Rendu 1, L'approche différences finies

Zouiche Omar et David Czarnecki

31 janvier 2021

1 Introduction

1.1 Rappels et définitions

Nous rappelons quelques définitions et opérateurs différentiels utilisés dans ce premier rendu. Soit $u : \mathbb{R}^3 \to \mathbb{R}$ une fonction. On définit le gradient de u par $\nabla u = \begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \\ \frac{\partial u}{\partial z} \end{pmatrix}$ Le laplacien de u noté Δu par $\Delta u = div(\nabla u) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}$ qui est l'application de l'opérateur divergence sur le vecteur ∇u. De même on définit la divergence d'un vecteur $A = (A_x, A_y, A_z) \in \mathbb{R}^3$ par $div(A) = \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y} + \frac{\partial A_z}{\partial z}$.
On prend :

$$u = \begin{cases} \mathbb{R} \longrightarrow \mathbb{R} \\ x \longmapsto x^4 \end{cases}$$

on a alors :

$$\nabla u(x) = \frac{\partial u}{\partial x}(x) = u'(x) = 4x^3$$

$$\Delta u(x) = \frac{\partial^2 u}{\partial x^2}(x) = u''(x) = 12x^2$$

1.2 Premiers codes et définitions

Nous codons u, ∇u et Δu en python d'abord, on utilise les librairies suivantes

```python
import matplotlib.pyplot as plt
import numpy as np
from scipy import misc
from math import sqrt
```

code0.py

On code u. ∇u et Δu seront définit à l'aide de la librairie misc de scipy

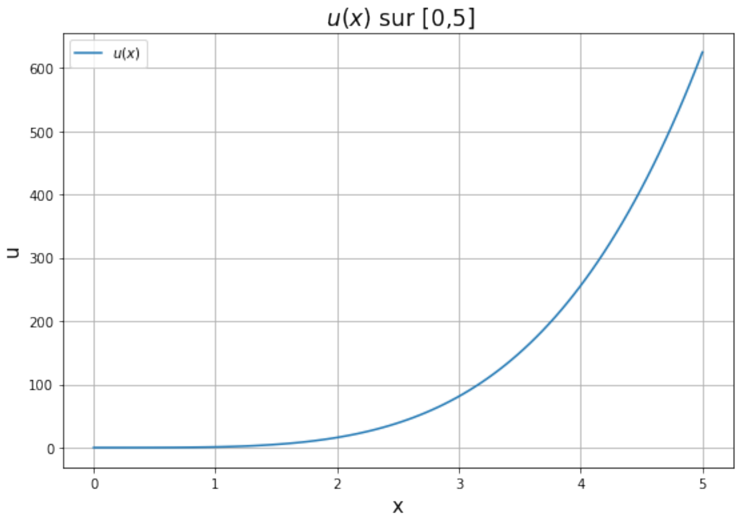
```python
def u(x): # définition de la fonction u
    return x**4

def grad1(f, x0, dx = 1.0):
    #Donne le gradient en dimension 1
    return misc.derivative(f, x0, dx, n = 1)

def laplac1(f, x0, dx = 1.0):
    #Donne le Laplacien en dimension 1
    return misc.derivative(f, x0, dx, n = 2)

# On remarque que dans notre cas
    # Le gradient est la dérivée
    # Le Laplacien est la dérivée seconde
```

code1.py

On visualise ensuite u sur $]0, 5[$:

```python
fig1 = plt.figure(figsize=(9, 6))

N = 100
x = np.linspace(0, 5, N)
y = u(x)

plt.plot(x,y, label = "$u(x)$")
plt.title(" $ u(x) $ sur [0,5] ", fontsize = 17)
plt.xlabel('x', fontsize = 15)
plt.ylabel('u', fontsize = 15)
plt.legend()
plt.grid(True)
plt.show()
```

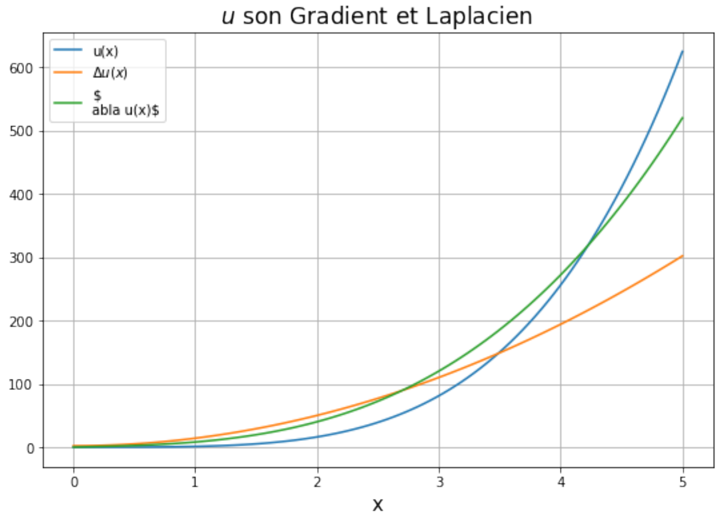
code2.py



On rajoute le graphe de Δu et ∇u :

```python
fig2 = plt.figure(figsize = (9, 6))

gradu = np.zeros(N)
laplacu = np.zeros(N)
x = np.linspace(0, 5, N)

for i in range(N):
    laplacu[i] = laplac1(u, x[i])
    gradu[i] = grad1(u, x[i])

plt.plot(x, y, label = 'u(x)')
plt.plot(x, laplacu, label = '$ \Delta u(x) $')
plt.plot(x, gradu, label = '$\nabla u(x)$')
plt.title("$u$ son Gradient et Laplacien ", fontsize = 17)
plt.xlabel('x', fontsize = 15)
plt.legend()
plt.grid(True)

plt.show()
```

code3.py



2 Approximation par différences finies

2.1 Objectif

On veut tracer l'erreur Laplacien, pour cela on approche les dérivées de u avec un deux développement limités, le premier est un développement de Taylor à l'ordre 1, avec un pas h :

$$u'(x_i) \simeq \frac{u(x_i + h) - u(x_i)}{h}$$

Le deuxième est une développement de Taylor à l'ordre 2 :

$$u''(x_i) \simeq \frac{u(x_i + h) - 2u(x_i) + u(x_i - h)}{h^2}$$

On code les deux développements :

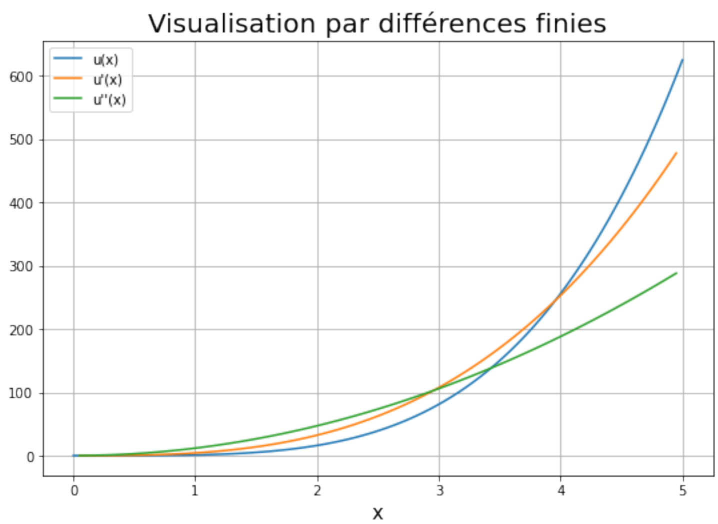
```
def diff_finie1(u, h, a, b):

    x = np.arange(a, b, h)
    N = np.size(x)
    u_x = np.zeros(N-2)

    for i in range(1,N-1):
        u_x[i-1] = (u(x[i+1]) - u(x[i]))/h

    return u_x

def diff_finie2(u, h, a, b):

    x = np.arange(a, b, h)
    N = np.size(x)
    u_xx = np.zeros(N-2)

    for i in range(1,N-1):
        u_xx[i-1] = (u(x[i+1]) - 2*u(x[i]) + u(x[i-1]))/(h**2)

    return u_xx
```

code4.py

Ce qui nous permettra d'appliquer cette méthode à la fonction u définie au début du rendu.

```python
fig3 = plt.figure(figsize = (9, 6))

N = 100
x = np.linspace(0, 5, N)
y = u(x)
u_x = diff_finie1(u, 5/N, 0, 5)
u_xx = diff_finie2(u, 5/N, 0, 5)

plt.plot(x,y, label = 'u(x)')
plt. plot(x[1:N-1], u_x, label = "u'(x)")
plt.plot(x[1:N-1], u_xx, label = "u''(x)")
plt.legend()
plt.grid(True)
plt.xlabel('x', fontsize = 15)
plt.title("Visualisation par différences finies", fontsize = 20)


plt.show()
```

code5.py



Pour trouver l'erreur d'approximation de la méthode, on doit définir analytiquement (dans ce cas c'est possible) u' et u'' :

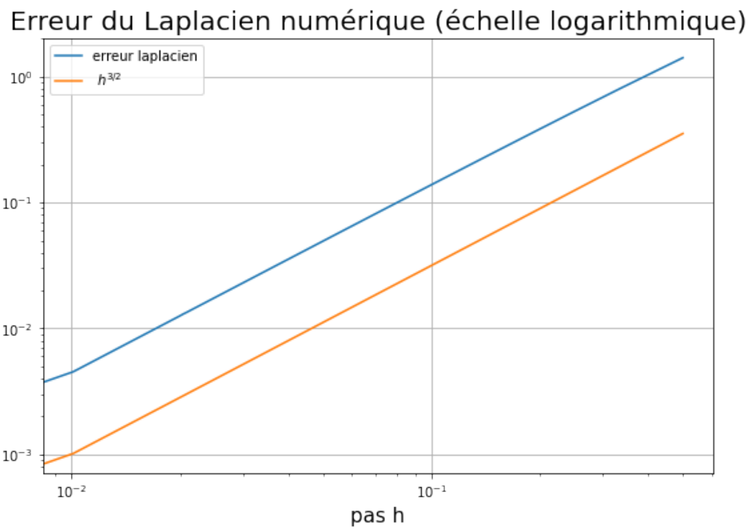
```python
def du(x):
     return 4*x**3


def d2u(x):
    return 12*x**2
```

code6.py

```python
erreur_Laplacien = np.zeros(99)
erreur_gradient = np.zeros(99)
h = np.zeros(99)

for n in range(3,101):
    h[n-3] = 1/(n-1)
    x = np.arange(0, 5, 1/(n-1))
    k = np.size(x)
    grad = du(x[1:k-1])
    lapla = d2u(x[1:k-1])
```

```python
    gradh = diff_finie1(u, 1/(n-1), 0, 5)
    laplah = diff_finie2(u, 1/(n-1), 0, 5)
    erreur_Laplacien[n-3] = sqrt(sum((lapla - laplah)**2))          # pour calculer la norme 2 discrète
    erreur_gradient[n-3] = sqrt(sum((grad - gradh)**2))             # pour calculer la norme 2 discrète

fig4 = plt.figure(figsize = (9, 6))
plt.plot(h, erreur_Laplacien, label = 'erreur laplacien')
plt.plot(h,h**(3/2), label = " $h^{3/2}$")
plt.legend()
plt.grid(True)
plt.xlabel('pas h', fontsize = 15)
plt.xscale('log')
plt.yscale('log')
plt.title("Erreur du Laplacien numérique (échelle logarithmique)", fontsize = 20)

fig5 = plt.figure(figsize = (9, 6))
plt.plot(h, erreur_gradient, label = "erreur gradient")
plt.plot(h,h**(1/2), label = "h")
plt.legend()
plt.xlabel('pas h', fontsize = 15)
plt.grid(True)
plt.xscale('log')
plt.yscale('log')
plt.title("Erreur du gradient numérique (échelle logarithmique)", fontsize = 20)

plt.show()
```

code7.py



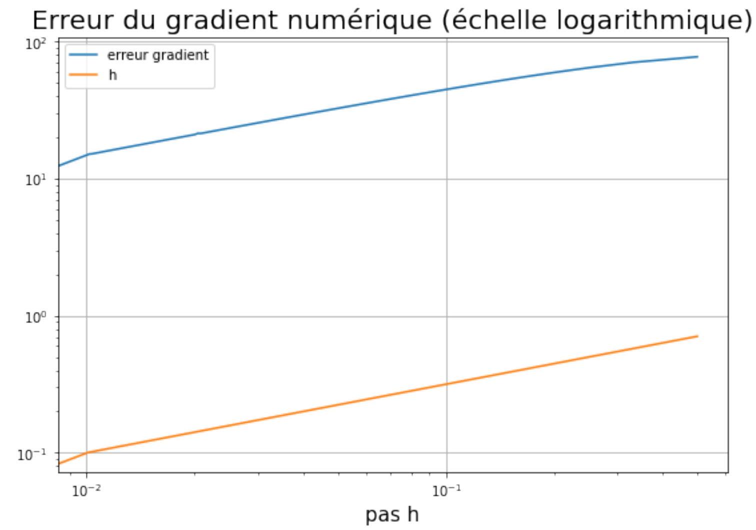
Ce qui nous donne un ordre de convergence de $\frac{3}{2}$ pour le Laplacien et $\frac{1}{2}$ pour le gradient. Analytiquement on a que $u(x+h) = u(x) + hu'(x) + O(h)$, si on développe à l'ordre deux on obtient :

$$\begin{cases} u(x+h) = u(x) + hu'(x) + \frac{h^2}{2}u''(x) + O(h^2) \\ u(x-h) = u(x) - hu'(x) + \frac{h^2}{2}u''(x) + O(h^2) \end{cases}$$

La somme nous donne :

$$u(x+h) + u(x-h) = 2u(x) + h^2u''(x) + O(h^2)$$

Ce qui montre une cohérence entre les résultats qui donne une convergence proche de et l'ordre de convergence donné par les développement limités qui est de 1.



3 Amélioration de derivatives1.py

La denière tâche est d'améliorer le code derivtives1.py fourni afin de :

- Paramétrer NX et étudier l'évolution de l'erreur L_1 pour 20 valeurs de NX entre 3 et 200
- Tracer les courbes d'erreur en fonction de NX en échelle log10

```
import numpy as np
import matplotlib.pyplot as plt


# NUMERICAL PARAMETERS

#Number of grid points
L=2*3.141592    #2*math.pi
err_Tx = np.zeros(199)
err_Txx = np.zeros(199)
pas = np.zeros(199)

for NX in range(3, 201):
    # Initialisation
    dx = L/(NX-1)  #Grid step (space)
    pas[NX-3] = dx
    x = np.linspace(0.0,L,NX)

    T = np.sin(x)
    Txe = np.cos(x)
    Txxe = -np.sin(x)

    Tx = np.zeros((NX))
    Txx = np.zeros((NX))

    #discretization of the second order derivative (Laplacian)
    for j in range (1, NX-1):
        Tx[j] = (T[j+1]-T[j-1])/(dx*2)
        Txx[j] = (T[j-1]-2*T[j]+T[j+1])/(dx**2)

    #Tx and Txx on boundaries
    # use extrapolation in order to have (T,x),xx=0
    #(T,x),xx= (Tx0 -2*Tx1+Tx2) =0
    Tx[0] = 2*Tx[1]-Tx[2]
    #use lower order formula (1st order)
    Tx[NX-1] = (T[NX-2]-T[NX-3])/dx
```

```python
    Txx[0] = 2*Txx[1]-Txx[2]
    Txx[NX-1] = 2*Txx[NX-2]-Txx[NX-3]

    err_Tx[NX-3] = np.sum(abs(Tx-Txe))
    err_Txx[NX-3] = np.sum(abs(Txx-Txxe))

plt.figure(1)
plt.plot(x,T, label = "graphe de sinus")
plt.title(u'Function sinus')
plt.xlabel(u'$x$', fontsize=20)
plt.ylabel(u'$T$', fontsize=26, rotation=90)
plt.legend()

plt.figure(2)
plt.xlabel(u'$x$', fontsize=26)
plt.ylabel(u'$Tx$', fontsize=26, rotation=90)
plt.plot(x,Tx, label='Tx')
plt.plot(x,np.log10(abs(Tx-Txe)), label='Error')
plt.title(u'First Derivative Evaluation (NX = 200)')
plt.legend()

plt.figure(3)
plt.xlabel(u'$x$', fontsize=26)
plt.ylabel(u'$Txx$', fontsize=26, rotation=90)
plt.plot(x,Txx,label='Txx')
plt.plot(x,np.log10(abs(Txx-Txxe)),label='Error')
plt.title(u'Second Derivative Evaluation (NX = 200)')
plt.legend()

plt.figure(4)
plt.plot(pas, err_Tx, label = "err_Tx")
plt.plot(pas, 1/2*pas**(3/2), label = '$h^{3/2}$')
plt.xlabel('pas')
plt.xscale('log')
plt.yscale('log')
plt.title("Courbe d'erreur de la dérivée numérique en fonction du pas")
plt.legend()

plt.figure(5)
plt.plot(pas, err_Txx, label = "err_Txx")
plt.plot(pas, 1/5*pas, label = 'h')
plt.xlabel('pas')
plt.xscale('log')
plt.yscale('log')
plt.title("Courbe d'erreur de la dérivée seconde numérique en fonction du pas")
plt.legend()

plt.show()
```

code8.py

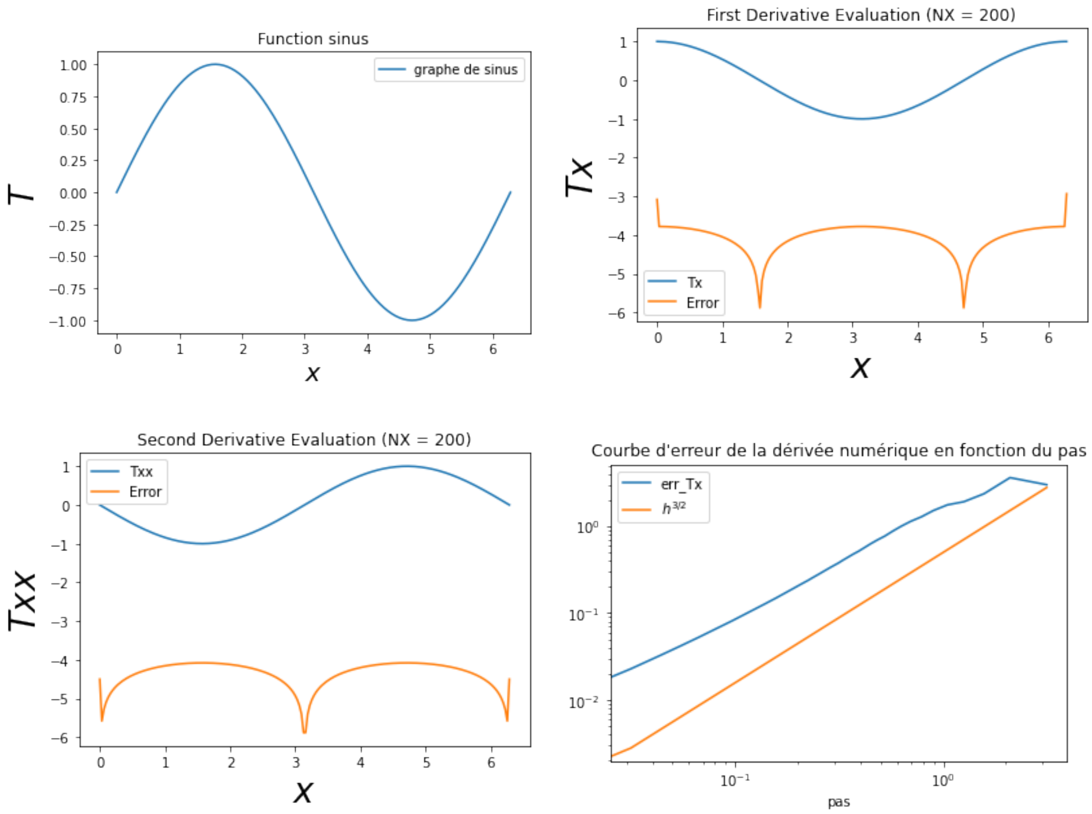


Figure 1: Graphes après améliorations du code derivatives.py