Implementation_1D_Fond

March 21, 2022

1 ImplÃľmentation du schÃľma de Rusanov

1.1 Importations

[1]:
```
import math as math
import matplotlib.pyplot as plt
import matplotlib.patches as patches

from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from matplotlib.figure import Figure

import numpy as np
import imageio

import time
```

1.2 ParamÃĺtres du problÃĺme

[2]:
```
g=1 # Constante gravitationnelle

xMin = 0
xMax = 100

N=128
h=(xMax-xMin)/(N+1) # Pas du maillage spatial

Tmax=20 # Temps Max
t=0 # Temps dans la simulation
n=0 # Nombre d'itÃľrations

X = np.linspace(xMin,xMax,N+2) # DiscrÃľtisation de [xMin, xMax]

# Tableau avec 2 lignes et N+2 colonnes : discrÃľtisation du vecteur (h, q) en N␣
 ↪parties + 2 aux bords
```

1.3 Nombre de sauvegardes durant la simulation

[3]:
```
nSauvegarde=100
tSauvegarde=[False for i in range(nSauvegarde)]
images=[]
j=0 #Nombre de sauvegardes dÃľjÃă effectuÃľes
```

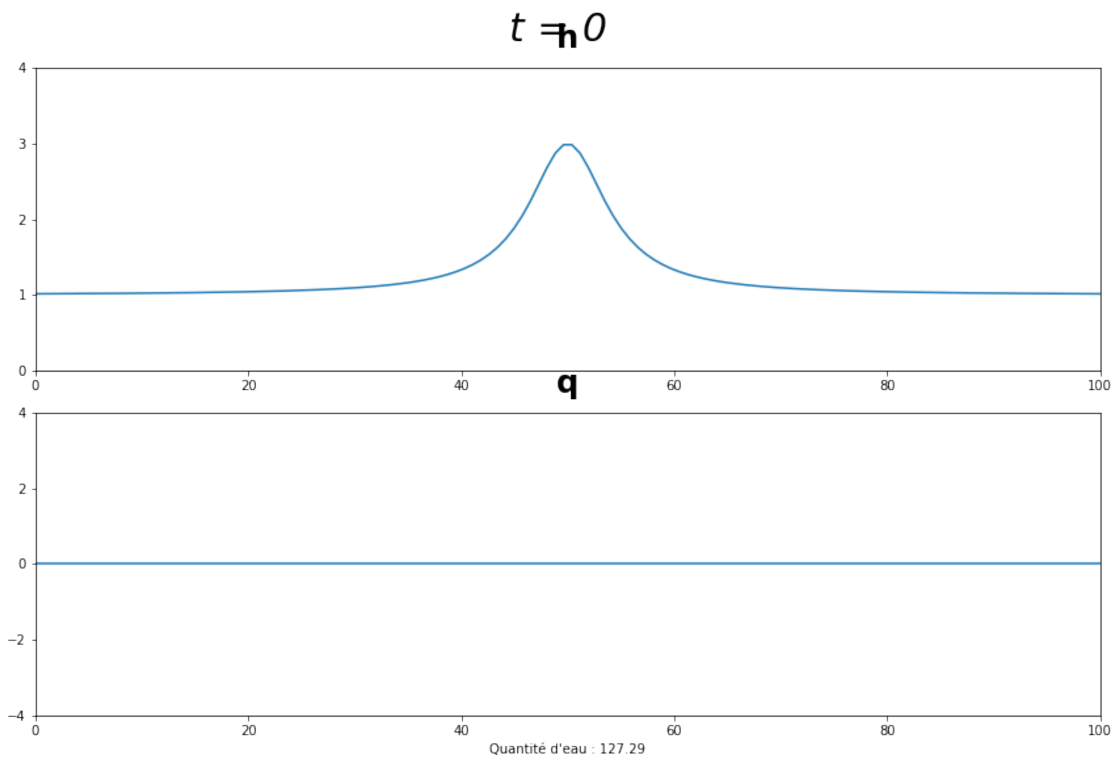
1.4 Conditions initiales et Topographie

[4]:
```
# Tableau avec 2 lignes et N+2 colonnes : discrÃľtisation du vecteur (h, q) en N␣
 ↪parties + 2 aux bords
U = np.ones((2,N+2))
Uprime = np.zeros((2,N+2)) # Pour faire les calculs dans la boucle

Z = np.zeros((1,N+2))



##CONDITIONS SUR Z

# Conditions initiales avec Z = 0
Z[0,:]= 0

# Conditions initiales avec Z en crÃľneau
# Z[0,(N+2)//5:2*(N+2)//5]= 0.3
# Z[0,3*(N+2)//5:4*(N+2)//5]= 0.3

# Conditions initiales avec Z en bosse C infini
# Z[0,:]= 1/ ( 1+.1*(X-50)**2 )

# Conditions initiales avec Z en escalier descendant
# Z[0,0:(N+2)//4]= .75
# Z[0,(N+2)//4:3*(N+2)//4]= 0.25

# Conditions initiales avec Z en tangente hyperbolique
# i=0
# for x in X:
#     Z[0,i]= np.tanh(5-x)/2 + .5
#    i+=1



##CONDITIONS SUR H

# Hauteur de l'eau en escalier
```

```
# U[0, 0 :(N+2)//3]= 1.75 - Z[0,:(N+2)//3]
# U[0,(N+2)//3:N+2]= 1.25 - Z[0,(N+2)//3:N+2]

# Hauteur de l'eau constante
# U[0,:]= 1

# Hauteur de l'eau en bosse
U[0,:]= 1+ 2 / ( 1+.05*(X-50)**2 )



##CONDITIONS SUR Q

# DÃľbit de l'eau constant
U[1,:]= 0
```

1.5 Affichage des images

[5]:
```
def affiche_U():

    fig, axs = plt.subplots( 2 , 1 , figsize=(12,8) )

    fig.suptitle("t = 0", style='italic',size=30)
    fig.tight_layout()

    plt.rc('font', size=20)

    h_z = U[0,:]+Z[0,:]

    axs[0].plot(X,h_z)
    axs[0].set_title("h", fontweight="bold", pad=15)
    axs[0].set_xlim([xMin,xMax])
    axs[0].set_ylim([0,4])
    axs[0].fill_between(X, Z[0,:], step="pre", alpha=0.5, color="grey")

    axs[1].plot(X,U[1,:])
    axs[1].set_title("q", fontweight="bold", pad=15)
    axs[1].set_xlim([xMin,xMax])
    axs[1].set_ylim([-4,4])

    axs[1].set_xlabel("QuantitÃľ d'eau : "+str(round( sum(U[0,:])*h , 2 )))

    plt.show()

affiche_U()
```



1.6 Sauvegarde des images

[6]:
```python
def enregistre_U(n,t):

    t_int=round(t,4) # Troncature de t aprÃĺs la 4eme dÃľcimale

    fig, axs = plt.subplots( 2 , 1 , figsize=(12,8) )

    fig.suptitle("t = "+str(t_int) , style='italic',size=30)
    fig.tight_layout()

    plt.rc('font', size=20)

    h_z = U[0,:]+Z[0,:]

    axs[0].plot(X,h_z)
    axs[0].set_title("h", fontweight="bold", pad=15)
    axs[0].set_xlim([xMin,xMax])
    axs[0].set_ylim([0,4])
    axs[0].fill_between(X, Z[0,:] , step="pre", alpha=0.5,color="grey")
```

```
    axs[1].plot(X,U[1,:])
    axs[1].set_title("q", fontweight="bold", pad=15)
    axs[1].set_xlim([xMin,xMax])
    axs[1].set_ylim([-4,4])

    axs[1].set_xlabel("QuantitÃľ d'eau : "+str(round( sum(U[0,:])*h , 2 )))

    # To remove the huge white borders
    axs[0].margins(0)
    axs[1].margins(0)

    fig.canvas.draw()
    image_from_plot = np.frombuffer(fig.canvas.tostring_rgb(), dtype=np.uint8)
    image_from_plot = image_from_plot.reshape( fig.canvas.get_width_height()[::
↪-1] + (3,))

    images.append(image_from_plot)

    #Sauvegarde dans un fichier png
    #plt.savefig("etape"+str(n)+".png")

    plt.close()
```

1.7 Fonctions qui interviennent dans le schÃľma :

Notons $\mathbf{U} = \begin{pmatrix} h \\ q \end{pmatrix} \in \mathbb{R}^2$ le vecteur inconnu. Les diffÃľrentes fonctions du schÃľma qu'il s'agit d'implÃľmenter sont $\mathbf{F}(\mathbf{U})$, $\mathbf{B}(\mathbf{U})$ et $\mathcal{F}^n_{i+\frac{1}{2}}$. Rappelons les diffÃľrentes dÃľfinitions :

$$\mathbf{F}(\mathbf{U}) = \begin{pmatrix} q \\ \frac{q^2}{h} + g\frac{h^2}{2} \end{pmatrix} \in \mathbb{R}^2 \text{ et } \mathbf{B}(\mathbf{U}) = \begin{pmatrix} 0 \\ -gh\frac{\partial Z}{\partial x} \end{pmatrix} \in \mathbb{R}^2.$$

$\mathbf{F}(\mathbf{U})$ dÃľsigne la fonction flux et $\mathbf{B}(\mathbf{U})$ le terme source. Avec ces notations, le systÃĺme de dÃľpart se rÃľÃľcrit :

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial}{\partial x}(\mathbf{F}(\mathbf{U})) = \mathbf{B}(\mathbf{U}).$$

Et si l'on suppose que $Z \equiv 0$, alors le schÃľma s'Ãľcrit simplement :

$$\frac{\mathbf{U}_i^{n+1} - \mathbf{U}_i^n}{\Delta t^n} + \frac{\mathcal{F}^n_{i+\frac{1}{2}} - \mathcal{F}^n_{i-\frac{1}{2}}}{\Delta x_i} = 0$$

Avec

$$\mathcal{F}^n_{i+\frac{1}{2}} = \frac{\mathbf{F}(\mathbf{U}_i^n) + \mathbf{F}(\mathbf{U}_{i+1}^n)}{2} - \max_{j\in\{i,i+1\}} \max_{k\in\{1,2\}} |\lambda_k(\mathbf{U}_j^n)| \frac{\mathbf{U}_{i+1}^n - \mathbf{U}_i^n}{2}.$$

En prenant une topographie Z non triviale, on peut essayer de prendre le mÃłme schÃľma en rajoutant le terme source. Si l'on a accÃĺs Ãă la dÃľrivÃľe Z' de Z, le schÃľma s'Ãľcrit alors :

$$\frac{\mathbf{U}_i^{n+1} - \mathbf{U}_i^n}{\Delta t^n} + \frac{\mathcal{F}^n_{i+\frac{1}{2}} - \mathcal{F}^n_{i-\frac{1}{2}}}{\Delta x_i} = \begin{pmatrix} 0 \\ -gh_i Z_i' \end{pmatrix}$$

ou bien en utilisant des diffÃľrences finies centrÃľes pour approcher les dÃľrivÃľes :

$$\frac{\mathbf{U}_i^{n+1} - \mathbf{U}_i^n}{\Delta t^n} + \frac{\mathcal{F}^n_{i+\frac{1}{2}} - \mathcal{F}^n_{i-\frac{1}{2}}}{\Delta x_i} = \begin{pmatrix} 0 \\ -gh_i \frac{Z_{i+1} - Z_{i-1}}{2\Delta x_i} \end{pmatrix}$$

[7]:
```
# Fonction F du schÃľma :
# EntrÃľe : vecteur U = (h, q) (type numpy.ndarray)
# Sortie : F(U) (type numpy.ndarray)
def F(U):
    return np.array([U[1],U[1]**2/U[0]+g*U[0]**2/2])

# Fonction B du schÃľma :
# EntrÃľes : * vecteur U = (h, q) (type numpy.ndarray)
#           * valeur dxZ de la dÃľrivÃľe spatiale de Z
# Sortie : B(U) (type numpy.ndarray)
def B(U,i):
    return np.array([0,-g*U[0]*(Z[0,i+1]-Z[0,i-1])/2])

# Fonction qui retourne la valeur propre max en module
# EntrÃľes : * U solution discrÃľtisÃľe (type numpy.ndarray de taille 2*(N+2))
#           * indice i de la position spatiale
# Sortie : max_j max_k |Îż_k(U_j^n)|
def vmax(U,i):
    res = abs(U[1,i]/U[0,i]+math.sqrt(g*U[0,i]))
    res = max(res,abs(U[1,i]/U[0,i]-math.sqrt(g*U[0,i])))
    res = max(res,abs(U[1,i+1]/U[0,i+1]+math.sqrt(g*U[0,i+1])))
    res = max(res,abs(U[1,i+1]/U[0,i+1]-math.sqrt(g*U[0,i+1])))
    return res

# Fonction flux numÃľrique :
# EntrÃľes : * U solution discrÃľtisÃľe (type numpy.ndarray de taille 2*(N+2))
#           * indice i de la position spatiale
# Sortie : F_{i+1/2}^n
def F_ronde(U,i):
    return (F(U[:,i])+F(U[:,i+1]))/2-vmax(U,i)*(U[:,i+1]-U[:,i])/2
```

1.8 Boucle de rÃľsolution numÃľrique :

Une fois le pas de temps Δt^n dÃľterminÃľ, on passe du temps t^n au temps t^{n+1} suivant la relation explicite suivante :

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n - \Delta t^n \frac{\mathcal{F}_{i+\frac{1}{2}}^n - \Delta t^n \mathcal{F}_{i-\frac{1}{2}}^n}{\Delta x_i} + \Delta t^n \begin{pmatrix} 0 \\ -gh_i \frac{Z_{i+1}-Z_{i-1}}{2\Delta x_i} \end{pmatrix}$$

[8]:
```
enregistre_U(n,t)

print("Calcul en cours ...")

while(t<Tmax): # Tant que le temps max n'est pas atteint :

    M = vmax(U, 0)
    for i in range(1, N+1):
        M = max(M, vmax(U, i))

    # Pour assurer la stabilitÃľ, tau doit Ãłtre infÃľrieur Ãă h/(2* max(vp))
    tau = 0.8*h/(2*M)

    for i in range(1, N+1):
        Uprime[:,i] = U[:,i]+tau/h*(F_ronde(U,i-1)-F_ronde(U,i))
        Uprime[:,i] -= tau*B(U[:,i],i)

    U = Uprime

    # Conditions aux bords
    U[:,0] = U[:,1]
    U[:,N+1] = U[:,N]

    t+=tau
    n+=1

    if (t > Tmax/nSauvegarde*j and (j<nSauvegarde) and (not tSauvegarde[j])):
        enregistre_U(n,t)
        #affiche_U()
        tSauvegarde[j]=True
        j+=1

print("Nombre d'itÃľrations : " + str(n))

imageio.mimsave('movie.gif', images)

print("Gif SauvegardÃľ dans le dossier sous le nom : movie.gif")
```

```
Calcul en cours ...
Nombre d'itÃĺrations : 121
Gif SauvegardÃĺ dans le dossier sous le nom : movie.gif
```

[ ]:

```

```

[ ]:

```

```