1-D Simulation of Western Boundary Current

Chuning Wang

February 26, 2017

1 Model setup

1.1 Vorticity equation

The numerical model in this report aims to solve the 1-D vorticity equation

$$\frac{\partial}{\partial t}\phi_{xx} + \beta\phi_x = WC - \gamma\phi_{xx} \tag{1}$$

where ϕ is the stream function, the terms on the left side are the 'storage term' and 'β effect term', while terms on the right side are wind curl and 'Stommel' bottom friction, respectively. To solve this equation numerically, the domain is set to a 1-D grid consists of $X+1$ points, and the spatial derivation terms must be discretized with finite difference method:

$$(\phi_x)_i^n = \frac{\phi_{i+1}^n - \phi_{i-1}^n}{2\Delta x} \tag{2}$$

$$(\phi_{xx})_i^n = \frac{\phi_{i+1}^n - 2\phi_i^n + \phi_{i-1}^n}{\Delta x^2} \tag{3}$$

and temporal derivation can be discretized with a variety of schemes, for example, forward scheme

$$\frac{\partial}{\partial t}(\phi_{xx})_i = \frac{(\phi_{xx})_i^{n+1} - (\phi_{xx})_i^n}{\Delta t} \tag{4}$$

or leap frog scheme

$$\frac{\partial}{\partial t}(\phi_{xx})_i = \frac{(\phi_{xx})_i^{n+1} - (\phi_{xx})_i^{n-1}}{2\Delta t} \tag{5}$$

The superscript n and subscript i are indices for time steps and spatial grid points, respectively. Substitute Eq 4 into Eq 1 and rearrange terms

$$(\phi_{xx})_i^{n+1} = (\phi_{xx})_i^n + \Delta t(-\beta(\phi_x)_i^n + WC - \gamma(\phi_{xx})_i^n) \tag{6}$$

then substitute Eq 3 into Eq 6 for time step $n+1$, the forward scheme, discretized 1-D vorticity equation becomes

$$\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1} = \Delta x^2(\phi_{xx})_i^n + \Delta x^2 \Delta t(-\beta(\phi_x)_i^n + WC - \gamma(\phi_{xx})_i^n) \tag{7}$$

or using leap frog scheme

$$\phi_{i+1}^{n+1} - 2\phi_i^{n+1} + \phi_{i-1}^{n+1} = \Delta x^2(\phi_{xx})_i^{n-1} + 2\Delta x^2 \Delta t(-\beta(\phi_x)_i^n + WC - \gamma(\phi_{xx})_i^n) \tag{8}$$

For a 1-D grid consists of $X+1$ points, Eq 7 and 8 satisfy everywhere except at boundaries $i = 0$ or $i = X$.

1.2 Boundary condition

To close the system and complete the linear equations, boundary condition are required. For this particular exercise, we use two sets of boundary conditions: first of which is 'no net transport'

$$\phi = 0 \quad @ \quad x = 0,\ 1 \tag{9}$$

and second is for velocity v at each boundary, which is chosen from either non-slip

$$\phi_x = 0 \quad @ \quad x = 0,\ 1 \tag{10}$$

or free-slip

$$\phi_{xx} = 0 \quad @ \quad x = 0,\ 1 \tag{11}$$

Discretizing Eq 9, 10 and 11 for time step $n+1$ gives

$$\phi_0^{n+1} = 0,\ \phi_X^{n+1} = 0 \tag{12}$$

$$\phi_1^{n+1} - \phi_{-1}^{n+1} = 0,\ \phi_{X+1}^{n+1} - \phi_{X-1}^{n+1} = 0 \tag{13}$$

$$\phi_{-1}^{n+1} - 2\phi_0^{n+1} + \phi_1^{n+1} = 0,\ \phi_{X-1}^{n+1} - 2\phi_X^{n+1} + \phi_{X+1}^{n+1} = 0 \tag{14}$$

where $i = -1$ and $i = X+1$ are two imaginary grid points brought into the system to satisfy the second set of boundary condition.

1.3 Matrix construction

At this point, we have constructed a linear equation system with $X+3$ equations and $X+3$ unknowns. It can be expressed with a matrix form

$$A\phi^{n+1} = B \tag{15}$$

where A is a sparse matrix dependent on the choice of the second boundary condition and B is a 1-D array dependent on the choice of temporal discretization scheme. An example of A with forward scheme, non-slip condition is

$$A = \begin{bmatrix} -1 & 0 & 1 & & & & & & \\ 0 & 1 & 0 & & & & & & \\ & 1 & -2 & 1 & & & & & \\ & & 1 & -2 & 1 & & & & \\ & & & \ddots & \ddots & \ddots & & & \\ & & & & 1 & -2 & 1 & & \\ & & & & & 1 & -2 & 1 & \\ & & & & & & 0 & 1 & 0 \\ & & & & & & -1 & 0 & 1 \end{bmatrix} \tag{16}$$

and B with forward scheme is

$$B = \begin{bmatrix} 0 \\ 0 \\ \Delta x^2(\phi_{xx})_1^n + \Delta x^2 \Delta t(-\beta(\phi_x)_1^n + WC - \gamma(\phi_{xx})_1^n) \\ \Delta x^2(\phi_{xx})_2^n + \Delta x^2 \Delta t(-\beta(\phi_x)_2^n + WC - \gamma(\phi_{xx})_2^n) \\ \vdots \\ \Delta x^2(\phi_{xx})_{X-1}^n + \Delta x^2 \Delta t(-\beta(\phi_x)_{X-1}^n + WC - \gamma(\phi_{xx})_{X-1}^n) \\ \Delta x^2(\phi_{xx})_{X-2}^n + \Delta x^2 \Delta t(-\beta(\phi_x)_{X-2}^n + WC - \gamma(\phi_{xx})_{X-2}^n) \\ 0 \\ 0 \end{bmatrix} \tag{17}$$

The unknown ϕ^{n+1} is also a 1-D array

$$\phi^{n+1} = \begin{bmatrix} \phi_{-1}^{n+1} \\ \phi_0^{n+1} \\ \phi_1^{n+1} \\ \phi_2^{n+1} \\ \vdots \\ \phi_{X-2}^{n+1} \\ \phi_{X-1}^{n+1} \\ \phi_X^{n+1} \\ \phi_{X+1}^{n+1} \end{bmatrix} \tag{18}$$

Iteratively solve Eq 16 for $n = n + 1$ to move forward in temporal domain. Update velocity $v = \phi_x$ and vorticity $\zeta = \phi_{xx}$ for each step and store the solution.

1.4 Numerical tips

- When N is a very large number, matrix A becomes larger accordingly and solving the system eats up a lot of memory. However, since A is a sparse matrix, Python and MATLAB both offer special algorithms to solve the equation system much more efficiently. Make sure to use these algorithms and save some time.

- In this test case five temporal discretization schemes are provided, three explicit (forward, leap frog, Adams-Bashforth) and two implicit (backward, trapezoidal). Note that to use leap frog and Adams-Bashforth scheme two previous time steps are required - thus the first step (from $n = 0$ to $n = 1$) is always solved with trapezoidal scheme.

2 Results

The model is coded in Python, with only standard libraries and modules (Numpy, Scipy, and Matplotlib for plotting) to increase generality. Source code is attached in Appendix A.

Using parameters listed in Appendix A, the model is ran over a total period of 2000 hrs, and the modeled velocity is plotted as Hovmöller diagram in Fig 1. For this run, non-slip boundary condition and trapezoidal discretization scheme is used; other boundary condition and discretization schemes performs similarly thus not presented here. A Rossby wave is generated at the beginning of the run, which propagates westwards and decays quickly in about 50 days. A strong western boundary current is formed, which balances negative transport across the basin.

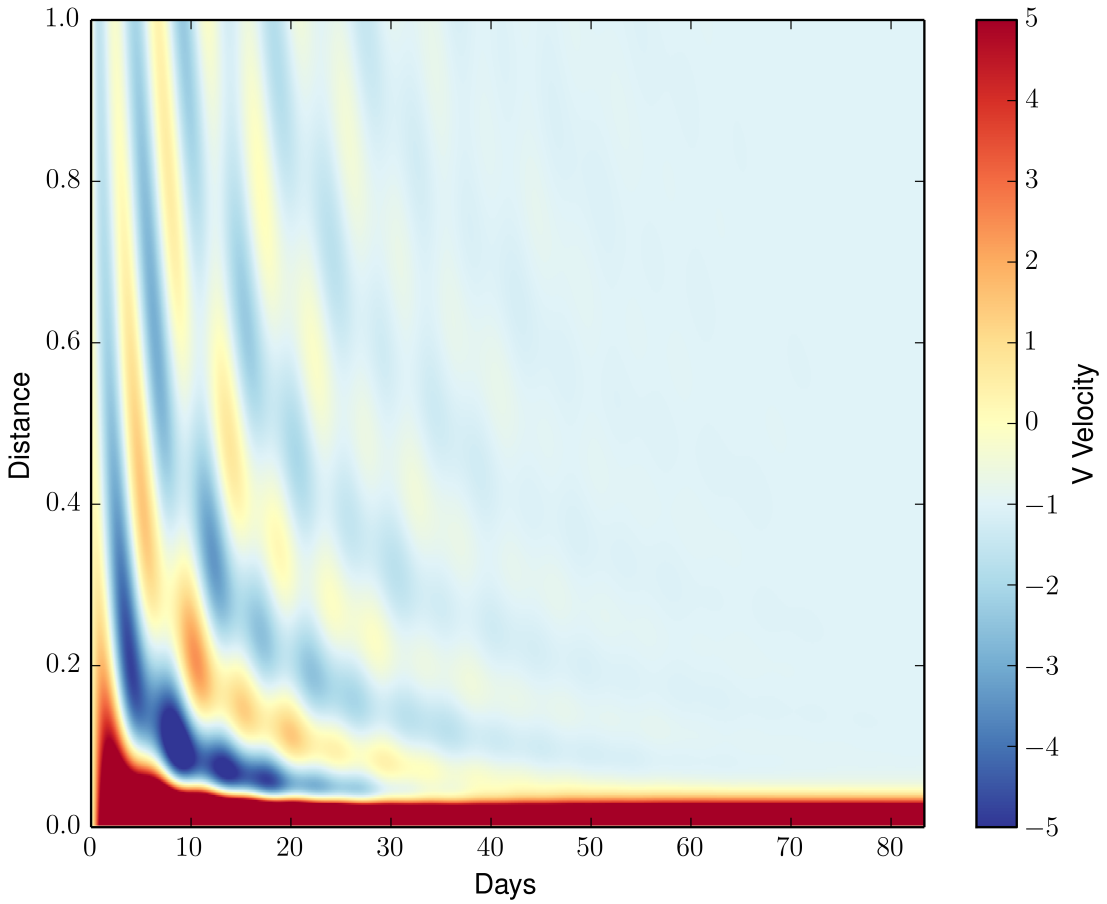

Figure 1: Hovmöller Diagram of modeled velocity with trapezoidal discretization scheme and non-slip boundary condition. Y-axis is distance in terms of 'basin dimension', where 0 is the western boundary and 1 is the eastern boundary.

A Source code

```
import numpy as np
from scipy import sparse
from scipy.sparse.linalg import spsolve
import matplotlib.pyplot as plt

def cal_phi(dt, tn, t, \
            l, xi, dx, x, \
            beta, gamma, wc, \
            bc, dmethod \
           ):
    #-------------------------------------------------------------------
    # initiate variables
```

```
# stream funciton
phi = np.zeros((tn, xi+2))  # [db2s-1]
# v-velocity
v = np.zeros((tn, xi+2))  # [db s-1]
# vorticity
zeta = np.zeros((tn, xi+2))  # [s-1]

# initial condition (if not zero)
phi[0, :] = np.zeros(xi+2)  # [db2s-1]
v[0, 1:-1] = (phi[0, 2:] - phi[0, :-2]) / (2*dx)
zeta[0, 1:-1] = (phi[0, 2:] + phi[0, :-2] - 2*phi[0, 1:-1]) / (dx**2)

# boundary condition
# no penetration
phiw = np.zeros(tn)  # west
phie = np.zeros(tn)  # east
if bc=='noslip':
    # no slip
    vw = np.zeros(tn)  # west
    ve = np.zeros(tn)  # east
elif bc=='freeslip':
    # free slip
    zetaw = np.zeros(tn)  # west
    zetae = np.zeros(tn)  # east

# choose finite differencing parameters
if dmethod == 'forward':
    alpd = 1.
    betd = 0.
    gamd = 0.
    deld = 1.
    epsd = 0.
elif dmethod == 'backward':
    alpd = 1.
    betd = 0.
    gamd = 1.
    deld = 0.
    epsd = 0.
elif dmethod == 'leapfrog':
    alpd = 0.
    betd = 1.
    gamd = 0.
    deld = 2.
    epsd = 0.
elif dmethod == 'trape':
    alpd = 1.
    betd = 0.
    gamd = 0.5
    deld = 0.5
    epsd = 0.
elif dmethod == 'AB':
    alpd = 1.
    betd = 0.
    gamd = 0.
```

```
    deld = 1.5
    epsd = -0.5

#-------------------------------------------------------------------------
# iterate to solve the equation
# construct linear equation matrix
A = -2*sparse.eye(xi+2)*(1+dt*gamd*gamma) + \
    sparse.eye(xi+2, k=1)*(1+dt*gamd*(gamma+beta*dx/2)) + \
    sparse.eye(xi+2, k=-1)*(1+dt*gamd*(gamma-beta*dx/2))
# setup boundary condition
# no penetration
A[1, 0], A[1, 2], A[-2, -1], A[-2, -3] = 0, 0, 0, 0
A[1, 1], A[-2, -2] = 1, 1
# second order
if bc=='noslip':
    A[0, 0] = -1
    A[0, 2] = 1
    A[-1, -3] = -1
    A[-1, -1] = 1
    A[0, 1] = 0
    A[-1, -2] = 0
elif bc=='freeslip':
    A[0, 0] = 1
    A[0, 2] = 1
    A[-1, -3] = 1
    A[-1, -1] = 1
    A[0, 1] = -2
    A[-1, -2] = -2

# for first step
A0 = -2*sparse.eye(xi+2)*(1+dt*0.5*gamma) + \
     sparse.eye(xi+2, k=1)*(1+dt*0.5*(gamma+beta*dx/2)) + \
     sparse.eye(xi+2, k=-1)*(1+dt*0.5*(gamma-beta*dx/2))
# setup boundary condition
# no penetration
A0[1, 0], A0[1, 2], A0[-2, -1], A0[-2, -3] = 0, 0, 0, 0
A0[1, 1], A0[-2, -2] = 1, 1
# second order
if bc=='noslip':
    A0[0, 0] = -1
    A0[0, 2] = 1
    A0[-1, -3] = -1
    A0[-1, -1] = 1
    A0[0, 1] = 0
    A0[-1, -2] = 0
elif bc=='freeslip':
    A0[0, 0] = 1
    A0[0, 2] = 1
    A0[-1, -3] = 1
    A0[-1, -1] = 1
    A0[0, 1] = -2
    A0[-1, -2] = -2

#-------------------------------------------------------------------------
```

```
for n in range(tn-1):

    if n==0:
        # first step always trapezoidal
        B = np.zeros(xi+2)
        B[2:-2] = zeta[n, 2:-2]*dx*dx + \
                  dt*dx*dx*(0.5*wc + \
                            0.5*(-beta*v[n, 2:-2] + wc - gamma*zeta[n, 2:-2]))
        # setup boundary condition
        # no penetration
        B[1] = phiw[n]
        B[-2] = phie[n]
        # second order
        if bc=='noslip':
            B[0] = 2*dx*vw[n]
            B[-1] = 2*dx*ve[n]
        elif bc=='freeslip':
            B[0] = dx**2*zetaw[n]
            B[-1] = dx**2*zetae[n]

        # solve linear equation
        phi[n+1, :] = spsolve(A0, B)
        # update v and zeta
        v[n+1, 1:-1] = (phi[n+1, 2:] - phi[n+1, :-2]) / (2*dx)
        zeta[n+1, 1:-1] = (phi[n+1, 2:] + phi[n+1, :-2] - 2*phi[n+1, 1:-1]) / (dx**2)
        continue

    # ------------------------------------------------------------------------

    B = np.zeros(xi+2)
    B[2:-2] = alpd*zeta[n, 2:-2]*dx*dx + \
              betd*zeta[n-1, 2:-2]*dx*dx + \
              dt*dx*dx*(gamd*wc + \
                        deld*(-beta*v[n, 2:-2] + wc - gamma*zeta[n, 2:-2]) + \
                        epsd*(-beta*v[n-1, 2:-2] + wc - gamma*zeta[n-1, 2:-2])
                       )
    # setup boundary condition
    # no penetration
    B[1] = phiw[n]
    B[-2] = phie[n]
    # second order
    if bc=='noslip':
        B[0] = 2*dx*vw[n]
        B[-1] = 2*dx*ve[n]
    elif bc=='freeslip':
        B[0] = dx**2*zetaw[n]
        B[-1] = dx**2*zetae[n]

    # solve linear equation
    phi[n+1, :] = spsolve(A, B)
    # update v and zeta
    v[n+1, 1:-1] = (phi[n+1, 2:] - phi[n+1, :-2]) / (2*dx)
    zeta[n+1, 1:-1] = (phi[n+1, 2:] + phi[n+1, :-2] - 2*phi[n+1, 1:-1]) / (dx**2)
    return  phi, v, zeta
```

```
# ------------------------------------------------------------------------------
# customized parameters
dt = 0.2*60*60  # delta t [s]
tn = 10000  # number of time steps
t = np.arange(0, dt*tn, dt)

L = 5.0e6  # length of basin [m]
l = 1.  # basin unit length [bd]
xi = int(5e3+1)  # number of horizontal grids
dx = l/(xi-1.)  # grid length in terms of bd [bd]
# add 2 imaginary points, one at each end to deal with boundary condition
x = np.linspace(-dx/l, 1+dx/l, xi+2)  # [db]

beta = 1.0e-4  # rossby parameter [bd-1s-1]
gamma = 1.0e-6  # friction coefficient [s-1]
wb_width = gamma/beta  # west boundary layer width [bd]
wc = -1e-4  # wind curl [s-2]

# methodology switchs
bc1 = 'npslip'  # noslip or freeslip
bc2 = 'freeslip'
dmethod1 = 'trape'  # forward, backward, leapfrog, trape, or AB
dmethod2 = 'leapfrog'
dmethod3 = 'AB'

# ------------------------------------------------------------------------------
# iterate to get solution
phi, v, zeta = cal_phi(dt, tn, t, \
                       l, xi, dx, x, \
                       beta, gamma, wc, \
                       bc1, dmethod1 \
                      )

# phi21, v21, zeta21 = cal_phi(dt, tn, t, \
#                        l, xi, dx, x, \
#                        beta, gamma, wc, \
#                        bc1, dmethod2 \
#                       )
#
# phi31, v31, zeta31 = cal_phi(dt, tn, t, \
#                        l, xi, dx, x, \
#                        beta, gamma, wc, \
#                        bc1, dmethod3 \
#                       )
#
# phi12, v12, zeta12 = cal_phi(dt, tn, t, \
#                        l, xi, dx, x, \
#                        beta, gamma, wc, \
#                        bc2, dmethod1 \
#                       )
#
# phi22, v22, zeta22 = cal_phi(dt, tn, t, \
```

```
#                             l, xi, dx, x, \
#                             beta, gamma, wc, \
#                             bc2, dmethod2 \
#                            )
#
# phi32, v32, zeta32 = cal_phi(dt, tn, t, \
#                             l, xi, dx, x, \
#                             beta, gamma, wc, \
#                             bc2, dmethod3 \
#                            )

# ------------------------------------------------------------------------
# make plots

pltv = 0
if pltv == 1:
    # plot v interactively
    plt.figure()
    plt.show(block=False)
    for i in range(len(t)):
        if (i % 100 == 0):
            plt.plot(v[i, :])
            plt.draw()

# plot the output as hovmuller diagram
plt_hov = 1
if plt_hov == 1:
    plt.figure()
    plt.pcolor(t[::5]/24/60/60, x[::10], v[::5, ::10].T, cmap=plt.cm.RdYlBu_r)
    plt.xlabel('Days')
    plt.ylabel('Distance')
    plt.xlim(0, t[-1]/24/60/60)
    plt.ylim(0, 1)
    plt.clim(-5, 5)
    cb = plt.colorbar()
    cb.ax.set_ylabel(r'V Velocity')
    plt.savefig('v_hovmoller_'+dmethod+'_'+bc+'.png', format='png', dpi=900)
    plt.close()

# plt_all = 0
# if plt_all == 1:
#     f, axarr = plt.subplots(3, 2)
#     axarr[0, 0].pcolor(t[::5]/24/60/60, x[::10], \
#                        v12[::5, ::10].T-v[::5, ::10].T, \
#                        cmap=plt.cm.RdYlBu_r)
#     axarr[0, 0].set_xlabel('Days')
#     axarr[0, 0].set_ylabel('Distance')
#     axarr[0, 0].set_xlim(0, t[-1]/24/60/60)
#     axarr[0, 0].set_ylim(0, 1)
#     # axarr[0, 0].clim(-5, 5)
#     # cb = plt.colorbar()
#     # cb.ax.set_ylabel(r'V Velocity')
#     plt.savefig('v_hovmoller_cmp.png', format='png', dpi=900)
#     plt.close()
```