Computation of Shallow Water Equation

Student ID: r07525117 Student Name: Chang-Syuan Syu

Question1: Plot the initial conditions, i.e., η(x; 0) and U(x; 0), to be used in the simulations.

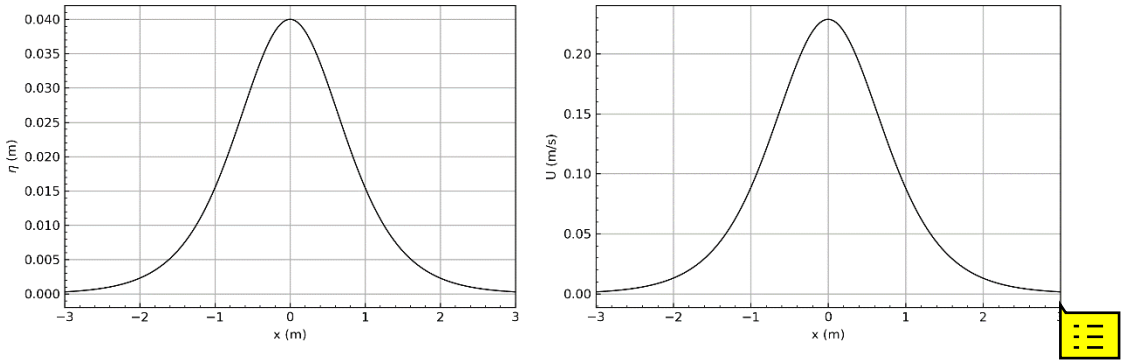

Figure 1: Reference plot for problem 1.

Question2: Write a program to solve the 1DH LSWE using these initial conditions, for -12 < x < 24 (m) and 0 < t < 6.95 (s). You can try using the step size Δx = 0.06 m as a start.

The Result Show in the appendix

Question3: Make sure your code runs correctly. To gain confidence in your results, you may want to compare your numerical results for η against the analytical solution at t = 6.95 s, i.e., (7).

Answer: The Result Show in Figure 2

Question4: Run two additional simulations with different step sizes Δx. Then plot the analytical solution and all your numerical results in one plot at t = 6.95 s.

Answer: The Result Show in Figure 2

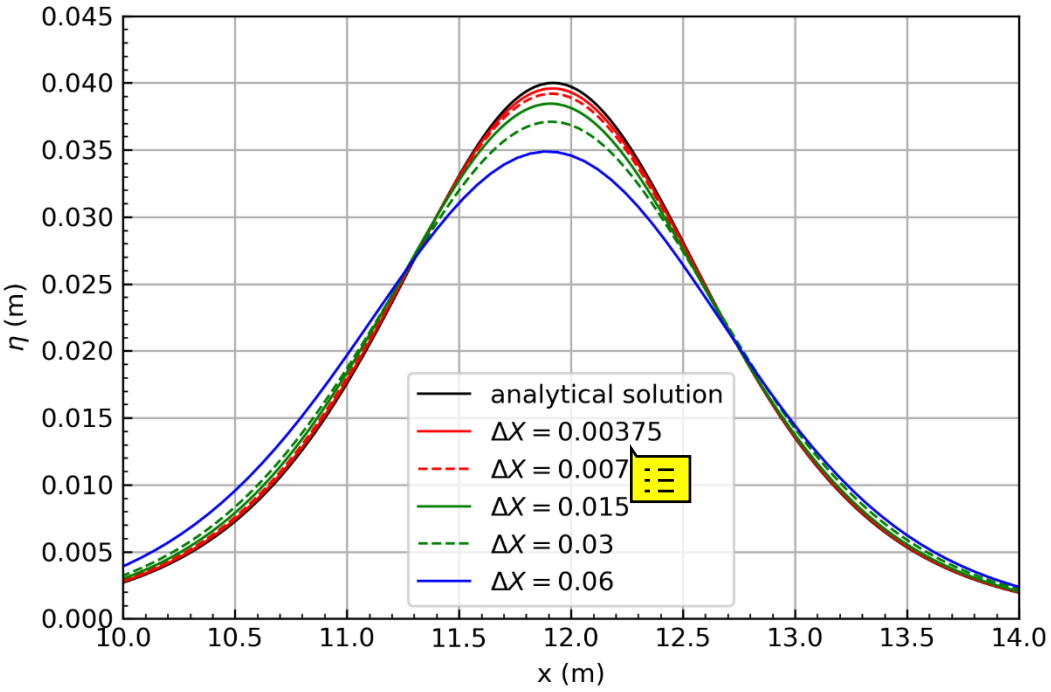

Figure 2: Reference plot for problem 3 and 4.

Question5: Show that the convergence rate of the Lax-Friedrichs numerical scheme is first-order; i.e., $\Delta x \ \alpha \ L^2$-norm, by plotting the three different step sizes against the corresponding L^2-norms.

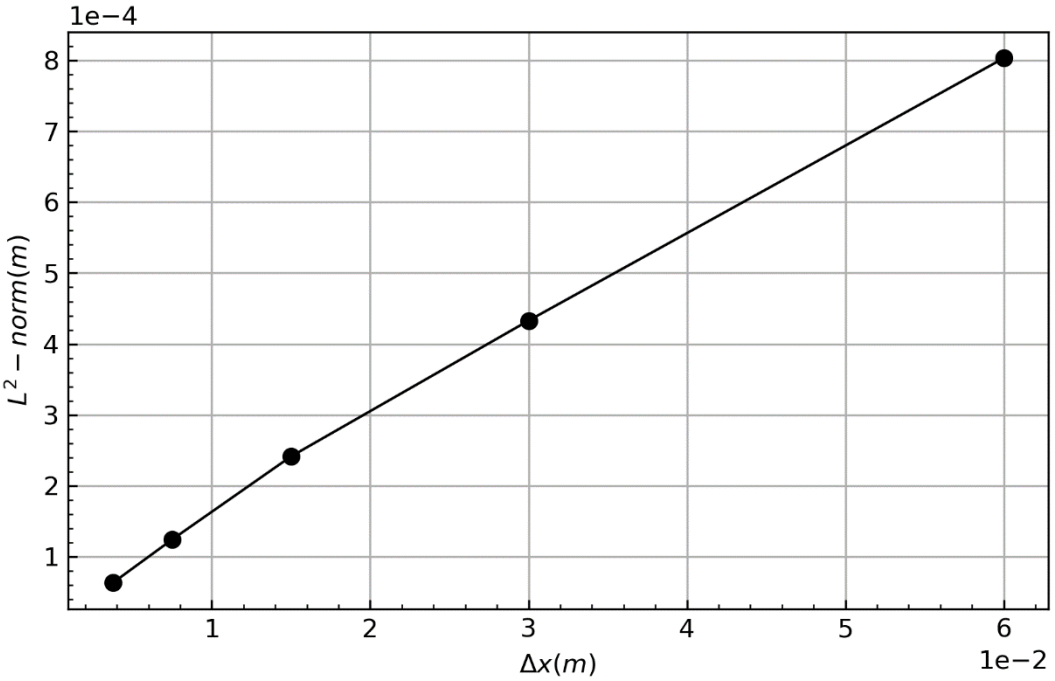

Figure 3: Reference plot for problem 5.

Appendix

Python Code:

```
import numpy as np
import matplotlib.pyplot as plt
import os
os.chdir('D:\ShallowWaterComputation\HW2')
from numba import jit

g = 9.806
h = 0.3

def sech(x):
    return 1.0/np.cosh(x)

def eta(x, t=0, H=0.04):
    K = (1.0/h)*np.sqrt( (3*H) / (4*h) )
    C = np.sqrt(g*h)
    return H*np.power( sech( K*( x-C*t ) ) , 2 )

def U(x, t=0):
    TEMP = eta(x, t)
    return TEMP *np.sqrt(g*h) /float(h)

@jit
def Numeri(deltaX,Ccfl=1):
    deltaT = float(deltaX)/np.sqrt(g*h)*Ccfl

    x = np.arange(-12, 24, deltaX)
    t = np.arange(0, 6.95, deltaT)

    tlen = len(t)
    xlen = len(x)
```

```
    ETA = np.empty( ( tlen, xlen+2 ) )
    Vel_U = np.empty( ( tlen, xlen+2 ) )

    #####Initialize

    ##Cell
    x = np.append(x, x[-1]+deltaX )
    x = np.append(x[0]-deltaX, x )
    for i in range(1,xlen-1):
        ETA[0][i] = eta(x[i])

    for i in range(1,xlen-1):
        Vel_U[0][i] = U(x[i])

    ##BC
    ETA[0][0] = ETA[0][2]
    ETA[0][-1] = ETA[0][-3]
    Vel_U[0][0] = 0
    Vel_U[0][-1] = 0


    n = 0
    #######Numerical Solution
    while( n < (tlen-1) ):
        #cell
        for i in range(1,xlen-1):
            ETA[n+1][i] = ( ETA[n][i+1] + ETA[n][i-1] )/2.0 -
(deltaT*h)/(2*deltaX)*( Vel_U[n][i+1] - Vel_U[n][i-1] )
            Vel_U[n+1][i] = ( Vel_U[n][i+1] + Vel_U[n][i-1] )
/2.0 - (deltaT*g)/(2*deltaX)*( ETA[n][i+1] - ETA[n][i-1] )
        #BC
        ETA[n+1][0] = ETA[n+1][2]
        ETA[n+1][-1] = ETA[n+1][-3]
        Vel_U[n+1][0] = 0
        Vel_U[n+1][-1] = 0
        n = n+1
```

```
    return x, ETA

####Initial Plot
deltaX = 0.00375
x = np.arange(-12, 24, deltaX)
U0 = U(x)
ETA0 = eta(x, t=0, H=0.04)


fig, ax1 = plt.subplots()
l1 = ax1.plot( x, ETA0, '-', color = 'black', linewidth= 1)
ax1.set_xlim( [-3,3] )
ax1.tick_params(which='both',direction='in')
ax1.minorticks_on()
ax1.grid()
ax1.set_xlabel( 'x (m)' )
ax1.set_ylabel( '${\eta}$ (m)' )
fig.tight_layout()
fig.savefig( 'ETA0.png', dpi=300)


fig, ax2 = plt.subplots()
l2 = ax2.plot( x, U0, '-', color = 'black', linewidth= 1)
ax2.set_xlim( [-3,3] )
ax2.tick_params(which='both',direction='in')
ax2.minorticks_on()
ax2.grid()
ax2.set_xlabel( 'x (m)' )
ax2.set_ylabel( 'U (m/s)' )

fig.tight_layout()
fig.savefig( 'U0.png', dpi=300)




####
Cfl = 0.9
```

```
x1,ETA1= Numeri(0.00375, Cfl)
x2,ETA2= Numeri(0.0075, Cfl)
x3,ETA3= Numeri(0.015, Cfl)
x4,ETA4= Numeri(0.03, Cfl)
x5,ETA5= Numeri(0.06, Cfl)



#Analytical Solution
x = np.arange(-12, 24, 0.015)
AnalyticalSol = eta(x,t=6.95)


fig, ax1 = plt.subplots()
l1 = ax1.plot( x, AnalyticalSol, '-', color = 'black',
linewidth= 1)
l2 = ax1.plot( x1, ETA1[-1], '-', color = 'r', linewidth= 1)
l3 = ax1.plot( x2, ETA2[-1], '--', color = 'r', linewidth= 1)
l4 = ax1.plot( x3, ETA3[-1], '-', color = 'g', linewidth= 1)
l5 = ax1.plot( x4, ETA4[-1], '--', color = 'g', linewidth= 1)
l6 = ax1.plot( x5, ETA5[-1], '-', color = 'b', linewidth= 1)


ax1.set_xlim( [10,14] )
ax1.set_ylim( [0,0.045] )
ax1.tick_params(which='both',direction='in')
ax1.minorticks_on()
ax1.grid()
ax1.set_xlabel( 'x (m)' )
ax1.set_ylabel( '${\eta}$ (m)' )

lns =  l1 + l2 + l3 + l4 + l5 +l6
labels = [  'analytical solution', '${\Delta}X =
0.00375$' ,'${\Delta}X = 0.0075$' ,'${\Delta}X =
0.015$' ,'${\Delta}X = 0.03$','${\Delta}X = 0.06$' ]
ax1.legend(lns ,labels , loc = 'lower center' )
```

```
fig.tight_layout()
fig.savefig( 'Q1.png', dpi=300)


###L-norm
@jit
def norm(EtaNum, EtaTheory):
    temp = 0
    for i , j in zip(EtaNum, EtaTheory):
        temp = temp + np.power(i-j, 2)
    N = len(EtaNum)
    temp = np.sqrt( float(temp) / float(N) )
    return temp



Cfl = 0.9
x1,ETA1= Numeri(0.00375, Cfl)
x2,ETA2= Numeri(0.0075, Cfl)
x3,ETA3= Numeri(0.015, Cfl)
x4,ETA4= Numeri(0.03, Cfl)
x5,ETA5= Numeri(0.06, Cfl)

AnalySol1 = eta(x1,t=6.95)
AnalySol2 = eta(x2,t=6.95)
AnalySol3 = eta(x3,t=6.95)
AnalySol4 = eta(x4,t=6.95)
AnalySol5 = eta(x5,t=6.95)


e1 = norm(ETA1[-1],AnalySol1 )
e2 = norm(ETA2[-1],AnalySol2 )
e3 = norm(ETA3[-1],AnalySol3 )
e4 = norm(ETA4[-1],AnalySol4 )
e5 = norm(ETA5[-1],AnalySol5 )

x = np.array([0.00375, 0.0075, 0.015, 0.03, 0.06])
Error = np.array([e1,e2,e3,e4,e5])
```

```
fig, ax1 = plt.subplots()
l1 = ax1.plot( x, Error, '-o', color = 'black', linewidth= 1)

#ax1.set_xlim( [10,14] )
#ax1.set_ylim( [0,0.045] )
ax1.tick_params(which='both',direction='in')
ax1.minorticks_on()
ax1.grid()
ax1.yaxis.get_major_formatter().set_powerlimits((0,2))
ax1.xaxis.get_major_formatter().set_powerlimits((0,1))
ax1.set_xlabel( '${\Delta}x (m)$ ' )
ax1.set_ylabel( '${L^2}-norm (m)$' )


fig.tight_layout()
fig.savefig( 'Q2.png', dpi=300)
```