Computation of Shallow Water Equation-HW3

Student ID: r07525117 Student Name: Chang-Syuan Syu

1. Governing Equations

In this study, I will use basic numerical methods to solve the linear shallow water equation (LSWE) in 1DH:

$$\begin{cases} \dfrac{\partial \eta}{\partial t} + \dfrac{\partial hU}{\partial x} = 0 \\ \dfrac{\partial U}{\partial t} = -g\dfrac{\partial \eta}{\partial x} \end{cases} \tag{1}$$

Where η is the free surface elevation, U is the horizontal flow velocity, h is the still water depth, and g is the gravitational acceleration.

2. Discretization

We can employ the third-order Strong Stability-Preserving Runge-Kutta method in time and fourth-order central difference method in space to discretize (1) as

At each time step (n), we need to perform three rounds of calculations in order to get the information at the next time step (n+1); two intermediate are needed, (*) and (**).

The numerical scheme to update $\eta^{(n)}$ and $U^{(n)}$ to $\eta^{(n+1)}$ and $U^{(n+1)}$ looks like this:

First round:

$$\begin{cases} \eta_i^{(*)} = \eta_i^{(n)} - \dfrac{\Delta t}{12\Delta x}\left(-U_{i+2}^{(n)}h_{i+2} + 8U_{i+1}^{(n)}h_{i+1} - 8U_{i-1}^{(n)}h_{i-1} + U_{i-2}^{(n)}h_{i-2}\right) \\ U_i^{(*)} = U_i^{(n)} - \dfrac{\Delta t}{12\Delta x}g\left(-\eta_{i+2}^{n} + 8\eta_{i+1}^{n} - 8\eta_{i-1}^{n} + \eta_{i-2}^{n}\right) \end{cases}$$

Second round:

$$\begin{cases} \eta_i^{(**)} = \dfrac{3}{4}\eta_i^{(n)} + \dfrac{1}{4}\eta_i^{(*)} - \dfrac{\Delta t}{48\Delta x}\left(-U_{i+2}^{(*)}h_{i+2} + 8U_{i+1}^{(*)}h_{i+1} - 8U_{i-1}^{(*)}h_{i-1} + U_{i-2}^{(*)}h_{i-2}\,)\right) \\ U_i^{(**)} = \dfrac{3}{4}U_i^{(n)} + \dfrac{1}{4}U_i^{(*)} - \dfrac{\Delta t}{48\Delta x}g\left(-\eta_{i+2}^{(*)} + 8\eta_{i+1}^{(*)} - 8\eta_{i-1}^{(*)} + \eta_{i-2}^{(*)}\right) \end{cases}$$

Third round:

$$\begin{cases} \eta_i^{(n+1)} = \frac{1}{3}\eta_i^{(n)} + \frac{2}{3}\eta_i^{(**)} - \frac{\Delta t}{18\Delta x}\left(-U_{i+2}^{(**)}h_{i+2} + 8U_{i+1}^{(**)}h_{i+1} - 8U_{i-1}^{(**)}h_{i-1} + U_{i-2}^{(**)}h_{i-2}\right) \\ U_i^{(n+1)} = \frac{1}{3}U_i^{(n)} + \frac{2}{3}U_i^{(**)} - \frac{\Delta t}{18\Delta x}g\left(-\eta_{i+2}^{(**)} + 8\eta_{i+1}^{(**)} - 8\eta_{i-1}^{(**)} + \eta_{i-2}^{(**)}\right) \end{cases} \tag{2}$$

where (n) denotes variables at the current time step, (n + 1) denotes variables at the new time step, i denotes the i-th discretized node in space, Δx is the step size in space, and Δt is the step size in time

The relation between Δx and Δt is required by the CFL (Courant, Friedrichs, and Lewy) condition:

$$\Delta t = C_{\text{CFL}} \frac{\Delta x}{\sqrt{g h_0}}, \tag{3}$$

where the constant C_{CFL} is often referred to as the *Courant number* or the CFL number, and h_0 is the *maximum water depth* in the problem.

In practice, $C_{\text{CFL}} = 0.9$ is found to be an optimal choice, which I will use in this assignment.

3. Boundary Conditions

Let us consider a numerical wave flume spanning from x = -12 m to x = 24 m. The water depth is constant, $h = h_0 = 0.3$ m. The two ends of the wave flume are solid walls. At the left end, x = -12 m for example, the wall boundary condition means

$$\begin{cases} U_{i=1} = 0 \\ \dfrac{\partial U}{\partial t} = 0 = -g\dfrac{\partial \eta}{\partial x} \rightarrow \left(\dfrac{\partial \eta}{\partial x}\right)_{i=1} = 0 \end{cases} \tag{4}$$

This also means the ghost cells to the left of x = -12 have the values

$$\begin{cases} \eta_{i=0} = \eta_{i=4} \\ \eta_{i=1} = \eta_{i=3} \\ U_{i=0} = 0 \\ U_{i=1} = 0 \end{cases} \tag{5}$$

4. Initial Conditions

I specify the initial conditions - η (x, 0) and U(x, 0). In constant water depth, we know that any wave of translation (平移波) moving at the speed $\sqrt{gh}$ is a solution to the 1DH LSWE (1). This can be verified by checking that a function of the form f(x - $\sqrt{gh} \cdot$ t) is a solution to (1) in constant water depth.

In an addition, based on the linear wave theory the horizontal flow velocity for linear shallow water waves can be determined as

$$U(x,t) = \frac{\eta(x,t)}{h}\sqrt{gh} \tag{6}$$

As the initial conditions for this assignment, I will use a wave of translation of the form

$$\eta(x,t) = H sech^2\big(K(x - Ct)\big), \qquad K = \frac{1}{h}\sqrt{\frac{3H}{4h}}, \tag{7}$$

where H is the wave height, C denotes the wave speed ($C = \sqrt{gh}$ for LSWE), and sech(x) is the hyperbolic secant function. K can be seen as the effective wave number for this wave, and an effective wavelength L can be defined as

$$L = \frac{2\pi}{K} \tag{8}$$

An effective wave period T can also be defined:

$$T = \frac{L}{C} \tag{9}$$

A wave of the form (7), whose flow velocity can be calculated from (6), is called the *solitary wave* (孤立波). It is often used as a benchmark wave in many long-wave studies. I will use H = 0.04 m. In a water depth of h = 0.3 m, this means that the effective wavelength is L = 5.961 m, and the effective wave period is T = 3.475 s.

The initial conditions, i.e., η(x; 0) and U(x; 0), to be used in the simulations shown in the Figure 1.

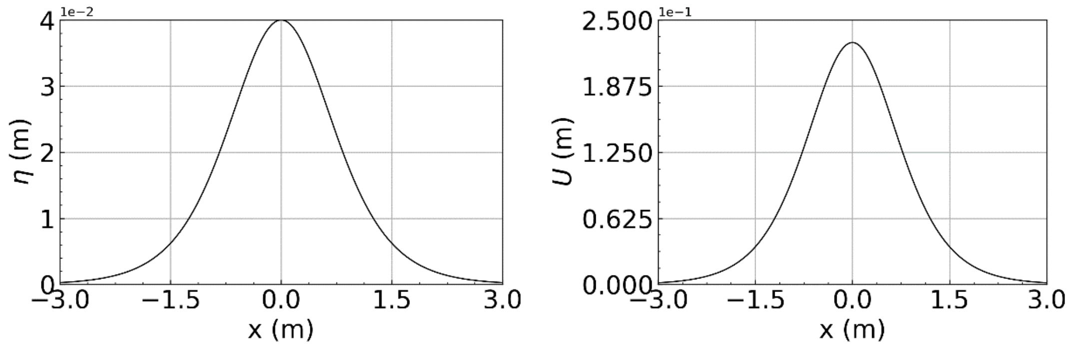

Figure 1: Initial value plot

5. Solve the 1DH LSWE

In this study, I written a program to solve the 1DH LSWE using these initial conditions, for -12 < x < 24 (m) and 0 < t < 6.95 (s). I try using the step size △x = 0.03 m as a start. The codes are shown in the appendix.

6. Validation

In order to make sure my code runs correctly. I compared my numerical results for

η against the analytical solution at t = 6.95 s, i.e., (7). The results are shown in Figure 2

7. Behavior on Different Step Sizes

I run several simulations with different step sizes Δ x at physical time equal 6.95. The results are shown in Figure 2.

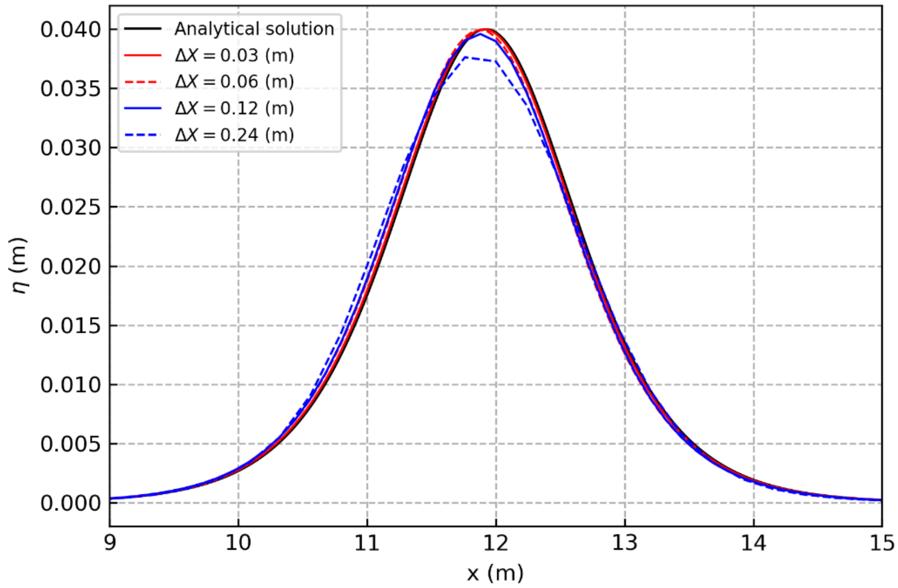

Figure 2: Reference plot for topic 6 and 7.

8. Grid Dependency Test

The results are shown in Figure 3 which shown that the convergence rate of the Lax-Friedrichs numerical scheme is first-order.

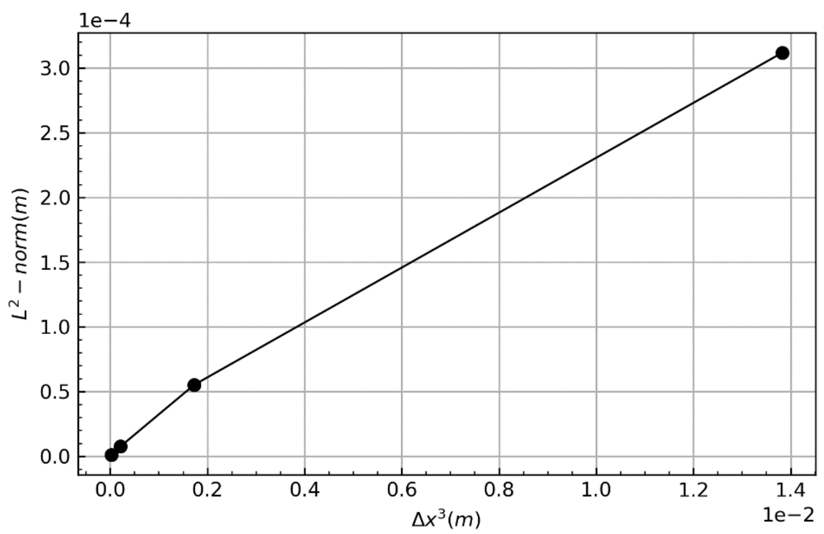

Figure 3: Reference plot for topic 8.

9. Behavior of wave with stepped depth

I pick the $\Delta x = 0.03$ and $t = 13.9s$, and change the water depth for $x > 12\ m\ to\ h_2 = 0.1\ m\ (so\ that\ the\ water\ depth\ for - 12 \leq x \leq 12\ m\ is\ h_1 = 0.3\ m)$. I use the hyperbolic tangent function, to obtain a "smooth jump". The function of stepped depth as

$$h(x) = 0.3 - 0.2\frac{1 + \tanh\big(a(x - 12)\big)}{2},$$

where the constant a controls the smoothness of this jump (set a = 8 as a start).

The wave height ratios determined from my numerical simulation are $\frac{H_T}{H_I} = 1.2741$, $\frac{H_R}{H_I} = 0.2465$, while the analytical solutions are

$$\frac{H_T}{H_I} = \frac{2C_1}{C_1 + C_2} \cong 1.268, \qquad \frac{H_R}{H_I} = \frac{C_1 - C_2}{C_1 + C_2} \cong 0.268$$

Appendix

```
# -* coding: utf-8 -*-
"""
Created on Mon Mar 30 11:37:18 2020

@author: Kingsley
"""



import os
os.chdir('D:\ShallowWaterComputation\HW3')
import numpy as np
import matplotlib.pyplot as plt
from numba import jit

g = 9.81

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

def eta(x, h=0.1, 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, h=0.1, t=0):
    TEMP = eta(x,h,t)
    return TEMP *np.sqrt(g*h) / float(h)

def h(x , a = 8):
    return 0.3-0.2*( 1 + np.tanh( a*(x-12) ) )/2.0
# def h(x):
#    return 0.3
```

```
Ccfl = 0.9

@jit
def Numeric(deltaX,End_Time):
    th = 0.3
    deltaT = float(deltaX)/np.sqrt(g*th)*Ccfl

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

    tlen = len(t)
    xlen = len(x)+4

    ETA = np.zeros( ( tlen, xlen ) )
    Vel_U = np.zeros( ( tlen, xlen ) )

    ETAs1 = np.zeros( ( tlen, xlen ) )
    Vel_Us1 = np.zeros( ( tlen, xlen ) )

    ETAs2 = np.zeros( ( tlen, xlen ) )
    Vel_Us2 = np.zeros( ( tlen, xlen ) )

    ###get ghost cell and BCs
    x = np.append(x, [ x[-1]+deltaX, x[-1]+2*deltaX ] )
    x = np.append( [ x[0]-2*deltaX, x[0]-deltaX ], x )

    ##Initialization
    for i in range(2,xlen-2):
        ETA[0][i] = eta( x[i], h(x[i]) )

    for i in range(2,xlen-2):
        Vel_U[0][i] = U( x[i], h(x[i]) )

    n = 0
    while( n < (tlen-1) ):
        #BC
        ETA[n][0] = ETA[n][4]
```

```
        ETA[n][1] = ETA[n][3]
        ETA[n][-1] = ETA[n][-5]
        ETA[n][-2] = ETA[n][-4]
        Vel_U[n][0] = 0
        Vel_U[n][1] = 0
        #Vel_U[n][2] = 0
        Vel_U[n][-1] = 0
        Vel_U[n][-2] = 0
        #Vel_U[n][-3] = 0
        #Cell
        for i in range(2,xlen-2):
            ETAs1[n][i] = ETA[n][i] - 
( (deltaT)/(12*deltaX) )*( -Vel_U[n][i+2]*h(x[i+2]) + \
            8*Vel_U[n][i+1]*h(x[i+1]) - 8*Vel_U[n][i-1]*h(x[i-
1]) + Vel_U[n][i-2]*h(x[i-2]) )
            Vel_Us1[n][i] = Vel_U[n][i] - 
( (deltaT*g)/(12*deltaX) ) * \
            ( -ETA[n][i+2] + 8*ETA[n][i+1] - 8*ETA[n][i-1] + 
ETA[n][i-2] )
        ETAs1[n][0] = ETAs1[n][4]
        ETAs1[n][1] = ETAs1[n][3]
        ETAs1[n][-1] = ETAs1[n][-5]
        ETAs1[n][-2] = ETAs1[n][-4]
        Vel_Us1[n][0] = 0
        Vel_Us1[n][1] = 0
        #Vel_Us1[n][2] = 0
        Vel_Us1[n][-1] = 0
        Vel_Us1[n][-2] = 0
        #Vel_Us1[n][-3] = 0
        for i in range(2,xlen-2):
            ETAs2[n][i] = (3.0/4.0)*ETA[n][i] + 
(1.0/4.0)*ETAs1[n][i] - ( deltaT/(4.0*12*deltaX) ) \
            * ( -Vel_Us1[n][i+2]*h(x[i+2]) + 
8*Vel_Us1[n][i+1]*h(x[i+1]) \
            -8*Vel_Us1[n][i-1]*h(x[i-1]) + Vel_Us1[n][i-
2]*h(x[i-2]) )
            Vel_Us2[n][i] = (3.0/4.0)*Vel_U[n][i] + 
(1.0/4.0)*Vel_Us1[n][i] - ( (deltaT*g) / (4*12*deltaX) ) \
```

```
           *( -ETAs1[n][i+2] + 8*ETAs1[n][i+1] - 8*ETAs1[n][i-
1] + ETAs1[n][i-2] )
        ETAs2[n][0] = ETAs2[n][4]
        ETAs2[n][1] = ETAs2[n][3]
        ETAs2[n][-1] = ETAs2[n][-5]
        ETAs2[n][-2] = ETAs2[n][-4]
        Vel_Us2[n][0] = 0
        Vel_Us2[n][1] = 0
        #Vel_Us2[n][2] = 0
        Vel_Us2[n][-1] = 0
        Vel_Us2[n][-2] = 0
        #Vel_Us2[n][-3] = 0
        for i in range(2,xlen-2):
            ETA[n+1][i] = (1.0/3.0)*ETA[n][i] +
(2.0/3.0)*ETAs2[n][i] - ( ( 2.0*deltaT )/( 3.0*12*deltaX ) )*
\
            ( - Vel_Us2[n][i+2]*h(x[i+2]) +
8*Vel_Us2[n][i+1]*h(x[i+1]) - 8*Vel_Us2[n][i-1]*h(x[i-1]) \
            + Vel_Us2[n][i-2]*h(x[i-2]) )
            Vel_U[n+1][i] = (1.0/3.0)*Vel_U[n][i] +
(2.0/3.0)*Vel_Us2[n][i] - ( (2.0*deltaT*g)/(3*12*deltaX) ) \
            * ( -ETAs2[n][i+2] + 8*ETAs2[n][i+1] -8*ETAs2[n][i-
1] + ETAs2[n][i-2] )
        n = n+1
    return x,t,ETA,Vel_U


x1,t1,Et1,Vu1 = Numeric(0.03,6.95)
x2,t2,Et2,Vu2 = Numeric(0.06,6.95)
x3,t3,Et3,Vu3 = Numeric(0.12,6.95)
x4,t4,Et4,Vu4 = Numeric(0.24,6.95)

#Analytical Solution
x = np.arange(-12, 24, 0.03)
AnalyticalSol = eta(x,h=0.3,t=6.95)


fig, ax1 = plt.subplots()
```

```
l1 = ax1.plot(x ,AnalyticalSol, '-', color = 'black',
linewidth= 1 )
l2 = ax1.plot( x1, Et1[-1], '-', color = 'r', linewidth= 1)
l3 = ax1.plot( x2, Et2[-1], '--', color = 'r', linewidth= 1)
l4 = ax1.plot( x3, Et3[-1], '-', color = 'b', linewidth= 1)
l5 = ax1.plot( x4, Et4[-1], '--', color = 'b', linewidth= 1)
ax1.set_xlim([9,15])
ax1.grid(linestyle = '--')
ax1.tick_params(which='both',direction='in')
ax1.set_xlabel( 'x (m)' )
ax1.set_ylabel( '${\eta}$ (m)' )

lns =  l1 + l2 + l3 + l4+ l5
labels = [  'Analytical solution', '${\Delta}X = 0.03$ (m)',
'${\Delta}X = 0.06$ (m)' \
         ,'${\Delta}X = 0.12$ (m)' , '${\Delta}X = 0.24$ (m)']
ax1.legend(lns ,labels , loc = 'upper left',prop={'size':8} )

fig.tight_layout()
fig.savefig( 'Q1' , 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

 #Analytical Solution
x1,t1,Et1,Vu1 = Numeric(0.03,6.95)
x2,t2,Et2,Vu2 = Numeric(0.06,6.95)
x3,t3,Et3,Vu3 = Numeric(0.12,6.95)
x4,t4,Et4,Vu4 = Numeric(0.24,6.95)
```

```
AnalySol1 = eta(x1,h=0.3,t=t1[-1])
AnalySol2 = eta(x2,h=0.3,t=t2[-1])
AnalySol3 = eta(x3,h=0.3,t=t3[-1])
AnalySol4 = eta(x4,h=0.3,t=t4[-1])

e1 = norm(Et1[-1],AnalySol1 )
e2 = norm(Et2[-1],AnalySol2 )
e3 = norm(Et3[-1],AnalySol3 )
e4 = norm(Et4[-1],AnalySol4 )


x = np.array([ 0.000027,0.000216, 0.001728, 0.013824])
Error = np.array([e1,e2,e3,e4])

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( '${\Delta}x^3 (m)$ ' )
ax1.set_ylabel( '${L^2}-norm (m)$' )


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


####Q3
x1,t1,Et1,Vu1 = Numeric(0.03,13.9)
fig, ax1 = plt.subplots()
y = h(x1)
```

```
l1 = ax1.plot( x1, Et1[-1]+0.3, '-', color = 'r', linewidth=
1)
l2 = ax1.plot( x1, y, '-', color = 'black', linewidth= 1)

ax1.set_xlim([5,15])


HI = 0.04
HT = max(Et1[-1]) #0.050963
HR = max(Et1[-1][0:600]) ##0.00986167

HT_HI = 1.274075
HR_HI = 0.24654
```