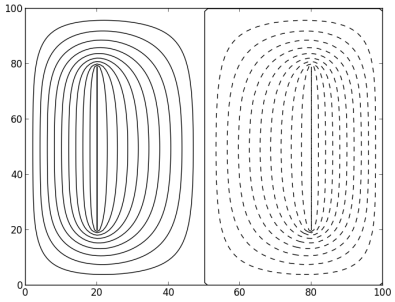
PHY407 2015 Computational Lab 08
Solving PDEs, Part 1

In the solutions, figures, pseudocode, and model output are reproduced in small size to reduce the size of the output file. Small text and figures can be rescaled or cut and paste to another document for improved viewing.

1. You were asked just to produce a contour plot. This is provided with contours every 0.1 V below.



Remarks - not for grading: I notice, for reference, that the difference between running with and without Gauss-Seidel iteration is substantial: for the requested convergence, G-S saves about 50% of the iterations. However, this does not account for the substantial savings obtained by vectorizing using numpy arrays (which is still possible to do even with this domain with voltage plates in the middle).

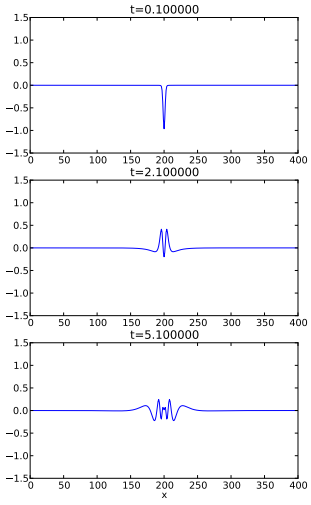
2. Problem 2

   (a) Here is some pseudocode for the problem:

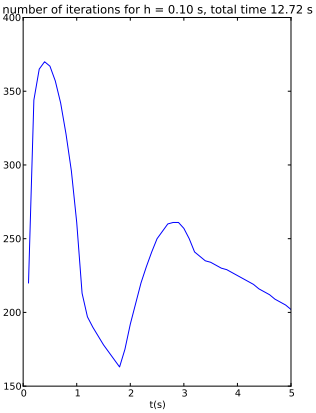
```
#gravity_wave.py
#By Paul Kushner for PHY407
#Generate the displacement of the free surface under the action of gravity in an infinitely deep layer

#Solve Laplace equation \nabla^2\phi=0 by iterative relaxation subject to the boundary conditions that
#i) phi = 0 for all boundary
#ii) The top and bottom values of $\psi$ are fixed for all $x$.
#The bottom boundary condition is actually incorrect for the shallow water wave problem. In this simulation we impose the Dirichlet boundary condition $\

#import necesssary routines from numpy, pylab, time

#Define routine to calculate Laplacian
   #create a copy array phicopy
   #define target in, target = 1e-4 #m^2/s
   #keep iterating until target is reached or until failure
   #for each iteration
      #Do nearest neighbour averaging in interior domain away from boundaries
      #calculate difference between current and previous
      #if difference is within target then break out of this loop
      #if not then just continue

   #if convergence isn't reached after the number of iterations is exceeded then quit


#The tendencies are at the surface
#$\partial\phi(x,0,t)/\partial t=-g\eta$ and $\partial\eta(x,t)/\partial t=\partial\phi(x,0,t)/\partial z$.
#The latter expression requires a finite difference calculation implemented below
#define function to return time rate of change of surface fields
   #implement w = deta/dt=dphi/dz, dphi/dt = -g*eta
   #the phiz is a finite difference involving surface quantities.
   #phiz=(phi[-1,:]-phi[-2,:])/dz
   #return -g*eta,phiz

#Define Geometry, parameters for integration
#define density, gravity
```

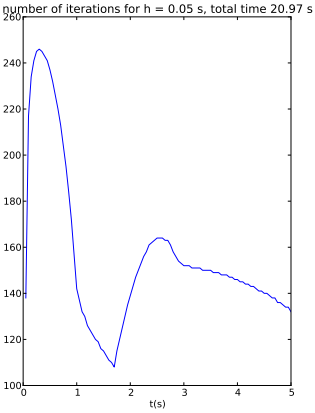
```
#define domain parameters
#define numerical parameters
#h, t, nsteps
#define nsteps in x and z

#define eta(x,t), phi(x,z), surface phi0(x,t)
#Define eta_array, phi_array for all time points for plotting purposes
#define time axis
#define grid of points in x and z
#define resolution in x and z


#define width of pulse, height of pulse
#height of pulse
#initialize $\eta$
#find initial time on clock
#
#initial eta (on full timestep)
#eta_full is the full timestep version of eta
#eta_full = A*exp(-(x-L/2.0)**2/(Delta)**2)
#save this to final array for analysis
#set up eta_half

#Set initial phi to zero indicating displaced interface and state of rest.
#set phi0_full to zero, and phi0_half


#Initialize phi to zero. The boundary conditions are zero for all times.

#initial time step
#variables that will be time stepped are eta, phi0
#need separate eta_0 and phi_0
#simple euler forward for first half step
#this gives eta_half and phi0_half


#phi0_half is surface condition

#solve for laplacian(phi)=0 for #phi field consistent with this


#proceed into stepping loop
#for each time step
   #update from t to t + h
   #update surface condition at t+h
   #solve phi consistent with this
   #update time
   #save diagnostics

   #update from t+h/2 to t+3h/2
   #plot
```

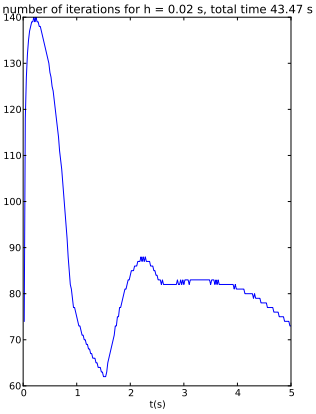
(b) The figure below shows the plot for times 0, 2, and 5 seconds. The pulse is gradually spreading. If you look in detail you will see that some features aren't very well resolved.

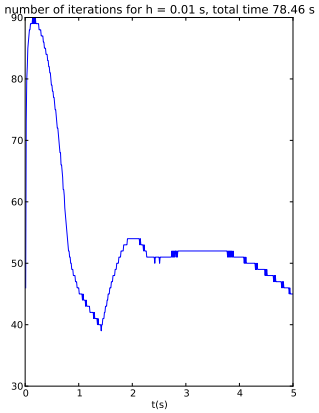


(c) The figures below show the number of iterations and the run time for `h` values listed.


number of iterations for h = 0.10 s, total time 12.72 s
400
350
300
250
200
150
0
1
2
3
4
5
t(s)

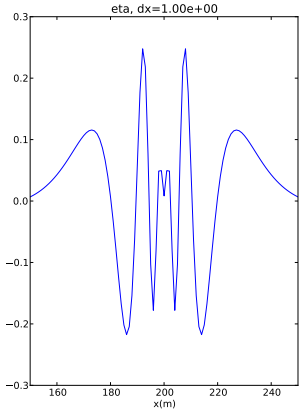

number of iterations for h = 0.05 s, total time 20.97 s
260
240
220
200
180
160
140
120
100
0
1
2
3
4
5
t(s)

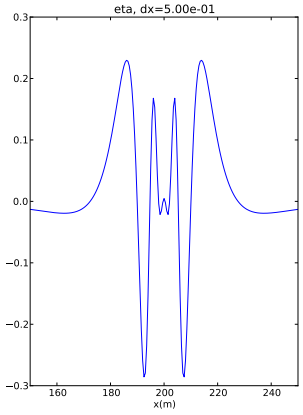

number of iterations for h = 0.02 s, total time 43.47 s
140
130
120
110
100
90
80
70
60
0
1
2
3
4
5
t(s)

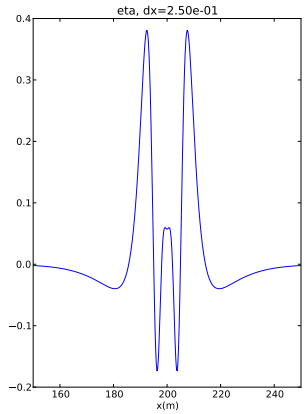


There are evident tradeoffs. For $h = (0.10, 0.02, 0.01)$, number of iterations at the 3s mark is about $(250, 80, 50)$ and the runtime is $(13, 43, 78)$. For now we will settle on the $h = 0.02$ value as a compromise.

Below are a couple of figures for different values of dx for the same domain width.







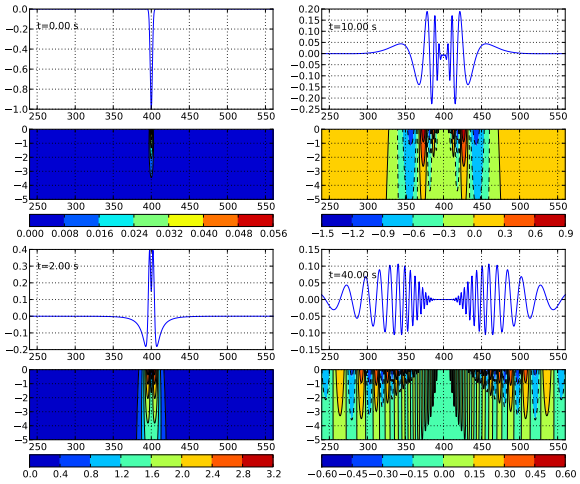
In these plots, the evolution is quite different and in the 1m resolution case the features are not well resolved. In fact there is not a clear sense that the solution has converged. But the runtime is 256 s for the case with dx=25cm resolution, for five seconds, for h=0.01. If we are two double the domain width, we could expect that the run time will be 100 s per 1 s of simulation time. So a 40 s simulation will take about 4000 s, or an hour, which is a long time.

I ran a long simulation with the following parameters:

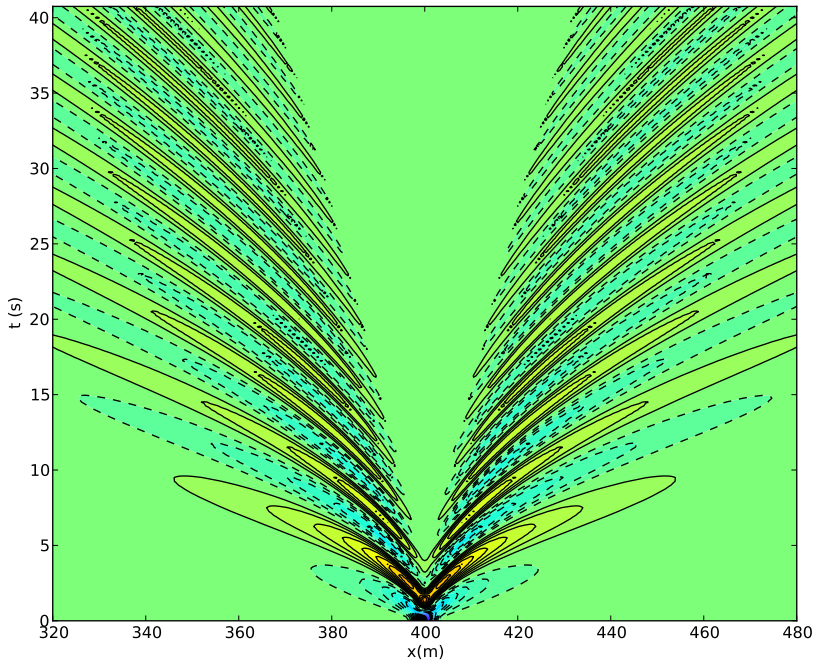
```
#define domain parameters
L=800.0
D=50.0
#define numerical parameters
h = 0.01 #seconds
#define initial time
t=0
#define nsteps
nsteps = 4100
#number of snapshots
snap_int = 25
nsnaps = nsteps/snap_int
#define nsteps in x and z
npts_x   = 1601
npts_z   = 101
```

To use less memory I saved snapshots every 25 time steps (0.25 s). The resolution in this case was 50cm in the horizontal and vertical. The domain was 800 m which is probably a bit narrow. This took about 30 minutes on a 15" MacBookPro. The number of iterations reduced considerably as the solution proceeded.

The following shows η and ϕ at $t = 0, 2, 10$, and 40s. The typical pattern is that in the area of a trough the gradient of ϕ is toward the trough, indicating that the water is moving to fill in the trough. This becomes less obvious later in the simulation as the structure becomes finer.



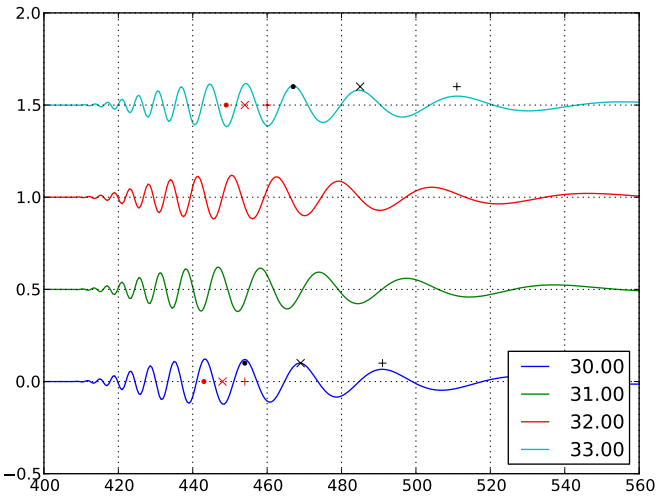
A contour plot of the wave eta as a function of x and t is shown below:



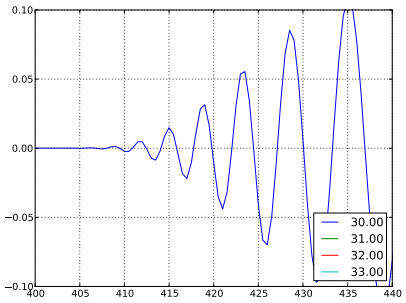
Here is some of what I observed:

- The simulation yields counter propagating packets whose leading edges feature longer waves than the trailing edge. This makes sense because the phase speed scales as $\sqrt{\lambda}$. Features start getting less resolved as you approach x=0, because the wave length falls below the grid scale. The largest amplitude waves get shorter over time, indicating that the group speed is slower than the phase speed of the waves in the envelope. Put another way, waves are seen to travel through the envelope. This

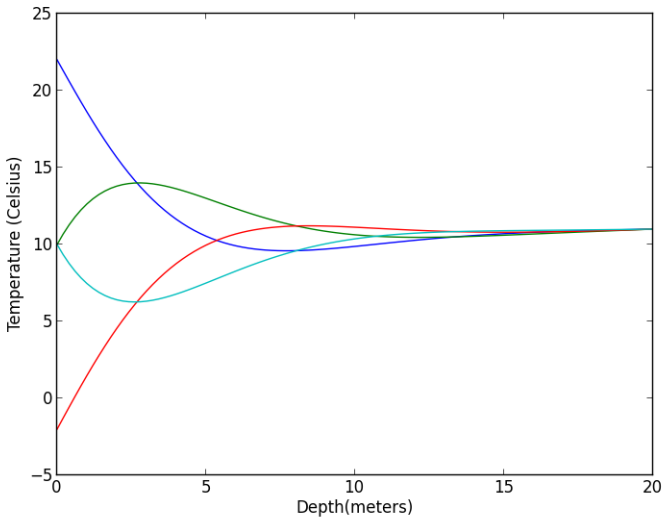
is known as *anomalous dispersion*, because for many dispersive waves you would see the wave crests and troughs being slower than the wave group.



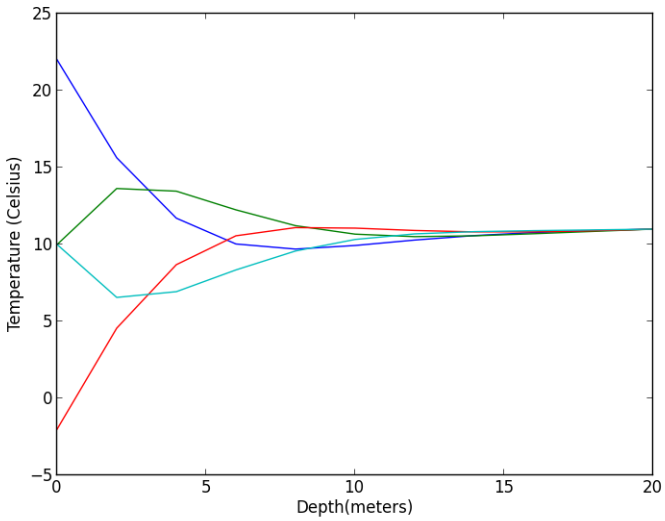
- In the figures above, I used features at the leading edge, which were the best resolved because they were the longest wavelength. I estimated the location of three crests indicated by the black symbols at two different times, and from these calculated approximate phase speeds and wavelengths. I found the phase speed of these features to be about 4.83 m/s for the average of the first and second crests and about 6.00 m/s for the second and third crests. Then I calculated the approximate phase speed using $c = \sqrt{g/k}$, where the wavenumber $k = 2\pi/lambda$ was calculated by taking the distance between the crests as the wavelength. For the first pair of crests I found the phase speed here to be 5.08 m/s, which is 5% greater than the first estimate, and for the second pair of crests I found the phase speed to be 6.12 m/s, which is only 2% greater than the second estimate.
- To estimate the group speed, the feature I tracked was the peak amplitude of the wave packet, which I estimate based on the location of the three largest antinodes in the packets. These are marked by the red symbols. I compared this to the dispersion relation $c_g = \sqrt{g/4k}$, using k calculated from the wavelengths estimated as the distance between the first and third red symbol. The packet group velocity estimated the first way was 2.00 m/s, and the second way 2.07 m/s, which is 4% greater. So the numerics is basically seen to conform to theoretical expectations.
- In the zoom below you see how some of the features are only resolved in a limited way as we approach the trailing edge of the packet.



3. Suppose we choose a vertical spacing of $a = 20$cm, then $a^2/2D = 0.2$days. This suggests we should choose a time step of less than 0.2 days. It's confirmed that if we choose the time step h=0.3 d the code is unstable. Choose instead h=0.01d, which is well below this limit. We then get the plot below:



We can also try a run with coarse resolution of $a = 2$m. It is surprisingly similar to the fine resolution simulation, although of course more poorly resolved.


25
20
15
10
5
0
−5
Temperature (Celsius)
0
5
10
15
20
Depth(meters)