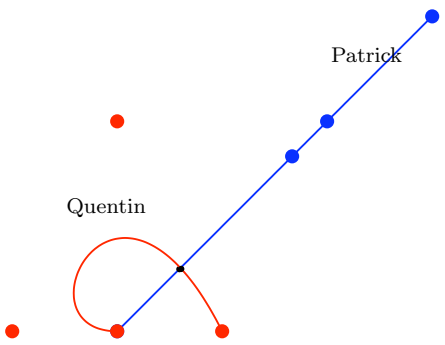
On conclut donc que Quentin croise la trajectoire de Patrick en $\boxed{t_* = \dfrac{\sqrt{3}}{2}}$

au point défini par $\boxed{x_* = y_* = \dfrac{9}{2}\Big(1 - \dfrac{\sqrt{3}}{2}\Big)}$

5. Patrick doit être à ce point de croisement au même instant que Quentin.
Il faut donc exiger que $x_p(t_*) = x_*$.

$$\begin{array}{rcl} 3\Big(1-\frac{\sqrt{3}}{2}\Big)^3 + 6\frac{\sqrt{3}}{2}\Big(1-\frac{\sqrt{3}}{2}\Big)^2 + 3\frac{3}{4}\alpha\Big(1-\frac{\sqrt{3}}{2}\Big) & = & \frac{9}{2}\Big(1-\frac{\sqrt{3}}{2}\Big) \\ \downarrow & & \\ 3\Big(1-\frac{\sqrt{3}}{2}\Big)^2 + 6\Big(1-\frac{\sqrt{3}}{2}\Big)\frac{\sqrt{3}}{2} + \frac{9\alpha}{4} & = & \frac{9}{2} \\ 3 + \frac{9}{4} - 3\sqrt{3} + 3\sqrt{3} - \frac{9}{2} + \frac{9\alpha}{4} & = & \frac{9}{2} \\ 9\alpha & = & 18 + 18 - 9 - 12 \\ 9\alpha & = & 15 \end{array}$$

On conclut donc $\boxed{\alpha = \dfrac{5}{3}}$



94 Janvier 2008 : Equation de la chaleur

1. Les unités de la diffusivité thermique et l'expression β en termes de α, Δx et Δt sont données par :

$\boxed{\alpha = 1\left[\frac{cm^2}{s}\right]}$ et $\boxed{\beta = \frac{\alpha \Delta t}{\Delta x^2}}$

De manière assez surprenante, une majorité des étudiants n'ont pas pu fournir les unités de la diffusivité thermique. Plus surprenant, certains pensent que c dans la relation $\alpha = k/(\rho c)$ est une vitesse... Mais que vous apprennent Grégoire Winckelmans et Jean-François Remacle ?

2. La perturbation évoluera comme suit lors d'un pas de temps :

$$\begin{aligned} U_j^{n+1} &= U_j^n + \beta\Big(U_{j+1}^n + U_{j-1}^n - 2U_j^n\Big) \\ &= U_j^n\Big(1 + \beta\Big(e^{ik\Delta x} + e^{-ik\Delta x} - 2\Big)\Big) \\ &= U_j^n\Big(1 + \beta\Big(2\cos(k\Delta x) - 2\Big)\Big) \end{aligned}$$

Pour que la perturbation ne s'accroisse pas, il faut que : $|1 + \beta\Big(2\cos(k\Delta x) - 2\Big)| \leq 1$

En prenant le cas le plus défavorable $k\Delta x = \pi$, ↓

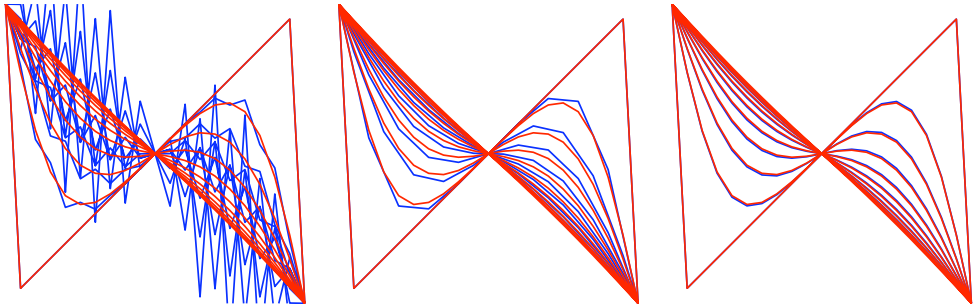
$$\begin{aligned} -1 &\leq 1 - 4\beta \\ \beta &\leq 1/2 \end{aligned}$$

3-4. Une implémentation possible[7] des deux fonctions demandées de manière simultanée est :

```
function edp(m)
  if (nargin == 0) m = 21; end
  Tf = 0.1; Lf = 1; nfig = 10;
  X = linspace(0,Lf,m);  dx = 1/(m-1);   Uo = X;  Uo(1) = 1;  Uo(m) = 0;

  dt = dx * dx / 2.0;   beta = dt / (dx * dx);   U = Uo; Unew = Uo;
  for i = 1:nfig
      plot(X,U,'-b'); hold on;
      for isub = 1:Tf / (dt * nfig)
         for j = 2:m-1
             Unew(j) = U(j) + beta*(U(j-1) + U(j+1) - 2 * U(j));
         end
         U(2:m-1) = Unew(2:m-1);
      end
  end

  [T,U]=ode45(@dfdt,[0 Tf],Uo);
  Uplot = interp1(T,U,linspace(0,Tf,nfig+1));
  plot(X,Uplot,'r-');
end

function dUdt=dfdt(t,U);
  m = length(U); dUdt = zeros(m,1); dx = 1/ (m-1);
  for j = 2:m-1
      dUdt(j) = (U(j-1) + U(j+1) - 2 * U(j)) / (dx*dx);
  end
end
```

[7] Il n'était ni demandé, ni indispensable, ni utile, ni astucieux de construire une matrice tridiagonale creuse pour les deux sous-questions... C'est inutilement compliqué, ici ! On ne vous en a jamais tenu rigueur toutefois.

Choisir $\beta = 0.5$ permet d'éviter l'apparition d'instabilités... Toutefois obtenir une intégration suffisamment précise requiert parfois de choisir une valeur plus petite. Pour une discrétisation spatiale relativement grossière ($m = 21$), la superposition des solutions obtenues par `ode45` (courbes rouges) et par Euler explicite pour des valeurs de $\beta = 0.527,\ 0.500$ et 0.125 (courbes bleues) illustre la nécessité de parfois réduire davantage le pas de temps pour une question de précision. Pour un maillage fin, le critère de stabilité est toutefois toujours dominant !



Appliquer un critère de stabilité n'est donc pas toujours suffisant. Il faut aussi se soucier de la précision ! Bravo aux quelques (rares :-) étudiants qui l'ont remarqué.
Honte à tous ceux (très largement majoritaires :-) qui ont choisi d'utiliser froidement dix pas de temps.

95 Janvier 2009 : Herman ou l'intégration numérique

1. Tout d'abord, il est utile d'observer que seules les puissances paires ont une contribution non nulle sur un intervalle symétrique :-) En intégrant l'erreur de discrétisation du polynôme passant par $-h/3$ et h, on obtient :

$$\int_{-h}^{h} (x + \frac{h}{3})(x - h)\ dx \ = \int_{-h}^{h} \Big(x^2 + \cdots - \frac{h^2}{3}\Big) dx = \Big[\ \frac{x^3}{3} - \frac{xh^2}{3}\Big]_{-h}^{h} = \frac{2h^3}{3} - \frac{2h^3}{3} = 0.$$

Cela signifie donc que la méthode est d'un ordre trois. Il faut donc intégrer un polynôme passant par trois points. On ajoute donc une abscisse quelconque ($x = 0$:-) et on obtient :

$$\int_{-h}^{h} (x + \frac{h}{3})(x - h)x\ dx = \int_{-h}^{h} \Big(\cdots + (\frac{h}{3} - h)x^2 + \ldots\Big) dx = \Big[\ \frac{-2h}{3}\frac{x^3}{3}\Big]_{-h}^{h} = -\frac{4h^4}{9}$$

On obtient donc une méthode composite d'ordre trois :

$$\begin{array}{rcl} E^h & = & n\dfrac{C_3}{6}\dfrac{4h^4}{9} \\ & \downarrow & \text{En sachant que } 2nh = (b-a) \\ & = & \dfrac{C_3\ (b-a)\ h^3}{27} \end{array}$$

2. Le degré de précision d'une méthode numérique est le nombre entier positif d tel que l'erreur d'intégration est nulle pour tous les polynômes de degré inférieur ou égal à d mais est non nulle pour au moins un polynôme de degré $d + 1$.

$$\int_{-h}^{h} (dx^3 + cx^2 + bx + a)dx \quad \neq \quad \frac{h}{2}\left(\left(-3d\frac{h^3}{27} + 3c\frac{h^2}{9} - 3b\frac{h}{3} + 3a\right) + \left(dh^3 + ch^2 + bh + a\right)\right)$$

$$\frac{2ch^3}{3} + 2ah \quad \neq \quad \frac{12dh^4}{27} + \frac{2ch^3}{3} + 2ah$$

Le degré de précision de la méthode d'Herman est 2 (et non 3 !) car on intègre parfaitement un polynôme de degré deux, mais pas un polynôme de degré trois (l'intégrale exacte du terme de degré trois est nulle, mais pas la contribution de la quadrature).

De manière très surprenante (eh oui !), l'immense majorité des étudiants a expliqué de manière quasi-rituelle qu'on gagne un ordre de précision car l'intégrale d'un terme de degré impair était toujours nulle sur un intervalle symétrique.

3. Comme il s'agit d'éliminer un terme d'erreur en h^3, il faut écrire la combinaison suivante :

$$\boxed{H_{extr} = \frac{8H_h - H_{2h}}{7}}$$

96 Janvier 2009 : Equation aux dérivées partielles

1. Comme c'était la même question qu'en 2008, les étudiants ont évidemment trouvé cela facile !
2. Ici, il y avait une toute petite astuce :-)

$$\boxed{\gamma = 1\left[\frac{m^2}{s}\right]} \text{ et } \boxed{\beta = \frac{\gamma \Delta t}{4\Delta x^2}}$$

De manière assez surprenante, une majorité des étudiants redonnent exactement la réponse à la question posée en janvier 2008 : mais, nous sommes en janvier 2009.
C'était facile, mais pas totalement idiot quand-même.
L'oubli du facteur "4" est considéré comme une erreur impardonnable.

3. Une implémentation possible alliant simplicité et efficacité consiste à écrire.

```
function U = edpEuler(m,n,beta)

alpha = linspace(0,1,m);
[X,Y] = meshgrid(alpha,alpha);
U = X.^2 + Y.^2;
for t=1:n
     U(2:m-1,2:m-1) = U(2:m-1,2:m-1) + beta*(U(3:m,3:m) - U(1:m-2,3:m) - ...
                                             U(3:m,1:m-2) + U(1:m-2,1:m-2));
end
```

On peut aussi écrire le programme avec trois boucles imbriquées, mais c'est nettement moins efficace. Dans ce cas, il faut impérativement deux variables distinctes pour effectuer l'itération temporelle : c'était l'unique difficulté à voir !

```
function U = edpEuler(m,n,beta)

alpha = linspace(0,1,m);
[X,Y] = meshgrid(alpha,alpha);
U = X.^2 + Y.^2;
Unew = U;
for t=1:n
     for i=2:m-1
          for j = 2:m-1
                 Unew(i,j) = U(i,j) + beta*(U(i+1,j+1) - U(i-1,j+1) - ...
                                            U(i+1,j-1) + U(i-1,j-1));
          end
     end
     U = Unew;
end
```

4. La perturbation évoluera comme suit lors d'un pas de temps :

$$\begin{aligned}
U_{j,k}^{n+1} &= U_{j,k}^n + \beta\Big(U_{j+1,k+1}^n + U_{j-1,k-1}^n - U_{j+1,k-1}^n - U_{j-1,k+1}^n\Big) \\
&= U_{j,k}^n\Big(1 + \beta\Big(e^{ik_x\Delta x}e^{ik_y\Delta x} + e^{-ik_x\Delta x}e^{-ik_y\Delta x} - e^{ik_x\Delta x}e^{-ik_y\Delta x} - e^{-ik_x\Delta x}e^{ik_y\Delta x}\Big)\Big) \\
&= U_{j,k}^n\Big(1 + \beta\Big(\Big(e^{ik_x\Delta x} - e^{-ik_x\Delta x}\Big)\Big(e^{ik_y\Delta x} - e^{-ik_y\Delta x}\Big)\Big)\Big) \\
&= U_{j,k}^n\Big(1 - 4\beta\Big(\sin(k_x\Delta x)\sin(k_y\Delta x)\Big)\Big)
\end{aligned}$$

Pour que la perturbation ne s'accroisse pas, il faut toujours que :

$$|1 - 4\beta\sin(k_x\Delta x)\sin(k_y\Delta x)| \leq 1$$

Mais en prenant, par exemple, $k_x\Delta x = \pi/2$ et $k_y\Delta x = -\pi/2$, ↓

$$|1 + 4\beta| \leq 1$$

Cette dernière inégalité ne sera jamais satisfaite puisque β est toujours positif... La méthode est donc inconditionnellement instable : il est impossible de trouver un quelconque β qui permet d'être certain d'avoir un comportement stable. Essayez d'exécuter le programme ci-dessus et vous obtiendrez des résultats totalement incohérents !
De manière vraiment très surprenante, quasiment tous les étudiants ne peuvent même pas imaginer que l'enseignant puisse imaginer une méthode aussi délirante et se trompent quasiment de manière systématique sur les trois dernières lignes même après avoir obtenu le bon facteur d'amplification : à méditer !

97 Janvier 2010 : Approximation de Fourier

1. La fonction demandée est : $J(a_1, a_2, a_3) = \sum_{i=0}^{7}\left(U_i - \sum_{j=1}^{3}\cos(jX_i)\ a_j\right)^2$
   Cette question est vraiment très simple. Il faut les deux sommes écrites de manière adéquate. Ne considérer que les 3 premiers points en effectuant donc une simple interpolation est impardonnable (et n'a donc pas été admis !)
2. Il suffit d'annuler les dérivées partielles de J par rapport à a_1, a_2 et a_3 afin d'obtenir :

$$\begin{bmatrix}
\sum_{i=0}^{7}\cos(X_i)\cos(X_i) & \sum_{i=0}^{7}\cos(X_i)\cos(2X_i) & \sum_{i=0}^{7}\cos(X_i)\cos(3X_i) \\
\sum_{i=0}^{7}\cos(2X_i)\cos(X_i) & \sum_{i=0}^{7}\cos(2X_i)\cos(2X_i) & \sum_{i=0}^{7}\cos(2X_i)\cos(3X_i) \\
\sum_{i=0}^{7}\cos(3X_i)\cos(X_i) & \sum_{i=0}^{7}\cos(3X_i)\cos(2X_i) & \sum_{i=0}^{7}\cos(3X_i)\cos(3X_i)
\end{bmatrix}
\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} =
\begin{bmatrix}
\sum_{i=0}^{7}\cos(X_i)U_i \\
\sum_{i=0}^{7}\cos(2X_i)U_i \\
\sum_{i=0}^{7}\cos(3X_i)U_i
\end{bmatrix}$$

3. Le système linéaire devient simplement

$$\begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix}
\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} =
\begin{bmatrix} 1-\sqrt{2} \\ 3 \\ 1+\sqrt{2} \end{bmatrix}$$

On obtient immédiatement $a_1 = (1-\sqrt{2})/4$, $a_2 = 3/4$ et $a_3 = (1+\sqrt{2})/4$

4. Voici un exemple de programme utilisant les données. (Certains petits taquins codent directement les valeurs analytiques : c'est gentiment se foutre de la gueule du correcteur qui n'en a pas tenu rigueur, toutefois)

```
X = [0 1 2 3 4 5 6 7] * pi / 4;
U = [2 1 0 2 1 1 0 0];
x = linspace(0,2*pi,100);
a = [cos(X) * U'  cos(2*X) * U'  cos(3*X) * U'] /4;
uh = a * cos([1 2 3]' * x);
plot(x,uh);
```

98 Janvier 2010 : Méthode de Steffensen

1. Si les itérations convergent et s'il existe deux constantes positives $C < 1$ et $r \geq 1$ telles que

$$\lim_{i\to\infty} \frac{|e_i|}{|e_{i-1}|^r} = C$$

on dit que la séquence converge avec un *taux de convergence* r. Il est aussi judicieux que la constante r soit supérieure à l'unité.

2. Si la méthode converge, la fonction f tend progressivement vers zéro, on peut donc écrire :

$$\begin{aligned} x_{i+1} &= x_i - \frac{f(x_i)f(x_i)}{f(x_i + f(x_i)) - f(x_i)} \\ &\downarrow \quad \text{En effectuant un développement en série de Taylor :} \\ &\quad f(x+f(x)) = f(x) + f'(x)f(x) + f''(x)\frac{f^2(x)}{2} + \dots \\ x_{i+1} &= x_i - \frac{f(x_i)f(x_i)}{f'(x_i)f(x_i) + f''(x_i)\frac{f^2(x_i)}{2} + \dots} \\ &\downarrow \quad \text{En observant que } f \gg f^2 \text{ lorsqu'on arrive à la convergence !} \\ x_{i+1} &= x_i - \frac{f(x_i)f(x_i)}{f'(x_i)f(x_i)} = x_i - \frac{f(x_i)}{f'(x_i)} \end{aligned}$$

Lorsqu'on converge, cette méthode tend vers celle de Newton-Raphson et on a donc $r = 2$. □

Très très peu d'étudiants avaient réussi à obtenir ce résultat.
Steffensen utilise simplement la définition usuelle de la dérivée avec comme incrément f.

3. Ici, il faut observer qu'une itération de Newton-Raphson nécessite le calcul de $f(x_i)$ et de $f'(x_i)$, tandis qu'une itération de la sécante ne nécessite que le calcul de $f(x_i)$ (puisque $f(x_{i-1})$ a été calculé à l'itération précédente). En d'autres mots, il faut comparer le gain d'une itération de Newton-Raphson à celui des deux itérations de la sécante.

   Le coût d'une itération de Steffensen est identique à celui de Newton-Raphson, mais ne nécessite pas l'expression analytique de la dérivée. C'est donc une méthode plus intéressante que celle de Newton-Raphson dans le cas scalaire !

	Nombre d'évaluations	Taux de convergence
Newton-Raphson	2 : $f(x_i)$ et $f'(x_i)$	2
sécante	1 : $f(x_i)$ (car $f(x_{i-1})$ a déjà été évalué !)	$(1.618)^2 = 2.62$
Steffensen	2 : $f(x_i)$ et $f(x_i + f)$	2

Honte aux multiples étudiants qui m'expliquent que les méthodes de la sécante et de Steffensen requièrent respectivement 3 et 5 évaluations de la fonction...

4. Voici un exemple de programme. C'est vraiment élémentaire et pourtant...

```
function x= steffensen(x,tol,nmax,f);
n = 0; delta = tol + 1;
while abs(delta) >= tol && n < nmax
   n = n + 1;
   fx = f(x);
   dfdx = (f(x+fx) - fx) / fx;
   delta = -fx/dfdx;
   x = x + delta;
end
if (n >= nmax)  error();
end
```

Ecrire un programme qui effectue 5 évaluations de f en recopiant bêtement la formule est considéré comme une erreur peu pardonnable et fait perdre la majorité des points. A mon grand dam (et à celui d'un nombre important d'étudiants), c'est malheureusement l'option de la plupart des étudiants. Et parfois, ces étudiants viennent, de manière parfaitement correcte, d'estimer le nombre requis d'évaluations de f pour la méthode de Steffensen dans la question qui précédait (si, si !).

Il n'est pas interdit d'utiliser le même symbole pour la valeur d'entrée et la solution finale, malgré ce que pensait l'un ou l'autre d'entre vous.

99 Janvier 2010 : Méthodes BDF

1. Le polynôme d'interpolation s'écrit simplement :

$$\begin{aligned} u_h(x) &= U_{i+1}\,\frac{(x-X_i)(x-X_{i-1})(x-X_{i-2})}{6h^3} + U_i\,\frac{(x-X_{i+1})(x-X_{i-1})(x-X_{i-2})}{-2h^3} \\ &\quad + U_{i-1}\,\frac{(x-X_{i+1})(x-X_i)(x-X_{i-2})}{2h^3} + U_{i-2}\,\frac{(x-X_{i+1})(x-X_i)(x-X_{i-1})}{-6h^3} \end{aligned}$$

2. En évitant de développer bêtement et mécaniquement, la dérivée s'écrit simplement :

$$\begin{aligned} u_h'(x) &= U_{i+1}\,\frac{(x-X_{i-1})(x-X_{i-2})+(x-X_i)(x-X_{i-2})+(x-X_i)(x-X_{i-1})}{6h^3} \\ &\quad + U_i\,\frac{(x-X_{i-1})(x-X_{i-2})+\dots}{-2h^3} \\ &\quad + U_{i-1}\,\frac{(x-X_i)(x-X_{i-2})+\dots}{2h^3} \\ &\quad + U_{i-2}\,\frac{(x-X_i)(x-X_{i-1})+\dots}{-6h^3} \end{aligned}$$

$\downarrow$ En évaluant en $x = X_{i+1}$

$$u_h'(X_{i+1}) = U_{i+1}\,\frac{6h^2+3h^2+2h^2}{6h^3} + U_i\,\frac{6h^2+0+0}{-2h^3} + U_{i-1}\,\frac{3h^2+0+0}{2h^3} + U_{i-2}\,\frac{2h^2+0+0}{-6h^3}$$

$\downarrow$

$$u_h'(X_{i+1}) = \frac{11U_{i+1}-18U_i+9U_{i-1}-2U_{i-2}}{6h}$$

On obtient donc :

$$\boxed{U_{i+1} = \frac{18}{11}U_i - \frac{9}{11}U_{i-1} + \frac{2}{11}U_{i-2} + \frac{6h}{11}F_{i+1}}$$

3. En observant que la méthode BDF0 est la méthode d'Euler implicite d'ordre un, on pouvait immédiatement déduire que l'ordre de cette formule est 3. Si nécessaire un petit développement de Taylor (non demandé !) pouvait vous rassurer complètement !

4. Le programme s'écrit simplement en recherchant les valeurs possibles du facteur d'amplification α lorsqu'on pose $U_i = \alpha^i U_0$ et qu'on développe :

$$\begin{aligned} U_{i+1} &= \frac{18}{11}U_i - \frac{9}{11}U_{i-1} + \frac{2}{11}U_{i-2} + \frac{6h}{11}F_{i+1} \\ &\downarrow \\ \alpha^3 U_0 &= \frac{18}{11}\alpha^2 U_0 - \frac{9}{11}\alpha U_0 + \frac{2}{11}U_0 + \frac{6h}{11}\lambda\alpha^3 U_0 \\ 0 &= (6h\lambda - 11)\alpha^3 + 18\alpha^2 - 9\alpha + 2 \end{aligned}$$

La méthode sera stable si le module de la plus grande racine est inférieur à un. La méthode BDF2 (ou méthode de Gear d'ordre 3) est stable pour l'entièreté du domaine réel.

```
[x,y]=meshgrid([-8:0.5:8],[-8:0.5:8]);
z = x+i*y;
alpha = z;   % pre-allocation de alpha :-)
for k=1:size(z,1)
    for l =1:size(z,2)
       c = [(z(k,l)*6 -11)  18 -9  2];
       r = roots(c);
       alpha(k,l) = max(abs(r));
    end
end
figure; contourf(x,y,-alpha,[-1:0.1:0]); grid;
```

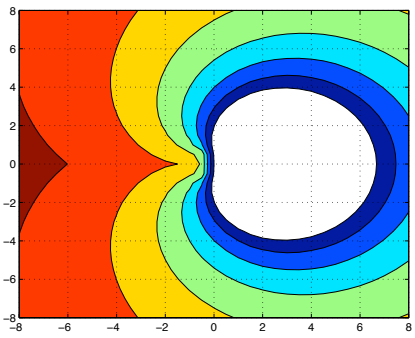
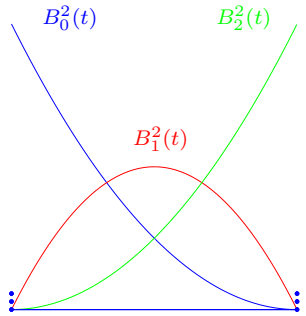


Figure 12.10: Zone de stabilité de la méthode de Gear d'ordre 3

100 Janvier 2011 : Approximation de Bézier

1. Fonctions de Bézier $B_i^2(t)$ définies pour les six noeuds $[T_0, T_1, T_2, T_3, T_4, T_5] = [0, 0, 0, 1, 1, 1]$.



Les trois fonctions sont :

$$\begin{array}{rcl} B_0^2(t) & = & (1-t)^2 \\ B_1^2(t) & = & 2(1-t)t \\ B_2^2(t) & = & t^2 \end{array}$$

Connaissant l'allure des deux fonctions extrêmes et le fait que la somme des trois fonctions doive toujours valoir l'unité, ce résultat peut être obtenu immédiatement sans faire aucun calcul.
Le maximum de la fonction centrale est 0.5 et celui des deux autres fonctions est 1.
Beaucoup d'esquisses étaient en contradiction flagrante avec cette observation !

2. Fonction $J(a_0, a_1, a_2)$ qu'il faut minimiser pour résoudre un tel problème.

$$J(a_0, a_1, a_2) = \sum_{i=0}^{3} \left(U_i - \sum_{j=0}^{2} a_j \; B_j^2(T_i) \right)^2$$

Cette question est vraiment très simple.
En plus c'est exactement la même que celle posée en janvier 2010...
Il faut l'exposant deux et les deux sommes écrites de manière adéquate. Ne considérer que les trois premiers points en effectuant donc une simple interpolation était impardonnable.

3. Comme $\{B_0^2(t), B_1^2(t), B_2^2(t)\}$ et $\{t^2, t, 1\}$ sont deux bases du même sous-espace vectoriel (l'ensemble des polynômes de degré inférieur ou égal à deux), les deux résultats seront identiques.

   En termes de coût calcul ou d'erreurs d'arrondi, il n'y a strictement aucune différence pertinente ! Certains étudiants ont été particulièrement imaginatifs et créatifs à ce sujet :-)
   Au passage, je signale aussi que les fonctions de Bézier ne sont pas orthogonales ! Eh oui, la question ressemblait à celle de janvier 2010, mais c'était quand même un brin différent.

4. Sur base de l'observation précédente, la courbe peut être obtenue avec une unique ligne de code

```
T = [0 1 2 3] /3;
U = [0 1 0 2];
t = linspace(0,1,100)
uh = polyval(polyfit(T,U,2),t);
plot(t,uh);
```

   Ecrire un programme n'utilisant pas `polyval` mais calculant les équations normales était aussi parfaitement admissible. Voilà une autre implémentation qui fournit exactement la même courbe que le précédent :

```
T = [0 1 2 3] /3;
U = [0 1 0 2];
t = linspace(0,1,100);
Phi = [(1-T).^2 ; 2*T.*(1-T); T.^2];
A = Phi * Phi';
```

```
b = Phi * U';
a = A \ b;
phi = [(1-t).^2 ; 2*t.*(1-t); t.^2];
uh = a' * phi;
plot(t,uh);
```

Par contre, s'inspirer directement du programme fourni dans la solution de la première question de janvier 2010 était vraiment une très mauvaise idée, car les fonctions de Bézier ne sont pas orthogonales (contrairement aux cosinus !). Pas mal d'étudiants ont pourtant choisi cette option. Cela a bêtement irrité le correcteur.

101 Janvier 2011 : Méthode d'ordre quatre de Johan

1. Il suffit d'écrire les développements de Taylor (où il est possible de directement omettre tous les termes impairs en raison du caractère symétrique des équations de Johan !)

$$\begin{aligned} U_{i\pm1} &= U_i + \cdots + \frac{h^2}{2}U_i'' + \cdots + \frac{h^4}{24}U_i'''' + \cdots + \mathcal{O}(h^6) \\ U_{i\pm1}'' &= U_i'' + \cdots + \frac{h^2}{2}U_i'''' + \cdots + \mathcal{O}(h^4) \end{aligned}$$

Ensuite, on écrit les équations de Johan :

$$\frac{U_{i-1} - 2U_i + U_{i+1}}{h^2} = \frac{F_{i-1} + \alpha F_i + F_{i+1}}{\gamma}$$

$\downarrow$ En remplaçant F par U''

$$\frac{U_{i-1} - 2U_i + U_{i+1}}{h^2} = \frac{U_{i-1}'' + \alpha U_i'' + U_{i+1}''}{\gamma}$$

$\downarrow$ En y incluant les développements de Taylor de $U_{i\pm1}$ et $U_{i\pm1}''$

$$U_i'' + \frac{h^2}{12}U_i'''' + \mathcal{O}(h^4) = \frac{2+\alpha}{\gamma}U_i'' + \frac{h^2}{\gamma}U_i'''' + \mathcal{O}(h^4)$$

En identifiant les termes, on conclut donc que : $\boxed{\begin{array}{rcl} \alpha &=& 10 \\ \gamma &=& 12 \end{array}}$

2. Une implémentation possible est :

```
function U = johan(n,f)
  h = 1/n; A = speye(n+1,n+1);
  for i=2:n
     A(i,i)   = -2;
     A(i,i+1) =  1;
     A(i,i-1) =  1;
  end

  F = f(0:h:1);
  b = ones(n+1,1);
  b(2:n) = (h*h)*(F(1:n-1) + 10*F(2:n) + F(3:n+1))/12;
```

```
  U =A \b;
end
```

Quelques étudiants proposent également une implémentation de la méthode du tir en considérant que la méthode de Johan est une méthode d'intégration à pas multiples. Il faut alors effectuer, par exemple, une méthode de bissection pour trouver la valeur de U_2 requise pour lancer le calcul. Une implémentation correcte d'une telle approche a aussi été admise. Par contre, proposer un programme qui fournit une solution qui ne satisferait pas une des deux conditions aux limites est considéré comme une erreur impardonnable.

102 Janvier 2011 : Schéma symplectique de Bart

1. Il suffit juste de résoudre :

$$\begin{aligned} det\begin{bmatrix} -\lambda & 1 \\ -1 & -\lambda \end{bmatrix} &= 0 \\ \downarrow & \\ \lambda^2 + 1 &= 0 \end{aligned}$$

On conclut finalement que : $\boxed{\lambda = \pm i}$

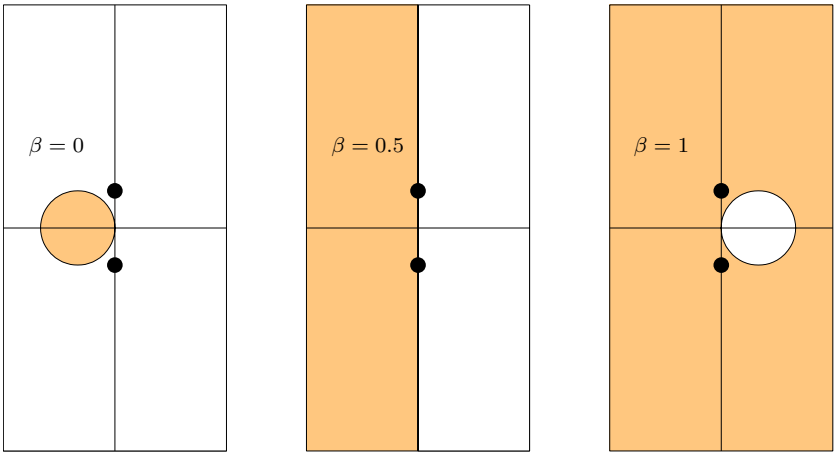
De manière un peu prévisible, certains étudiants estiment impossible d'avoir des valeurs complexes et les corrigent d'autorité par deux valeurs réelles unitaires : c'est pas une bonne idée!

2. La représentation paramétrique adéquate du cercle est :

$$\boxed{\begin{aligned} x(t) &= cos(-t) \\ y(t) &= sin(-t) \end{aligned}}$$

Les conditions initiales et les deux équations différentielles ordinaires sont bien satisfaites.
L'écriture en termes d'exponentielles complexes est aussi admise.
L'oubli du signe négatif n'a été que très faiblement pénalisé :-)
Par contre, l'ajout de termes constants a fait bondir le correcteur !

3. La méthode est explicite si $\beta = 0$ et implicite dans tous les autres cas.
4. Zones de stabilité numérique pour $\beta = 0$, $\beta = 1$ et $\beta = \frac{1}{2}$ dans le plan complexe $h\lambda$.



5. Comme la solution s'écrit comme une combinaison linéaire des deux fonctions propres et que le facteur d'amplification est identique pour les deux valeurs propres complexes conjuguées, l'évolution du rayon est directement donnée par ce facteur d'amplification.
   On conclut immédiatement que :
   - avec $\beta = 0$ (Euler explicite), on a une spirale qui tend vers l'infini,
   - avec $\beta = 0.5$ (Crank-Nicolson), la solution reste toujours sur le cercle de rayon un,
   - avec $\beta = 1$ (Euler implicite), on a une spirale qui tend vers zéro.

   Il est important d'observer que la méthode d'Euler implicite est aussi mauvaise que la méthode d'Euler explicite ! Honte donc à l'immense majorité des étudiants qui ont condamné, sans appel, le pauvre Euler explicite et qui ont acclamé un Euler implicite tout aussi pervers...
   En effet, si on effectue une itération avec chaque méthode, on observe que :

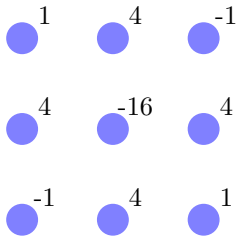
Euler explicite : $\mathcal{O}(h)$ $R_2 = \sqrt{1+h^2} > 1$	$\begin{bmatrix} X_2 \\ Y_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -h \end{bmatrix}$
Euler implicite : $\mathcal{O}(h)$ $R_2 = \dfrac{1}{\sqrt{1+h^2}} < 1$	$\begin{bmatrix} 1 & -h \\ h & 1 \end{bmatrix} \begin{bmatrix} X_2 \\ Y_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ $\downarrow$ $\begin{bmatrix} X_2 \\ Y_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{1+h^2} \\ \frac{-h}{1+h^2} \end{bmatrix}$
Crank-Nicolson : $\mathcal{O}(h^2)$ $R_2 = \sqrt{\dfrac{16+h^4-8h^2+16h^2}{(4+h^2)^2}} = 1$	$\begin{bmatrix} 1 & -h/2 \\ h/2 & 1 \end{bmatrix} \begin{bmatrix} X_2 \\ Y_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -h/2 \end{bmatrix}$ $\downarrow$ $\begin{bmatrix} X_2 \\ Y_2 \end{bmatrix} = \begin{bmatrix} \frac{4-h^2}{4+h^2} \\ \frac{-4h}{4+h^2} \end{bmatrix}$

   Strictement personne n'a effectué le calcul algébrique pour déterminer β à partir des équations et je reconnais bien volontiers que c'est monstrueusement calculatoire...
   Toutefois, poser correctement le problème était apprécié par le correcteur !
   Dernière toute petite remarque : le cercle du diagramme de phase et les deux cercles des diagrammes de stabilité n'ont strictement aucun lien entre eux :-)
6. La méthode de Bart est tout simplement celle de Crank-Nicolson $\beta = 0.5$.
7. L'ordre de précision de Crank-Nicolson est deux.
   Si, si ! Bien combiner deux méthodes d'ordre un permet d'obtenir une méthode d'ordre deux...

   Il est évidemment totalement impossible de répondre à cette question, si vous n'avez aucune idée de la réponse à la question précédente. Beh oui et quoi encore, vous rêvez d'un gouvernement pour le retour de vos vacances ? Que nenni, bois de l'eau claire et révise ta chimie.
   Choisir -au hasard- un ordre de précision est se foutre de la gueule du correcteur !

103 Janvier 2012 : Une bête équation aux dérivées partielles

1. Les différences finies centrées d'ordre deux sont :



$$\frac{\partial^2 U_{i,j}}{\partial x^2} \approx \frac{U_{i+1,j} + U_{i-1,j} - 2U_{i,j}}{h^2}$$

$$\frac{\partial^2 U_{i,j}}{\partial y^2} \approx \frac{U_{i,j+1} + U_{i,j-1} - 2U_{i,j}}{h^2}$$

$$\frac{\partial^2 U_{i,j}}{\partial x \partial y} \approx \frac{1}{2h}\left(\frac{\partial U_{i+1,j}}{\partial y} - \frac{\partial U_{i-1,j}}{\partial y}\right)$$

En remplaçant $\frac{\partial U_{i+1,j}}{\partial y}$ par $\frac{U_{i+1,j+1} - U_{i+1,j-1}}{2h}$

$$\approx \frac{U_{i+1,j+1} - U_{i-1,j+1} - U_{i+1,j-1} + U_{i-1,j-1}}{4h^2}$$

Les $(n-1)^2$ équations pour les noeuds intérieurs sont donc :

$$\frac{U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j}}{h^2} + \frac{U_{i+1,j+1} - U_{i-1,j+1} - U_{i+1,j-1} + U_{i-1,j-1}}{4h^2} = 1$$

$$i = 1, \ldots, n-1$$
$$j = 1, \ldots, n-1$$

Pour les noeuds frontières, il suffit d'imposer $U_{i,j} = 0$, lorsque l'un des indices vaut 0 ou n. Oublier le terme croisé est évidemment impardonnable : ceci n'est pas un laplacien ! Quelques étudiants semblent croire que la dérivée croisée est le produit des deux dérivées premières: ce n'est vraiment pas une bonne idée :-(

2. Une implémentation possible est :

```
function u = edp(n);
h = 1/n;
n = n+1;       % Petite facétie :-)
A = speye(n^2);
b = zeros(n^2,1);
indexpt = [n-1 n n+1 -1   0 1 -n-1 -n -n+1];
pattern = [ -1 4   1  4 -16 4    1  4   -1];
for i = 2:n-1
  for j = 2:n-1
    index = i + (j-1)*n;
    A(index,index+indexpt) = pattern;
    b(index) = 4*h*h;
  end
end
u = reshape(A\b,n,n);
end
```

Quasiment tous les étudiants écrivent les 9 équations comme je l'avais fait au cours : cela est évidemment admis, mais nécessite un petit travail fastidieux de scribe. Les points importants sont l'initialisation d'une matrice creuse et la prise en compte des conditions aux frontières. Remplacer `speye` par `sparse` est une erreur. Ne pas utiliser de matrices creuses pour ce

problème est aussi considéré comme une erreur. Oublier de redimensionner le vecteur final est aussi une erreur.

La taille de la matrice est $(n+1)^2$. Un maillage de $m \times m$ points est caractérisé par un pas $h = 1/(m-1)$. Cette toute petite astuce n'était que très légèrement pénalisée pour ceux qui ne l'avaient pas observée. Attention, remplacer `h=1/n; n=n+1;` par `h=1/(n-1)` est une erreur !

Recopier le programme de Poisson des transparents sans y faire les adaptations utiles pour le problème posé et la définition de `n` ne rapporte quasiment rien. En général, l'humeur du correcteur devient vraiment très négative si l'étudiant est -en outre- incapable de fournir les équations qui sont résolues par ce programme :-(

Quelques étudiants proposent des solutions d'une inutile complexité : c'est souvent l'occasion d'écrire beaucoup de bêtises et donc de perdre des points. Un programme simple et compact est bien plus souvent la clé du succès.

104 Janvier 2012 : Newton, Raphson et Euler tous ensemble !

1. L'itération d'Euler implicite s'écrit : $\boxed{U^{n+1} = U^n + h\Big(\cos((n+1)h) - \big(U^{n+1}\big)^3\Big)}$

2. Le problème non linéaire à résoudre est $\underbrace{U^{n+1} + h\big(U^{n+1}\big)^3 - U^n - h\ \cos((n+1)h)}_{f(U^{n+1}) = f(x)} = 0.$

   Bien noter que l'inconnue est U^{n+1} pour l'itération de Newton $x_{i+1} = x_i - \dfrac{f(x_i)}{f'(x_i)}$

   On conclut donc : $\boxed{U_{i+1}^{n+1} = U_i^{n+1} - \dfrac{U_i^{n+1} + h\big(U_i^{n+1}\big)^3 - U^n - h\cos((n+1)h)}{1 + 3h\big(U_i^{n+1}\big)^2}}$

   Il ne faut évidemment pas dériver le cosinus !
   Inclure un indice i à la valeur précédente U^n est une erreur !
   La plupart des étudiants n'arrivent pas à écrire cette relation :-(
   Comme annoncé, simplement changer les notations vous perturbe totalement comme prévu.

3. L'application mécanique de la formule donne :

$$\boxed{\begin{aligned} U_0^1 &= U^0 = 0 \\ U_1^1 &= h\cos(h) \\ U_2^1 &= \frac{h\cos(h) + 2h^4\cos^3(h)}{1 + 3h^3\cos^2(h)} \end{aligned}}$$

4. Le critère d'erreur doit impliquer un incrément maximal toléré et un nombre maximal d'itérées. On peut choisir deux valeurs arbitraires.

```
while (abs(delta) >= tol && i < imax)
```

Toutefois, cela aurait du sens de lier l'incrément maximal avec l'erreur locale commise à chaque pas de temps par le schéma d'Euler implicite. Cette erreur locale peut être estimée de deux manières. Elle est proportionnelle à h^2. Elle peut aussi être estimée par $\mid U^{n-1} - U^n \mid$ obtenue au pas précédent.

Comme d'habitude, la plupart des étudiants pensent utile de baratiner le correcteur en lui déversant une quantité de banalités, alors qu'une seule phrase suffit :-) C'est inutile et c'est souvent l'occasion d'écrire des âneries qui vous font perdre des points... Ne dites que ce qui vous semble pertinent.

5. Une implémentation possible est :

```
function U = eulerImplicite(h,m)
T = linspace(0,h*m,m+1);   U = zeros(1,m+1);
U(1) = 0; imax =  50; du = 1;
for n=1:m
    tol = 0.01 * max(du,1);
    i = 0; delta = tol + 1; x = U(n);
    while (abs(delta) >= tol && i < imax)
        i = i + 1;
        delta = - (x + h*x^3 - U(n) - h*cos(T(n)))/(1+3*h*x*x);
        x = x + delta;
    end
    U(n+1) = x;
    du = abs(U(n+1)-U(n));
end
end
```

105 **Janvier 2012 : Trajectoire d'un monstre maléfique**

1. Comme $B_0^2(0) = B_3^2(0) = 0$ et $B_1^2(0) = B_2^2(0) = \frac{1}{2}$, on a :

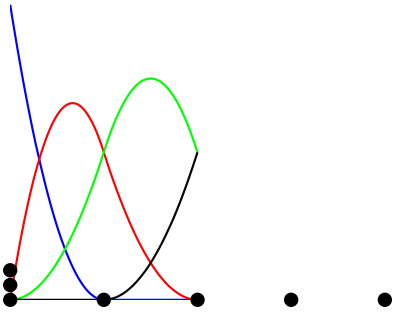
$$\boxed{\mathbf{x}(0) \;=\; \Big(-2, 8, \alpha\Big)}$$

2. En observant que sur l'intervalle $[T_2, T_3]$, seules les fonctions $B_1^1(t) = -t$ et $B_2^1(t) = (t+1)$ sont non nulles, on obtient en appliquant posément la définition :

$$\boxed{\begin{array}{lcccccl} B_0^2(t) & = & 0 & + & \frac{(0-t)}{1} B_1^1(t) & = & t^2 \\ B_1^2(t) & = & \frac{(1+t)}{1} B_1^1(t) & + & \frac{(1-t)}{2} B_2^1(t) & = & (1 - 2t - 3t^2)/2 \\ B_2^2(t) & = & \frac{(1+t)}{2} B_2^1(t) & + & 0 & = & (1 + 2t + t^2)/2 \\ B_3^2(t) & = & 0 & + & 0 & = & 0 \end{array}}$$

La plupart des étudiants arrivent à réaliser ce petit calcul élémentaire :-)

3. L'esquisse des fonctions $B_0^2(t)$, $B_1^2(t)$, $B_2^2(t)$ et $B_3^2(t)$ pour $t = [T_2, T_4] = [-1, 1]$....



Comme prévu, réaliser cette esquisse a été une tâche compliquée ! Pourtant, en sachant que

$$\sum_{i=0}^{3} B_i^2(t) = 1$$

sur les deux intervalles, le graphe pouvait être très aisément construit de manière tout-à-fait intuitive des valeurs connues aux noeuds.

Il faut avoir l'allure générale des 4 courbes et ne pas dessiner de points de rebroussement sur vos courbes qui sont $\mathcal{C}^2$. Seule, la courbe de $B_0^2(t)$ atteint une valeur unitaire en $t = -1$, puisque ce noeud est triple. L'obtention de graphes complètement bizarres ne semble pas trop inquiéter les étudiants. Beaucoup dessinent des courbes symétriques : ce qui est illogique pour les noeuds choisis. Par contre, c'est normal que la figure ne soit pas symétrique : l'inverse serait plus étonnant !

4. Comme toutes les fonctions B-splines entre $[0, 1[$ étaient fournies dans l'énoncé, on écrit :

$$\begin{aligned} 8 &= \alpha\, B_1^2(\tfrac{1}{2}) + \alpha\, B_2^2(\tfrac{1}{2}) \\ 8 &= \alpha\, \frac{1}{8} + \alpha\, \frac{3}{4} \\ &\downarrow \\ 8 &= \frac{7}{8}\, \alpha \end{aligned}$$

On conclut donc que : $\boxed{\alpha = \dfrac{64}{7}}$

Il est donc possible d'obtenir ce résultat en n'ayant pas répondu aux trois sous-questions précédentes. Conclusion : commencer par la fin est souvent une bonne idée avec moi (mais, pas toujours :-)

106 **Janvier 2013 : Hervé s'emberlificote avec Heun**

1. On écrit le système sous forme matricielle : $\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \underbrace{\begin{bmatrix} -10 & -9 \\ -9 & -10 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$

Le problème sera stable si les valeurs propres de la matrice jacobienne $\mathbf{A}$ sont négatives.
Il suffit donc simplement de calculer :

$$\det \begin{bmatrix} -10-\lambda & -9 \\ -9 & -10-\lambda \end{bmatrix} = 0$$

$$\Downarrow$$

$$(10+\lambda)^2 - 81 = 0$$

$$\lambda^2 + 20\lambda + 19 = 0$$

$$\Downarrow \quad \text{En observant que } 324 = 81 \times 4 \text{ :-)}$$

$$\lambda = \frac{-20 \pm \sqrt{400-76}}{2} = -10 \pm \sqrt{81} = -10 \pm 9$$

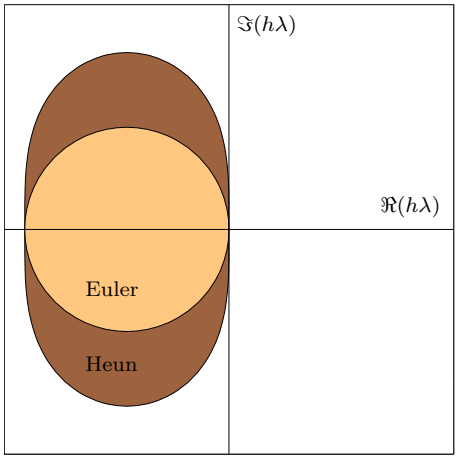
On conclut que le problème est stable, puisque $\lambda' = -1$ et $\lambda'' = -19$ sont négatives.

Un très grand nombre d'étudiants se sont lamentablement plantés dans cette question pourtant élémentaire. Dans le rayon du prix des horreurs, citons juste les étudiants justifiant la stabilité car la matrice est négative...
Ah bon, c'est quoi le signe d'une matrice ?
Ils vous ont appris quoi en algèbre, Kouider, Michel, Roland et tous les autres...

2. On évalue simplement U_{i+1} pour le problème modèle :

$$\begin{aligned} K_1 &= \lambda U_i \\ K_2 &= \lambda\big(U_i + hK_1\big) = \lambda\big(U_i + h\lambda U_i\big) \\ &\Downarrow \\ U_{i+1} &= U_i + \frac{h}{2}\Big(K_1 + K_2\Big) = \Big(1 + h\lambda + \frac{h^2\lambda^2}{2}\Big)\, U_i \end{aligned}$$

La zone de stabilité de la méthode de Heun est donc : $\boxed{\left|1 + h\lambda + \frac{h^2\lambda^2}{2}\right| \leq 1}$



Il est essentiel d'indiquer correctement les axes !

Non, la zone de stabilité de Heun n'a pas de petites oreilles comme la méthode de Runge-Kutta d'ordre 4. Noter qu'une très grande tolérance a été admise pour les dessins d'ellipse un brin approximative. La plupart des horribles patates mal foutues ont été admises :-)

Avoir les dessins des zones de stabilité sur son formulaire était parfaitement licite (et ici bien utile !)

3. Il faut considérer la valeur propre la plus négative et écrire :

$$\begin{aligned} \left|1 - 19h + \frac{361h^2}{2}\right| &\leq 1 \\ -1 \leq 1 - \frac{361}{2}h\left(\frac{38}{361} - h\right) &\leq 1 \\ 2 \geq \frac{361}{2}h\left(\frac{2}{19} - h\right) &\geq 0 \\ \downarrow & \\ h \leq \frac{2}{19} & \end{aligned}$$

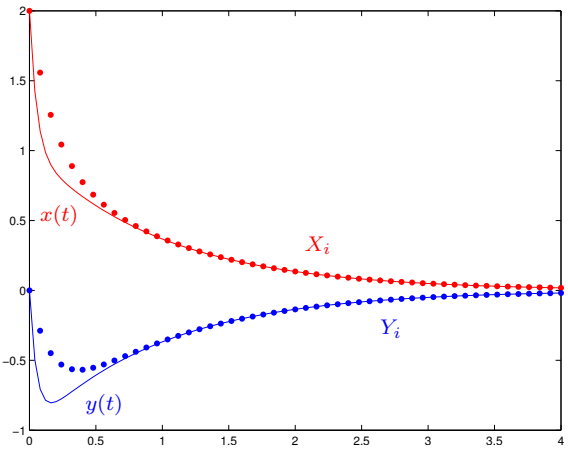
C'est exactement la même condition que celle obtenue avec la méthode d'Euler explicite. Il est donc possible d'obtenir immédiatement le résultat en observant le graphique, en n'effectuant aucun calcul, mais en se rappelant juste que pour une valeur propre réelle, la condition de stabilité pour Euler explicite était $|h\lambda| < 2$. C'est évidemment la valeur propre la plus raide qu'il faut considérer :-)

4. Une implémentation possible est donnée par :

```
function [x y] =  HeunHerve(n,h)
X  = [[2 0] ; zeros(n,2)];
for k=1:n
    K1 = f(X(k,:));
    K2 = f(X(k,:) + h * K1);
    X(k+1,:) = X(k,:) + h * (K1 + K2)/2;
end
x = X(:,1); y = X(:,2);
end

function dxdt = f(x)
dxdt = x;
dxdt(1) = -10*x(1) -9*x(2);
dxdt(2) =  -9*x(1)-10*x(2);
end
```

Comme la méthode de Heun pour un problème linéaire (ce qui est le cas ici !) est strictement équivalente à la méthode de Taylor, on peut aussi se contenter d'implémenter celle-ci (c'est

encore plus simple et c'est évidemment admis :-). Par contre, effectuer d'abord tout le calcul pour `x` et ensuite pour `y` est incorrect : il faut calculer les deux composantes du vecteur `K1` avant d'entamer le calcul de `K2`. Alors que ce programme est quasiment la copie conforme de celui d'un exercice effectué en séance tutorée, un nombre (vraiment !) très élevé d'étudiants n'arrivent pas à l'écrire...



La solution analytique du problème d'Hervé est donnée par

$$\begin{cases} x(t) &= \exp(-19t) + \exp(-t) \\ y(t) &= \exp(-19t) - \exp(-t) \end{cases}$$

Même si cela ne faisait pas partie de l'examen, c'était vraiment facile de l'obtenir.

107 Janvier 2013 : Stable ou pas stable ?

1. Le nombre entier k est le nombre d'onde (quelconque) de la perturbation (quelconque). Alors que l'enseignant pensait (naïvement) que cette question était vraiment facile, la plupart des étudiants n'ont pas été capables d'y répondre. Citons en vrac quelques perles particulièrement significatives : l'amplitude de la perturbation, le vecteur d'onde (c'est un scalaire: hein !), la direction de la propagation de l'onde, le coefficient de dilatation ou la conductivité thermique (sic !).
2. La méthode est explicite
3. En incluant l'expression $U_j^n = U^n \exp(ikX_j)$ dans le schéma numérique, on obtient :

$$\frac{U_j^{n+1} - U_j^n}{\Delta t} = -c\,\frac{U_j^n - U_{j-1}^n}{\Delta x}$$

$$\Big\downarrow \quad \text{En remplaçant } U_j^n \text{ par } U^n \exp(ikX_j)$$

$$U^{n+1}\exp(ikX_j) - U^n\exp(ikX_j) = -\frac{c\Delta t}{\Delta x}\Big(U^n\exp(ikX_j) - U^n\exp(ikX_j)\exp(-ik\Delta x)\Big)$$

$$U^{n+1} = U^n\left(1 - \frac{c\Delta t}{\Delta x}\Big(1 - \exp(-ik\Delta x)\Big)\right)$$

On en déduit donc que : $\boxed{G = \left(1 - \frac{c\Delta t}{\Delta x}\Big(1 - \exp(-ik\Delta x)\Big)\right)}$

4. Pour obtenir la condition de stabilité, il faut effectuer le calcul du carré du module de G et vérifier que ce module soit inférieur ou égal à l'unité.

$$
\begin{aligned}
|G|^2 &= \left(1-\frac{c\Delta t}{\Delta x}\Big(1-\cos(-k\Delta x)\Big)\right)^2+\left(\frac{c\Delta t}{\Delta x}\sin(-k\Delta x)\right)^2 \\
&\Big\downarrow \quad \text{En définissant } \beta=\frac{c\Delta t}{\Delta x} \text{ et en faisant judicieusement appel à quelques formules trigonométriques :-)} \\
&= \left(1-2\beta\sin^2\left(\frac{k\Delta x}{2}\right)\right)^2+\left(2\beta\sin\left(\frac{k\Delta x}{2}\right)\cos\left(\frac{k\Delta x}{2}\right)\right)^2 \\
&= 1-4\beta\sin^2\left(\frac{k\Delta x}{2}\right)+4\beta^2\sin^4\left(\frac{k\Delta x}{2}\right)+4\beta^2\sin^2\left(\frac{k\Delta x}{2}\right)\cos^2\left(\frac{k\Delta x}{2}\right) \\
&= 1-4\beta\sin^2\left(\frac{k\Delta x}{2}\right)+4\beta^2\sin^2\left(\frac{k\Delta x}{2}\right)\underbrace{\left(\sin^2\left(\frac{k\Delta x}{2}\right)+\cos^2\left(\frac{k\Delta x}{2}\right)\right)}_{=1} \\
&= 1+4\beta(\beta-1)\sin^2\left(\frac{k\Delta x}{2}\right)
\end{aligned}
$$

Maintenant, il est possible de déduire une condition de stabilité sur le pas de temps...

$$
\begin{aligned}
|G| &\leq 1 \\
4\beta(\beta-1)\sin^2\left(\frac{k\Delta x}{2}\right) &\leq 0 \\
\beta-1 &\leq 0 \\
\beta &\leq 1
\end{aligned}
$$

Le schéma est conditionnellement stable et la condition de stabilité est : $\boxed{\dfrac{c\Delta t}{\Delta x} \leq 1}$

Certains étudiants trouvent le cas critique en analysant quelques angles particuliers : cette démarche est admise:-) De manière assez surprenante, cette question assez calculatoire (et que l'enseignant estimait difficile) a été relativement bien réussie... alors qu'expliquer ce qu'était le nombre d'onde (ce que l'enseignant estimait élémentaire) a posé problème à beaucoup d'étudiants.

108 Janvier 2013 : Différences finies compactes

1. Comme la formule est parfaitement symétrique, il n'y aura que des termes d'erreurs pairs. En choisissant de manière astucieuse α, β et γ, on éliminera les termes en $\mathcal{O}(h^0)$, $\mathcal{O}(h^2)$ et $\mathcal{O}(h^4)$.

Donc, on peut espérer -sauf miracle numérique- que : $\boxed{p = 6}$

2. En ne gardant que les termes utiles, on écrit les développements en série de Taylor :

$$\begin{aligned}
U_{i\pm1} &= U_i \pm hU_i' + \ldots \pm \frac{h^3}{6}U_i^{(3)} + \ldots \pm \frac{h^5}{120}U_i^{(5)} + \ldots \pm \mathcal{O}(h^7) \\
U_{i\pm2} &= U_i \pm 2hU_i' + \ldots \pm \frac{8h^3}{6}U_i^{(3)} + \ldots \pm \frac{32h^5}{120}U_i^{(5)} + \ldots \pm \mathcal{O}(h^7) \\
U_{i\pm1}' &= U_i' \pm \ldots + \frac{h^2}{2}U_i^{(3)} \pm \ldots + \frac{h^4}{24}U_i^{(5)} \pm \ldots + \mathcal{O}(h^6)
\end{aligned}$$

Ensuite, on substitue ces développements dans la formule de Christian :

$$\begin{aligned}
(1+2\alpha)U_i' + 2\alpha\frac{h^2}{2}U_i^{(3)} + 2\alpha\frac{h^4}{24}U_i^{(5)} &= \frac{2\beta}{2h}\left(hU_i' + \frac{h^3}{6}U_i^{(3)} + \frac{h^5}{120}U_i^{(5)}\right) + \\
&\quad + \frac{2\gamma}{4h}\left(2hU_i' + \frac{8h^3}{6}U_i^{(3)} + \frac{32h^5}{120}U_i^{(5)}\right) + \mathcal{O}(h^6)
\end{aligned}$$

En identifiant les termes en U_i', $U_i^{(3)}$ et $U_i^{(5)}$, on obtient finalement :

$$\begin{aligned}
2\alpha + 1 &= \beta + \gamma \\
6\alpha &= \beta + 4\gamma \\
10\alpha &= \beta + 16\gamma
\end{aligned}$$

On obtient bien la précision pressentie, car le terme d'ordre six ne s'annule pas :-)

3. Il faut juste résoudre le système des 3 équations...

La solution unique est : $\alpha = \frac{1}{3}, \quad \beta = \frac{14}{9}, \quad \gamma = \frac{1}{9}$

Comme γ était fourni dans l'énoncé, on pouvait obtenir α et β à partir de deux équations. Inclure les trois valeurs dans la troisième équation permettait de vérifier si aucune erreur malencontreuse d'algèbre ne s'était pas glissée dans le calcul. La plupart des étudiants obtiennent ce résultat... tout en se trompant souvent sur l'ordre de précision de la première question.

4. Une implémentation possible est :

```
function [dU] = compactDerivative(U,alpha,beta,gamma,h)
n = length(U);
A = sparse(n,n);
for i=2:n-1
        A(i,[i-1 i i+1]) = [alpha 1 alpha];
end
A(1,[n-1 1 2]) = [alpha 1 alpha];
A(n,[n-1 n 2]) = [alpha 1 alpha];
b = beta*(U([2:n 2]) - U([n-1 1:n-1]))/(2*h) ...
      + gamma*(U([3:n 2 3]) - U([n-2 n-1 1:n-2]))/(4*h);
dU = A\b;
end
```

Il faut résoudre un système pour obtenir la solution.
Ne pas l'observer fait perdre la totalité des points pour le programme.
Evidemment, il est essentiel d'utiliser une matrice creuse.
Quelques étudiants construisent deux autres matrices pour calculer le membre de droite : ce n'est pas formellement une erreur, mais c'est inutilement compliqué.

Il faut évidemment inclure les conditions de périodicité pour la matrice et le membre de droite. Notons que la valeur U_0 qui précède U_1 n'est pas U_n, mais U_{n-1}. La plupart des étudiants n'observent pas cette toute petite astuce, mais le correcteur ne leur en a pas tenu rigueur. Les plus astucieux observeront que le vecteur `U` doit avoir une taille minimale et être un vecteur colonne pour que le code fonctionne...

109 Janvier 2014 : It is a piece of integration cake :-)

1. On écrit tout simplement :

$$\boxed{u(x) = U_0 + xU_0' + \frac{x^2}{2}U"_0 + \frac{x^3}{6}U_0^{(3)} + \frac{x^4}{24}U_0^{(4)} + \frac{x^5}{120}U_0^{(5)} + \frac{x^6}{720}U_0^{(6)} + \dots}$$

2. Pour estimer l'erreur locale, il suffit donc d'écrire :

$$\begin{aligned}\int_{-h}^{h} u(x)\,dx &\approx 2hU_0 + \frac{h^3}{3}U"_0 + \frac{h^5}{60}U_0^{(4)} + \dots \\ I^h_{midpoint} &= 2hU_0\end{aligned}$$

Il est impardonnable de ne pas observer que l'intégration des termes impairs est nul ! Ensuite, on obtient l'expression de l'erreur locale en comparant les deux lignes.

$$\boxed{I - I^h_{midpoint} = \frac{h^3}{3}U"_0 + \frac{h^5}{60}U_0^{(4)} + \dots}$$

Comme l'erreur est en $\mathcal{O}(h^3)$, l'ordre de précision de la méthode est deux (et pas trois !).

3. On procède de même en écrivant $U_{\pm h}$ à partir d'un développement en série à l'origine :

$$\begin{aligned}\int_{-h}^{h} u(x)dx &\approx 2hU_0 + \frac{h^3}{3}U"_0 + \frac{h^5}{60}U_0^{(4)} + \dots \\ I^h_{trapeze} = h\big(U_{-h} + U_h\big) &\approx 2h\left(U_0 + \frac{h^2}{2}U_0^{"} + \frac{h^4}{24}U_0^{(4)} + \dots\right)\end{aligned}$$

Ensuite, on obtient l'expression de l'erreur locale en comparant les deux lignes.

$$\boxed{I - I^h_{trapeze} = -\frac{2h^3}{3}U"_0 - \frac{4h^5}{60}U_0^{(4)} - \dots}$$

Comme l'erreur est à nouveau en $\mathcal{O}(h^3)$, l'ordre de précision de la méthode est deux. Notons que l'erreur est maintenant de signe opposé ! Cela peut se voir graphiquement en dessinant l'erreur commise pour intégrer une simple parabole. Conclusion, c'est vraiment toujours utile de faire un tout petit dessin :-)

4. On choisit α et β afin de supprimer les deux premiers termes d'erreur en $\mathcal{O}(h^3)$:

$$I - I_{extr} \quad \approx \quad 2h\,(1-\alpha-\beta)\,U_0 + (\beta - 2\alpha)\,\frac{h^3}{3}U_0^{"} + (\beta - 4\alpha)\,\frac{h^5}{60}U_0^{(4)} + \ldots$$

En annulant les termes en U_0, et U_0'', on obtient directement : $\boxed{\alpha \;=\; \frac{1}{3}, \quad \beta \;=\; \frac{2}{3}}$

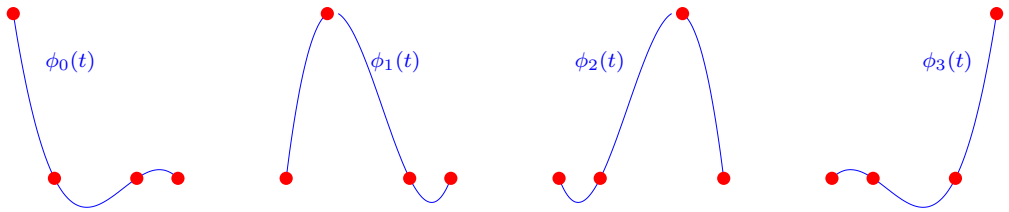
On retrouve ainsi la méthode de Simpson qui est bien une méthode d'ordre quatre.
Quelques étudiants ont subtilement deviné la réponse de cette manière : c'était admis !
Ce n'est pas une extrapolation de Richardson, même si il existe une grosse analogie : les deux coefficients sont ici positifs ! Evidemment, dans ce jeu pervers, pas mal d'étudiants sont tombés dans un piège assez grossier... en recopiant servilement et bêtement des coefficients de Richardson.

5. On déduit servilement : $\boxed{\left(\frac{2}{3} - \frac{4}{3}\right)\frac{h^5}{60}U_0^{(4)} \quad = \quad -\frac{h^5}{90}U_0^{(4)}}$

Notons qu'on retrouve exactement l'expression de l'erreur de la méthode de Simpson du syllabus. Recopier bêtement le formulaire fournissait la bonne réponse (pour une fois :-).

110 Janvier 2014 : Méditation rectorale

1. C'est une question élémentaire !
Et pourtant, beaucoup d'étudiants produisent des dessins vraiment lamentables...



Il n'y a vraiment aucune raison d'imposer une dérivée nulle (erreur fréquente) aux deux extrémités. Le dessin peut être très approximatif, mais doit ressembler à une fonction cubique avec un unique maximum et un unique minimum. Même si une très grande tolérance a été admise pour le dessin, c'était vraiment facile et même en n'ayant strictement rien étudié, vous auriez dû tous réussir cette question : c'est très loin d'être le cas :-)

On note -au passage- que quelques étudiants frustrés de ne pouvoir dessiner des NURBS ont décidé d'autorité de changer la question. Faut-il le répéter ? Oui, il faut lire l'énoncé !

Il n'était ni nécessaire, ni demandé d'écrire les expressions des fonctions de Lagrange.

2. Il suffit d'écrire :

$$\begin{array}{rclcrcl} x(t) & = & \displaystyle\sum_{i=0}^{3} X_i\phi_i(t) & \qquad & y(t) & = & \displaystyle\sum_{i=0}^{3} Y_i\phi_i(t) \\ & \downarrow & & & & \downarrow & \\ & = & 2\phi_1(t) & & & = & \phi_1(t)+\phi_2(t) \\ & = & \dfrac{t(t-3)(t-4)}{3} & & & = & \dfrac{t(t-3)(t-4)}{6} - \dfrac{t(t-1)(t-4)}{6} \end{array}$$

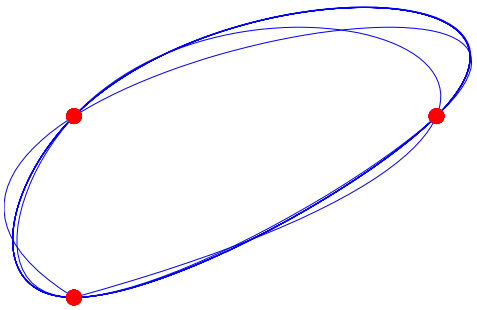
En $t = \frac{1}{2}$, on en déduit la position : $\boxed{\left(\frac{35}{24}, \frac{14}{24}\right)}$

Globalement, cette question purement calculatoire et fortement inspirée d'un exercice du syllabus a été remarquablement bien réussie :-) Comme quoi, malgré la fatigue de fin de session, certains d'entre vous ont eu le courage de faire ce petit effort qui leur fera sauver deux mois de vacances, pour pas mal d'entre vous.

3. Il suffit juste d'ajouter les quatre lignes suivantes sans autre forme de procès :

```
function theRectorMeditationCurve()
T = [0 1 3 4];
X = [0 2 0 0];
Y = [0 1 1 0];
t = linspace(0,4,100);
plot(spline(T,X,t),spline(T,Y,t),'-r'); end
```

Alors, comme dirait Léopold : "ça m'a l'air trop simple, j'ai zappé quelque chose ?"
Eh ben non, j'ai rien demandé d'autre. Espérer imposer la stricte périodicité de la trajectoire (ce qui n'était pas demandé !) en répétant les points de passage avant et après ne permettra pas de l'obtenir en plus.... A titre d'illustration, répéter dix fois les points d'intégration va bien générer des trajectoires de plus en plus proches mais jamais parfaitement et strictement périodiques : d'ailleurs, la solution 26.4 du syllabus d'exercice est erronée au passage et je devrais la modifier. Comme quoi pour tous les étudiants avides de bonus, oui il avait encore des erreurs que vous n'avez pas trouvées : et na :-)
Toutefois, le correcteur a quand-même été vraiment très compréhensif envers les quelques étudiants si enthousiastes de montrer qu'ils avaient étudié tous les exercices du syllabus et qui reproduisaient cette solution non demandée et erronée : oui, l'enseignant est parfois de bien mauvaise foi.



Oui : cette question était vraiment honteusement simple et a d'ailleurs permis de compenser la difficulté de la suivante :-)

111 Janvier 2014 : Une convergence en or !

1. Une implémentation possible est :

```
function x= secante(x0,x1,tol,nmax,f)
n = 0; delta = tol + 1; f0 = f(x0);
while abs(delta) >= tol && n < nmax
     n = n + 1;
     f1 = f(x1);
     delta = -f1*(x1-x0)/(f1-f0);
     x0 = x1;  f0 = f1; x1 = x1 + delta;
 end
 x = x1;
```

   Une implémentation efficace ne nécessite qu'un unique calcul de la fonction f par itération. C'est cela qui définit le vrai coût de la méthode par itération. En conséquence, le correcteur a retiré un point par évaluation inutile de cette fonction. Et donc (sans doute à l'étonnement de beaucoup d'étudiants), cette question qui paraissait très simple a été très souvent désastreuse pour tous ceux qui se sont bêtement contentés de recopier la formule de manière servile.

   En d'autres mots, il ne sert à rien de venir voir sa copie pour constater le désastre, si vous avez obtenu un seul point malgré l'écriture d'un très long code agrémenté de multiples commentaires totalement inutiles !

2. Si les itérations convergent et s'il existe deux constantes strictement positives $C < 1$ et $\alpha \geq 1$ telles que

$$\lim_{i \to \infty} \frac{|e_i|}{|e_{i-1}|^\alpha} = C$$

   on dit que la séquence converge avec un *taux de convergence* α.

3. Lorsque $i \to \infty$, on peut écrire que :

$$\begin{aligned} e_{i+1} &= D\, e_i e_{i-1} \\ &\downarrow \quad \text{Asymptotiquement, on peut écrire } e_{i+1} = C\,(e_i)^\alpha \\ C\,(e_i)^\alpha &= DC\,(e_{i-1})^\alpha\, e_{i-1} \\ CC^\alpha\,(e_{i-1})^{\alpha\alpha} &= DC\,(e_{i-1})^{\alpha+1} \end{aligned}$$

   Cette égalité ne peut être satisfaite que si $\alpha^2 = \alpha + 1$.
   L'unique racine positive de cette équation du second degré est le nombre d'or.

   Nous avons donc bien une convergence en or : $\boxed{\alpha = \dfrac{1+\sqrt{5}}{2} = 1.618}$

   C'est un peu futé, mais parfaitement faisable et on utilise la même idée que celle adoptée pour l'analyse de stabilité de méthode à pas liés. Quelques étudiants (pas beaucoup, j'en conviens) ont trouvé la solution. Il existe d'autres manières de démontrer le même résultat en particulier en faisant appel à des logarithmes...

4. Si la méthode converge, la fonction f tend progressivement vers zéro, on peut donc écrire :

$$\begin{aligned}
x_{i+1} &= x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})} \\
e_{i+1} &= e_i - \frac{f(x+e_i)(e_i - e_{i-1})}{f(x+e_i) - f(x+e_{i-1})} \\
e_{i+1} &= \frac{e_i f(x+e_i) - e_i f(x+e_{i-1}) - e_i f(x+e_i) + e_{i-1} f(x+e_i)}{f(x+e_i) - f(x+e_{i-1})} \\
e_{i+1} &= \frac{e_{i-1} f(x+e_i) - e_i f(x+e_{i-1})}{f(x+e_i) - f(x+e_{i-1})}
\end{aligned}$$

$\downarrow$ En effectuant les développements en série de Taylor :

$$e_{i+1} = \frac{e_{i-1}\left(e_i f'(x) + e_i^2 \frac{f''(x)}{2}\right) - e_i\left(e_{i-1} f'(x) + e_{i-1}^2 \frac{f''(x)}{2}\right))}{\left(e_i - e_{i-1}\right) f'(x)}$$

$$e_{i+1} = \left(e_{i-1} e_i\right) \frac{\left(e_i - e_{i-1}\right) f''(x)}{\left(e_i - e_{i-1}\right) 2 f'(x)}$$

On obtient alors finalement : $\boxed{D = \frac{f''(x)}{2f'(x)}}$

Obtenir le bon rapport n'est pas aisé, je le reconnais bien volontiers ! C'est une question difficile que peu d'étudiants ont vraiment abordée. Toutefois, j'ai été assez ou très agréablement surpris d'observer que quelques étudiants ont parfaitement résolu l'entièreté de cette troisième question.

112 Janvier 2015 : Un petit logarithme :-)

1. Tout d'abord, on écrit simplement l'interpolation linéaire passant par a et b :

$$u^h(x) = U_a \frac{b-x}{b-a} + U_b \frac{x-a}{b-a}$$

Et ensuite, on effectue l'intégration de cette interpolation...

$$\begin{aligned}
I_h = \int_a^b u^h(x)dx &= \frac{U_a}{b-a}\int_a^b b - x \; dx + \frac{U_b}{b-a}\int_a^b x - a \; dx \\
&= \frac{U_a}{b-a}\left[bx - \frac{x^2}{2}\right]_a^b + \frac{U_b}{b-a}\left[\frac{x^2}{2} - ax\right]_a^b \\
&= \frac{U_a}{b-a}\frac{(b-a)^2}{2} + \frac{U_b}{b-a}\frac{(b-a)^2}{2}
\end{aligned}$$

On retrouve bien la règle du trapèze : $\boxed{I_h = (U_a + U_b)\frac{(b-a)}{2}}$

Une explication basée sur un petit dessin d'un trapèze est admise, mais un propos totalement vague n'est pas accepté. Faire le calcul pour l'exemple du logarithme ne répond pas à la question !

2. Il suffit d'écrire :

$$\int_a^b \frac{1}{x}\,dx \approx \frac{b-a}{4}\left(u(a)+2u\left(\frac{a+b}{2}\right)+u(b)\right)$$

$$\downarrow$$

$$\ln(2) = \int_1^2 \frac{1}{x}\,dx \approx \frac{1}{4}\left(1+\frac{4}{3}+\frac{1}{2}\right) = \frac{1}{4}\left(\frac{6+8+3}{6}\right)$$

On conclut tout simplement : $\boxed{\ln(2) \approx \frac{17}{24}}$

La calcul est élémentaire, mais il faut vraiment obtenir le résultat final et de manière très surprenante, tous les étudiants n'arrivent pas à obtenir cela !

3. En sachant que $nh = (b-a)$, il suffit de choisir n tel que :

$$|E^h| \leq \frac{C_2(b-a)^3}{12n^2} \leq 10^{-8}$$

$$\downarrow$$ Car la valeur maximale de $u''(x) = \frac{2}{x^3}$ vaut 2 sur l'intervalle $[1,2]$.

$$\frac{1}{6n^2} \leq 10^{-8}$$

$$\downarrow$$

$$\frac{10^4}{\sqrt{6}} \leq n$$

Et on obtient le nombre requis d'intervalles : $\boxed{n \geq 4083}$

Le calcul est élémentaire et il était vraiment requis ici de réellement estimer C_2 par une valeur numérique. Le correcteur a été très déçu par la difficulté que rencontrent vraiment beaucoup trop d'étudiants pour dériver deux fois un logarithme et borner cette dérivée sur un intervalle élémentaire. Il faut donc bien obtenir une valeur numérique du bon ordre et faire tout le calcul!

113 Janvier 2015 : Charlie veut faire de l'ordre trois !

1. En définissant $v(t) = u'(t)$ et $w(t) = u''(t)$,

on obtient immédiatement : $\boxed{\begin{cases} u'(t) = v(t) \\ v'(t) = w(t) \\ w'(t) = 2w(t) - v(t) + \cos(t) \end{cases}}$

Les conditions initiales sont $u(0) = v(0) = w(0) = 1$.

2. Le schéma d'Euler explicite est :

$$\boxed{\begin{array}{rcl} U_{i+1} & = & U_i + hV_i \\ V_{i+1} & = & V_i + hW_i \\ W_{i+1} & = & W_i + h\Big(2W_i - V_i + \cos(T_i)\Big) \end{array}}$$

Les conditions initiales sont $U_0 = V_0 = W_0 = 1$. Il existe d'autres possibilités en faisant des combinaisons linéaires des trois équations, mais il faut vraiment être un peu tordu pour imaginer cela. C'est une extension purement mécanique d'un exemple fait au cours et cela a posé pas mal de difficultés à beaucoup d'étudiants.

3. Pour obtenir une méthode de Taylor d'ordre deux, il faut ajouter :

$$\begin{array}{rcl} \cdots & + & \dfrac{h^2}{2} V_i' \\ \cdots & + & \dfrac{h^2}{2} W_i' \\ \cdots & + & \dfrac{h^2}{2}\Big(2W_i' - V_i' - \sin(T_i)\Big) \end{array}$$

Comme $v'(t) = w(t)$ et $w'(t) = 2v(t) - w(t) + \cos(t)$, on peut écrire simplement :

$$\begin{array}{rcl} \cdots & + & \dfrac{h^2}{2} W_i \\ \cdots & + & \dfrac{h^2}{2}\Big(2W_i - V_i + \cos(T_i)\Big) \\ \cdots & + & \dfrac{h^2}{2}\Big(2\Big(2W_i - V_i + \cos(T_i)\Big) - W_i - \sin(T_i)\Big) \end{array}$$

Le schéma de Taylor est donc défini par :

$$\boxed{\begin{array}{rcl} U_{i+1} & = & U_i + hV_i + \dfrac{h^2}{2} W_i \\ V_{i+1} & = & V_i + hW_i + \dfrac{h^2}{2}\Big(2W_i - V_i + \cos(T_i)\Big) \\ W_{i+1} & = & W_i + h\Big(2W_i - V_i + \cos(T_i)\Big) + \dfrac{h^2}{2}\Big(3W_i - 2V_i + 2\cos(T_i) - \sin(T_i)\Big) \end{array}}$$

Il est indispensable de faire le calcul complet et d'obtenir une expression correcte du terme en rouge pour être crédité de l'ensemble des points. Ici,l'algèbre est élémentaire et vous aviez largement le temps de la faire avec calme et précision. Certains étudiants devraient noter la dérivée du cosinus sur leur formulaire :-)

Attention, la méthode de Heun est équivalent à la méthode de Taylor d'ordre deux uniquement pour des problèmes linéaires homogènes : ce qui n'est pas le cas ici. Me raconter plein de jolies histoires et me donner un programme MATLAB Heun à la place de ce qui est demandé, n'était

pas une bonne idée... Se méfier des réponses qui semblent trop simples, même si parfois les énoncés des examens souffrent de petites coquilles de l'enseignant !

4. Une implémentation possible est :

```
function [T,U] = taylorSystem(n,h)
T = h*(0:n);
U = ones(3,n+1);
for i = 1:n
    u = U(1,i); v = U(2,i); w = U(3,i); t = T(i);
    f = 2*w - v + cos(t); df = 2*f - w - sin(t);
    U(1,i+1) = u + h*v + h*h*w/2;
    U(2,i+1) = v + h*w + h*h*f/2;
    U(3,i+1) = w + h*f + h*h*df/2;
end
```

Fournir l'implémentation d'une autre méthode n'était vraiment pas une bonne idée et ne rapportait rien ! C'est juste une perte de temps et d'énergie. Il était essentiel d'observer qu'il fallait fournir `f` et `df`. C'était aussi utile de veiller à ne pas calculer deux fois la dérivée `f`. Se contenter d'écrire la méthode d'Euler explicite est évidemment totalement insuffisant !

Même si vous n'avez pas tous les termes requis dans la question précédente, il est possible d'écrire la plus grande partie réellement utile de ce programme !

5. Pour obtenir une méthode de Taylor d'ordre trois, il faut ajouter :

$$\begin{aligned} \cdots &+ \frac{h^3}{6} W_i' \\ \cdots &+ \frac{h^3}{6}\Big(2W_i' - V_i' - \sin(T_i)\Big) \\ \cdots &+ \frac{h^3}{6}\Big(3W_i' - 2V_i' - 2\sin(T_i) - \cos(T_i)\Big) \end{aligned}$$

Comme $v'(t) = w(t)$ et $w'(t) = 2v(t) - w(t) + \cos(t)$, on peut écrire simplement :

$$\begin{aligned} \cdots &+ \frac{h^3}{6}\Big(2W_i - V_i + \cos(T_i)\Big) \\ \cdots &+ \frac{h^3}{6}\Big(2\Big(2W_i - V_i + \cos(T_i)\Big) - W_i - \sin(T_i)\Big) \\ \cdots &+ \frac{h^3}{6}\Big(3\Big(2W_i - V_i + \cos(T_i)\Big) - 2W_i - 2\sin(T_i) - \cos(T_i)\Big) \end{aligned}$$

Les trois termes supplémentaires sont donc :

$$\begin{aligned} \cdots &+ \frac{h^3}{6}\Big(2W_i - V_i + \cos(T_i)\Big) \\ \cdots &+ \frac{h^3}{6}\Big(3W_i - 2V_i + 2\cos(T_i) - \sin(T_i)\Big) \\ \cdots &+ \frac{h^3}{6}\Big(4W_i - 3V_i + 2\cos(T_i) - 2\sin(T_i)\Big) \end{aligned}$$

Il est indispensable de faire le calcul complet et d'obtenir une expression correcte du terme en rouge pour être crédité de l'ensemble des points : pas mal d'étudiants y sont parvenus. Oui,

c'est de l'algèbre un peu bêtifiante, je le reconnais bien volontiers, mais il faut vraiment de tout pour faire le bonheur de Charlie. Notons au passage que les deux premiers termes d'ordre trois sont simplement les deux derniers termes d'ordre deux. Il ne fallait donc en réalité que calculer deux termes en tout et pour tout pour faire le bonheur complet de Charlie !

114 Janvier 2015 : Une méthode spéciale pour des intégrales pondérées

1. Il suffit de prendre comme polynôme :

$$u(x) = (x - X_1)^2(x - X_2)^2$$

qui est une fonction toujours positive sur l'intervalle et qui ne s'annule qu'en X_1 et X_2. La fonction cosinus étant strictement positive sur l'intervalle, on en déduit que l'intégrale exacte doit être strictement positive, alors que la formule de quadrature fournira toujours un résultat nul. Ce contre exemple démontre de manière indiscutable qu'il était impossible que le degré de précision pondérée soit égal à quatre !

Moi qui pensait naïvement que c'était simple. Un seul étudiant a trouvé mon contre-exemple : toutes mes félicitations à ce petit malin qui a malheureusement piteusement échoué dans la suite de la question. Bon, il était aussi possible qu'observer qu'on avait quatre paramètres et qu'un polynôme de degré quatre a cinq degrés de libertés : il est possible de montrer que la cinquième relation ne sera quasiment jamais satisfaite...

C'était vraiment compliqué et j'aurais dû mettre la question tout à la fin, mais j'espérais que vous trouviez le contre exemple plus facilement : eh oui :-)
J'ai donc ajouté des étoiles dans la solution : ce qui n'était pas présent dans l'énoncé.

2. Il faut calculer les deux poids et abscisses afin que la quadrature fournisse des valeurs exactes pour $u(x) = 1$, $u(x) = x$, $u(x) = x^2$ et $u(x) = x^3$. C'est exactement le même raisonnement que nous avons fait pour Gauss-Legendre, il faut uniquement ajouter le cosinus et bien observer que ce cosinus est toujours positif, pair (et donc symétrique) sur l'intervalle considéré. On fera donc les mêmes simplifications et on obtiendra un résultat très proche de ce qu'on a obtenu avec Gauss-Legendre.

$$\int_{-\pi/2}^{\pi/2} \cos(x)\ dx = a_1 + a_2 \qquad \int_{-\pi/2}^{\pi/2} x\ \cos(x)\ dx = a_1X_1 + a_2X_2$$

$$\int_{-\pi/2}^{\pi/2} x^2\ \cos(x)\ dx = a_1X_1^2 + a_2X_2^2 \qquad \int_{-\pi/2}^{\pi/2} x^3\ \cos(x)\ dx = a_1X_1^3 + a_2X_2^3$$

Ecrire les équations était une partie importante de la réponse, il suffisait ensuite de calculer les quatre intégrales.

- Les deux intégrales impliquant x et x^3 sont nulles par symétrie !
- Normalement, on obtient très facilement :

$$\int_{-\pi/2}^{\pi/2} \cos(x)\ dx = \Big[\sin(x)\Big]_{-\pi/2}^{\pi/2} = 2$$

- La dernière intégrale est un tout peu plus compliquée à obtenir :

$$\begin{aligned}
\int_{-\pi/2}^{\pi/2} x^2 \ \cos(x)\ dx &= \Big[x^2 \sin(x)\Big]_{-\pi/2}^{\pi/2} - \int_{-\pi/2}^{\pi/2} 2x \ \sin(x)\ dx \\
&= \frac{\pi^2}{2} - \int_{-\pi/2}^{\pi/2} 2x \ \sin(x)\ dx \\
&= \frac{\pi^2}{2} + \Big[2x\cos(x)\Big]_{-\pi/2}^{\pi/2} - \int_{-\pi/2}^{\pi/2} 2 \ \cos(x)\ dx \\
&= \frac{\pi^2}{2} - 4
\end{aligned}$$

Les quatre équations sont :

$$\boxed{\begin{aligned}
2 &= a_1 + a_2 \\
0 &= a_1 X_1 + a_2 X_2 \\
\frac{\pi^2}{2} - 4 &= a_1 X_1^2 + a_2 X_2^2 \\
0 &= a_1 X_1^3 + a_2 X_2^3
\end{aligned}}$$

Avec $X_1 = -X_2$ et $a_1 = a2 = 1$, la dernière et les deux premières équations sont satisfaites. La troisième équation se réduit simplement à

$$\frac{\pi^2}{2} - 4 = 2X_1^2$$

On obtient finalement :

$$\boxed{X_1 = \sqrt{\frac{\pi^2}{4} - 2}, \quad X_2 = -\sqrt{\frac{\pi^2}{4} - 2},}$$

Si beaucoup d'étudiants échouent à cette question un peu plus compliquée, il y a quand-même un nombre non négligeable (plus ou moins un quart des étudiants) qui arrivent au résultat final correct. Cela a plutôt été une bonne surprise pour moi. Par contre, honte à tous ceux qui sont incapables d'intégrer la fonction cosinus entre $-\pi/2$ et $\pi/2$: ça c'est vraiment pas bien du tout et il y a un nombre vraiment incompréhensible d'étudiants qui ne savent pas (ne veulent pas ?) faire cela une fois comme on dit chez moi :-)

3. C'était honteusement simple....

```
function I = weightedIntegral(u,a,X)
I = u(X(1))*a(1) + u(X(2))*a(2);
end;
```

Les étudiants qui ont implémenté une méthode composite n'ont pas été pénalisés. Quelques petits futés ont remarqué que c'était une idée stupide de vouloir faire une méthode composite : c'est très bien vu. Bon, avoir voulu ajouter cette dernière sous-question était une mauvaise idée. Oui, oui je le reconnais et donc il n'y avait quasiment aucun point associé à cette sous-question :-)

115 Janvier 2016 : Orbite d'une planète mystérieuse

1. Il s'agit de minimiser la fonction suivante :

$$J(a) = \sum_{i=1}^{n} \left(R_i - \frac{10}{(6 - a\cos\Theta_i + \cos\Theta_i)} \right)^2$$

et d'annuler sa dérivée première :

$$0 \;=\; J'(a) \;=\; \sum_{i=1}^{n} 2 \left(R_i - \frac{10}{(6 - a\cos\Theta_i + \cos\Theta_i)} \right) \left(\frac{10\cos\Theta_i}{(6 - a\cos\Theta_i + \cos\Theta_i)^2} \right)$$

La réponse est donc $\boxed{K(a) \;=\; \sum_{i=1}^{n} \frac{R_i \cos\Theta_i}{(6 - a\cos\Theta_i + \cos\Theta_i)^2} - \sum_{i=1}^{n} \frac{10\cos\Theta_i}{(6 - a\cos\Theta_i + \cos\Theta_i)^3}}$

Ecrire $J(a)$ et non $K(a)$ est évidemment une erreur impardonnable :-)

2. Le schéma de Newton-Raphson s'écrit :

$$\boxed{\begin{array}{c} \text{On calcule } a_{k+1} \text{ à partir de } a_k \text{ avec} \\ K'(a_k)\ \Delta a \;=\; -K(a_k) \\ a_{k+1} \;=\; a_k + \Delta a \end{array}}$$

avec la dérivée : $K'(a) \;=\; \sum_{i=1}^{n} \frac{2R_i \cos^2\Theta_i}{(6 - a\cos\Theta_i + \cos\Theta_i)^3} - \sum_{i=1}^{n} \frac{30\cos^2\Theta_i}{(6 - a\cos\Theta_i + \cos\Theta_i)^4}$

3. Il suffit de calculer :

$$K(5) \;=\; \left(\frac{5}{4}\right) - \left(\frac{10}{8} - \frac{10}{1000}\right) \;=\; \frac{1}{100}$$

$$K'(5) \;=\; \left(\frac{10}{8}\right) - \left(\frac{30}{16} + \frac{30}{10000}\right) \;=\; \frac{-3 - 625}{1000} \;=\; -\frac{314}{500}$$

On peut finalement en déduire que : $-\frac{314}{500}\ \Delta a \;=\; \frac{1}{100}$.

et écrire que : $\boxed{a_1 \;=\; 5 + \frac{5}{314} \;=\; \frac{1575}{314} \;=\; 5.016}$

L'algèbre n'était vraiment pas calculatoire et était parfaitement faisable :-)
Un nombre tout-à-fait honnête d'étudiants a obtenu la fraction finale.

Finalement, une option astucieuse choisie par quelques étudiants consistait à directement résoudre toute la question avec les données. Cela revient à écrire :

$$J(a) \;=\; \left(5 - \frac{10}{(7 - a)}\right)^2 + \left(\frac{10}{(5 + a)}\right)^2 \;=\; 25 - \frac{100}{(7 - a)} + \frac{100}{(7 - a)^2} + \frac{100}{(5 + a)^2}$$

et à annuler la dérivée première :

$$0 \quad = \quad J'(a) \quad = \quad -\frac{100}{(7-a)^2} + \frac{200}{(7-a)^3} - \frac{200}{(5+a)^3}$$

Les étudiants qui ont procédé de la sorte n'ont pas été pénalisés, même si ce n'est pas l'approche qu'avait imaginé le concepteur de la question. Globalement, les calculs sont un peu plus simples surtout si on divise $K(a)$ par un facteur 50 avant de faire la somme des fractions ! Par contre, l'approche est moins générale et cette résolution peut donner l'impression d'être très calculatoire et un peu simpliste...

116

Janvier 2016 : Le train arrivera-t-il ?

1. La solution du problème homogène est $C\exp(t)$ et une solution particulière du problème non-homogène est $exp(2t)$. La condition initiale permet de déduire immédiatement que $C = 1$.

   On conclut donc : $\boxed{u(t) \quad = \quad \exp(t) + \exp(2t)}$

2. Pour effectuer la démonstration, il suffit d'écrire les trois développements de Taylor :

$$U_{i-1} \quad = \quad U_i - h\,U_i' + \frac{h^2}{2}U_i'' + \mathcal{O}(h^3)$$

$$U_{i+1} \quad = \quad U_i + h\,U_i' + \frac{h^2}{2}U_i'' + \mathcal{O}(h^3)$$

$$F_{i+1} \quad = \quad U_{i+1}' \quad = \quad U_i' + h\,U_i'' + \mathcal{O}(h^2)$$

   Ensuite, on les incorpore dans la formule du cheminot :

$$U_{i+1} \quad = \quad \frac{1}{3}\left(-U_{i-1} + 4U_i\right) + \frac{2h}{3}\,F_{i+1}$$

   En remplaçant U_{i+1}, U_{i-1} et F_{i+1} par leur développement

$$U_i + h\,U_i' + \frac{h^2}{2}U_i'' + \mathcal{O}(h^3) \quad = \quad \underbrace{\left(-\frac{1}{3} + \frac{4}{3}\right)}_{=1}\,U_i + \underbrace{\left(\frac{h}{3} + \frac{2h}{3}\right)}_{=h}\,U_i' + \underbrace{\left(-\frac{h^2}{6} + \frac{2h^2}{3}\right)}_{=h^2/2}\,U_i'' + \mathcal{O}(h^3)$$

   En identifiant les termes, on observe donc une erreur locale en $\mathcal{O}(h^3)$ à chaque pas temps. L'erreur globale est donc en $\mathcal{O}(h^2)$ et la méthode est donc bien d'ordre deux.

   □

   Juste reconnaître la méthode de Gear d'ordre deux n'est pas une démonstration rigoureuse ! Démontrer que l'erreur locale est en $\mathcal{O}(h^2)$ est évidemment une erreur :-)

3. On applique simplement $U_1 = U_0 + hF_0$ avec $h = 1$.

   Et on conclut : $\boxed{U_1 \quad = \quad 2 + (2 + \exp(0)) \quad = \quad 5}$

4. Il suffit à nouveau de simplement écrire la formule avec $h = 1$.

$$\begin{array}{rcl} U_2 & = & \frac{1}{3}\Big(-U_0 + 4U_1\Big) + \frac{2}{3}\,\Big(U_2 + \exp(4)\Big) \\ & \Big\downarrow & \\ U_2 & = & \frac{18}{3} + \frac{2}{3}U_2 + \frac{2}{3}\exp(4) \end{array}$$

Et on conclut : $\boxed{U_2 \;=\; 18 + 2\exp(4) \;=\; 127.2}$

Il est normal d'ignorer que $\exp(4) \approx 55$, il est pitoyable de ne pas savoir que $\exp(0) = 1$!
Sans calculatrice, moi aussi je suis incapable d'obtenir 127 !
Par contre, il fallait écrire $U_1 = 5$ pour recevoir le point pour la question d'Euler explicite !

5. Une implémentation possible est :

```
function [T,U] = cheminot(n,h)

   T = linspace(0,n*h,n+1)
   U = zeros(1,n+1);
   U(1) = 2;
   U(2) = 2 + 3*h;
   for i=3:n+1
        U(i) = (-U(i-2) + 4*U(i-1) + 2*h*exp(2*h*(i-1)))/ (3*(1-2*h/3));
   end

end
```

Fournir l'implémentation d'une autre méthode n'était vraiment pas une bonne idée !
Il faut évidemment utiliser un pas h pour l'itération d'Euler et ne pas bêtement utiliser la valeur numérique de 5 trouvée à la question précédente.
Pour valider la question, la fraction dans l'itération du cheminot doit être correcte.
Oublier de préallouer le vecteur `U` est une erreur !

117 Janvier 2016 : Polynômes d'Hermite

1. Imposer $\phi_0(0) = 1$ et $\phi_1(0) = \phi_2(0) = \phi_3(0) = 0$ permet d'obtenir $U_0 = u^h(0)$.
Imposer $\phi_1(1) = 1$ et $\phi_0(1) = \phi_2(1) = \phi_3(1) = 0$ permet d'obtenir $U_1 = u^h(1)$.
Imposer $\phi_2'(0) = 1$ et $\phi_0'(0) = \phi_1'(0) = \phi_3'(0) = 0$ permet d'obtenir $U_0' = (u^h)'(0)$.
Imposer $\phi_3'(1) = 1$ et $\phi_0'(1) = \phi_1'(1) = \phi_2'(1) = 0$ permet d'obtenir $U_1' = (u^h)'(1)$.

Pour déterminer les 4 coefficients du polynôme $\phi_0(x) = ax^3 + bx^2 + cx + d$, on dispose de quatre contraintes qu'il est très aisé d'écrire :

$$\begin{array}{rcrcl}
\phi_0(0) & = & d & = & 1 \\
\phi_0'(0) & = & c & = & 0 \\
\phi_0(1) & = & a+b+c+d & = & 0 \\
\phi_0'(1) & = & 3a+2b+c & = & 0
\end{array}$$

$$\downarrow$$

$$\begin{array}{rcl}
a+b+1 & = & 0 \\
3a+2b & = & 0
\end{array}$$

$$\downarrow$$

$$\begin{array}{rcl}
a-2 & = & 0 \\
3a+2b & = & 0
\end{array}$$

On déduit finalement que $a = 2$, $b = -3$ et $d = 1$. Il était possible d'observer que le polynôme s'écrivait sous la forme $(x-1)^2(ex+f)$ puisqu'il a une racine double en $x = 1$: cela facilitait un peu les calculs. Par contre, utiliser mécaniquement les formules de Lagrange assurait une catastrophe automatique :-) Eh oui, pour une fois, il fallait résoudre le système pour trouver les quatre polynômes. Noter toutefois que les trois autres polynômes sont encore nettement plus faciles à obtenir et que le quatrième permettait de vérifier que l'on avait bien compris la démarche :-)

En procédant de la même manière pour les trois autres polynômes,

on obtient :

$$\boxed{\begin{array}{lcll} \phi_0(x) & = & (x-1)^2(2x+1) & = & 2x^3-3x^2+1 \\ \phi_1(x) & = & x^2(3-2x) & = & -2x^3+3x^2 \\ \phi_2(x) & = & x(x-1)^2 & = & x^3-2x^2+x \\ \phi_3(x) & = & x^2(x-1) & = & x^3-x^2 \end{array}}$$

2. Il suffit juste d'intégrer les quatre polynômes d'Hermite pour obtenir le résultat :-)

$$\begin{array}{rclcccccl} a & = & \displaystyle\int_0^1 \phi_0(x)\,dx & = & \Big[\frac{2}{4}x^4-\frac{3}{3}x^3+x\Big]_0^1 & = & \frac{1-2+2}{2} & = & \frac{1}{2} \\ b & = & \displaystyle\int_0^1 \phi_1(x)\,dx & = & \Big[-\frac{2}{4}x^4+\frac{3}{3}x^3\Big]_0^1 & = & \frac{-1+2}{2} & = & \frac{1}{2} \\ c & = & \displaystyle\int_0^1 \phi_2(x)\,dx & = & \Big[\frac{1}{4}x^4-\frac{2}{3}x^3+\frac{1}{2}x^2\Big]_0^1 & = & \frac{3-8+6}{12} & = & \frac{1}{12} \\ d & = & \displaystyle\int_0^1 \phi_3(x)\,dx & = & \Big[\frac{1}{4}x^4-\frac{1}{3}x^3\Big]_0^1 & = & \frac{3-4}{12} & = & -\frac{1}{12} \end{array}$$

Et on conclut donc :

$$\boxed{I^h = \frac{1}{2}\Big(U_0+U_1\Big)+\frac{1}{12}\Big(U_0'-U_1'\Big)}$$

Il est intéressant d'observer qu'imposer que la quadrature soit de degré trois (ce qui est assez prévisible :-) permettait d'obtenir exactement le même résultat ou de confirmer ce que vous aviez obtenu : on voit immédiatement que la quadrature sera exacte pour des polynômes de degré un à trois puisque $u^h(x)$ représente exactement n'importe quel polynôme de degré trois.

Un peu d'intuition permettait de deviner assez facilement b lorsqu'on avait a pour pouvoir intégrer le polynôme constant. Vérifier que la règle de quadrature intègre parfaitement un polynôme de degré trois permettait également d'obtenir ces quatre coefficients même sans avoir les polynômes d'Hermite : vous aviez largement le temps de valider vos calculs et d'obtenir une réponse correcte ! Pas mal d'étudiants y sont d'ailleurs arrivé !

3. Comme $u^h(x)$ sera une approximation parfaite de n'importe quel polynôme de degré trois, la règle d'intégration sera toujours de degré trois.

Il suffisait donc tout simplement d'écrire : $\boxed{d = 3}$

Il n'était pas nécessaire de justifier votre réponse et quand on ne demande pas quelque chose, il ne faut pas le faire !

Toutefois pour les inquiets, les angoissé, les incrédules, les malveillants et les sceptiques, on pouvait très aisément le vérifier en calculant :

$$
\begin{array}{rcccccccl}
I & = & \displaystyle\int_0^1 dx & = & 1 & = & \frac{1}{2}+\frac{1}{2}+0-0 & = & I_h \\
I & = & \displaystyle\int_0^1 x\ dx & = & \frac{1}{2} & = & 0+\frac{6}{12}+\frac{1}{12}-\frac{1}{12} & = & I_h \\
I & = & \displaystyle\int_0^1 x^2\ dx & = & \frac{1}{3} & = & 0+\frac{6}{12}+0-\frac{2}{12} & = & I_h \\
I & = & \displaystyle\int_0^1 x^3\ dx & = & \frac{1}{4} & = & 0+\frac{6}{12}+0-\frac{3}{12} & = & I_h \\
I & = & \displaystyle\int_0^1 x^4\ dx & = & \frac{1}{5} & \neq & 0+\frac{6}{12}+0-\frac{4}{12}- & = & I_h
\end{array}
$$

Eh non : il n'y a pas de degré bonus car on utilise une polynôme de degré trois : c'est pas une règle de quadrature optimale puisqu'avec trois degrés de liberté, on pourrait obtenir le même degré de précision. Donc tous les étudiants qui attribuent bêtement un degré bonus ont reçu un malus correspondant dans leur note. Conclusion : écoute le professeur mais n'oublie pas qu'au jeu subtil du plus crétin, c'est toujours lui qui gagne !

118 **Janvier 2017 : Hillary perdue dans les différences finies**

1. Il suffit d'écrire les développements en série de Taylor comme suit :

$$
\begin{aligned}
U_h &= U_0 + hU_0' + \frac{h^2}{2}U_0'' + \frac{h^3}{6}U_0''' + \frac{h^4}{24}U_0'''' \dots \\
U_{2h} &= U_0 + 2hU_0' + \frac{4h^2}{2}U_0'' + \frac{8h^3}{6}U_0''' + \frac{16h^4}{24}U_0'''' \dots \\
U_{3h} &= U_0 + 3hU_0' + \frac{9h^2}{2}U_0'' + \frac{27h^3}{6}U_0''' + \frac{81h^4}{24}U_0'''' \dots
\end{aligned}
$$

En substituant ces développements dans la formule d'Hillary, on obtient alors :

$$
\frac{(2+\alpha+\beta-1)}{h^2}U_0+\frac{(\alpha+2\beta-3)}{h}U_0'+\frac{(\alpha+4\beta-9)}{2}U_0''+\frac{(\alpha+8\beta-27)}{6}hU_0'''+\frac{(\alpha+16\beta-81)}{24}h^2U_0''''
$$

Pour que la formule d'Hillary aie du sens, il faut annuler les coefficients de U_0 et de U_0'. Il faut aussi exiger que le coefficient de U_0'' soit unitaire. Il y a trois conditions et deux inconnues: il est donc vraiment possible de vérifier que la réponse trouvée est bien correcte. C'est donc vraiment impardonnable de ne pas obtenir la solution exacte !

De la première condition sur U_0, on peut déduire que $\alpha = -(1+\beta)$...

... et de la seconde condition sur U_0', on déduit immédiatement que : $\boxed{\begin{array}{rcl}\alpha & = & -5 \\ \beta & = & 4\end{array}}$

2. Ensuite, on substitue α et β dans la formule d'Hillary pour obtenir le terme d'erreur :-)

$$\underbrace{\frac{(2-5+4-1)}{h^2}}_{=0}U_0+\underbrace{\frac{(-5+8-3)}{h}}_{=0}U_0'+\underbrace{\frac{(-5+16-9)}{2}}_{=1}U_0''+\underbrace{\frac{(-5+32-27)}{6}}_{=0}hU_0'''+\underbrace{\frac{(-5+64-81)}{24}}_{=11/12}h^2U_0'''$$

On conclut finalement : $\boxed{u'''0) \quad = \quad \frac{2U_0+\alpha U_h+\beta U_{2h}-U_{3h}}{h^2}-\frac{11}{12}h^2u^{(4)}(\zeta)}$

La méthode d'Hillary est donc d'ordre deux et correspond à une différence amont usuelle :-)

3. Il suffit de minimiser $E(h) \; = \; \frac{(2+5+4+1)}{h^2}\epsilon+\frac{11}{12}h^2$.

Attention : les erreurs ne font que s'additionner et donc on prend la valeur absolue de tous les coefficients de la formule d'Hillary et on prend la somme de l'erreur d'arrondi et de l'erreur de discrétisation même si cette dernière est négative !

$$\text{On exige donc que } f'(h) \; = \; -\frac{24}{h^3}\epsilon+\frac{22h}{12} \quad = \quad 0$$

$$\downarrow$$

$$h^4 \quad = \quad \frac{144}{11}\,\epsilon$$

On conclut que le pas optimal est donné par : $\boxed{h \quad = \quad \sqrt[4]{\frac{144}{11}\,\epsilon}}$

119 **Janvier 2017 : Donald veut calculer pi !**

1. Il suffit d'écrire : $\boxed{x_{i+1} \quad = \quad x_i+2\,\frac{\cos\left(\frac{x_i}{2}\right)}{\sin\left(\frac{x_i}{2}\right)}}$

2. Si les itérations convergent et s'il existe deux constantes positives $C<1$ et $r \geq 1$ telles que

$$\lim_{i\to\infty}\frac{|e_i|}{|e_{i-1}|^r}=C$$

on dit que la séquence converge avec un *taux de convergence* r.
Il est aussi judicieux que la constante r soit supérieure à l'unité.

3. Cela provient du fait que : $f''(\pi)=-\frac{1}{4}\,\cos\left(\frac{\pi}{2}\right)=0$

Pour effectuer la démonstration, il suffit tout simplement d'écrire

$$
\begin{aligned}
x_{i+1} &= x_i - \frac{f(x_i)}{f'(x_i)} \\
\underbrace{x_{i+1} - x}_{e_{i+1}} &= \underbrace{x_i - x}_{e_i} - \frac{f(x+e_i)}{f'(x+e_i)} \\
&\downarrow \quad \text{En effectuant un développement en série de Taylor pour } g(x) = \frac{f(x)}{f'(xi)} : \\
e_{i+1} &= e_i - g(x) - g'(x)\ e_i - g''(x)\ \frac{e_i^2}{2} + g'''(x)\ \frac{e_i^3}{6} + \ldots \\
&\downarrow \quad \text{En observant que } g(x) = g''(x) = 0 \text{ et } g'(x) = 1 \text{ car } f(x) = f''(x) = 0 \\
e_{i+1} &= g'''(x)\ \frac{e_i^3}{6}
\end{aligned}
$$

□

Pour les sceptiques, les grincheux et les fans d'algèbre,
la justification détaillée s'effectue comme suit :

$$
\begin{aligned}
g(x) &= \frac{f}{f'} = 0 \\
g'(x) &= \frac{f'}{f'} - \frac{ff''}{(f')^2} = 1 - \frac{ff''}{(f')^2} = 1 \\
g''(x) &= -\frac{f'f''}{(f')^2} - \frac{ff'''}{(f')^2} + \frac{2ff''f''}{(f')^3} = 0
\end{aligned}
$$

Pour cette question (difficile), plusieurs réponses et démonstrations étaient possibles. Pas mal d'étudiants ont la bonne intuition, mais peu arrivent à rédiger proprement une démonstration convaincante. Il est assez décevant d'observer qu'une immense majorité d'étudiants ne répondent pas du tout à cette question, alors que citer l'intuition que cela provenait du caractère particulier de la dérivée seconde de la fonction cosinus rapportait déjà pas mal de points !

4. Une implémentation possible est donnée par :

```
function [x error] = cosinus(x0,tol,nmax)
  n=1; delta = tol + 1; x(1) = x0;
  while abs(delta) >= tol && n < nmax
      error(n) = abs(pi - x(n));
      delta = 2 * cot(x(n)/2.0);
      x(n+1) = x(n) + delta;
      n = n + 1;
  end
end
```

5. Le taux de convergence observé est trois (et même un peu plus) car on triple (et même un peu plus) le nombre de chiffres significatifs à chaque itération.
On pouvait aussi simplement noter que $(10^{-4})^3 = 10^{-12}$.

Il ne fallait pas faire des calculs détaillés, mais une très brève justification était requise !
Certains étudiants se lancent dans des calculs inutilement complexes et hasardeux...
L'estimation (non demandée évidemment :-) via MATLAB

```
[x error] = cosinus(3,1e-13,100);
Order = log(error(2:end-1)./error(1:end-2))./log(error(1:end-2)) + 1
```

fournit les valeurs suivantes : `Order = 4.2702 3.2977`
C'est donc bien en accord avec la théorie !

120 Janvier 2017 : Heun revisité par Barack

1. Il suffit d'effectuer le développement en série de Taylor :

$$\begin{array}{rcl} U_{i+1} & = & U_i + h\left(\frac{3}{4}F_i + \frac{1}{4}\left(F_i + \alpha\ \frac{\partial F_i}{\partial x} + \mathcal{O}(\alpha^2) + \beta\ \frac{\partial F_i}{\partial u}F_i + \mathcal{O}(\beta^2)\right)\right) \\ & \downarrow & \\ U_{i+1} & = & U_i + h\ F_i + \frac{h\alpha}{4}\frac{\partial F_i}{\partial x} + + \frac{h\beta}{4}\ \frac{\partial F_i}{\partial u}F_i + \mathcal{O}(h\beta^2) + \mathcal{O}(h\alpha^2) \end{array}$$

Maintenant, écrivons la valeur que l'on obtiendrait avec une méthode de Taylor d'ordre deux :

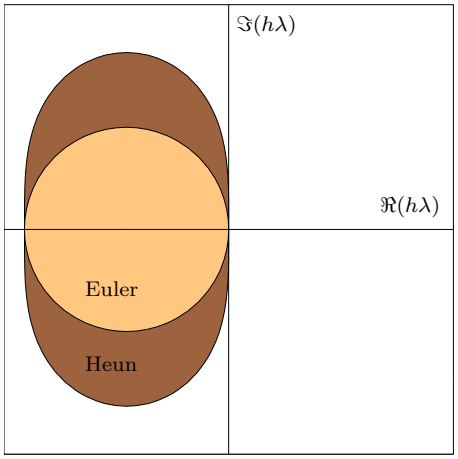
$$U_{i+1} = U_i + h\ F_i + \frac{h^2}{2}\ \frac{\partial F_i}{\partial x} + \frac{h^2}{2}\ \frac{\partial f}{\partial u}\ F_i + \mathcal{O}(h^3)$$

En identifiant terme à terme les deux développements, on obtient : $\boxed{\alpha = \beta = 2h}$

2. On évalue simplement U_{i+1} pour le problème modèle :

$$\begin{array}{rcl} K_1 & = & \lambda U_i \\ K_2 & = & \lambda\big(U_i + 2hK_1\big) = \lambda\big(U_i + 2h\lambda U_i\big) \\ & \downarrow & \\ U_{i+1} & = & U_i + \frac{h}{4}\Big(3K_1 + K_2\Big) = \Big(1 + h\lambda + \frac{h^2\lambda^2}{2}\Big)\ U_i \end{array}$$

La zone de stabilité de la méthode de Barack est donc : $\boxed{\left|1 + h\lambda + \frac{h^2\lambda^2}{2}\right| \leq 1}$



Il est essentiel d'indiquer correctement les axes !

La zone de stabilité de la méthode de Barack est la même que celle de la méthode de Heun qui coïncide avec celle de Taylor d'ordre deux... Au passage, on pouvait obtenir très aisément β en exigeant que ces zones de stabilité soit identique...

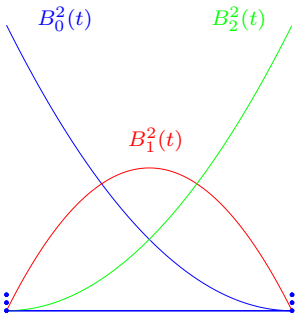
3. Une implémentation possible est donnée par :

```
function [X,U] =  barackIntegrate(n,h,U0,f)
  X = linspace(0,n*h,n+1);
  U = zeros(n+1,1);  U(1) = U0;
  for i=1:n
      K1 = f(X(i),U(i));
      K2 = f(X(i)+2*h,U(i)*2*h*K1);
      U(i+1) = U(i) + h * (3*K1 + K2)/4;
  end
end
```

Ce programme était vraiment simple à écrire.
Ne pas oublier de pré-allouer les deux vecteurs !
Il fallait impérativement une fonction f avec deux arguments
et ne pas bêtement recopier la solution d'un autre exercice du syllabus :-)

121 Janvier 2018 : Faut-il rapatrier Bézier ?

1. Fonctions de Bézier $B_i^2(t)$ définies pour les six noeuds $[T_0, T_1, T_2, T_3, T_4, T_5] = [0, 0, 0, 1, 1, 1]$.



Les trois fonctions sont :

$$\begin{array}{lcl} B_0^2(t) & = & (1-t)^2 \\ B_1^2(t) & = & 2(1-t)t \\ B_2^2(t) & = & t^2 \end{array}$$

Connaissant l'allure des deux fonctions extrêmes et le fait que la somme des trois fonctions doive toujours valoir l'unité, ce résultat peut être obtenu immédiatement sans faire aucun calcul.
Le maximum de la fonction centrale est 0.5 et celui des deux autres fonctions est 1.
Beaucoup d'esquisses étaient en contradiction flagrante avec cette observation !

2. Il faut minimiser la fonction suivante :

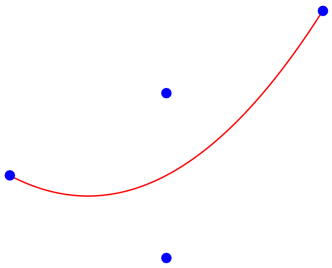
$$J(a_0, a_1, a_2) = \sum_{i=0}^{3} \left(U_i - \sum_{j=0}^{2} a_j \; B_j^2(T_i) \right)^2$$

Cette question est vraiment très simple.
Il faut l'exposant deux et les deux sommes écrites de manière adéquate. Ne considérer que les trois premiers points en effectuant donc une simple interpolation était impardonnable.

3. Comme $\{B_0^2(t), B_1^2(t), B_2^2(t)\}$ et $\{t^2, t, 1\}$ sont deux bases du même sous-espace vectoriel (l'ensemble des polynômes de degré inférieur ou égal à deux), les deux résultats seront identiques.

   En termes de coût calcul ou d'erreurs d'arrondi, il n'y a strictement aucune différence pertinente ! Certains étudiants ont été particulièrement imaginatifs et créatifs à ce sujet :-)
   Au passage, je signale aussi que les fonctions de Bézier ne sont pas orthogonales ! On peut éventuellement remarquer que les trois fonctions de Bézier s'annulent à une ou au deux extrémités de l'intervalle : ce qui facilite un tout petit peu le calcul qui suit :-) Par contre, le calcul des équations normales avec les monômes est peut-être un peu léger au niveau du calcul à la main.... mais, qui fait encore ce genre de calcul sans un ordinateur ?

4. Faire un petit dessin était une bonne idée pour cette question !



Il est très simple de remarquer
que comme deux des quatre données correspondent
à une même abscisse $\hat{T}_1 = \hat{T}_2$,
l'approximation demandée
correspond donc à l'interpolation par les points

$$\left[\hat{T}_0, \hat{U}_0\right], \qquad \left[\hat{T}_1, \frac{\hat{U}_1 + \hat{U}_2}{2}\right], \qquad \left[\hat{T}_3, \hat{U}_3\right].$$

Pour les données du tableau, on peut immédiatement écrire que le polynôme d'interpolation :

$$u^h(t) = 4t(t - 0.5)$$

puisque $u^h(0) = u^h(0.5) = 0$ et $u^h(1) = 1$. L'obtention des coefficients peut donc s'obtenir très facilement en identifiant terme à terme ce polynôme avec la combinaison des 3 polynômes de Bézier. Il est utile d'observer qu'il n'y a virtuellement aucun calcul à réaliser.

$$\begin{aligned} 4t^2 - 2t &= a_0(1 - 2t + t^2) + a_1(2t - 2t^2) + a_2 t^2 \\ 4t^2 - 2t &= (a_0 - 2a_1 + a_2)t^2 + (2a_1 - 2a_0)t + a_0 \end{aligned}$$

$$\downarrow$$

$$\begin{aligned} 4 &= a_0 - 2a_1 + a_2 \\ -2 &= 2a_1 - 2a_0 \\ 0 &= a_0 \end{aligned}$$

On conclut donc : $\boxed{\begin{array}{rcl} a_0 &=& 0 \\ a_1 &=& -1 \\ a_2 &=& 2 \end{array}}$

Evidemment, il est aussi possible de calculer le système de l'interpolation ou même les équations normales. De manière assez décevant, la plupart des étudiants choisissent cette alternative calculatoire. De manière assez surprenante, pas mal d'étudiants obtiennent toutefois la bonne solution des équations normales en procédant de la sorte... Pour être donc complet et exhaustif, les équations normales s'écrivent sour la forme suivante :

$$\begin{bmatrix} \sum_{i=0}^{3} B_0^2(\hat{T}_i)B_0^2(\hat{T}_i) & \sum_{i=0}^{3} B_0^2(\hat{T}_i)B_1^2(\hat{T}_i) & \sum_{i=0}^{3} B_0^2(\hat{T}_i)B_2^2(\hat{T}_i) \\ \sum_{i=0}^{3} B_1^2(\hat{T}_i B_0^2(\hat{T}_i) & \sum_{i=0}^{3} B_1^2(\hat{T}_i)B_1^2(\hat{T}_i) & \sum_{i=0}^{3} B_1^2(\hat{T}_i)B_2^2(\hat{T}_i) \\ \sum_{i=0}^{3} B_2^2(\hat{T}_i)B_0^2(\hat{T}_i) & \sum_{i=0}^{3} B_2^2(\hat{T}_i)B_1^2(\hat{T}_i) & \sum_{i=0}^{3} B_2^2(\hat{T}_i)B_2^2(\hat{T}_i) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{3} B_0^2(\hat{T}_i)\hat{U}_i \\ \sum_{i=0}^{3} B_1^2(\hat{T}_i)\hat{U}_i \\ \sum_{i=0}^{3} B_3^2(\hat{T}_i)\hat{U}_i \end{bmatrix}$$

$$\frac{1}{8}\begin{bmatrix} 9 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 9 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$$

Et on obtient ainsi les mêmes coefficients mais en passant vraiment nettement plus de temps en algèbre totalement stérile ! Comme la persévérance est une qualité, les étudiants qui ont choisi cette voie délicate n'ont toutefois pas été pénalisés.

Ecrire directement l'interpolation est une option vachement plus rapide. Ici, le système devient vraiment trivial a résoudre. Quelques étudiants mais trop peu nombreux choisissent cette option.

$$\begin{bmatrix} B_0^2(0) & B_1^2(0) & B_2^2(0) \\ B_0^2(1) & B_1^2(1) & B_2^2(1) \\ B_0^2(2) & B_1^2(2) & B_2^2(2) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \hat{U}_0 \\ (\hat{U}_1 + \hat{U}_2)/2 \\ \hat{U}_3 \end{bmatrix}$$

$$\frac{1}{4}\begin{bmatrix} 4 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$$

122 Janvier 2018 : Théo a expulsé la division !

1. Il faut évidemment tenir compte que vous ne disposez plus de la division !

Il suffit d'écrire : $\boxed{\begin{array}{rcl} x_{n+1} & = & x_n - \left(b - \dfrac{1}{x_n}\right) x_n^2 \\ & = & 2x_n - bx_n^2 \end{array}}$

On observe qu'il n'est pas nécessaire d'avoir la division pour effectuer cette itération ! Un nombre incalculable d'étudiants n'arrivent pas à obtenir ce résultat qui était clairement annoncé dans l'énoncé dans la question. Pour obtenir les points associés à cette question élémentaire, il fallait impérativement écrire l'itération sans faire apparaître une division.

2. L'ordre de convergence théorique et observé est deux.
   Asymptotiquement le nombre de chiffres significatifs double à chaque itération.
   On observe bien que $(10^{-5})^2 = 10^{-10}$.

Il suffit d'écrire : $\boxed{\text{ordre} \;=\; 2}$

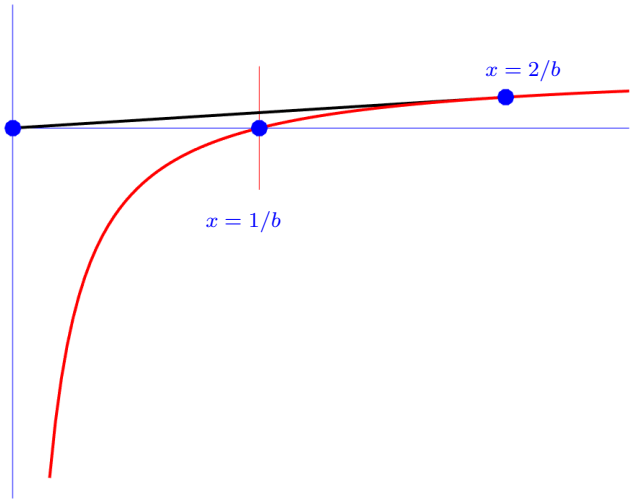
Il ne fallait pas faire des calculs détaillés, mais une très brève justification était requise ! C'est donc bien en accord avec la théorie de la méthode de Newton-Raphson !

L'estimation (non demandée évidemment :-) via MATLAB

```
[x error] = inverse(1.5,1e-13,15);
order = log(error(2:end-1))./log(error(1:end-2))
```

fournit les valeurs suivantes : `order = 2.1859 2.2649 2.1411 2.0667`

3. Faire un dessin était la clé du succès pour cette question : on observe immédiatement que la convergence sera assurée pour toutes les valeurs comprises entre 0 et $1/b$. De l'autre côté, il faut choisir un candidat initial tel que l'itération suivante produise une nouvelle estimation positive.



Cela peut s'exprimer sous la forme suivante :

$$\begin{aligned} 0 &< x_1 \\ 0 &< 2x_0 - bx_0^2 \\ 0 &< 2 - bx_0 \\ &\downarrow \\ x_0 &< \frac{2}{b} \end{aligned}$$

Il suffit d'écrire : $\boxed{\text{intervalle d'encadrement} \quad = \quad \left]0, \frac{2}{b}\right[}$

Très peu d'étudiants ont trouvé cette réponse qui est évidente lorsqu'on fait le dessin.
Beaucoup d'étudiants font beaucoup d'algèbre inutile avec la condition de Lipschitz.
Je savais que Théo pouvait être source de soucis, mais ici il n'a vraiment pas démérité :-)

4. Une implémentation possible est donnée par :

```
function [x error] = inverse(x0,tol,nmax)
  b = 0.5;
  n=1; delta = tol + 1; x(1) = x0;
  while abs(delta) >= tol && n < nmax
    delta = -0.5*x(n)*x(n) + x(n);
    error(n) = abs(delta);
    fprintf('\n x = %16.14f : Error %13.7e at %i iter. ',x(n),error(n),n);
    x(n+1) = x(n) + delta;
    n = n + 1;
  end
end
```

Dans ce programme, on ajouté l'estimation de l'erreur pour obtenir une estimation du taux de convergence, mais ce n'était évidemment ni demandé, ni requis. Par contre, écrire un programme avec des divisions est considéré comme une erreur impardonnable, en particulier si l'expression de la première sous-question n'était pas fournie. Toutefois, le correcteur a été parfois honteusement indulgent lorsque une expression correcte apparaissait dans le programme et pas dans la première sous-question pour créditer l'ensemble de points des deux sous-questions : l'inverse est parfois aussi appliqué en fonction des justifications et de l'allure générale de la réponse....

123 Janvier 2018 : Quadratures de Rajoy et Puigdemont

1. Il faut exiger que : $\left\{ \begin{array}{rcl} w_0 + w_1 + w_2 & = & 2 \\ -w_0\alpha + w_2\alpha & = & 0 \\ w_0\alpha^2 + w_2\alpha^2 & = & 2/3 \end{array} \right.$

La première équation est requise pour intégrer exactement un terme constant.
La seconde équation permet d'intégrer exactement x.
La dernière équation correspond à x^2.

$$\int_{-1}^{1} x^2 \, dx = \frac{2}{3} = w_0\alpha^2 + w_2\alpha^2 = 2w_0\alpha^2 \longrightarrow \boxed{w_0 = w_2 = \frac{1}{3\alpha^2} \qquad w_1 = 2 - \frac{2}{3\alpha^2}}$$

Notons que l'on intègre aussi exactement n'importe quel polynôme de degré 3 !

2. Il suffit tout simplement d'écrire :

$$\boxed{I_h = \frac{1}{3\alpha^2}\left(\frac{1}{2-\alpha}+\frac{1}{2+\alpha}\right)+\left(2-\frac{2}{3\alpha^2}\right)\frac{1}{2}}$$

3. L'erreur est minimale lorsqu'elle est nulle :-)
Il suffit donc juste de calculer l'intégrale exacte de g et d'imposer que $I = I_h$.
En d'autres mots, on recherche α tel que :

$$\int_{-1}^{1}\frac{1}{x+2}\,dx \;=\; \ln(3) \;=\; \frac{1}{3\alpha^2}\left(\frac{1}{2-\alpha}+\frac{1}{2+\alpha}\right)+1-\frac{1}{3\alpha^2}$$

$$\downarrow$$

$$\begin{aligned}
3\alpha^2(4-\alpha^2)\,\ln(3) &= 4+3\alpha^2(4-\alpha^2)-4+\alpha^2\\
3\alpha^2(4-\alpha^2)\,\ln(3) &= 3\alpha^2(4-\alpha^2)+\alpha^2\\
3(4-\alpha^2)\,\ln(3) &= 3(4-\alpha^2)+1\\
3(4-\alpha^2)\,\ln(3) &= 3(4-\alpha^2)+1\\
(12\ln(3)-13) &= (3\ln(3)-3)\,\alpha^2
\end{aligned}$$

On conclut finalement : $\boxed{\alpha = \sqrt{\frac{12\ln(3)-13}{3\ln(3)-3}}}$

Il n'y a vraiment strictement aucune raison pour que cette valeur corresponde à la valeur habituelle de Gauss-Legendre : ce que va proposer notre compère Puigdemont. On impose ici de pouvoir intégrer g et non plus un quelconque polynôme... Observons que la valeur trouvée tombe dans l'intervalle $[0,1]$: ce qui n'était pas évident a priori !

4. C"est exactement le même principe que pour la première question.

$$\int_{-1}^{1}x^4\,dx = \frac{2}{5} = w_0\alpha^4+w_2\alpha^4 = \frac{2}{3\alpha^2}\alpha^4 \quad\longrightarrow\quad \boxed{\alpha=\sqrt{\frac{3}{5}}}$$

Notons à nouveau que l'on intègre aussi exactement n'importe quel polynôme de degré 5 ! Il n'y a vraiment strictement aucune raison pour que cette valeur corresponde à la valeur de Rajoy. Rajoy impose de pouvoir intégrer g tandis que Puigdemont veut pouvoir intégrer un quelconque polynôme... Notons au passage que la méthode de Puigdemont correspond à une méthode de Gauss-Legendre habituelle : il n'a donc rien inventé de neuf. Sans faire beaucoup d'algèbre, on pouvait prévoir que les deux protagonistes ne seraient pas d'accord.

5. Pour la quadrature composite de Gauss-Legendre à trois points :

$$\boxed{m = 6}$$

Comme on intègre exactement un polynôme de degré cinq, on doit intégrer, sur chaque intervalle, une erreur $u - u^h$ qui est un polynôme de degré six: cela génère une erreur locale sur chaque intervalle en $\mathcal{O}(h^7)$ et une erreur globale $\mathcal{O}(h^6)$ en additionnant les $n = 2/h$ erreurs locales !

6. Comme il s'agit d'éliminer un terme d'ordre 6 :

$$\boxed{\frac{\left(64 I_{h/2} - I_h\right)}{63}}$$

L'erreur de cette extrapolation sera en $\mathcal{O}(h^8)$, puisqu'il n'y a pas de termes d'erreur de degré impair. La règle de Gauss-Legendre est, en effet, une formule parfaitement symétrique.

124 **Janvier 2019 : C'est fini, les différences pour Charles...**

1. Il suffit d'écrire les développements en série de Taylor :

$$\begin{aligned}
U_{-h} &= U_0 - hU_0' + \frac{h^2}{2}U_0'' - \frac{h^3}{6}U_0''' + \frac{h^4}{24}U_0'''' \dots \\
U_{-2h} &= U_0 - 2hU_0' + \frac{4h^2}{2}U_0'' - \frac{8h^3}{6}U_0''' + \frac{16h^4}{24}U_0'''' \dots \\
U_{-3h} &= U_0 - 3hU_0' + \frac{9h^2}{2}U_0'' - \frac{27h^3}{6}U_0''' + \frac{81h^4}{24}U_0'''' \dots
\end{aligned}$$

Et ensuite, on incorpore ces jolis développements dans la formule de Charles :

$$\frac{(2-5+\alpha+\beta)}{\gamma h^2}U_0 - \frac{(-5+2\alpha+3\beta)}{\gamma h}U_0' + \frac{(-5+4\alpha+9\beta)}{2\gamma}U_0'' - \frac{(-5+8\alpha+27\beta)}{6\gamma}hU_0''' + \frac{(-5+16\alpha+81\beta)}{24\gamma}h^2U_0''''$$

Il faut annuler les coefficients de U_0 et de U_0' et exiger que le coefficient de U_0'' soit unitaire.
Il y a trois conditions et trois inconnues !

Il faut donc imposer les conditions suivantes $\left\{\begin{array}{rcl} -3+\alpha+\beta &=& 0 \\ -5+2\alpha+3\beta &=& 0 \\ -5+4\alpha+9\beta &=& 2\gamma \end{array}\right.$

... on en déduit immédiatement que : $\boxed{\begin{array}{rcl} \alpha &=& 4 \\ \beta &=& -1 \\ \gamma &=& 1 \end{array}}$

Cette question peut être très largement qualifiée de simple et largement prévisible...
il est donc vraiment impardonnable d'échouer :-)

2. Ensuite, on substitue α et β dans la formule de Charles pour obtenir le terme d'erreur :-)

$$\underbrace{\frac{(2-5+4-1)}{h^2}}_{=0}U_0 - \underbrace{\frac{(-5+8-3)}{h}}_{=0}U_0' + \underbrace{\frac{(-5+16-9)}{2}}_{=1}U_0'' - \underbrace{\frac{(-5+32-27)}{6}}_{=0}hU_0''' + \underbrace{\frac{(-5+64-81)}{24}}_{=11/12}h^2U_0''''$$

On conclut finalement : $\boxed{u''(0) = \frac{2U_0 - 5U_{-h} + 4U_{-2h} - U_{-3h}}{h^2} - \frac{11}{12}h^2 u^{(4)}(\zeta)}$

La méthode de Charles est donc d'ordre deux et correspond à une différence amont usuelle :-)

3. Il suffit de minimiser $E(h) = \frac{(2+5+4+1)}{h^2}\epsilon + \frac{11}{12}h^2 10^4$.

Attention : les erreurs ne font que s'additionner. Donc on prend la valeur absolue de tous les coefficients et la somme de l'erreur d'arrondi et de l'erreur de discrétisation même si cette dernière est négative !

$$\text{On exige donc que } f'(h) = -\frac{24}{h^3}\epsilon + \frac{22h}{12}\,10^4 = 0$$
$$\downarrow$$
$$10^4\, h^4 = \frac{144}{11}\,\epsilon$$

On conclut que le pas optimal est donné par : $\boxed{h = \sqrt[4]{\frac{144}{110000}\,\epsilon}}$

125 Janvier 2019 : Newton et Raphson ont-ils des gilets jaunes ?

1. Il suffit juste de dire qu'il y a une infinité de solutions !
Pour un $x \in [0,1]$ quelconque, il suffit de choisir $z = -x$ et $y = \pm\sqrt{1-x^2}$.
Il y a aussi une solution particulièrement simple[8] : $x = z = 1$ et $y = 0$.

2. Le système à résoudre est :

$$\boxed{\begin{bmatrix} 2x_i & 2y_i & 0 \\ y_i & x_i + z_i & y_i \\ 0 & 2y_i & 2z_i \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = -\begin{bmatrix} x_i^2 + y_i^2 - 1 \\ y_i(x_i + z_i) \\ z_i^2 + y_i^2 - 1 \end{bmatrix}}$$

3. En prenant $x_0 = z_0 = 4$ et $y_0 = 0$,
on obtient un système linéaire diagonal qui se résout immédiatement...

[8] Et c'est avec cette stupide solution que toute la question a été construite ! Donc, c'était une bonne idée d'essayer de la trouver a priori :-)

$$\begin{bmatrix} 8 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 8 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = - \begin{bmatrix} 15 \\ 0 \\ 15 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} -15/8 \\ 0 \\ -15/8 \end{bmatrix}$$

Finalement on obtient : $\boxed{\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 4 \end{bmatrix} + \begin{bmatrix} -15/8 \\ 0 \\ -15/8 \end{bmatrix} = \begin{bmatrix} 17/8 \\ 0 \\ 17/8 \end{bmatrix}}$

4. C'est une conséquence immédiate du caractère diagonal du système linéaire de Newton-Raphson. Il suffit de refaire ce qui a été obtenu à la question précédente en introduisant un paramètre α_i.

$$\begin{bmatrix} 2\alpha_i & 0 & 0 \\ 0 & 2\alpha_i & 0 \\ 0 & 0 & 2\alpha_i \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = - \begin{bmatrix} \alpha_i^2 - 1 \\ 0 \\ \alpha_i^2 - 1 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \end{bmatrix} = \begin{bmatrix} \alpha_i \\ 0 \\ \alpha_i \end{bmatrix} - \frac{1}{2\alpha_i} \begin{bmatrix} \alpha_i^2 - 1 \\ 0 \\ \alpha_i^2 - 1 \end{bmatrix}$$

On conclut la démonstration en donnant la fonction de récurrence : $\boxed{\alpha_{i+1} = \underbrace{\frac{(\alpha_i^2 + 1)}{2\alpha_i}}_{g(\alpha_i)}}$

□

5. Tout d'abord, il est utile de noter qu'on peut se restreindre pour les valeurs positives de α sans aucune perte de généralité, car g est une fonction impaire : tout ce qui se passe pour les valeurs négatives est le parfait miroir de ce qui se passe pour les valeurs positives. Ensuite, il est judicieux de commencer par obtenir les points fixes de notre fonction de récurrence :

$$\alpha = \frac{(\alpha^2 + 1)}{2\alpha}$$

$$\downarrow$$

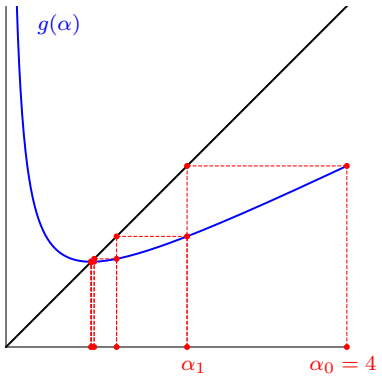
$$2\alpha^2 = \alpha^2 + 1$$

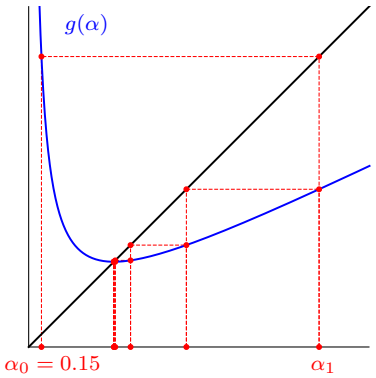
$$\alpha = \pm 1$$

Pour analyser la convergence d'une méthode itérative, on peut vérifier le caractère lipschistzien de la fonction g en calculant la dérivée de cette fonction :

$$g'(\alpha) = \frac{1}{2}\left(1 - \frac{1}{\alpha_i^2}\right)$$

Lorsque $\alpha > 1$, la valeur de la dérivée de g est incluse dans l'intervalle $[0, 0.5]$.
La fonction est donc lipschistzienne pour $\alpha > 1$.
En conséquence, la méthode itérative convergera vers le point fixe $\alpha = 1$, pour tout $\alpha_0 > 1$.
Réaliser un petit croquis de problème permettait de mieux comprendre la situation !





En effet, analyser ce qui se passe pour un candidat initial $\alpha_0 \in [0, 1]$ est un peu plus subtil. Dans ce cas, on observe que $g(\alpha) > \alpha$ et donc que la première itération va systématiquement envoyer la méthode itérative dans le domaine de convergence... Il n'y a donc jamais de divergence à l'exception du cas pathologique $\alpha_0 = 0$.

On peut donc conclure :

> La méthode convergera toujours sauf pour $\alpha_0 = 0$
>
> Avec $\alpha_0 = 4$, la méthode converge vers $\alpha = 1$

C'est une question difficile qui nécessitait de prendre du recul par rapport à l'algèbre et d'éventuellement réaliser un petit schéma pour effectuer la prédiction.... Citer et utiliser la condition de Lipschitz pemettait d'obtenir une bonne partie des points prévus pour cette question. Il était aussi utile de découvrir le point fixe avant de pouvoir terminer l'analyse du domaine de convergence de la méthode du point fixe :-)

126 Janvier 2019 : Une économie de décroissance... pour Nancy et Donald

1. Il suffit tout simplement d'écrire :

$$\begin{aligned} \frac{U_{n+1} - U_n}{h} &= -U_n\ (1 + V_n) \\ \frac{V_{n+1} - V_n}{h} &= -\frac{V_n}{U_n} \end{aligned}$$

Oui, c'était une sous-question vraiment facile et adaptée aux capacités de Donald :-)

2. Cela doit devenir évidemment un peu plus compliqué :-)

   Tout d'abord, il faut observer qu'il faut d'abord évaluer V_{n+1} qui ne dépend que de U_n et V_n avant d'évaluer U_{n+1} qui requiert V_{n+1}. En manipulant un peu les deux relations de récurrence de Nancy, on observe qu'on a bien une méthode semi-implicite puisqu'on peut en fait l'évaluer de manière purement explicite en enchaînant de manière adéquate les opérations.

   Plus précisément, il faut écrire :

$$\begin{array}{rcl} \frac{U_{n+1}-U_n}{h} & = & -U_n\ (1+V_{n+1}) \\ U_{n+1}-U_n & = & -U_n\ (h+hV_{n+1}) \\ U_{n+1} & = & U_n - U_n\ (h+hV_{n+1}) \\ & \downarrow & \\ U_{n+1} & = & U_n\ (1-h-hV_{n+1}) \end{array} \qquad \begin{array}{rcl} \frac{V_{n+1}-V_n}{h} & = & -\frac{V_{n+1}}{U_n} \\ V_{n+1}-V_n & = & -h\frac{V_{n+1}}{U_n} \\ V_{n+1}\left(1-h\frac{1}{U_n}\right) & = & V_n \\ & \downarrow & \\ V_{n+1} & = & V_n\left(\frac{U_n+h}{U_n}\right) \end{array}$$

   Ecrire ensuite le programme est immédiat.

```
from numpy import *

def integrateNancy(n,h) :
  U = zeros((n+1,2))
  U[0,:] = [1,1]
  for i in range(n):
     U[i+1,1] = U[i,1] * U[i,0] / (h + U[i,0])
     U[i+1,0] = U[i,0] * (1 - h - h * U[i,1])
  return U
```

   Il n'y avait strictement aucune astuce de programmation à inclure !

3. Pour savoir si la méthode sera stable, il faut analyser la valeur absolue des deux facteurs d'amplification U_* et V_* pour U_n et V_n respectivement.

$$U_{n+1} = U_n \underbrace{(1-h-hV_{n+1})}_{U_*} \qquad V_{n+1} = V_n \underbrace{\left(\frac{U_n+h}{U_n}\right)}_{V_*}$$

   Le comportement discret restera cohérent tant que $0 < U_* < 1$ et $0 < V_* < 1$. L'apparition d'un facteur négatif va générer des oscillations qui sont à proscrire a priori. Comme $0 < u(x) < 1$ et $0 < v(x) < 1$ pour tout $x > 0$, on observe que la condition sur V_* est toujours satisfaite. Par contre, pour U_*, il existe un pas de temps critique en dessous duquel le comportement du schéma deviendra incohérent :

$$
\begin{aligned}
0 &< 1 - h - hV_{n+1} \\
&\downarrow \quad \text{En considérant le cas le plus défavorable avec } V_{n+1} = 1, \\
0 &< 1 - 2h \\
h &< \tfrac{1}{2}
\end{aligned}
$$

On peut donc conclure :

> La méthode de Nancy sera stable lorsque $h < \frac{1}{2}$
> Le point le plus critique se situe en $x = 0$.

4. On analyse à nouveau la valeur absolue des deux facteurs d'amplification U_* et V_*.

$$
U_{n+1} = U_n \underbrace{(1 - h - hV_n)}_{U_*} \qquad V_{n+1} = V_n \underbrace{\left(1 - \frac{h}{U_n}\right)}_{V_*}
$$

Pour U_*, nous avons exactement la même condition que pour la méthode de Nancy : $h < \frac{1}{2}$. Par contre pour V_*, une nouvelle contrainte est apparue lorsque la valeur de U_n devient inférieure à h : à ce moment, la solution deviendra oscillante et puis un peu plus tard va exploser.

$$
\begin{aligned}
0 &< 1 - \frac{h}{U_n} \\
&\downarrow \quad \text{En considérant le cas le plus défavorable avec } U_{n+1} = 0.0035, \\
0 &< 1 - \frac{h}{0.0035} \\
h &< 0.0035
\end{aligned}
$$

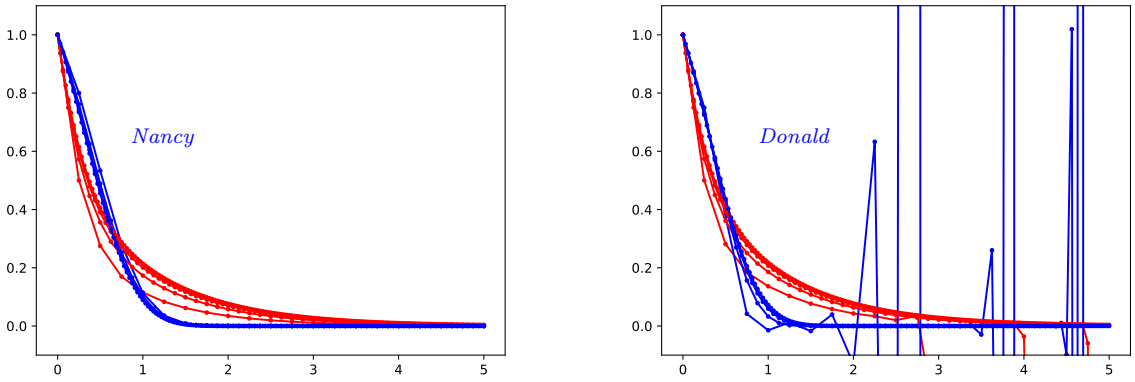
On peut donc conclure :

> La méthode de Donald sera stable lorsque $h < 0.0035$
> Le point le plus critique se situe en $x = 5$.

Calculer le pas de temps critique en imposant $-1 < U_* < 1$ et $-1 < V_* < 1$ est admis. Par exemple, on accepterait alors un pas $h = 0.007$ pour la méthode explicite de Donald en tolérant une solution oscillante mais dont l'amplitude n'augmente pas :-) Notons toutefois qu'utiliser un pas de temps $h = 1$ pour la méthode de Nancy va toutefois poser un vrai problème car la présence de valeurs négatives va introduire une instabilité pour V_i qui explosera dans ce cas.

Quand l'implémentation semble en contradiction avec la théorie....

L'implémentation avec $h = \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$ *et* $\frac{1}{32}$ *fournit les résultats suivants.*



Ici, on pourrait être vachement étonné que la méthode explicite de Donald semble fonctionner parfaitement sur l'intervalle $[0, 5]$ *avec* $h = 0.03125$ *alors que notre calcul avait prédit une instabilité.*

En effet, on voit sur la figure que la méthode de Donald explose successivement pour $h = 0.25, 0.125, 0.0625...$ *Par contre, le calcul semble parfaitement cohérent pour la plus petite valeur de* $h = 0.03125$ *sélectionnée !*

En réalité, l'instabilité est bien apparue, mais n'a pas encore eu le temps d'être visible sur la figure.

Pour bien comprendre le processus, nous avons légèrement instrumenté notre programme et nous imprimons les données pour chaque pas itération avec $h = 0.03125$ *largement supérieur (d'un facteur dix !) au pas de temps critique* $h = 0.0035$ *que nous venons d'estimer!*

```
nSet = [20,40,80,160]
for n in nSet:
  X = linspace(Xstart,Xend,n+1)
  U = zeros((n+1,2)); U[0,:] = Ustart
  h = (Xend - Xstart)/n
  print("Step : %10.8f " % h)
  for i in range(n):
     U[i+1,1] = U[i,1] - h * U[i,1] / U[i,0]
     U[i+1,0] = U[i,0] - h * U[i,0] * (1 + U[i,1])
     print("  == %4d : %14.7e %14.7e " % (i,U[i,1],(1 - h/U[i,0])))

  plt.plot(X, U[:,0],'.-r',X, U[:,1], '.-b',markersize='5.0')
  plt.ylim(-0.1,1.1)
```

En particulier, nous imprimons à chaque pas de temps : X_n, V_n *et* $(1 - h/U_n)$. *On observe bien avec précision le comportement suivant :*

```
==   90 :  2.81e+00  1.03e-17  8.24e-02
==   91 :  2.84e+00  8.52e-19  5.28e-02
==   92 :  2.88e+00  4.50e-20  2.22e-02
==   93 :  2.91e+00  1.00e-21 -9.30e-03
==   94 :  2.94e+00 -9.31e-24 -4.19e-02
==   95 :  2.97e+00  3.90e-25 -7.55e-02
==   96 :  3.00e+00 -2.94e-26 -1.10e-01
==   97 :  3.03e+00  3.24e-27 -1.46e-01
==   98 :  3.06e+00 -4.73e-28 -1.83e-01
```

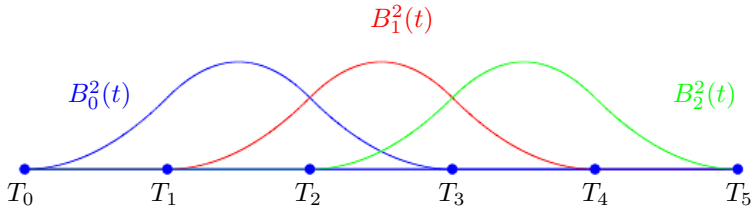
```
==  112 :  3.50e+00 -3.01e-33 -8.45e-01
==  113 :  3.53e+00  2.54e-33 -9.05e-01
==  114 :  3.56e+00 -2.30e-33 -9.66e-01
==  115 :  3.59e+00  2.22e-33 -1.03e+00
==  116 :  3.62e+00 -2.29e-33 -1.09e+00
==  117 :  3.66e+00  2.50e-33 -1.16e+00
==  118 :  3.69e+00 -2.91e-33 -1.23e+00
==  119 :  3.72e+00  3.59e-33 -1.30e+00
==  120 :  3.75e+00 -4.68e-33 -1.38e+00
==  121 :  3.78e+00  6.45e-33 -1.46e+00
==  122 :  3.81e+00 -9.38e-33 -1.53e+00
==  123 :  3.84e+00  1.44e-32 -1.62e+00
==  124 :  3.88e+00 -2.33e-32 -1.70e+00
==  125 :  3.91e+00  3.96e-32 -1.79e+00
==  126 :  3.94e+00 -7.07e-32 -1.88e+00
==  127 :  3.97e+00  1.33e-31 -1.97e+00
==  128 :  4.00e+00 -2.62e-31 -2.07e+00
==  129 :  4.03e+00  5.41e-31 -2.17e+00

==  156 :  4.88e+00 -1.62e-15 -6.46e+00
==  157 :  4.91e+00  1.05e-14 -6.70e+00
==  158 :  4.94e+00 -7.01e-14 -6.95e+00
==  159 :  4.97e+00  4.87e-13 -7.20e+00
```

Au 93ème pas, le facteur d'amplification devient négatif et la valeur de V_n oscille mais continue à décroître. Au 115ème pas, la valeur absolue du facteur d'amplification devient supérieur à l'unité et la valeur de V_n continue à osciller mais se met à grandir en amplitude. Toutefois, elle est tellement faible (10^{-33} !) que ses oscillations sont de l'ordre du bruit numérique des erreurs d'arrondis. Au 159ème pas, l'instabilité a réellement explosé puisque l'amplitude de la solution (10^{-13} !) a grandi d'un facteur 10^{20} même si elle n'est toujours pas visible à l'échelle de la figure.

Evidemment, ceci n'était pas demandé :-)
Pour être vraiment très honnête, je trouvais très surprenant que mon implémentation numérique ne correspondait à ma prédiction théorique et il m'a fallu un peu de temps pour le comprendre !

127 **Juin 2019 : Bart exige d'avoir des B-splines !**

1. Les fonctions $B_0^2(t)$, $B_1^2(t)$ et $B_2^2(t)$ sur l'intervalle $t \in [T_0, T_5] = [-2, 3]$ sont :



   Il ne faut pas un dessin précis, mais il est requis que les 3 fonctions soient plus ou moins semblables et de classe C_1 (pas de point anguleux !). En outre, la fonction $B_0^2(t)$ est définie sur l'intervalle $[T_0, T_3]$ (ce qui n'est pas l'intervalle $[0, 3]$: idem pour les deux autres fonctions !)
   Pas mal d'étudiants échouent dans cette question vraiment très facile.
   Et pourtant, vous étiez si déçus de ne pas avoir de B-splines dans l'interrogation :-)
   Et pourtant, il était permis d'avoir les dessins des fonctions B-splines sur votre formulaire !

2. Comme sur l'intervalle $[0, 1]$, seules $B_1^1(t)$ et $B_2^1(t)$ sont non-nulles, on doit juste évaluer :

$$\begin{array}{rcl} 2B_0^2(t) &=& (1-t)B_1^1(t) \\ 2B_1^2(t) &=& (t+1)B_1^1(t) + (2-t)B_2^1(t) \\ 2B_2^2(t) &=& tB_2^1(t) \end{array} \quad = \quad \begin{array}{r} (1-t)(1-t) \\ (t+1)(1-t)+(2-t)t \\ t^2 \end{array} \quad = \quad \begin{array}{l} 1-2t+t^2 \\ 1+2t-2t^2 \\ t^2 \end{array}$$

On conclut que : $\boxed{B_0^2(t) = \frac{1-2t+t^2}{2} \qquad B_1^2(t) = \frac{1+2t-2t^2}{2} \qquad B_2^2(t) = \frac{t^2}{2}}$

On peut vérifier le résultat en observant que la somme des trois fonctions vaut un.
On peut aussi noter qu'une bonne partie de la réponse se trouvait dans l'énoncé, puisque :

$$u(t) = \frac{3\alpha B_1^2(t)}{B_0^2(t) + \alpha B_1^2(t) + B_2^2(t)} = \frac{3\alpha(1+2t-2t^2)}{(\alpha+1)+2(\alpha-1)(t-t^2)},$$

3. On doit rechercher t_* et α tels que :

$$\begin{cases} u(t_*) &=& \frac{5}{2}, \\ u'(t_*) &=& 0. \end{cases}$$

Sur base de la symétrie des poids et points de contrôle, on peut immédiatement observer que le maximum sera toujours atteint en $t_* = \frac{1}{2}$. Ensuite, on calcule α en écrivant $u(\frac{1}{2}) = \frac{5}{2}$:

$$\begin{array}{rcl} \frac{3\alpha(1+1-\frac{1}{2})}{(\alpha+1)+2(\alpha-1)(\frac{1}{2}-\frac{1}{4})} &=& \frac{5}{2} \\ \downarrow & & \\ \frac{9\alpha}{2} &=& \frac{5\alpha}{2}+\frac{5}{2}+\frac{5\alpha}{4}-\frac{5}{4} \\ \frac{3\alpha}{4} &=& \frac{5}{4} \end{array}$$

On conclut donc que : $\boxed{\alpha = \frac{5}{3}}$

Sans intuition géométrique, il fallait obtenir la valeur de t^* annulant la dérivée première :

$$0 = u'(t^*) = \frac{3\alpha(2-4t)\big[(\alpha+1)+2(\alpha-1)(t-t^2)\big] - 3\alpha(1+2t-2t^2)2\alpha(1-2t)}{9\alpha^2(1+2t-2t^2)^2}$$

$\downarrow$ En observant que $(1-2t_*+2t_*^2) > 0$,

$$0 = 6\alpha(1-2t)\big[\alpha(1+2t-2t^2)+(1-2t+2t^2)\big] - 6\alpha^2(1+2t-2t^2)(1-2t)$$

$$0 = (1-2t_*)(1-2t_*+2t_*^2)$$

$\downarrow$ En observant à nouveau que $(1-2t_*+2t_*^2) > 0$,

$$t_* = \frac{1}{2}$$

Une autre manière astucieuse d'obtenir que $t_* = \frac{1}{2}$ consistait à observer que :

$$\begin{aligned} u(t) &= \frac{3\alpha B_1^2(t)}{B_0^2(t) + \alpha B_1^2(t) + B_2^2(t)} \\ &= \frac{3\alpha B_1^2(t)}{B_0^2(t) + B_1^2(t) + B_2^2(t) + (\alpha - 1) B_1^2(t)} \\ &= \frac{3\alpha B_1^2(t)}{1 + (\alpha - 1) B_1^2(t)} \\ &= \frac{3\alpha}{1/B_1^2(t) + (\alpha - 1)} \end{aligned}$$

On observe alors immédiatement que le maximum de $u(t)$ est obtenu lorsque la fonction $B_1^2(t)$ atteint son maximum en $t_* = \frac{1}{2}$.
En effet, le dénominateur de la grande (!) fraction devient alors minimal :-)
Bravo aux étudiants astucieux m'ont fait découvrir cette très jolie résolution :-)
Bon, je reconnais, c'est un tout petit peu calculatoire, mais pas totalement infaisable...
Toutefois, pas mal d'étudiants ont l'intuition géométrique et évitent le calcul de dérivée.
Obtenir la valeur numérique correcte pour α et t_* était requis pour valider la sous-question !
Non, non il ne faut pas juste se contenter d'expliquer comment il faut procéder !

128 Juin 2019 : Tom et Théo découvrent Newton-Raphson

1. Il s'agit de minimiser la fonction suivante :

$$f(a) = \sum_{i=0}^{n} \left(U_i - \frac{1}{(X_i + a)} \right)^2$$

L'équation que doit satisfaire a s'écrit donc :

$$0 = f'(a) = \sum_{i=0}^{n} 2 \left(U_i - \frac{1}{(X_i + a)} \right) \frac{1}{(X_i + a)^2}$$

On conclut donc : $\boxed{\displaystyle\sum_{i=1}^{n} \frac{U_i}{(X_i + a)^2} = \sum_{i=1}^{n} \frac{1}{(X_i + a)^3}}$

Attention : fournir $f(a)$ à la place de $f'(a)$ est impardonnable !
Au passage, cette question est une variation simplifiée de l'exemple des notes de cours et d'un exercice des séances : ce n'était donc vraiment pas compliqué : si, si !

2. Le schéma de Newton-Raphson s'écrit :

On calcule a_{k+1} à partir de a_k avec

$$\begin{aligned} f''(a_k)\, \Delta a &= -f'(a_k) \\ a_{k+1} &= a_k + \Delta a \end{aligned}$$

avec les dérivées données par les expressions suivantes :

$$f'(a) = 2\sum_{i=1}^{n}\frac{U_i}{(X_i+a)^2} - 2\sum_{i=1}^{n}\frac{1}{(X_i+a)^3}$$

$$f''(a) = -4\sum_{i=1}^{n}\frac{U_i}{(X_i+a)^3} + 6\sum_{i=1}^{n}\frac{1}{(X_i+a)^4}$$

Il est évidemment possible de simplifier toutes ces expressions par un facteur 2.
Par contre, on peut pas éliminer un facteur $(X_i-a)^2$ partout. Même si c'est une monstruosité que pas mal d'étudiants font ! En développant les expressions, on peut bien observer que c'est une horreur totale de faire cela !
Ah oui : il ne faut évidemment pas appliquer Newton-Raphson pour x :-(
Beaucoup de réponses très surréalistes mais pas toujours très créatives :-)

3. En remplacant $a = 1$ dans l'équation obtenue au second point, on constate que c'est une solution du problème !

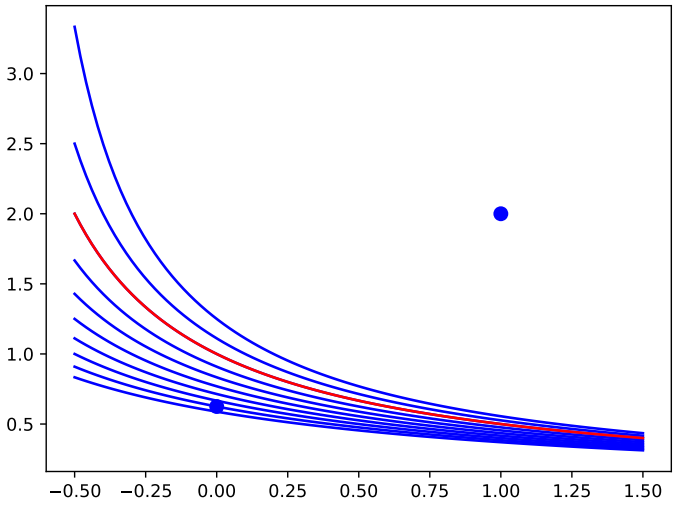
$$\sum_{i=0}^{1}\frac{U_i}{(X_i+a)^2} = \frac{5}{8}+\frac{2}{4} = \frac{5+4}{8} = \frac{9}{8}$$

$$\sum_{0=1}^{1}\frac{1}{(X_i+a)^3} = 1+\frac{1}{8} = \frac{9}{8}$$

Comme la dérivée seconde de f est non-nulle en ce point, le schéma de Newton-Raphson fournira directement comme résultat final $a_1 = a_0 = 1$ avec un diagnostic de convergence.

On conclut tout simplement : $\boxed{a = 1}$

L'énoncé de la question était un indice dont on pouvait ici tirer profit !
Imaginer que cet enseignant sadique vous fasse calculer des tas d'itérations et tout cela sans calculatrice : ce serait vraiment inacceptable pour Tom et Théo !
Représenter intuitivement les données peut permettre d'imaginer aisément une valeur probable de a et de mieux cerner le problème ! Il faut toutefois faire cela avec un minimum de prudence. Si on trace une série d'approximations possibles avec $a = 0.8$ jusqu'à $a = 1.7$ avec un incrément de 0.1, on observe qu'imposer le passage par le second point est nettement plus délicat que par le premier point.



Croire que la solution serait une courbe dont l'erreur en $x = 0$ et $x = 1$ est identique, n'est pas du tout correct ! Il faut donc parfois se méfier de certaines intuitions qui semblent trop évidentes !

129 Juin 2019 : Maggie et les différences finies compactes

1. Comme la formule est parfaitement symétrique, il n'y aura que des termes d'erreurs pairs. En choisissant de manière astucieuse α, β et γ, on éliminera les termes en $\mathcal{O}(h^0)$, $\mathcal{O}(h^2)$ et $\mathcal{O}(h^4)$.

   Donc, on peut espérer -sauf miracle numérique- que : $\boxed{p = 6}$

2. En ne gardant que les termes utiles, on écrit les développements en série de Taylor :

$$\begin{aligned}
U_{i\pm1} &= U_i \pm hU_i' + \ldots \pm \frac{h^3}{6}U_i^{(3)} + \ldots \pm \frac{h^5}{120}U_i^{(5)} + \ldots \pm \mathcal{O}(h^7) \\
U_{i\pm2} &= U_i \pm 2hU_i' + \ldots \pm \frac{8h^3}{6}U_i^{(3)} + \ldots \pm \frac{32h^5}{120}U_i^{(5)} + \ldots \pm \mathcal{O}(h^7) \\
U_{i\pm1}' &= U_i' \pm \ldots + \frac{h^2}{2}U_i^{(3)} \pm \ldots + \frac{h^4}{24}U_i^{(5)} \pm \ldots + \mathcal{O}(h^6)
\end{aligned}$$

   Ensuite, on substitue ces développements dans la formule de Christian :

$$\begin{aligned}
(1+2\alpha)U_i' + 2\alpha\frac{h^2}{2}U_i^{(3)} + 2\alpha\frac{h^4}{24}U_i^{(5)} &= \frac{2\beta}{2h}\left(hU_i' + \frac{h^3}{6}U_i^{(3)} + \frac{h^5}{120}U_i^{(5)}\right) + \\
&\quad + \frac{2\gamma}{4h}\left(2hU_i' + \frac{8h^3}{6}U_i^{(3)} + \frac{32h^5}{120}U_i^{(5)}\right) + \mathcal{O}(h^6)
\end{aligned}$$

   En identifiant les termes en U_i', $U_i^{(3)}$ et $U_i^{(5)}$, on obtient finalement :

$$\boxed{\begin{aligned} 2\alpha + 1 &= \beta + \gamma \\ 6\alpha &= \beta + 4\gamma \\ 10\alpha &= \beta + 16\gamma \end{aligned}}$$

   On obtient bien la précision pressentie, car le terme d'ordre six ne s'annule pas :-)

3. Il faut juste résoudre le système des 3 équations...

   La solution unique est : $\boxed{\alpha = \frac{1}{3}, \quad \beta = \frac{14}{9}, \quad \gamma = \frac{1}{9}}$

   Comme α était fourni dans l'énoncé, on pouvait obtenir β et γ à partir de deux équations. Inclure les trois valeurs dans la troisième équation permettait de vérifier si aucune erreur malencontreuse d'algèbre ne s'était pas glissée dans le calcul.

4. Une implémentation possible est :

```
from numpy import *
from scipy.sparse import dok_matrix
from scipy.sparse.linalg import spsolve
```

```
def compactDerivative(U,alpha,beta,gamma,h):

  n = len(U)
  A = dok_matrix((n,n),dtype=float32)
  B = zeros(n)
  for i in range(1,n-1) :
    A[i,[i-1,i,i+1]] = [alpha,1.0,alpha]
  A[0,[n-2,0,1]] = [alpha,1.0,alpha]
  A[n-1,[n-2,n-1,1]] = [alpha,1.0,alpha]

  U = array([U[n-3],U[n-2],*U,U[1],U[2]])
  B = (beta*(U[3:n+3]-U[1:n+1])/(2*h) +
       gamma*(U[4:n+4]-U[0:n])/(4*h))
  return spsolve(A.tocsr(),B)
```

Il faut résoudre un système linéaire creux pour obtenir la solution.
Ne pas l'observer fait perdre la totalité des points pour le programme.

Evidemment, il est essentiel d'utiliser une matrice creuse.
Quelques étudiants construisent deux autres matrices pour calculer le membre de droite : ce n'est pas formellement une erreur, mais c'est inutilement compliqué. Il n'était pas nécessaire d'avoir la syntaxe correcte pour les `import`, ainsi que les fonctions de librairie `sparse` pour obtenir la totalité des points !

Il faut évidemment inclure les conditions de périodicité pour la matrice et le membre de droite. Notons que la valeur U_{-1} qui précède U_0 n'est pas U_n, mais U_{n-1}. La plupart des étudiants n'observent pas cette toute petite astuce, mais le correcteur ne leur en a pas tenu rigueur. Les plus astucieux observeront que le vecteur `U` doit avoir une taille minimale et être un vecteur colonne pour que le code fonctionne...

Attention, la variable `n` du code correspond à la taille du vecteur `U` et donc à la valeur $n+1$ de l'énoncé :-) Il serait d'ailleurs possible et plus judicieux de ne pas inclure U_n comme inconnue et d'éviter de dédoubler stupidement une équation :-) On aurait alors bien un vecteur de taille n, et il faudrait alors répéter le premier élément du vecteur obtenu pour construire le vecteur requis par l'énoncé : réaliser cette petite variante du programme est laissé à votre sagacité :-)