

Universität Hamburg

63-721

Scientific Programming in Fortran

A numerical solution of 2D shallow water model

Dong Jian

MSc. Ocean and Climate Physics
Institute of Oceanography

dong.jian@studium.uni-hamburg.de

Abstract: This report describes the procedures of solving 2D shallow water equations model using FORTRAN and visualization and analysis using PYTHON.

1. Shallow Water Equations Model

The shallow water equations simulate a thin layer of constant density fluid in hydrostatic balance, rotating or not, bounded by a rigid bottom and a free surface. Such a model can describe the propagation of disturbance resulting from gravitational or rotational effects. It can well represent many processes in the ocean and atmosphere, such as the propagation of tsunamis in the ocean, since tsunamis are hundreds of kilometers long waves that propagate over a depth of a few kilometers.

The Shallow water model is a simplified version of the Navier-Stokes equation under the assumptions:

- A uniform water density and in hydrostatic balance.
- Horizontal length scale \gg vertical depth scale.
- The flow velocity is independent of depth and bottom friction is negligible.
- The surface elevation is small.
- The fluid flows over a weekly sloping bottom.

1.1 The 2D Shallow-Water Equations

The fluid motion is fully determined by the principles of conservation of mass and momentum. For simplicity, we ignore friction and nonlinear terms, then a shallow water system can be formulated as:

$$\frac{\partial u}{\partial t} - fv + g \frac{\partial h}{\partial x} = 0 \quad (1)$$

$$\frac{\partial v}{\partial t} + fu + g \frac{\partial h}{\partial y} = 0 \quad (2)$$

$$\frac{\partial h}{\partial t} + d \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) = 0 \quad (3)$$

Where variables $u(x,y,t)$ and $v(x,y,t)$ are components of horizontal velocity, $h(x,y,t)$ is perturbed sea-level elevation (total fluid depth minus mean fluid thickness d), t is time, d is the total water depth at rest, x and y are the two horizontal spatial dimensions. The forces acting on the fluid are gravity, g , and earth rotation contained in the Coriolis parameter f . We can solve the equations and derive velocity fields (u and v) and water elevation (h) using finite differences method given some initial and boundary conditions.

The systems of equations of 1-3 govern the linear wave dynamics of inviscid, homogeneous fluids under rotation.

1.2 Discretization

Solving the equations means to calculate the prognostic velocity field and water elevation at specific time and space coordinates. Using explicit method, all variables at the moment (n+1) are calculated directly from the variables at its previous moment n. we need to discretize the variables in space and time:

$$\begin{aligned} u_{i,j}^n &= u(\text{idx}, \text{jdy}, \text{ndt}) \\ v_{i,j}^n &= v(\text{idx}, \text{jdy}, \text{ndt}) \\ h_{i,j}^n &= h(\text{idx}, \text{jdy}, \text{ndt}) \end{aligned} \quad (4)$$

With the spatial discretization intervals dx and dy, and the temporal discretization interval dt. The spatial and temporal positions are then given by $x = i \, dx$ ($i = 1, 2, \dots, nx$), $y = j \, dy$ ($j = 1, 2, \dots, ny$), $t = n \, dt$ ($n = 1, 2, \dots, nt$)

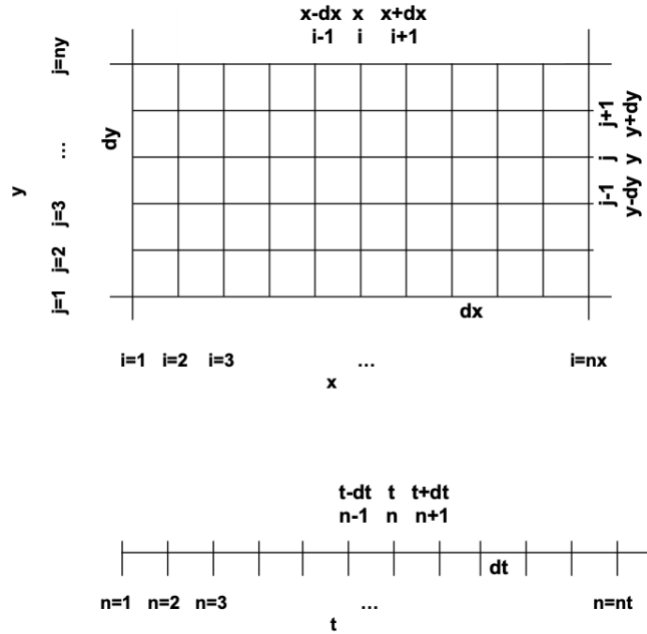


Figure 1 Discretization of space and time (Source: Dr. Nuno Serra)

1.3 Finite-Difference Equations

Using Taylor expansion, we can express derivatives in a discretized form with first order precision or even higher order precision depending on different methods.

$$f(x + \Delta x) = f(x) + \Delta x \frac{\partial f}{\partial x} + \frac{\Delta x^2}{1 \cdot 2} \frac{\partial^2 f}{\partial x^2} + \frac{\Delta x^3}{1 \cdot 2 \cdot 3} \frac{\partial^3 f}{\partial x^3} + \dots \quad (5)$$

the first derivative of a function can be approximated by:

$$\frac{\partial f}{\partial x} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \quad (6)$$

Called forward difference, or

$$\frac{\partial f}{\partial x} \approx \frac{f(x) - f(x - \Delta x)}{\Delta x} \quad (7)$$

Called back difference, or

$$\frac{\partial f}{\partial x} \approx \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} \quad (8)$$

Called centered difference.

For simplicity, the shallow water equations are discretized and integrated using an explicit FTCS (Forward Time Centered Space) scheme, which is conditionally stable depending on the CFL (Courant–Friedrichs–Lewy) condition $dt/2dx$, where dt is the time step of integration and dx is the horizontal spacing of discretization.

With the choice of equidistant grid spacing and index notation, the temporal first derivatives of output variables in shallow water model system can be approximated by forward finite differences:

$$\begin{aligned} \frac{\partial u}{\partial t} &\approx \frac{u_{i,j}^{n+1} - u_{i,j}^n}{dt} \\ \frac{\partial v}{\partial t} &\approx \frac{v_{i,j}^{n+1} - v_{i,j}^n}{dt} \\ \frac{\partial h}{\partial t} &\approx \frac{h_{i,j}^{n+1} - h_{i,j}^n}{dt} \end{aligned} \quad (9)$$

The spatial first derivatives can be approximated by centered finite-differences:

$$\begin{aligned} \frac{\partial u}{\partial x} &\approx \frac{u_{i+1,j}^n - u_{i-1,j}^n}{2dx} \\ \frac{\partial v}{\partial y} &\approx \frac{v_{i,j+1}^n - v_{i,j-1}^n}{2dy} \\ \frac{\partial h}{\partial x} &\approx \frac{h_{i+1,j}^n - h_{i-1,j}^n}{2dx} \\ \frac{\partial h}{\partial y} &\approx \frac{h_{i,j+1}^n - h_{i,j-1}^n}{2dy} \end{aligned} \quad (10)$$

Inserting expressions 5-6 into Equations 1-3 and solving for $u_{i,j}^{n+1}$, $v_{i,j}^{n+1}$ and $h_{i,j}^{n+1}$, the discretized shallow water equations become:

$$u_{i,j}^{n+1} = u_{i,j}^n + f_{i,j} dt v_{i,j}^n - g \frac{dt}{2dx} (h_{i+1,j}^n - h_{i-1,j}^n) \quad (11)$$

$$v_{i,j}^{n+1} = v_{i,j}^n - f_{i,j} dt u_{i,j}^{n+1} - g \frac{dt}{2dy} (h_{i,j+1}^n - h_{i,j-1}^n) \quad (12)$$

$$h_{i,j}^{n+1} = h_{i,j}^n - d_{i,j} \frac{dt}{2dx} (u_{i+1,j}^{n+1} - u_{i-1,j}^{n+1}) - d_{i,j} \frac{dt}{2dy} (v_{i,j+1}^{n+1} - v_{i,j-1}^{n+1}) \quad (13)$$

1.4 Boundary Conditions

The model domain is defined such that the prediction ranges from $i=1$ to $i=n_x$, from $j=1$ to $j=n_y$. The first and last grid cells should be allocated with values, one option is cyclic boundary condition that for variable A in x-direction reads:

$$\begin{aligned} A_{2,j}^n &= A_{n_x,j}^n \\ A_{1,j}^n &= A_{n_x-1,j}^n \end{aligned} \quad (14)$$

It is the same for y-direction. Boundary condition should be applied to all three prognostic variables. Other boundaries conditions are closed boundary, zero-gradient boundary etc.

2. Modelling using Fortran

2.1 Task description

We consider a square domain of 500×500 km in area extent using equidistant lateral grid spacings of $dx = dy = 1000$ meter, and the total water depth at rest is 1000 meter. The domain water is of uniform density, the initial condition will produce a tsunami-type wave train spreading towards southwest. The time step is chosen at $dt=1.0$ s, which satisfies the CFL stability criterion. The simulation is run for 60 minutes with data outputs every 30 seconds.

The solutions to the above finite difference equations subject to initial and boundary conditions.

Lateral boundary conditions are cyclic (also known as periodic), which consist in repeating the values of $i=2$ at $i=n_x$ and the values of $i=1$ at $i=n_x-1$ (same for y-direction) for all three prognostic variables.

The initial condition is specified as a geostrophically-balanced wave train modulated by a localized (at $i=350$, $j=350$) Gaussian function.

All variables are list as below:

Variables	Data type	meanings
$nx=500$	Integer	Total number of grid points in x-direction
$ny=500$	integer	Total number of grid points in y-direction
$nt=3600$	Integer	Total number of grid time steps
$dx=1000.0$	Real	Spatial discretization in meters in x-direction
$dy=1000.0$	Real	Spatial discretization in meters in y-direction
$dt=1.0$	Real	Time step size in seconds
$Dump=30$	Real	Output periodicity in seconds
$g=9.81$	Real	Gravity in meters per squared seconds
$f=0.8e-4$	Real 2D array	Coriolis parameter in one over squared second
$d=1000$	Real 2D array	Total water depth at rest in meters
u	Real 2D array	Velocity in x-direction in meters per second
ub	Real 2D array	Velocity in x-direction of previous time step
v	Real 2D array	Velocity in y-direction in meters per second
vb	Real 2D array	Velocity in y-direction of previous time step
h	Real 2D array	Perturbed elevation in meters
hb	Real 2D array	Perturbed elevation in previous time step
i,j	integer	Indices for x- and y- direction, respectively
n	integer	Time step index

Note: in finite difference equations, u,v,h are variables at time step $t + dt$, ub,vb,hb are variables at time step t .

2.2 Pseudocode

It makes sense to split the codes into several files containing the main program and other different parts. The main code is liked with other external subroutines via the command “CALL” (or encapsulate subroutines in modules and use command “USE”)

The main program will have the following structure:

Program start

Declaration of constants and variables

Initialization of all variables (initialization subroutine `init.f90`)

Open output files for u , v , h

Loop over time steps

Solve equation of u (subroutine)

Solve equation of v (subroutine)

Solve equation of h (subroutine)

Calculate doubly periodic boundary conditions (boundary subroutine)

```

Swap time steps
calculate and write to screen at specified times the maximum value of h
write unformatted at specified times the entire u,v,h to three individual
files
write the entire u,v,h and d to one NetCDF file (subroutine)
end of loop over time steps
close output files
program end

```

the other programs are subroutines including initial condition, boundary condition, equations and NetCDF output, as well as a head file which defines all constants and variables.

2.3 Experiments











Two integrations of model should be conducted:

Case 1: wave train over a flat bottom domain with $H = 1000\text{m}$

Case 2: wave train over a domain with two regions of constant H , in the northeast $H = 1000\text{m}$ and in the southwest $H = 500\text{m}$.

2.4 Results

We finally produce two CDF files for each experiment, which contain there prognostic variables. Each prognostic variable is 3 dimensional, with time, y-position and x-position. 120 time steps in total, each time step is 30 seconds. 500 grids in x-direction, each grid is 1 km, and same in y-direction.

► Dimensions:		(time: 120, xpos: 500, ypos: 500)	
▼ Coordinates:			
time	(time)	datetime64[ns] 2000-01-01 ... 2000-01-01T00:59:30	 
▼ Data variables:			
d	(ypos, xpos)	float32 ...	 
u	(time, ypos, xpos)	float32 ...	 
v	(time, ypos, xpos)	float32 ...	 
h	(time, ypos, xpos)	float32 ...	 
▼ Attributes:			
experiment :	swmPexperimentbase_dateAllocation would exceed memory limitArgument N		
base_date :	[2021 1 1]		

3. Visualization using Python

Python is a popular tool for scientific visualization, here we use Python to produce graphs and animations of model outputs.

3.1 Task Description

The aim of the Python part is to be familiar with some useful packages, such as Matplotlib, Numpy, Xraay etc. and know file input/output, data manipulation. We can write a program to read the NetCDF files and visualize the propagation of waves. We can produce

Hovmoeller diagrams for each experiment and thus compare the differences of the two cases.

3.2 Results

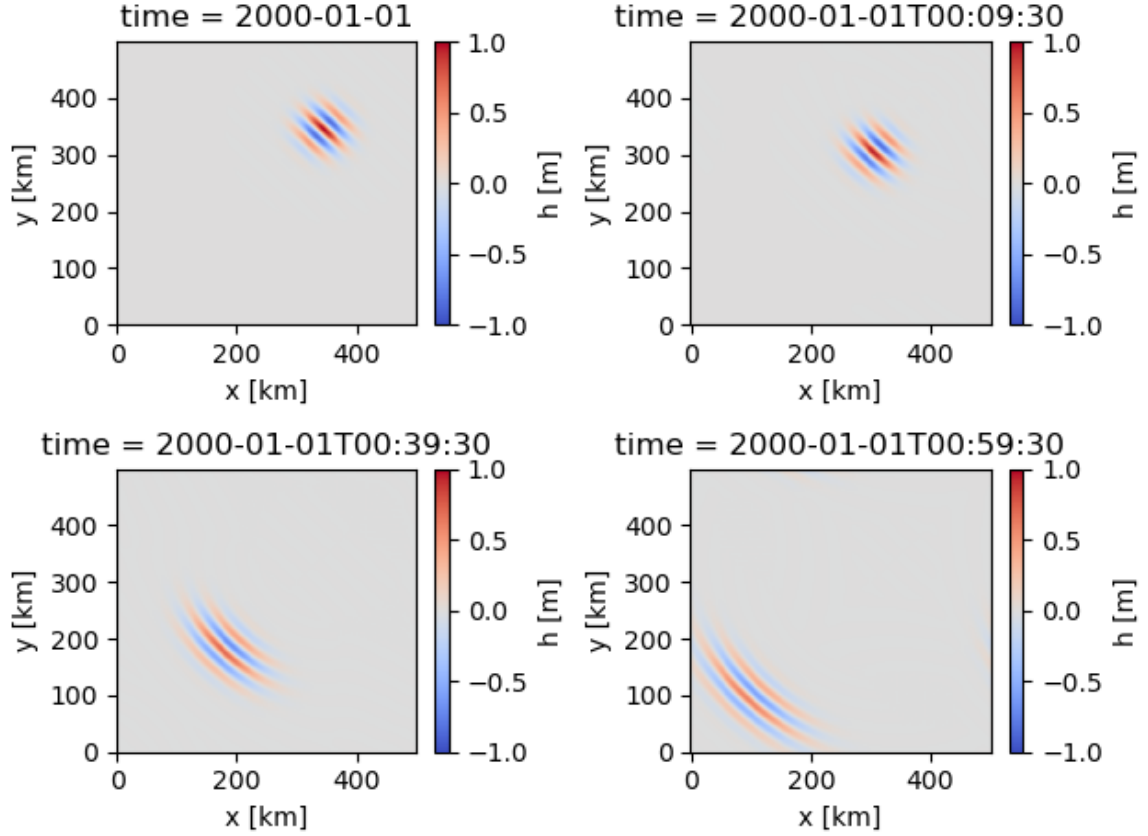


Figure 2 Case 1. Snapshots of water elevation at 1, 20, 80 and 120 time step

In case 1, a geostrophically balanced wave train modulated by a localized Gaussian function propagates toward the southwest. Snapshots at 1, 20, 80 and 120 all represent undisturbed wave, as there is no external forces or boundary.

We can see the sea surface elevation in the domain, dark red represents wave crest, dark blue represents wave trough. We see water elevation is decreasing as the wave propagates. The maximum sea elevation is about 0.99 meters (wave crest at starting position).

At snapshot of 120 time frame, when the wave meets the closed boundary, because we set up a cyclic/periodic boundary condition, we might notice the weak wave signals at the upper and eastern boundaries in the last subplot. (If you use NCVIEW for visualization, it is more obvious and you may notice a weak backward wave, that is produced due to numerical error.)

“A Hovmöller diagram is a common way of plotting meteorological data to highlight the behavior of waves, particularly tropical waves. The axes of Hovmöller diagrams depict changes over time of scalar quantities such as temperature, density, and other values of constituents in the atmosphere or ocean, such as depth, height, or pressure. Typically, in that case, time is recorded along the abscissa, or x-axis, while 'vertical' values (of depth, height, pressure, etc.) are plotted along the ordinate, or y-axis.” (Wikipedia)

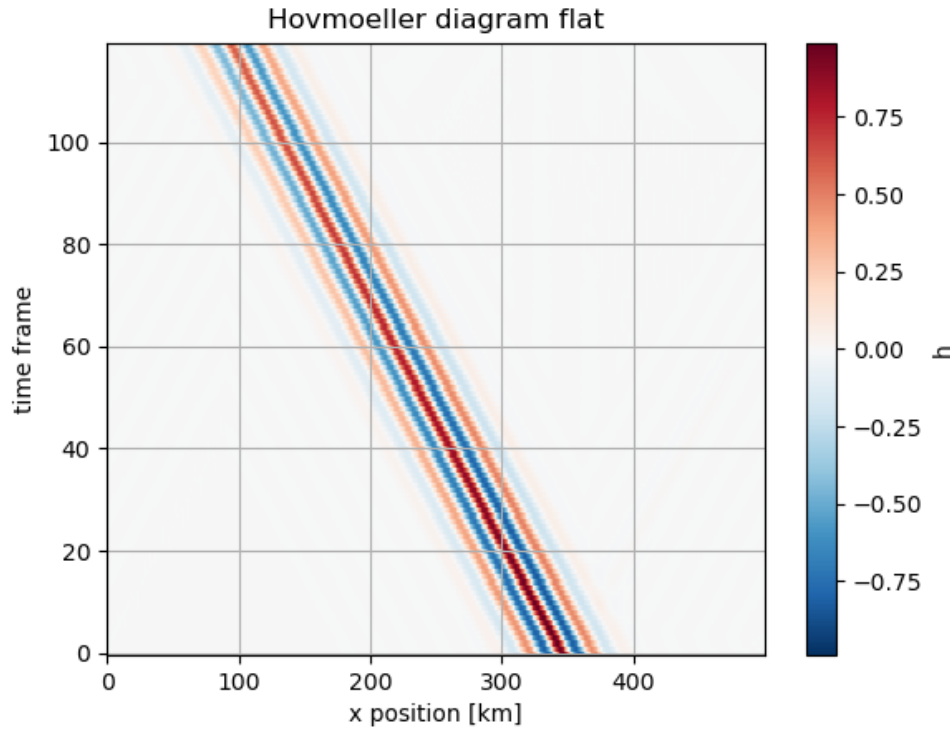


Figure 3 Hovmoeller diagram in case 1

From the Hovmoeller diagram, we can calculate phase speed, at the first-time step, the projection location of wave crest in the x-direction is 350 km, at the last time step, its project location in the x-direction is about 90 km. In 120 time steps, each step is 30 seconds, the wave move $\sqrt{2} \times (350 - 90)$ km. As there is no advection, the propagate speed is also phase speed.

$$\text{Phase speed} = \frac{\text{propagate distance}}{\text{time}} = \frac{(350 - 90)\text{km} \times 1000 \times \sqrt{2}}{120 \times 30\text{s}} \approx 102 \text{ m/s}$$

Theoretically, the phase speed of shallow water waves are governed by the well-known dispersion relation (Ocean modelling for Beginners):

$$c = \frac{\lambda}{T} = \sqrt{gH} \quad (15)$$

$$\text{theoretical value} = \sqrt{gH} = \sqrt{9.8 \times 1000} = 99 \text{ m/s}$$

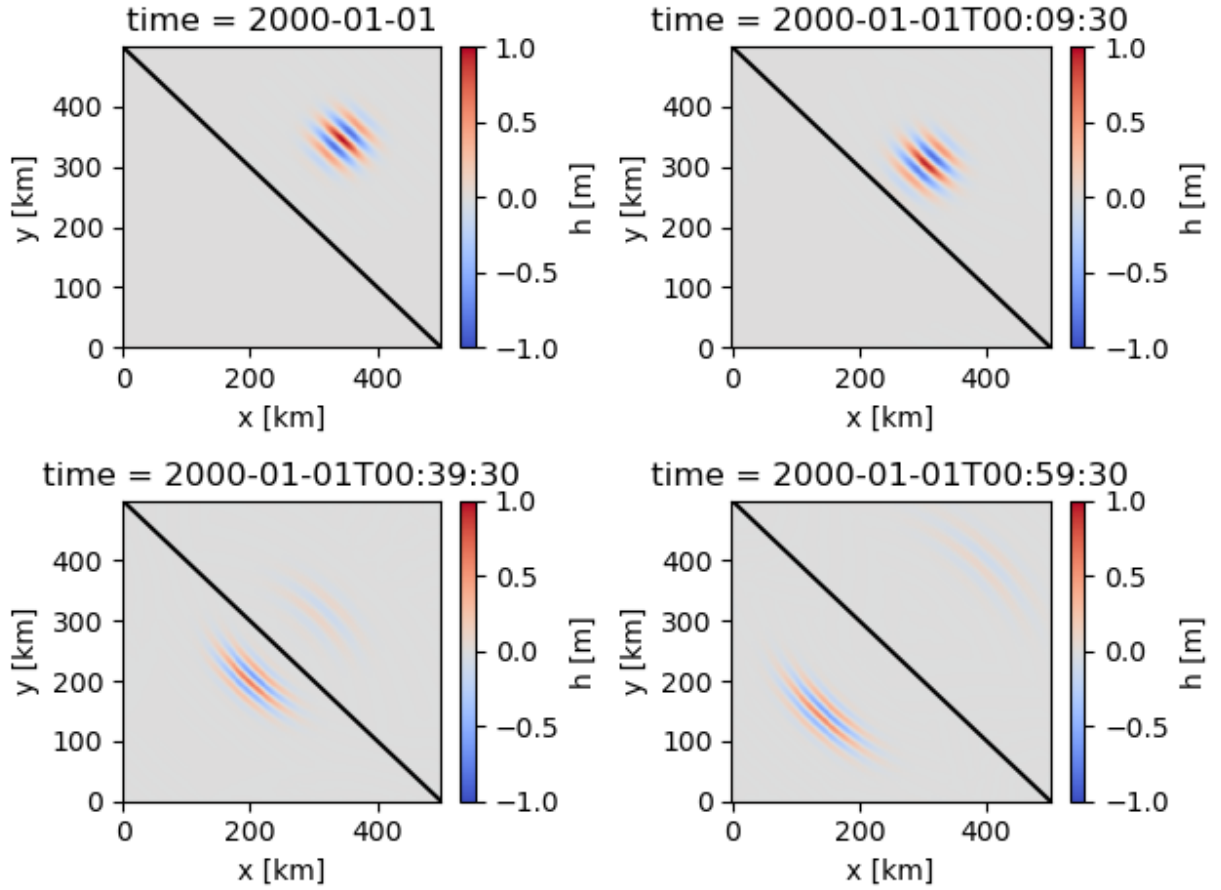


Figure 4 Case 2. Snapshots of water elevation at 1, 20, 80, 120 time step

In case 2, the wave train propagates toward the southwest, before crossing the diagonal, they are undisturbed waves (upper left and upper right). Because the southwest bottom is suddenly changed from 1000 m to 500m, we can see reflected waves in the upper triangles from the lower left and lower right subplots. And wave continues to propagate in the southwest domain is the transmitted wave.

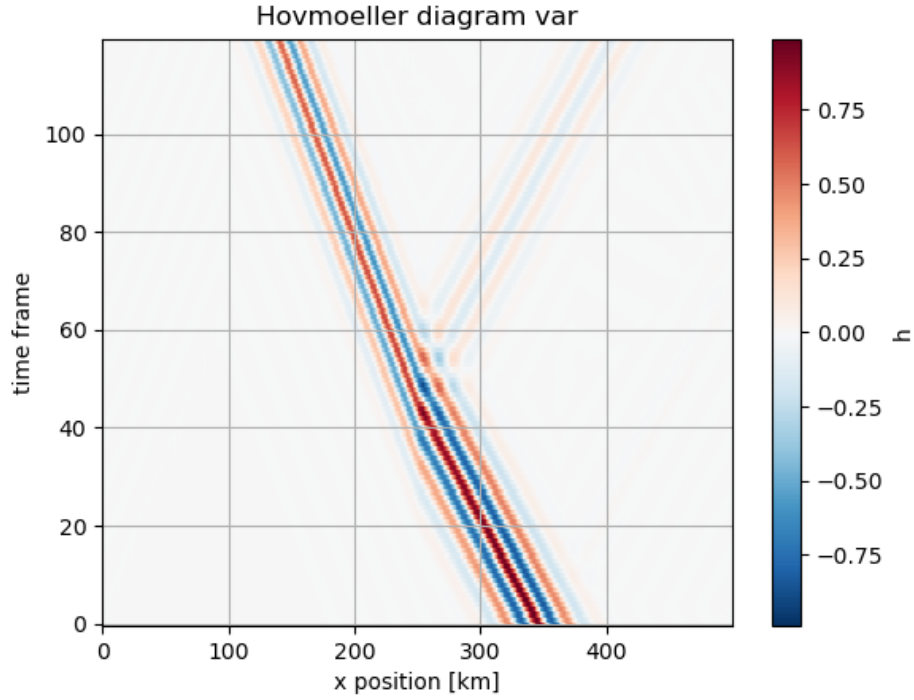


Figure 2 Hovmoeller diagram of case 2

We know waves in case 2 before crossing the diagonal propagates the same way with case 1. It is the undisturbed wave. Now we calculate the phase speed of the transmitted wave in the southwest domain and the reflected wave.

The transmitted wave should be the upper left part of the graph, at the 60-time step, the projection location of wave crest in the x-direction is 230 km, at the 120-time step, its project location in the x-direction is about 140 km.

$$\text{Phase speed} = \frac{\text{propagate distance}}{\text{time}} = \frac{(230 - 140)\text{km} \times 1000 \times \sqrt{2}}{(120 - 60) \times 30\text{s}} \approx 70 \text{ m/s}$$

Theoretical phase speed:

$$\text{theoretical value} = \sqrt{gH} = \sqrt{9.8 \times 500} = 70 \text{ m/s}$$

For the reflected wave, at the 60-time step, the projection location of wave crest in the x-direction is 260 km, at the last time step, its project location in the x-direction is about 390 km.

$$\text{Phase speed} = \frac{\text{propagate distance}}{\text{time}} = \frac{(390 - 260)\text{km} \times 1000 \times \sqrt{2}}{(120 - 60) \times 30\text{s}} \approx 102 \text{ m/s}$$

Which is close to theoretical value $\sqrt{gH} = \sqrt{9.8 \times 1000} = 99 \text{ m/s}$

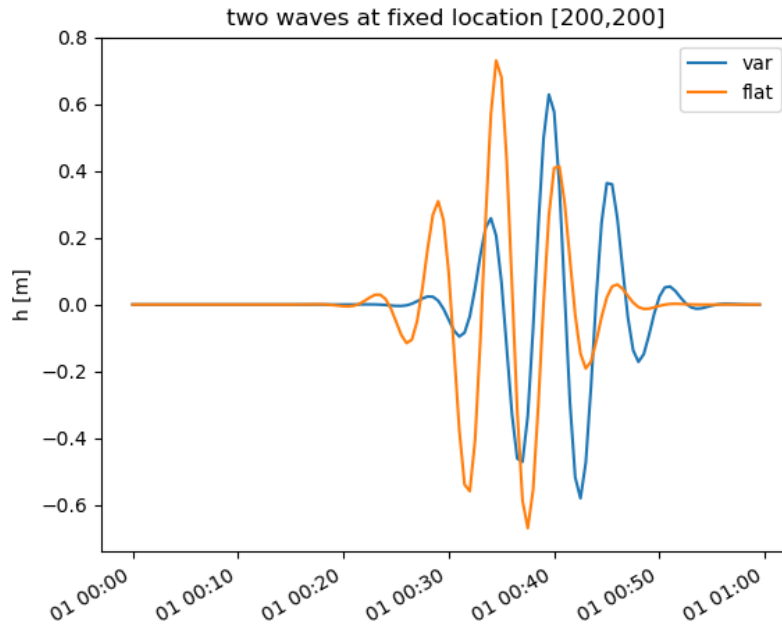


Figure 3 wave periods do not change in the southwestern domain

We randomly choose a fixed position along with the propagation diagonal and then compare how the waves change with time. From the figure, we can see the wave of case 1 moves faster than the wave of case 2 in the southwest domain, since it arrives earlier than the wave of case 2. But the wave period T is basically the same, if we change the location of the wave with flat bottom to [180, 180], we find the wave patterns are overlapped. (see the code in the appendix)

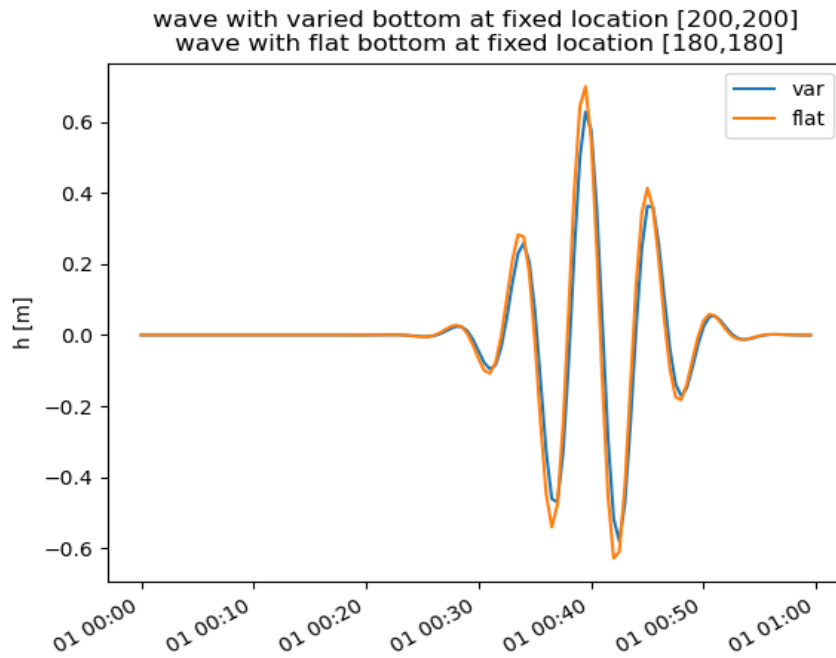


Figure 4 graph showing wave periods do not change

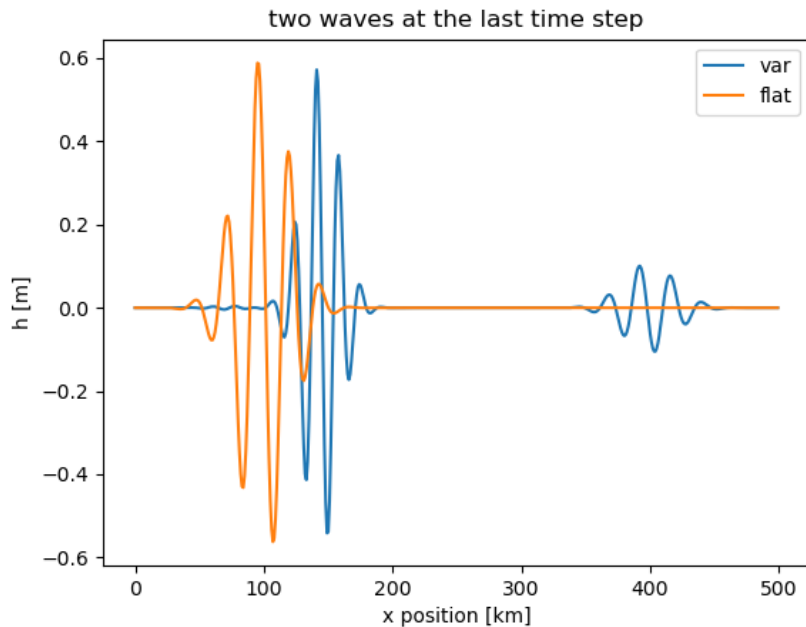


Figure 5 graph showing wavelength difference

we choose the last time step, this graph shows undisturbed wave in case 1 (yellow curve), transmitted wave (blue curve) when water depth changes to 500m in case 2, and reflected wave (blue curve on the right). In the southwestern domain, we find the projection of wavelength of case 2 in x-axis becomes shorter, so will the real wavelength, it looks like been squeezed in the animation, but for reflected wave, the wavelength becomes larger and water elevation is much lower.

4. Conclusion

From the course and this project, we should first understand the dynamics behind the shallow water model. We have learned the basic syntax, data manipulation, function, file input/output of Fortran language. Because Fortran is computationally efficient, we tend to choose Fortran to run a model and produce data. Considering the popularity of Python, it would be helpful to use Python for post-processing and visualization. We have been familiar with the netCDF file and know how to process data using Numpy, Pandas, and Xarray, to visualize data using Matplotlib. Learning to run a model in Fortran and analyzing data using Python is a fun journey. In the end, I would like to express my gratitude for the guidance of Dr. Nuno Serra, Remon Sadikni and my co-coding colleagues.

Appendix

Fortran code

```
swm.f90
1  PROGRAM swm
2    IMPLICIT NONE
3    ! 1. declaration o cons and vars
4    include 'param.h'
5    ! 2. initialization of all variables
6    call init(d, f, u, v, h, ub, vb, hb)
7    !*****
8    ! 3. open output files for u,v,h
9    open (10,file = 'u_exp2.bin', form= 'unformatted', access= 'sequential')
10   open (11,file = 'v_exp2.bin', form= 'unformatted', access= 'sequential')
11   open (12,file = 'h_exp2.bin', form= 'unformatted', access= 'sequential')
12   !*****
13   ! 4. loop over time
14   do n = 1,nt
15     call loop_u(f, ub, vb, hb, u)
16     call loop_v(f, u, vb, hb, v)
17     call loop_h(d, u, v, hb, h)
18     call bound(u, v, h)
19     !!swap time
20     ub = u
21     vb = v
22     hb = h
23     !write max h value to screen at specified time
24     if (mod(n*dt,dump) == 0) THEN
25       write(*,*) maxval(h)
26       write(10) u
27       write(11) v
28       write(12) h
29       !write to netcdf file
30       call outcdf(ub,vb,hb,d)
31     end if
32   end do
33   close(10)
34   close(11)
35   close(12)
36 end program swm
37
38 !===== init.f90 =====
39 SUBROUTINE init(d, f, u, v, h, ub, vb, hb)
40
41   IMPLICIT NONE
42   include 'param.h'
43   REAL :: kx, ky
44   kx = 10.*(2.*pi)/(nx*dx)
```

```

45     ky = 10.*(2.*pi)/(ny*dy)
46     DO i=1,nx
47         DO j=1,ny
48             f(i,j)= 0.8e-4
49             d(i,j)= 1000.
50             !case2
51             d(i,1:ny-i)=500.
52
53             u(i,j)=(-1./(d(i,j)*(kx**2+ky**2))) &
54             *(kx*sqrt(f(i,j)**2+g*d(i,j)*(kx**2+ky**2))) &
55             *cos(kx*i*dx+ky*j*dy)+f(i,j)*ky*sin(kx*i*dx+ky*j*dy)) &
56             *exp(-((0.2*kx*(i-350.)*dx)**2+(0.2*ky*(j-350.)*dy)**2))
57
58             v(i,j)=(1./(d(i,j)*(kx**2+ky**2))) &
59             *(-ky*sqrt(f(i,j)**2+g*d(i,j)*(kx**2+ky**2))) &
60             *cos(kx*i*dx+ky*j*dy)+f(i,j)*kx*sin(kx*i*dx+ky*j*dy)) &
61             *exp(-((0.2*kx*(i-350.)*dx)**2+(0.2*ky*(j-350.)*dy)**2))
62
63             h(i,j)=cos(kx*i*dx+ky*j*dy) &
64             *exp(-((0.2*kx*(i-350.)*dx)**2+(0.2*ky*(j-350.)*dy)**2))
65
66         END DO
67     END DO
68
69     ub = u
70     vb = v
71     hb = h
72 END SUBROUTINE init
73
74 !===== loop_u.f90 =====
75 SUBROUTINE loop_u(f, ub, vb, hb, u)
76     IMPLICIT NONE
77     include 'param.h'
78     DO i=2,nx-1
79         DO j=1,ny
80             u(i,j) = ub(i,j) + (f(i,j) * REAL(dt) * vb(i,j)) - &
81             (g * (REAL(dt)/(2.*dx)) * (hb(i+1,j)- hb(i-1,j)))
82         END DO
83     END DO
84 END SUBROUTINE loop_u

```

```

197  !===== loop_h.f90 =====
198  SUBROUTINE loop_h(d, u, v, hb, h)
199      IMPLICIT NONE
200      include 'param.h'
201      DO i=2,nx-1
202          DO j=2,ny-1
203              h(i,j) = hb(i,j) - (d(i,j) * (REAL(dt)/(2.*dx)) * (u(i+1,j) &
204                  - u(i-1,j))) - (d(i,j) * (REAL(dt)/(2.*dy)) * (v(i,j+1) &
205                  - v(i,j-1)))
206          END DO
207      END DO
208  END SUBROUTINE loop_h
209  !===== bound.f90 =====
210  SUBROUTINE bound(u, v, h)
211      IMPLICIT NONE
212      include 'param.h'
213      !cyclic boundary condition
214      DO j=1,ny
215          u(1,j) = u(nx-1,j)
216          u(nx,j) = u(2,j)
217
218          v(1,j) = v(nx-1,j)
219          v(nx,j) = v(2,j)
220
221          h(1,j) = h(nx-1,j)
222          h(nx,j) = h(2,j)
223      END DO
224
225      DO i=1,nx
226          u(i,1) = u(i,ny-1)
227          u(i,ny) = u(i,2)
228
229          v(i,1) = v(i,ny-1)
230          v(i,ny) = v(i,2)
231
232          h(i,1) = h(i,ny-1)
233          h(i,ny) = h(i,2)
234      END DO
235
236  END SUBROUTINE bound
237  !===== outcdf.f90 provided by Dr.Nuno=====
238

```


Python code

SWM

Import packages and read file using Xarray

```
In [ ]: %matplotlib notebook
import matplotlib.pyplot as plt
plt.ioff() # turn off interactive plotting mode
import numpy as np
import pandas as pd
import xarray as xr
ds_var_bottom = xr.open_dataset('/Users/dong/Desktop/HH/Semester_1/fortran_python_2020/data/swm_var_bottom.cdf')
ds_flat_bottom = xr.open_dataset('/Users/dong/Desktop/HH/Semester_1/fortran_python_2020/data/swm_flat_bottom.cdf')
```

produce animation

```
In [ ]: from matplotlib import animation
fig, ax = plt.subplots()
def animate(i): # i represents one frame in the animation
    data = ds_flat_bottom.h[i,:,:] # slice data advancing one timestep
    ax.cla()
    data.plot(ax=ax, cmap="coolwarm", add_colorbar=False, vmin=-1, vmax=1)
    # Instantiate the animator.
    anim = animation.FuncAnimation(fig,
                                   animate,
                                   frames=120, # 12 frames = 12 months
                                   interval=2, # 200 milliseconds interval between each frame
                                   blit=False, repeat=False) # for blit=True, only the changes are plotted
plt.show()
```

snapshot showing wave propagation of case 1 or 2

```
In [ ]: plt.subplot(2,2,1)
ds_var_bottom.h[0,:,:].plot(cmap="coolwarm",vmin=-1,vmax=1)
plt.xlabel("x [km]")
plt.ylabel("y [km]")
plt.plot([500,0],[0,500],"k-")

plt.subplot(2,2,2)
ds_var_bottom.h[19,:,:].plot(cmap="coolwarm",vmin=-1,vmax=1)
plt.xlabel("x [km]")
plt.ylabel("y [km]")
plt.plot([500,0],[0,500],"k-")

plt.subplot(2,2,3)
ds_var_bottom.h[79,:,:].plot(cmap="coolwarm",vmin=-1,vmax=1)
plt.xlabel("x [km]")
plt.ylabel("y [km]")
plt.plot([500,0],[0,500],"k-")

plt.subplot(2,2,4)
ds_var_bottom.h[119,:,:].plot(cmap="coolwarm",vmin=-1,vmax=1)
plt.xlabel("x [km]")
plt.ylabel("y [km]")
plt.plot([500,0],[0,500],"k-")
plt.tight_layout()

plt.show()
plt.savefig("swm_var_bottom"+"*.png")
```

plot Hovmoeller diagram

```
In [ ]: h_flat_diag = np.zeros((120,500))
        h_var_diag = np.zeros((120,500))
        for i in range(500):
            h_flat_diag[:,i]=(ds_flat_bottom.h[:,i,i]);
            h_var_diag[:,i]=(ds_var_bottom.h[:,i,i])
```

```
In [ ]: ds_flat_diag = xr.Dataset(data_vars={'h': ([ 'time step', 'xpos'], h_flat_diag)})
        ds_flat_diag.h.plot()
        plt.title("Hovmoeller diagram flat")
        plt.xlabel("x position [km]")
        plt.ylabel("time frame")
        plt.grid()
        plt.show()
```

```
In [ ]: ds_var_diag = xr.Dataset(data_vars={'h': ([ 'time', 'xpos'], h_var_diag)})
        ds_var_diag.h.plot()
        plt.title("Hovmoeller diagram var")
        plt.xlabel("x position [km]")
        plt.ylabel("time frame")
        plt.grid()
        plt.show()
```

showing wave period

```
In [ ]: ds_var_bottom.h[:,200,200].plot.line()
        ds_flat_bottom.h[:,180,180].plot.line()
        plt.xlabel("time [s]")
        plt.ylabel("h [m]")
        plt.title(" wave with varied bottom at fixed location [200,200] \n wave with flat bottom at fixed location [180,180]")
        plt.legend(['var', 'flat'])
        plt.xlabel("time")
        plt.show()
        plt.savefig("final.png")
```

showing wave length

```
In [ ]: ds_var_diag["h"][-1,:].plot.line(x="xpos")
        ds_flat_diag["h"][-1,:].plot.line(x="xpos")
        plt.xlabel("x position [km]")
        plt.ylabel("h [m]")
        plt.title("two waves at the last time step")
        plt.legend(['var', 'flat'])
        plt.show()
```