Project #3:

- Chapter 3: "Two-Layer Flows" Baines.
- **Part 1:** Numerically use the Riemann invariants (Eq 3.3.13) to get solutions for an initial smoothed tophat interface displacement.
  - Explore the parameter space shown in Fig 3.2. What happens as your initial interface is positive or negative?
  - Show examples of shocks formation.
- **Part 2:** See if the above agrees when simulating the interface in MITgcm.
- **Part 3:** In the MITgcm set up some hydraulic jumps and trace out one of the curves in Fig 3.7 for two values of r_u

Part 1:

Here is my code for part 1:

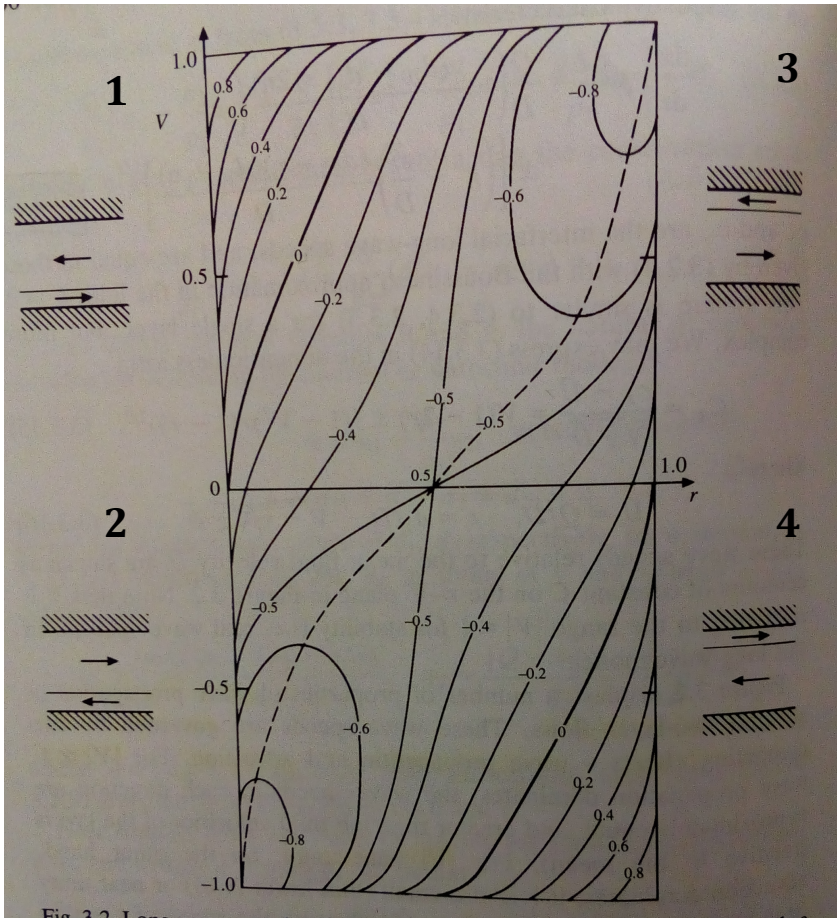
```
# Project 3 Two Layer Flows

from numpy import *
from scipy import *
from pylab import *
import numpy.matlib as matlib
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
from scipy.ndimage import gaussian_filter1d, gaussian_filter

# initializing
D = 2
time_step = 1

x = np.linspace(-100,100,1000)
t = np.r_[0:500:time_step]
NX=len(x)
NT=len(t)

g=9.81
rho_2=1020
rho_1=1000
g_prime=g*(rho_2-rho_1)/rho_2

h=0.1 # height of one depth              # Changing Variable
d_1o=h*D
d_2o=(1-h)*D

u_1o=0#np.sqrt(g_prime*d_1o)
```

```
u_2o=0#np.sqrt(g_prime*d_2o)

v_o=u_1o-u_2o
#eta_o=

# creating initial interface

boundary=np.empty((NX))
H=0.025 #amplitude of bump          # Changing Variable , either positive of
negative

width=10
sigma = 12

for j in range(0, NX):
    if abs(x[j]) > width:
        boundary[j]=0
    if abs(x[j]) < width:
        boundary[j]=H*D

#boundary = H*D*exp(-x*x/(sigma**2))+d_1o
#print shape(boundary)

boundary=gaussian_filter1d(boundary,sigma)

plt.plot(x,boundary)

def calculating_R(eta, v):
    """Calculating the Riemann invariants Eq. 3.3.13 in Baines"""

    R_p = np.arcsin((2*eta+d_1o-d_2o)/D) + np.arcsin(v/(np.sqrt(g_prime*D)))
    R_m = np.arcsin((2*eta+d_1o-d_2o)/D) - np.arcsin(v/(np.sqrt(g_prime*D)))

    return R_p, R_m

def calculating_dx_dt(eta, v):
    """Calculating the dx/dt=C_plus, C_minus the wave speed Eq. 3.3.14 in
Baines"""

    Q=d_1o*u_1o+d_2o*u_2o

    c_p = Q/D + v*(d_2o-d_1o-2*eta)/D + np.sqrt((g_prime-v**2/D)*((d_2o-
eta)*(d_1o+eta)/D))
    c_m = Q/D + v*(d_2o-d_1o-2*eta)/D - np.sqrt((g_prime-v**2/D)*((d_2o-
eta)*(d_1o+eta)/D))
```

```
    return c_p, c_m

def calculating_v(R_p, R_m):

    v=np.sin((R_p-R_m)/2)*np.sqrt(g_prime*D)

    return v

def calculating_eta(R_p, R_m):

    eta=np.sin((R_p+R_m)/2)*D/2-d_1o/2+d_2o/2

    return eta

v_total = np.empty((NT+1, NX))
v_total[0] = v_o*np.ones(NX)

eta_total = np.empty((NT+1, NX))
eta_total[0] = boundary

for j, t in enumerate(t, start=1):

    eta_j = eta_total[j-1]
    v_j = v_total[j-1]

    R_p_j, R_m_j = calculating_R(eta_j, v_j)

    R_p = interp1d(x, R_p_j,kind='linear',fill_value='extrapolate')
    R_m = interp1d(x, R_m_j,kind='linear',fill_value='extrapolate')

    c_p, c_m = calculating_dx_dt(eta_j, v_j)
    x_p = x - c_p*time_step
    x_m = x - c_m*time_step

    R_p_x_p = R_p(x_p)
    R_m_x_m = R_m(x_m)

    v_j = calculating_v(R_p_x_p, R_m_x_m)
    eta_j = calculating_eta(R_p_x_p, R_m_x_m)

    eta_total[j] = eta_j
    v_total[j] = v_j

# Plotting

fig,ax = plt.subplots(1)
```

```
dt=15

for k in range(0, NT, dt):
    plt.plot(x, eta_total[k, :] + 0.01*k/dt, 'b')

plt.xlabel('Distance')
plt.ylabel('Increaing time')
plt.title('Evolution of the interface in time')
ax.tick_params(labelleft='off')
plt.savefig('h_0.1_u1_0.25_u2_0.25_H_pos.pdf')
```

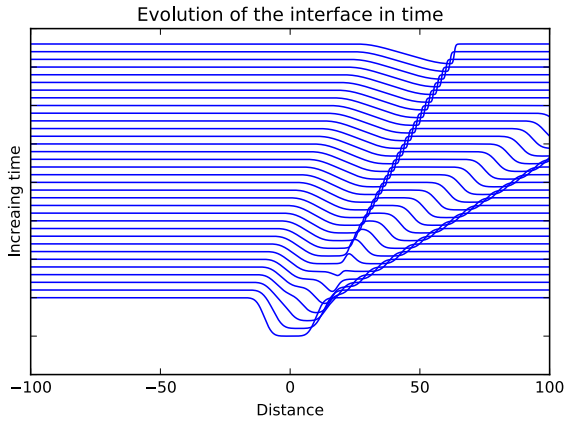
Baines Figure 3.2:



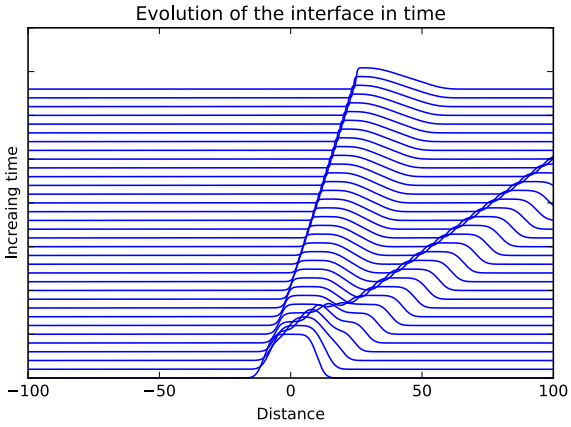
The parameter space I used for each of the 4 locations is given in the table below:

Location		r, thickness of bottom layer	u_{1o}	u_{2o}	Amplitude of initial interface
1	A	0.1	0.25	-0.25	-0.025
	B	0.1	0.25	-0.25	0.025
2	C	0.1	-0.25	0.25	-0.025
	D	0.1	-0.25	0.25	0.025
3	E	0.9	0.25	-0.25	-0.025
	F	0.9	0.25	-0.25	0.025
4	G	0.9	-0.25	0.25	-0.025
	H	0.9	-0.25	0.25	0.025

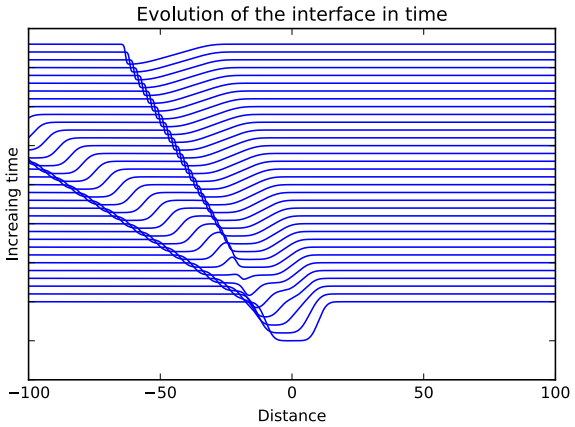
1A



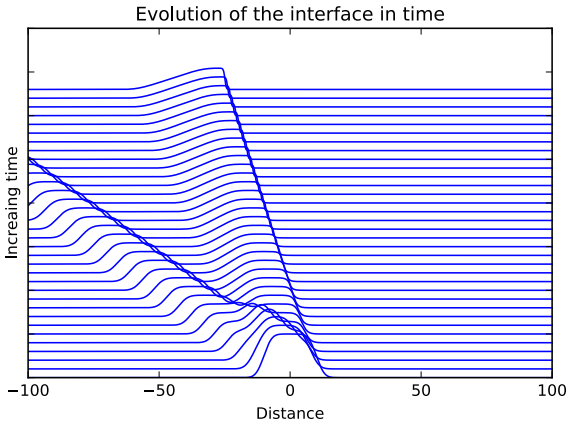
1B



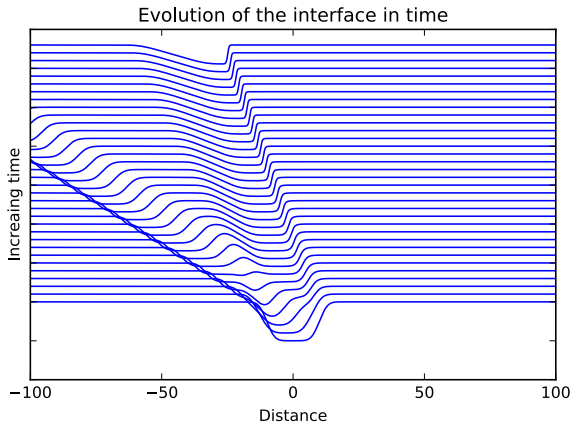
2C



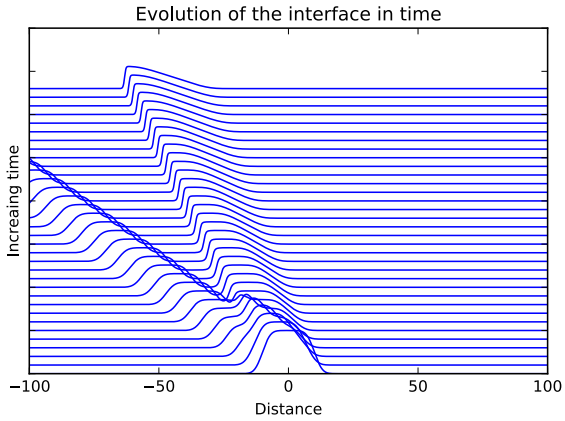
2D



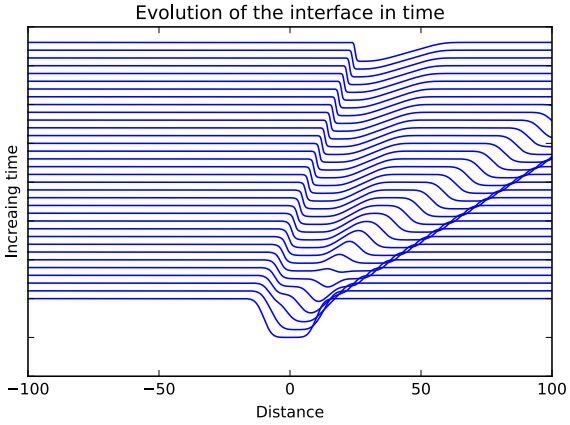
3E



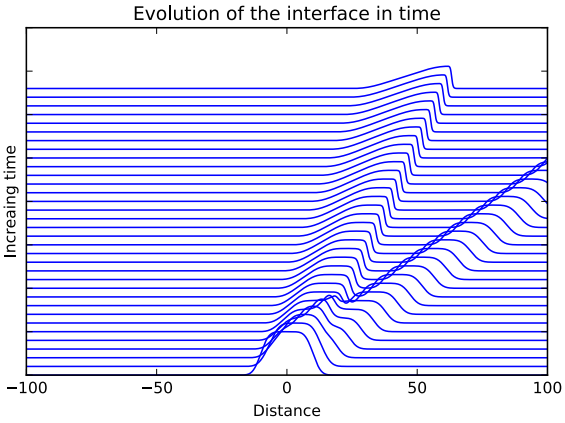
3F



4G



4H



From these plots one can see that the perturbation is advected with the velocity of the thinner layer. The perturbation propagates away as an upstream wave and a downstream wave. How fast they move depend on the relative velocities.

In the case of **1A**, the upstream-propagating wave is actually slowly moving the left because of the background flow. The shape of this wave changes, a shock forms on the right hand side of the wave. The downstream-propagating wave's shape stays the same. If the background velocities were both 0, then the upstream and downstream waves would be symmetrical and no shocks would form.
In the case of **1B**, it is almost identical to 1A, except that the shock forms on the other side of the upstream wave.

In addition one can see that location 1 and 4 (1A, 1B, 4G, 4H) are symmetric and that location 2 and 3 (2C, 2D, 3E, 3F) are also symmetric. This is as expected from Figure 3.2.

Part 2:

Here is my code for part 2:

Gendata.py

```
from numpy import *
from scipy import *
from scipy.ndimage import gaussian_filter1d, gaussian_filter
from pylab import *
import numpy.matlib as matlib
from shutil import copy
from os import mkdir
import os

outdir='../runs/Run_h_0.1_u1_-0.1_u2_0.3_H_pos'

# 4 different regimes
#h=0.1, 0.9
#H is positive or negative
#U is in the same direction or in opposite directions

try:
  mkdir(outdir)
except:
  print outdir+' Exists'
try:
  mkdir(outdir+'/figs')
except:
  print outdir+'/figs Exists'
copy('./gendata.py',outdir)

# These must match ../code/SIZE.h
ny = 1
nx = 4*20
nz = 100

# x direction
xt = 410e3

# dx
dx=zeros(nx)+1000
p1=np.arange(nx/2)
```

```
p2=np.flipud(p1)
p3=np.concatenate((p2,p1), axis=0)
dx=dx*1.02**p3
x=np.cumsum(dx)
x = x-x[nx/2]

with open(outdir+"/delXvar.bin", "wb") as f:
        dx.tofile(f)
f.close()

# plot
if 1:
   plot(x/1000.,dx)
   #xlim([-10,10])
   savefig(outdir+'/figs/dx.pdf')

D = 100

H=0.05 # amplitude of disturbance
width=5000
sigma = 1

h=0.1 # height of one depth
d_1o=h*D
d_2o=(1-h)*D

u_2o=0.3
u_1o=-0.1

# topo
topo=-D*np.ones(nx)


with open(outdir+"/topo.bin", "wb") as f:
        topo.tofile(f)
f.close()

# plot
if 1:
   clf()
   plot(x/1.e3,topo)
   # xlim([-20.,20.])
   savefig(outdir+'/figs/topo.pdf')

# dz
dz=zeros(nz)+1
```

```
z=-np.cumsum(dz)

with open(outdir+"/delZvar.bin", "wb") as f:
        dz.tofile(f)
f.close()

X,Z=np.meshgrid(x,z,indexing='ij')

# Boundary interface
boundary=np.empty(nx)
for j in range(0, nx):
   if abs(x[j]) > width:
      boundary[j]=0
   if abs(x[j]) < width:
      boundary[j]=H*D
boundary=gaussian_filter1d(boundary,sigma)-d_2o

Dbottom= z< -d_2o
Dbottom2= Z < boundary [:,None]

# Temperature profile
gravity=9.81
alpha = 2e-4
g_reduced=0.4E-3*gravity
dT=g_reduced/(gravity*alpha)


Tref=np.ones(nz)*dT
Tref[Dbottom]=0

T0 = dT * np.ones((nx, nz)) #T0 = Tref[:,np.newaxis]*np.ones((nz,nx))
T0[Dbottom2] = 0

# Initial Vel
U_initial = u_2o * np.ones(nz)
U_initial[Dbottom] = u_1o

U0 = u_2o * np.ones((nx, nz)) #U0=U_initial[:,np.newaxis]*np.ones((nz,nx))
U0[Dbottom2] = u_1o

T0=T0.transpose()
U0=U0.transpose()

with open(outdir+"/TRef.bin", "wb") as f:
        Tref.tofile(f)
f.close()
```

```
with open(outdir+"/T0.bin", "wb") as f:
        T0.tofile(f)

with open(outdir+"/Uin.bin","wb") as f:
  U_initial.tofile(f)

with open(outdir+"/U0.bin", "wb") as f:
  U0.tofile(f)

with open(outdir+"/Ue.bin","wb") as f:
  U_initial.tofile(f)

with open(outdir+"/Uw.bin", "wb") as f:
  U_initial.tofile(f)

with open(outdir+"/Te.bin", "wb") as f:
        Tref.tofile(f)
f.close()
with open(outdir+"/Tw.bin", "wb") as f:
        Tref.tofile(f)
f.close()

## Copy some other files
import shutil
shutil.copy('data', outdir+'/data')
shutil.copy('eedata', outdir)
shutil.copy('data.kl10', outdir)
shutil.copy('data.mnc', outdir)
shutil.copy('data.obcs', outdir)
shutil.copy('data.diagnostics', outdir)
shutil.copy('data.pkg', outdir+'/data.pkg')
# also store these.  They are small and helpful to document what we did
for nm in {'input','code','build_options','analysis'}:
    to_path = outdir+'/'+nm
    if os.path.exists(to_path):
        shutil.rmtree(to_path)
    shutil.copytree('../'+nm, outdir+'/'+nm)
```

Here is the matlab code to analyze the MITgcm simulation:

```
% Project 3 Part 2
% MITgcm Two layer flow
```

```
%% Read Data
clear;
NT=53;
data=000;
k=150;

for j=1:NT

   U=0;
   T=0;
   U_0=0;
   T_0=0;

   U = rdmds('U',data, 'n'); % Velocity [m/s]
   T = rdmds('T',data, 'n'); % Temperature [oC]

   U_0=U(:,1,1);
   T_0=T(:,1,1);

   for i=2:100;

      A=U(:,1,i);
      B=T(:,1,i);

      U_0=[U_0,A];
      T_0=[T_0,B];

   end

   U_0=U_0';
   T_0=T_0';

   eval(sprintf('U%d=U_0',j));
   eval(sprintf('T%d=T_0',j));
   eval(sprintf('Time%d=data*12.4/60/60',j));

   data=data+k

end


RC = rdmds('RC', 'n'); % Z [m]

Z=RC(:,1,1);

for i=2:100;
```

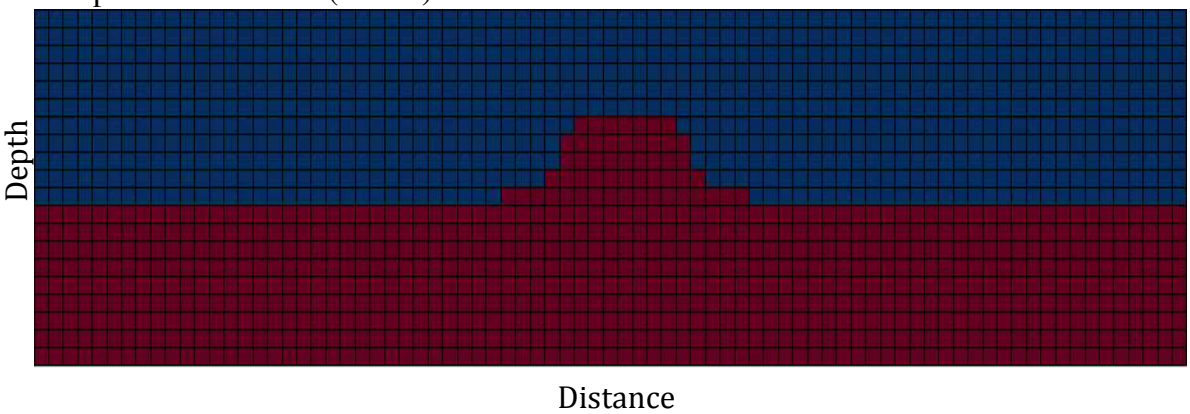
```
    D=RC(:,1,i);
    Z=[Z,D];

end
%%
Depth = rdmds('Depth', 'n');

XC = rdmds('XC', 'n'); % X [m]
XG = rdmds('XG', 'n');

RF = rdmds('RF', 'n');

XC= XC./1000; %distance in km


%% Removing values beneath seafloor

distance= XC;
range=-1*Z;

% for ii=1:80;
%     w=find(abs(Depth(ii)-range')<5);
%
%     for nn=1:50
%         if nn>w(1)
%             U_1(nn,ii)=NaN;
%             U_2(nn,ii)=NaN;
%             U_3(nn,ii)=NaN;
%             T_1(nn,ii)=NaN;
%             T_2(nn,ii)=NaN;
%             T_3(nn,ii)=NaN;
%         end
%     end
% end

%% Figures

figure(1)
% U contours


for j=1:NT
Variable=eval(sprintf('T%d',j));
k=1;
[C h]  = contourf(distance,range,Variable,k);
```

```
eval(sprintf('x%d=C(1,:)',j));
eval(sprintf('y%d=C(2,:)',j));

% Unique x's and their locations
x=eval(sprintf('x%d',j));
y=eval(sprintf('y%d',j));
[ux,~,idx] = unique(x);
% Accumulate
ymean = accumarray(idx,y,[],@mean);
movymean=movmean(ymean,3);

eval(sprintf('X%d=ux',j));
eval(sprintf('Eta%d=movymean',j));

end

%%
colormap(brewermap(21,'RdBu'))
v = VideoWriter('Run7.avi')
open(v)
for j=1:NT
Variable=eval(sprintf('T%d',j));
pcolor(Variable);
%k=[-2:0.3:2];
%contourf(distance, range, Variable,k)
axis ij;
ylim([0 100])
title('Temperature Interface')
c=colorbar; c.Label.String='T [^oC]'; caxis([0 2]);
pause(0.05)
Image(j)=getframe(gca)
writeVideo(v, Image(j))
end
hold on
ylabel('Depth [m]')
xlabel('Distance [km]');
set(gca,'LineWidth',2,'TickLength',[0.025 0.025]); set(gca,'fontsize',20)
%movie(Image)
close(v)
%%
figure (2)

for j=1:3:NT
    hold on
    X=eval(sprintf('X%d',j));
    Y=eval(sprintf('Eta%d',j));
```

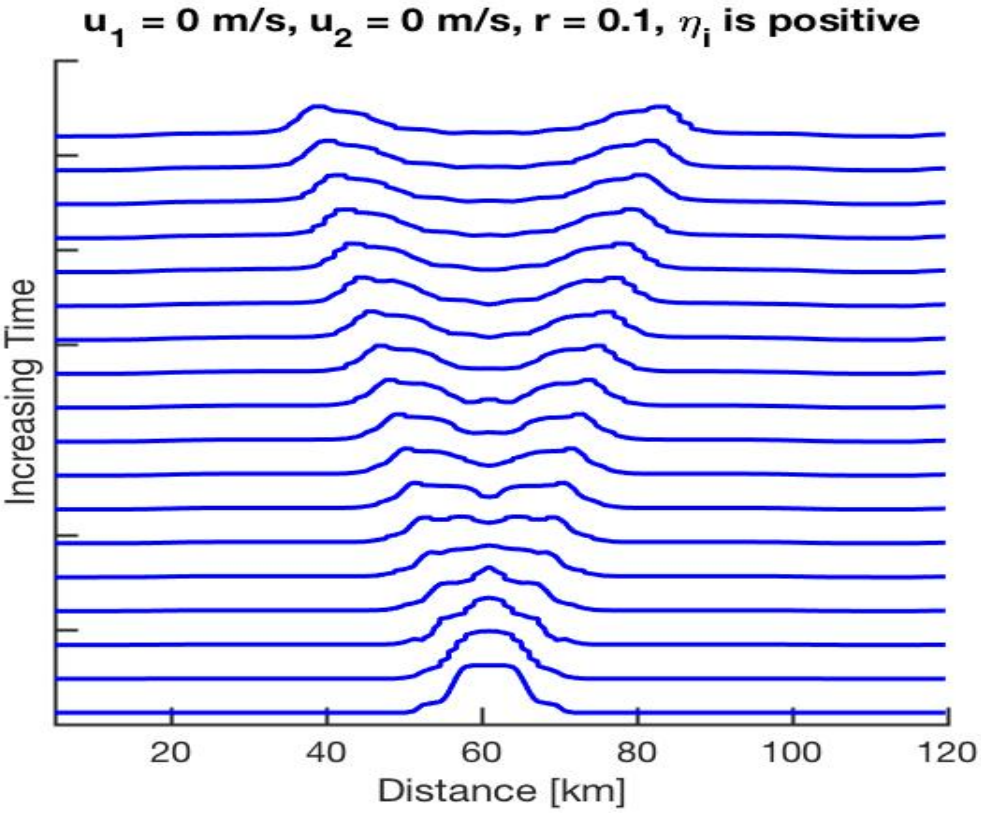
```
    plot(X,5*Y-j*6,'b-','LineWidth',3)
    axis ij
    hold on
end
 hold on
 ylabel('Increasing Time')
 xlabel('Distance [km]');
 title('u_1 = -0.1 m/s, u_2 = 0.3 m/s, r = 0.1, \eta_i is positive')
 %title('r_u = 0.1, u = 0.2')
 %axis ij
 set(gca,'LineWidth',2,'TickLength',[0.025 0.025]); set(gca,'fontsize',20);
 set(gca,'YTickLabel',[])
 xlim([5 120])
```

The first simulation I did was when $u_1 = u_2 = 0$ m/s and r = 0.1. As expected the two waves form and propagate upstream and downstream symmetrically, because there is no background velocity.

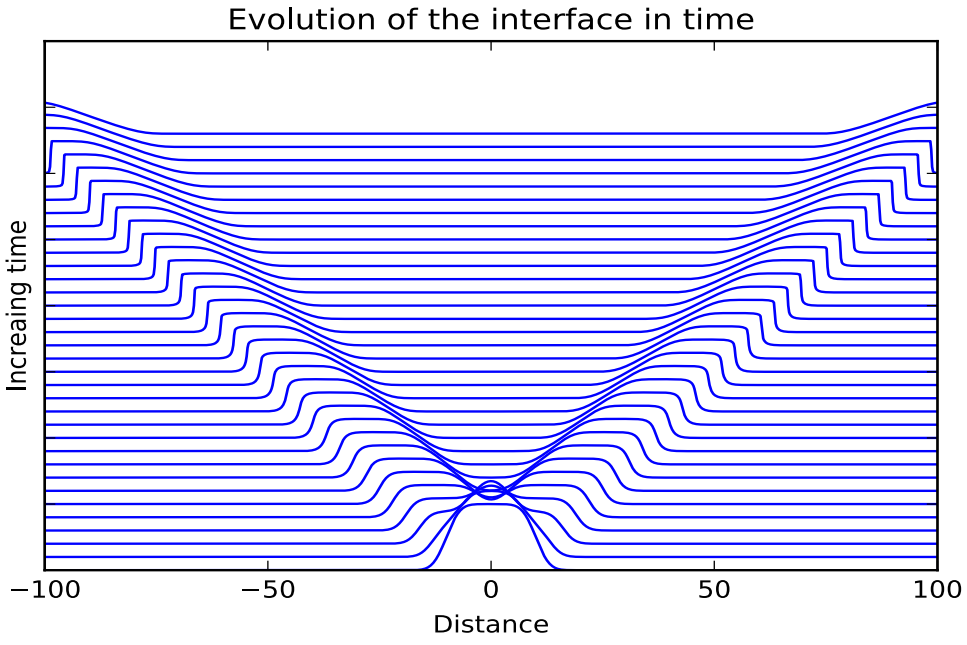
Temperature Interface (Movie)



A summary of the interface in the above movie clip is shown:



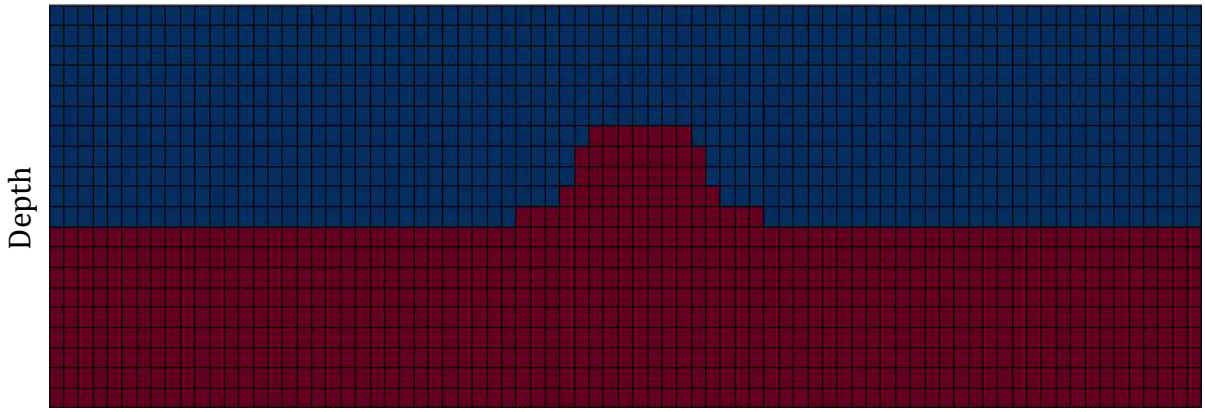
The results from the MITgcm simulation are similar to the result from the method of characteristics, except in the simulation the waves have bumpy, due to the mixing between layers.



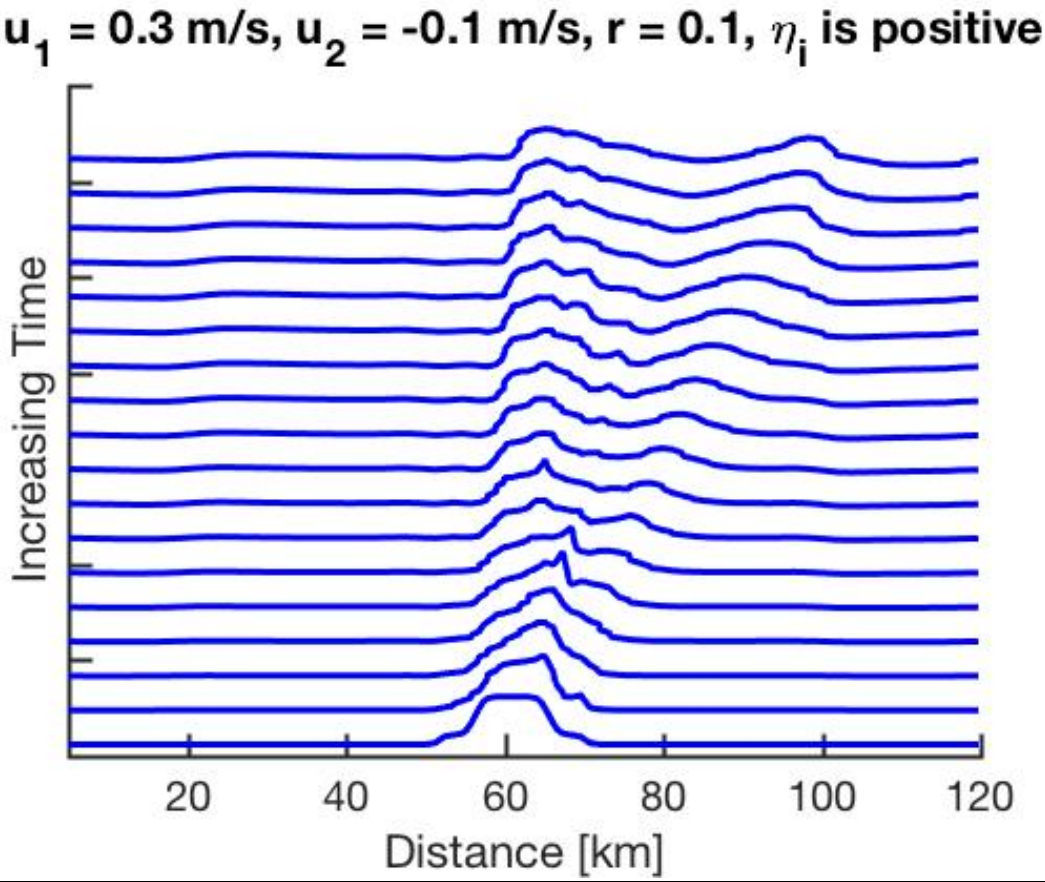
More examples are shown for simulations in Locations 1 and 3.

Location 1: $u_1 = 0.3$ m/s $u_2 = -0.1$ m/s and $r = 0.1$.

Temperature Interface (Movie)

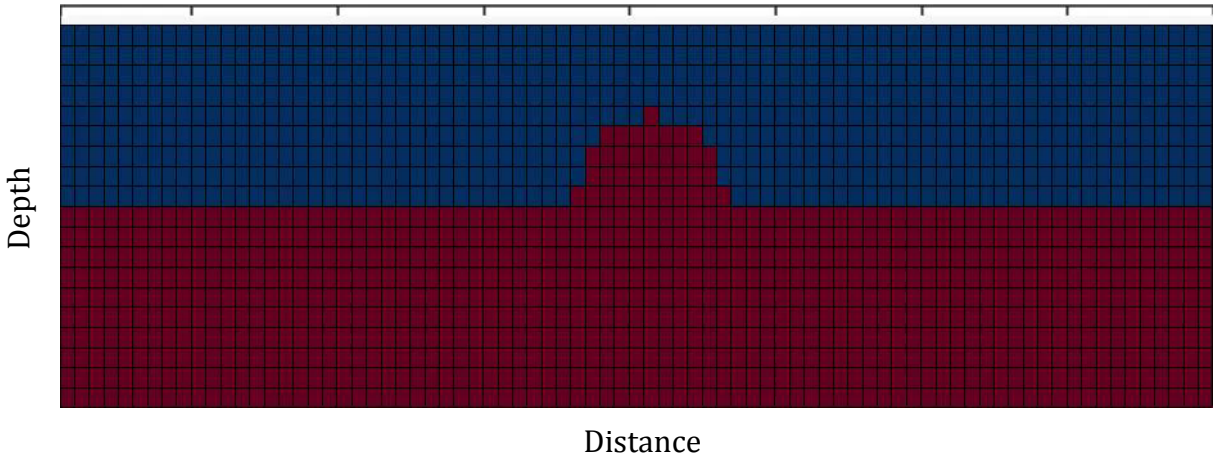


A summary of the interface in the above movie clip is shown:

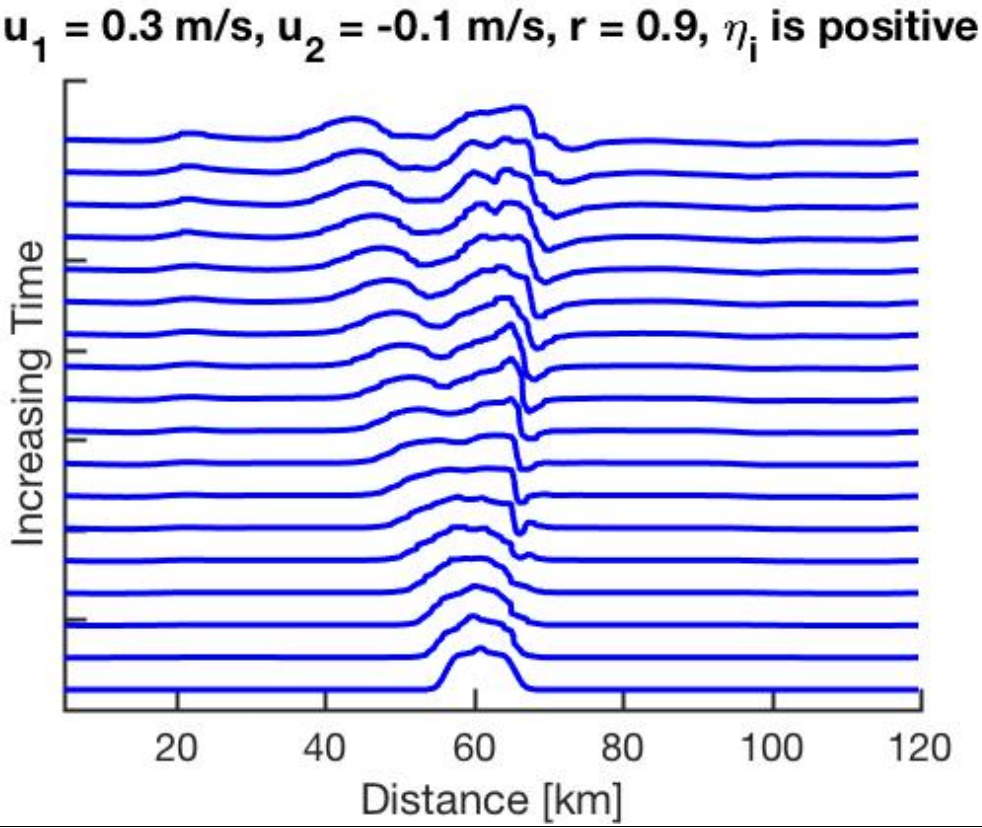


Location 3: $u_1 = 0.3$ m/s $u_2 = -0.1$ m/s and $r = 0.9$.

Temperature Interface (Movie)



A summary of the interface in the above movie clip is shown:



These simulations are similar to the results for the method of characteristics. You can slowly see the two waves being created, and the shock forming on either the upstream wave. Again, the perturbation is advected with the velocity of the thinner layer.

Part 3:

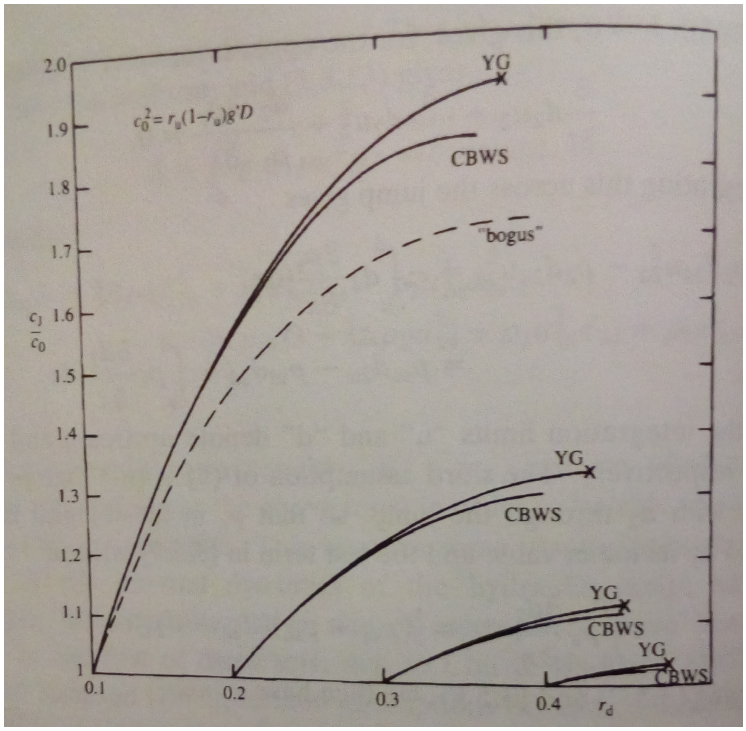
For the hydraulic jump I used the same gendata.py, except this time the topography has a tophat bump.

Here is the addition for the gendata.py code relating to the topography:

```
# topo
topo=-D*np.ones(nx)
width=5000
sigma = 1

for j in range(0, nx):
    if abs(x[j]) > width:
        topo[j]=0
    if abs(x[j]) < width:
        topo[j]=0.7*d_1o
topo=gaussian_filter1d(topo,sigma)-D

plt.plot(x,topo)
topo[topo>0.]=0.
```

I ran this simulation at $r_u = 0.1, 0.3$ and at many different initial velocities.

Baines Figure 3.7:

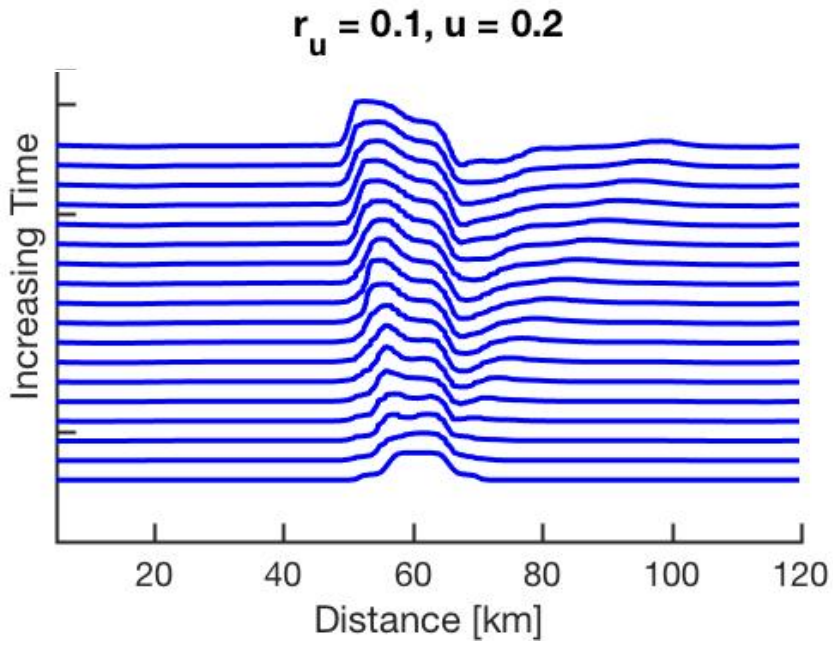


I have attached the code that let me theoretically recreate Figure 3.7:

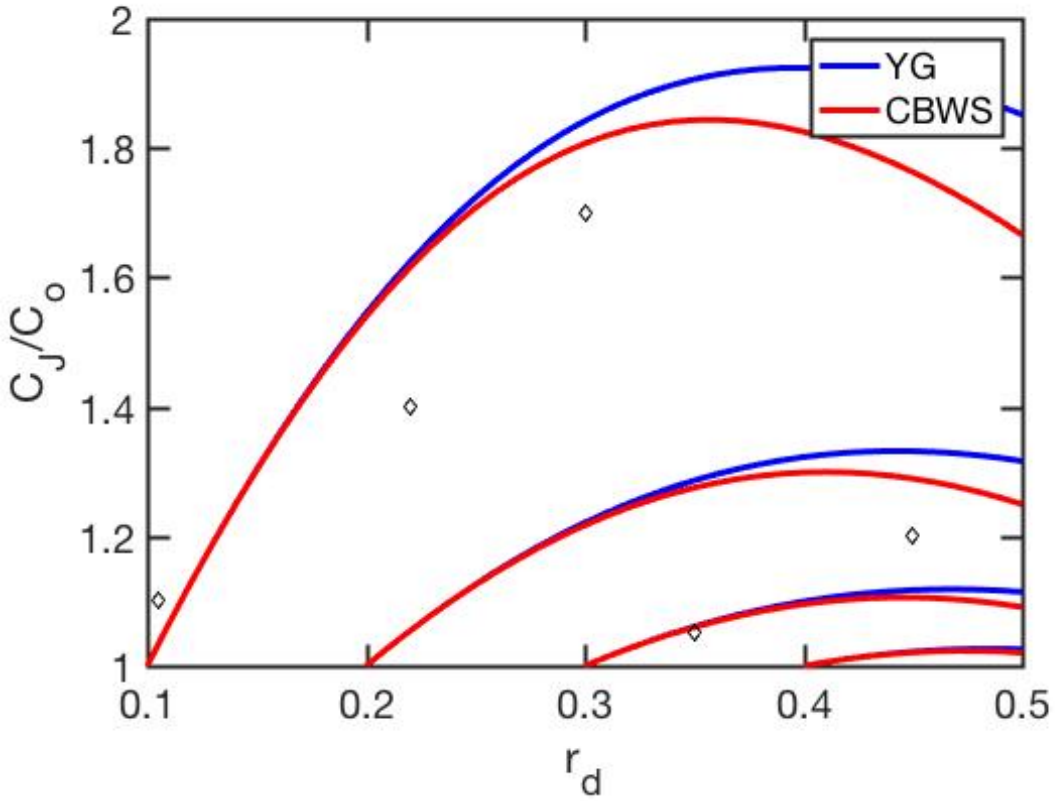
```
rho1 = 1025;
rho2 = 1026;

r_u = 0.1;
r_d = [0:0.01:0.5];

%YG formulation %Eq. 3.5.11 of Baines
top = (r_u + r_d)./2;
bottom = (r_u./r_d) + (rho2.*(1-r_u).*(r_u+r_d)./(rho1.*(1-r_d).*(2-r_u-r_d)));
CJ_YG = sqrt(top./bottom);

%CBWS formulation %Eq. 3.5.12 of Baines
bottom2 = (r_u./r_d) + ((rho2./rho1)*(r_u+r_d-2.*r_u.*r_d)./(2.*(1-r_d).^2));
cJ_CBWS = sqrt(top./bottom2);

scale = sqrt(r_u.*(1-r_u));

% Very rough estimation of C_J and r_d from MITgcm simulation
c=[0.105,0.22,0.3,0.35,0.45];
r=[1.1,1.4,1.7,1.05,1.2];

%Figure 3.7 of Baines

plot(r_d, CJ_YG/scale, 'b','LineWidth',3)
hold on
plot(r_d, cJ_CBWS/scale, 'r','LineWidth',3)
hold on
plot(c,r,'kd')
legend('YG', 'CBWS');
xlim([0.1 0.5])
ylim([1 2])
ylabel('C_J/C_o')
xlabel('r_d')
set(gca,'LineWidth',2,'TickLength',[0.025 0.025]); set(gca,'fontsize',20);
```

In each simulation I made very rough estimates of the jump speed and r_d. The jump speed was estimated by looking at two time snap shots and calculating the change in jump position. The jump speed was then scaled by c_0. r_d was calculated as the maximum interface value.

Below is an example of the results I got for the hydraulic jump. A jump of height r_d forms (this depends on the initial velocity) and moves at the jump speed upstream.



A summary of my results is shown:



Although the values match the trend, my estimations are C_J and r_d were very coarse. I need to a more accurate way to estimate the jump speed, before making any assumptions about the YG and CBWS formulations. In a couple simulations C_J/C_0 was less than 1, which did not match Figure 3.7 at all.