Sur la caractérisation d'outils relatif à la résolution d'équations aux dérivées partielles

Gaggini Lorenzo

Les résultas ci-aprés sont tous donné en coordoné cartésienne, Quand un résultats est affirmé, ça source est indiqué en annex par sont repére (1) associé.

On cherche dans un premier temps à appréhendé sout leur forme générales plusieurs outils en lien avec la résolution des EDP

Soit u: $\mathbb{R}^n \longrightarrow \mathbb{R}$ une application que l'on carractérise de «champs scalaire »(1)

Soit V: $\begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{pmatrix} \longmapsto \begin{pmatrix} V_1(x_1, x_2 \ldots, x_n) \\ V_2(x_1, x_2 \ldots, x_n) \\ . \\ . \\ V_n(x_1, x_2 \ldots, x_n) \end{pmatrix}$ que l'on carractérise de «champs vectoriel»

avec les V_i des application de $\mathbb{R}^n \longrightarrow \mathbb{R}^n$ toutes de classes C^n sur $\mathbb{R}^n$ (2)

On donne dans un premier temps $\Delta u = \sum_{i=1}^n \frac{\partial^2 u}{\partial x_i^2}(x_1, x_2, \ldots x_n)$ l'opérateur laplacien associé à u (3)

Avec par ailleur : Grad(u) le gradiant de u tel que $\mathrm{Grad}(u) = \begin{pmatrix} \frac{f}{\partial x_1} \\ \frac{f}{\partial x_2} \\ . \\ . \\ \frac{f}{\partial x_n} \end{pmatrix}$ (4)

Finalement on donne Div(V) la divergence de V donner par : $\mathrm{Div}(V) = \sum_{i=1}^n \frac{\partial V_{x_i}}{\partial x_i}$ (5)

Sur les pages suivantes sont joint les graph associé au proposistion qui précède illustrées par un exemple en dimention 1 et 2

Annex

(1) http://www.tangentex.com/OperateursDiff.htm

(2)https://fr.wikipedia.org/wiki/Champ_de_vecteurs

(3),,(5): http://www.tangentex.com/OperateursDiff.htm

(4) https://fr.wikipedia.org/wiki/Gradient

(6) https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html7

(7),(8) : http://www.tangentex.com/BibliothequeCodes.htm#Ind52

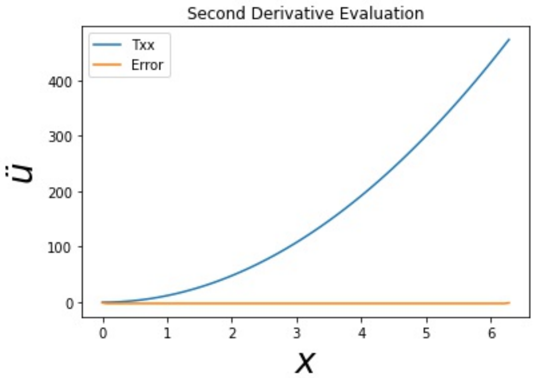
Exemple appliqué en dimention 1

Soit $u \quad : \mathbb{R} \longrightarrow \mathbb{R}$

$$x \longmapsto x^4$$

Dans ce cas en dimention 1, on a directement que $\Delta u = u''(x)$

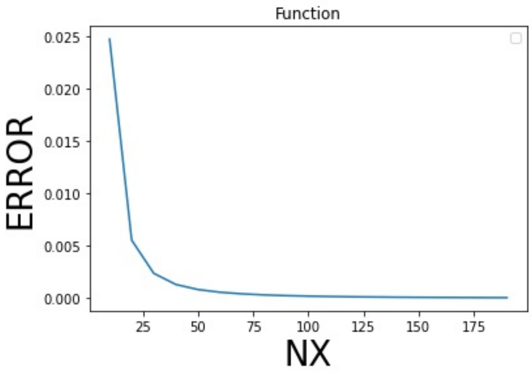
Ce faisant, il nous est fournis le code derivatives1d.py qui se propose de calculer la dérivé seconde de fonction d'une variable, ce dernier nous donnera la dérivé seconde et le taux d'érreur associé à son calcul :



Algorithme derivatives1d.py

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


# NUMERICAL PARAMETERS

NX = 100   #Number of grid points
L=2*np.pi
dx = L/(NX-1)  #Grid step (space)

# Initialisation
x = np.linspace(0.0,L,NX)

T = x**4
Txe = 4*x**3
Txxe = 12*x**2

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
Tx[0] = 2*Tx[1]-Tx[2]
Tx[NX-1] = (T[NX-2]-T[NX-3])/dx
Txx[0] = 2*Txx[1]-Txx[2]
Txx[NX-1] = 2*Txx[NX-2]-Txx[NX-3]


plt.figure(1)
plt.plot(x,T)
plt.title(u'Function')
plt.xlabel(u'$x$', fontsize=26)
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')
plt.legend()
plt.figure(3)
plt.xlabel(u'$x$', fontsize=26)
plt.ylabel(u'$ü$', 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')
plt.legend()
plt.show()
```

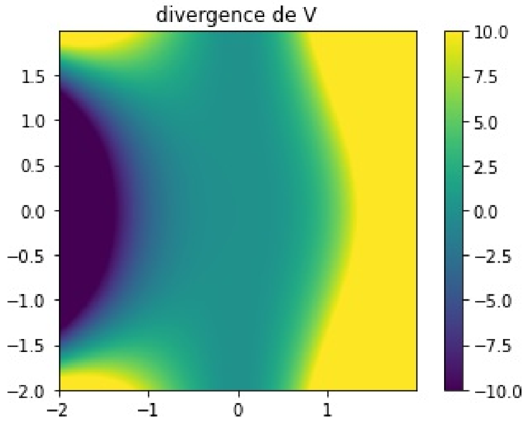
Ce faisant, pour calculer le rapport entre l'erreur du calcul, et la finesse de la grille de discretisation utilisé, on se propose une modification de derivatives1d.py nous offrant une visualisation de ce rapport :



On constate ainsi que plus le nombre de point NX est important, plus le pas de la grille de discretisation (noté dx dans l'algo) est petit, et donc plus grand est la précision du calcule.

Algorithme pour le calcule du rapport finesse/erreur de derivatives1d.py

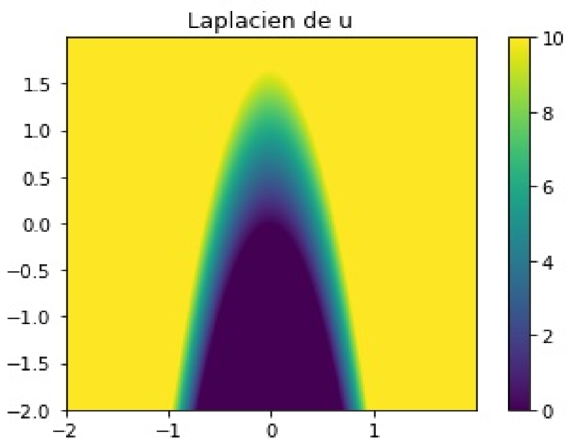
```
import numpy as np
import matplotlib.pyplot as plt

#EE et XX sont des liste où sont stocker les données offertes par le coeur du programe derivatives1d.py pour les restituer dans un graphique
EE=[]
XX=[]
for NX in range(10,200,10):
    L=1
    dx = L/(NX-1)  #Grid step (space)

    # Initialisation
    x = np.linspace(0.0,L,NX)

    T = x**4
    Txe = 4*x**3
    Txxe = 12*x**2

    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
    Tx[0] = 2*Tx[1]-Tx[2]
    Tx[NX-1] = (T[NX-2]-T[NX-3])/dx
    Txx[0] = 2*Txx[1]-Txx[2]
    Txx[NX-1] = 2*Txx[NX-2]-Txx[NX-3]

    print (NX,abs(Txx-Txxe)[1])
    EE.append(abs(Txx-Txxe)[1])
    XX.append(NX)
print (EE)
plt.figure(1)
plt.plot(XX,EE)
plt.title(u'Function')
plt.xlabel(u'NX', fontsize=26)
plt.ylabel(u'ERROR', fontsize=26, rotation=90)
plt.legend()
```



(7)

Algorithme pour la divergence de V

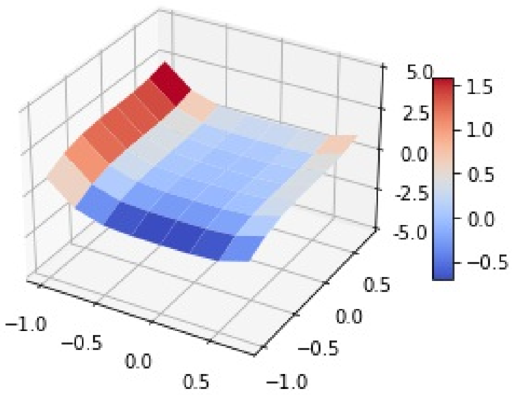
```
from scipy import meshgrid, diff, arange
from matplotlib.pyplot import *

# Définition du champ
def Champ(x,y):
    Ex = x**4
    Ey = x**2*y**3
    return Ex,Ey

# Calcul de la divergence
def Divergence(Fx,Fy):
    div = (diff(Fx,axis=1)/hx)[:-1,:] + (diff(Fy,axis=0)/hy)[:,:-1]
    return div


# Définition de la grille de calcul
hx = 0.01
hy = 0.01
X = arange(-2.,2.,hx)
Y = arange(-2.,2.,hy)
x,y = meshgrid(X,Y)

# Calcul du champ
Ex, Ey = Champ(x,y)

# calcul de la divergence du champ
divE = Divergence(Ex,Ey)

# tracé de la divergence
figure()
title("divergence")
pcolormesh(x, y, divE, vmin=-10, vmax=10)
colorbar()
gca().set_aspect("equal")
show()
```



(8)

Algorithme pour le laplacien de u

```
from scipy import meshgrid, diff, arange, sqrt
from matplotlib.pyplot import *

#Définition de la fonction scalaire
def Fonction(x,y):
    return 2*x**4 + y**3

#Routine de calcul du Laplacien
def LaplacienScalaire(x,y,F):
    dFx = (diff(F,n=2,axis=1)/hx**2)[:-2,:]
    dFy = (diff(F,n=2,axis=0)/hy**2)[:,:-2]
    return dFx + dFy

#Définition de la grille de calcul
hx = 0.001
hy = 0.001
X = arange(-2,2,hx)
Y = arange(-2,2,hy)
x,y = meshgrid(X,Y)

#Calcul de la fonction
f = Fonction(x,y)

#Calcul du laplacien
delta2 = LaplacienScalaire(x,y,f)

#Tracé du laplacien
figure()
title('Laplacien de u')
pcolormesh(x[:-2,:-2], y[:-2,:-2], delta2, vmin=0, vmax=10)
colorbar()
gca().set_aspect("equal")
show()
```



(6)

Algorithme pour le graph de u

```
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStr-
Formatter
import numpy as np


fig = plt.figure()
ax = fig.gca(projection='3d')

# Make data.
X = np.arange(-1, 1, 0.25)
Y = np.arange(-1, 1, 0.25)
X, Y = np.meshgrid(X, Y)
Z = 2*X**4+Y**3

# Plot the surface.
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=True)

# Customize the z axis.
ax.set_zlim(-5.01, 5.01)
ax.zaxis.set_major_locator(LinearLocator(5))
ax.zaxis.set_major_formatter
(FormatStrFormatter('%.1f'))

# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=10)

plt.show()
```