Computer methods in Engineering
Exercise on hyperbolic equations

Problem 1
The forward and backward numerical derivative of a function f can be expressed as

$$f'(k\Delta x + \Delta x/2) \approx d_x^+ f(k\Delta x) \quad = \quad d^+ f_k = \frac{f_{k+1} - f_k}{\Delta x}, \tag{1}$$

$$f'(k\Delta x - \Delta x/2) \approx d_x^- f(k\Delta x) \quad = \quad d^- f_k = \frac{f_k - f_{k-1}}{\Delta x}. \tag{2}$$

Here $f_k = f(k\Delta x)$ where $k = 0, 1, 2, \cdots, N-1$, Δx is the sampling interval and N is the number of samples.

a) Write two functions, Df and Db which performs backward and forward differentiation of a function. The input arguments to the functions are an array of length N with the function samples, an output array of length N and the sampling interval dx. The function Df should compute the derivatives of the input array at points $k\Delta x + \Delta x/2$ for $k = 0, 2, \cdots, N-2$ while Db should compute the derivative at points $k\Delta x - \Delta x/2$ for $k = 1, 2, \cdots, N-1$.

Solution:
A simple approximation for the derivative of a function $f(x)$ is the finite difference expression:

$$f'(x) = \frac{f(x+\Delta) - f(x)}{\Delta x}, \tag{3}$$

which approximates the derivative at x using a small stepsize Δx. To find the error of the approximate derivative we can expand $f(x)$ in a Taylor series:

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{f''(x)}{2}\Delta x^2 + \frac{f'''(x)}{6}\Delta x^3 + \ldots \tag{4}$$

If we solve the above equation for $f'(x)$ one gets:

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} - \frac{f''(x)}{2}\Delta x + \ldots \tag{5}$$

If we truncate the expression after the first term on the right hand side, we see that we are making an error which is proportional to Δx.

It is possible to improve the accuracy by using a Taylor expansion for $f(x - \Delta x)$:

$$f(x - \Delta x) \approx f(x) - f'(x)\Delta x + \frac{f''(x)}{2}\Delta x^2 - \frac{f'''(x)}{6}\Delta x^3 + \ldots \tag{6}$$

If we subtract equation (6) from equation (4) we get:

$$f(x + \Delta x) - f(x - \Delta x) \approx 2f'(x)\Delta x + 2\frac{f'''(x)}{6}\Delta x^3 + \ldots \tag{7}$$

Solving for $f'(x)$ we get

$$f'(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} + 2\frac{f'''(x)}{12}\Delta x^2 + \dots \tag{8}$$

This gives us the so-called central difference *approximation for the derivative:*

$$f'_c(x) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} \tag{9}$$

with an error proportional to Δx^2 which is more accurate then the approximation given in equation (3).

To adapt to the given problem we let $\Delta x \to \Delta x/2$ in equation (9) to get

$$f'_c(x) = \frac{f(x+\Delta x/2) - f(x-\Delta x/2)}{\Delta x} \tag{10}$$

However, in the given problem f is only known at $x = 0, \Delta, 2\Delta x, \dots$ so we have to change x to $x + \Delta x/2$ to get

$$f'_c(x+\Delta x/2) = \frac{f(x+\Delta x) - f(x)}{\Delta x}. \tag{11}$$

We see that if we use the finite difference approximation given in equation (3), but interpret the derivative to be given at $f(x + \Delta x/2)$ instead of at x, we can compute a derivative with better accuracy.

We can also construct the backward version of equation (11)

$$f'_c(x-\Delta x/2) = \frac{f(x) - f(x-\Delta x)}{\Delta x}. \tag{12}$$

Below is an implementation of the Df and Db functions.

```
"""Functions for first order forward and backward differentiation """
import numpy as np

def df(u,v,dx) :
  ''' df computes the first-order forward derivative

      Parameters:
        u  : Function to be differentiated
        v  : Derivative of u
        dx : grid interval

      Returns:
        The return value is a vector with
        the values of the derivative.
        The last grid point in the output is set to zero.

  '''
  nx=u.shape[0]
  for i in range(0,nx-1):
      v[i] = (u[i+1]-u[i])/dx
  return(v)

def db(u,v,dx) :
  ''' db computes the first-order backward derivative
```

```
    Parameters:
      u  : Function to be differentiated
      v  : Derivative of u
      dx : grid interval

    Returns:
      The return value is a vector with
      the values of the derivative.
      The first grid point is set to zero.

  '''
  nx=u.shape[0]
  for i in range(1,nx):
      v[i] = (u[i]-u[i-1])/dx
  return(v)
```

b) Check that the routines work by differentiating a known function.

Solution:

Here is a small test program for testing the df routine.

```
import numpy as np
import matplotlib.pyplot as pl
from diff import *

PI=3.14159
N=100
x = np.linspace(0,2*PI,N)
f = np.sin(x)

L=2.0*PI
dx=L/N
fdiff=np.zeros(N)
df(f,fdiff,dx)
fd=np.cos(x+dx/2)
fd2=np.cos(x)

pl.figure()
y=fdiff-fd
pl.plot(x,y,label="Error at x+dx/2")
y=fdiff-fd2
pl.plot(x,y,label="Error at x")
pl.title("Test of the df function")
pl.xlabel("x")
pl.ylabel("df(x)")
pl.legend()
pl.ylim((-0.1,0.1))
pl.savefig('df.pdf')
pl.show()
```

The output is given in figure 1.

Note the problem with the last point in the approximate derivative. Since we do not know the function outside the numerical grid, we can not compute the derivative at the endpoint.

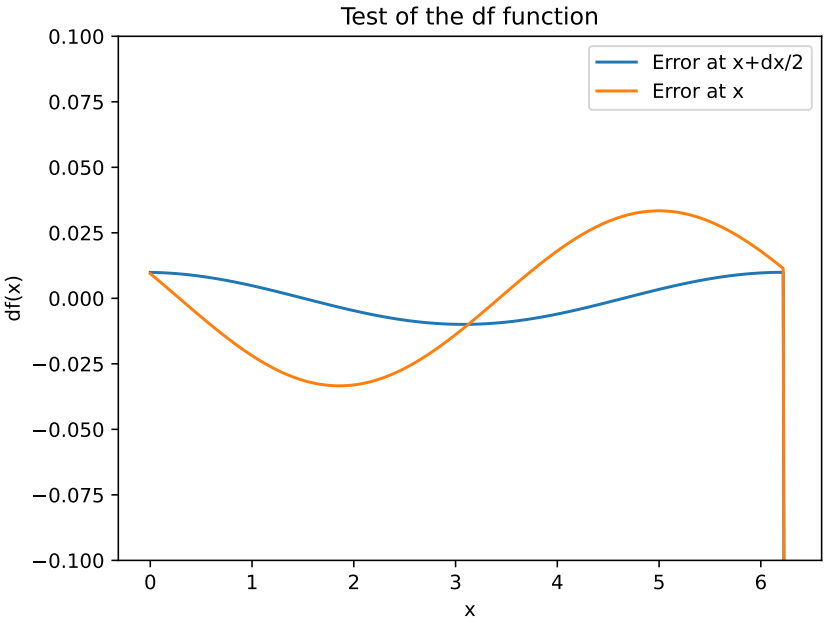

Figure 1: Test of differentiator function df showing the error between exact derivative and the finite difference result

Problem 2

The one-dimensional acoustic equations of motion are given as

$$\ddot{u}(x,t) = \frac{1}{\rho(x)}\frac{d\sigma(x,t)}{dx}, \tag{13}$$

$$\ddot{\sigma}(x,t) = \kappa(x)\frac{d\ddot{u}(x,t)}{dx} + s(x,t). \tag{14}$$

An algorithm for solving the equations numerically is

1. Set $t = 0, \sigma(x, t = 0) = 0$ and $\sigma(x, t = -\Delta t) = 0$ for all $x = l\Delta x, l = 0, \ldots, N-1$
2. Compute the accelerations $\ddot{u}(x,t)$ at time t, for all x=$l\Delta x, l = 0, \ldots, N-2$ by using the equations

$$\ddot{u}(x + \Delta x/2, t) = \rho^{-1}(x + \Delta x/2)d_x^+\sigma(x,t), \tag{15}$$

3. Compute the stress $\sigma(x, t + \Delta t)$ at the future time $t + \Delta t$ and for all $x = l\Delta x$, $l = 1, \ldots, N-1$ by using the equation

$$\begin{aligned}\sigma(x,t+\Delta t) &= 2\sigma(x,t) - \sigma(x,t-\Delta t) \\ &+ \Delta t^2\kappa(x)[d_x^-[\ddot{u}(x+\Delta x/2,t)]\end{aligned} \tag{16}$$

4. Add the source term for a single grid position $x_s = m_s\Delta x$

$$\sigma(x_s, t+\Delta t) = \sigma(x_s, t+\Delta t) + \Delta t^2 s(x_s,t)/\Delta x. \tag{17}$$

5. set $t = t + \Delta t$ and go to 2. Stop if $t = (N_t - 1)\Delta t$ is reached.

The derivative operators d_x^+ and d_x^- are the forward and backward derivatives computed by th Df and Db functions.

a) Implement the above scheme for solution of the one-dimensional wave equation. Use three arrays, sigma2, sigma1 and sigma0, of length N for the stresses corresponding to times $t + \delta t$, t, and $t - \delta t$. Use an additional array u for the acceleration and if necessary additional arrays for holding intermediate results.

The source s should be implemented as a separate function with the time index k $t = k\Delta t$ as an argument. Use a Ricker wavelet with peak frequency(fp) of 30Hz, and delay (tp) of 0.1 seconds. See the attached ricker jupyter notebook for the python code of a function implementing the ricker wavelet. The boundary conditions are implicit and no extra code is needed. Plot the pulse.

Make separate arrays to hold the density and bulk modulus κ. The stability condition for the algorithm is

$$c\Delta t/\Delta x \leq 1, \tag{18}$$

where $c = \sqrt{(}\kappa/\rho)$ is the wave velocity.

Solution:

The main difficulty in implementing the algorithm given above is to realize that conceptually we need to handle two different numerical grids. One grid is defined as a regular grid with x-coordinates given as $x = i\Delta x, i = 0, .., N - 1$ where N is the number of gridpoints. The other grid is defined as $x = i\Delta x + \Delta x/2, i = 0, ..., N - 1)$, i.e. a grid shifted an amount $\Delta x/2$ relative to the first grid. We see this from equation (15) where we use the centered forward differentiator d_x^+ which takes as input the stress $\sigma(x)$ defined on a regular grid $x = i\Delta x, i = 0, .., N - 1$. The output from the differentiation is, however, defined on a shifted, or staggered, grid given by $x = i\Delta x + \Delta x/2$. This means that the accelerations are given not on the regular grid but at the points $u(x + \ddot{\Delta} x/2)$. Also note that since we are multiplying with the density, ρ must also be defined on this staggered grid. Since the staggered grid is only conceptual, the output from the differentiation is stored on the regular grid. Doing this implies that the output at point $x = i\Delta x + \Delta x/2$ is actually stored in the regular grid at position $x = i\Delta i, i = 0, ..., N - 1$. This implies that the accelerations computed at $\ddot{u}(x + \Delta x/2)$ is actually stored in the regular grid at position $x = i\Delta x, i = 0, ..., N - 1$.

If we move on to equation (16), we see that we apply the backward differentiator d_x^- to the accelerations. The backward differentiator is defined in equation (12). However, the inputs to the backward differentiator is now defined on the staggered grid, i.e. at points $x = i\Delta x + \Delta x/2, i = 0, ..., N - 1$ We can arrange this by using equation (12), by changing the output point from $\Delta x - \Delta x/2$ to x. Equation (12) then becomes

$$f_c'(x) = \frac{f(x + \Delta x/2) - f(x - \Delta x/2)}{\Delta x}. \tag{19}$$

Keeping in mind that the outputs from applying the forward derivative d_x^+ was actually stored into the regular grid, it means that when we apply equation (12) to the accelerations we have to use the backward formula given above as

$$\ddot{e}(i\Delta x) = d_x^- \ddot{u}(i\Delta x + \Delta x/2) = \frac{\ddot{u}_i - \ddot{u}_{i-1}}{\Delta x} \tag{20}$$

where $\ddot{u}_i = \ddot{u}(i\Delta x)$.

A script which implements the algorithm is given below.

```
"""Wave1d solves the 1D elastic equations using stress-acceleration"""
import sys
import re
import struct
import numpy as np
import matplotlib.pyplot as pl
from math import *
from diff import *

def ricker(fp,tp,M,dt) :
  ''' ricker computes a Ricker source function

      Parameters:
        fp : Peak frequency
        tp : Time delay
        M  : No of output time samples
        dt : Time sampling interval

      Returns:
        The return value is an array of length M
        containing the source function values.

  '''

  s=np.zeros((M))
  for k in range(0,M):
     t=k*dt
     wp=2.0*3.14159*fp
     s[k] =(1-0.5*pow(wp,2)*pow((t-tp),2))*exp(-0.25*pow(wp,2)*pow(t-tp,2))
  return(s)

def fd1(N,M,dt,dx,s,xs,kappa,rho) :
  ''' fd1 computes the solution of the 1d wave equation
      using the staggered stress-acceleration method.

      Parameters:
        N : No of spatial (x) gridpoints.
        M : No of temporal (t) griudpoints.
        dx : Spatial (x) grid interval.
        dt : Temporal (t) grid interval.
        s  : vector with source pulse of length M.
        xs : Source position (in gridpoints)
        kappa: Vector with bulk modulus
        rho  : Vector with density

  '''

  # Stress vectors
  sigma0=np.zeros((N))
  sigma1=np.zeros((N))

  # displacement vector
  u=np.zeros((N))
  # Acceleration vector
  a = np.zeros((N))
  # Output stress vector
  out = np.zeros((N,M))
```

```
  for k in range(0,M):

      #compute the acceleration:
      u=df(sigma1,u,dx)
      u = u/rho

      #Differentiate the acceleration
      a=db(u,a,dx)

      #Integrate the stress
      sigma2 = 2.0*sigma1 -sigma0 + ((dt*dt)*kappa)*a

      #Add the source at source position
      sigma1[int(xs)]=sigma1[int(xs)]+dt*dt*s[k]/dx

      out[:,k]=sigma2

      #Leapfrog
      sigma0 = sigma1
      sigma1 = sigma2
  return(out)

def main() :
  dt = 0.0001  #Time sampling interval
  M = 10000 #No of time samples
  t = np.zeros((M))
  for i in range(0,M) :
    t[i] = dt*i

  fp=30.0
  tp=0.1
  s=ricker(fp,tp,M,dt)
  fs = np.fft.fft(s)
  fig=pl.figure()
  spec=abs(fs[0:int(M/2)])
  deltaf=1.0
  pl.xlabel("Frequency (Hz)")
  pl.ylabel("Amplitude")
  pl.plot(spec[0:100])
  pl.savefig('spect.pdf')
  pl.show()

  fig=pl.figure()
  pl.plot(t,s)
  pl.xlim(0.0,1.0)
  pl.xlabel("Time (sec)")
  pl.ylabel("Amplitude")
  pl.savefig('pulse.pdf')
  pl.show()
  wl=20.0

  fig=pl.figure()

  N = 1200   #Spatial grid size
  dt = 0.0001  #Time sampling interval
  dx = 2.5    #Spatial time smpling interval
  c0 = 2000.0 #Wave velocity
  c1 = 3000.0
  rho=np.zeros((N))
```

```
  kappa=np.zeros((N))

  # Case for homogeneous model c=2000.0
# rho[:N]=1000.0
# kappa[0:N] = c0*c0*rho[0:N]

  # Case for inhomogeneous model
  rho[:N]=1000.0
  rho[900:N]=2000.0
  kappa[0:N] = c0*c0*rho[0:N]
  kappa[900:N] = c1*c1*rho[900:N]

  fp=30.0
  tp=0.1
  s=ricker(fp,tp,M,dt)
  xs=int(N/2)
  out=fd1(N,M,dt,dx,s,xs,kappa,rho)
  #================
  #Plot figures
  #================
  fig=pl.figure()
  #Set aspect ratio
  ar=3000.0
  ax=pl.gca()
  asr = 1.0/(ax.get_data_ratio()*ar)
  im=pl.imshow(out,cmap='gray',extent=(0,1,0,3000))
  pl.Axes.set_aspect(ax,asr)

# Case for homogeneous model c=2000 m/s
  pl.savefig('sect.pdf')

# Case for inhomogeneous model
# pl.savefig('sect2.pdf')
  pl.show()

  x=np.zeros((N))
  c=np.zeros((N))
  for i in range(0,N):
    x[i] = i*dx
    c[i] = sqrt(kappa[i]/rho[i])
  pl.plot(x,c)
  pl.xlabel("Depth (km)")
  pl.ylabel("Velocity (km/s)")
  pl.savefig('vel.pdf')
  pl.show()

  fig=pl.figure()
  pl.plot(x,rho)
  pl.xlabel("Depth (km)")
  pl.ylabel("Density kg/$m^3$")
  pl.savefig('rho.pdf')
  pl.show()

if __name__ == "__main__":
    main()
```

b) Create a model with constant velocity and density with length of 3000m. The wave

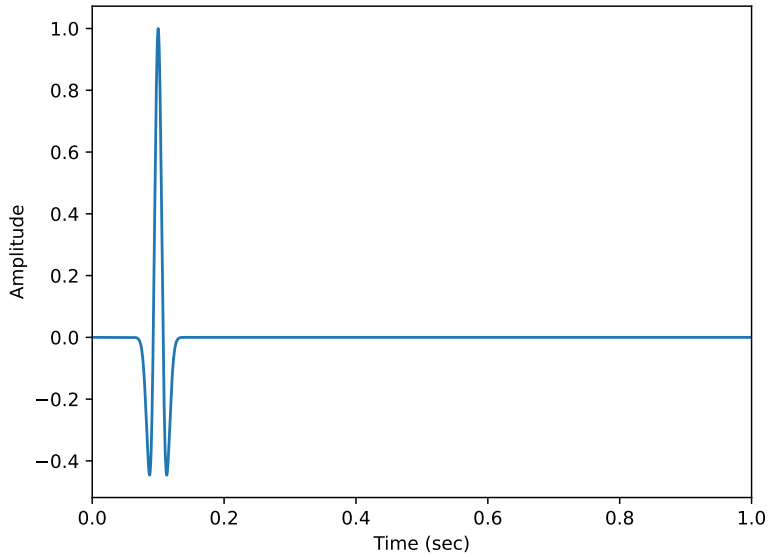

Figure 2: Ricker pulse with peak frequency 30Hz

velocity should be equal to 2000 m/s and the density set to 1000kg/m^3. The source should have a peak frequency of 30Hz. Select suitable $\Delta x = 5.0$m and $\Delta t = 0.0001$ seconds. Place the source at a position of 1500m and simulate the stress for approximately one second. Plot the resulting stress as function of position and time as an image using f.ex a gray scale color table.

Solution:
We use the code given above to produce the plot shown in figure 3.

Problem 3

a) Perform four simulations with $\Delta x = 10, 5, 2.5$ and 1m. Record the pressure at a depth of 2000m and compute the relative error between simulations for 10 and 5m, 5 and 2.5m and 2.5m and 1m. The error between two simulations can be computed using

$$\epsilon = \frac{\sqrt{\sum_{i=0}^{N}[p_1(i\Delta t) - p_2(i\Delta t)]^2}}{\sqrt{\sum_{i=0}^{N-1} p_1^2(i\Delta t)}} \tag{21}$$

Plot the resulting error as a function of the wavelength divided by Δx, i.e. the number of wavelengths per spatial grid point.

Solution:
A script for implementing the tests is shown below:

```
"""Error demonstrates fd errors for wave propagation"""
import sys
import re
import struct
```

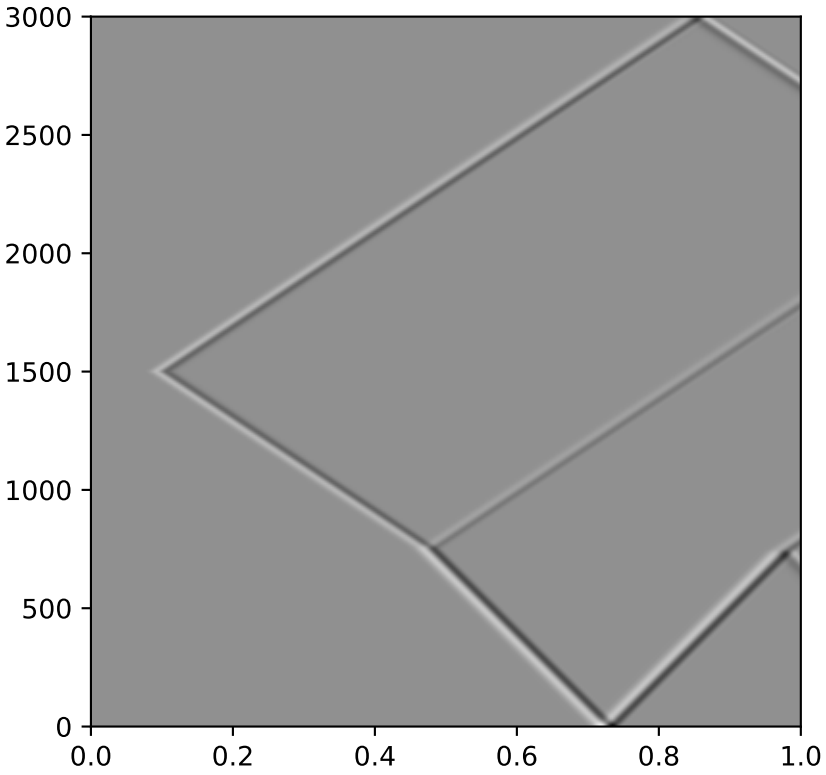

Figure 3: Stress for homogeneous model with $c = 2000$ m/s.

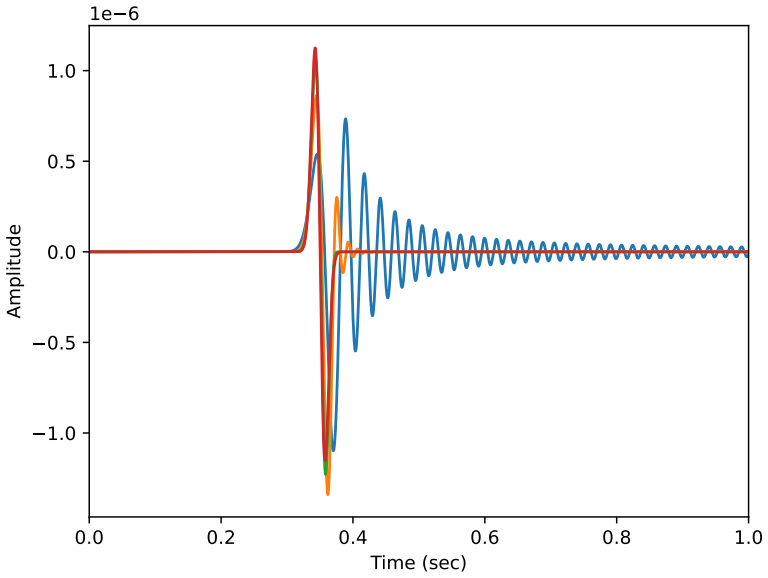

Figure 4: Pressure recorded at position of 2000m for $\Delta x = 10, 5, 2.5$ and 1m grid spacing.

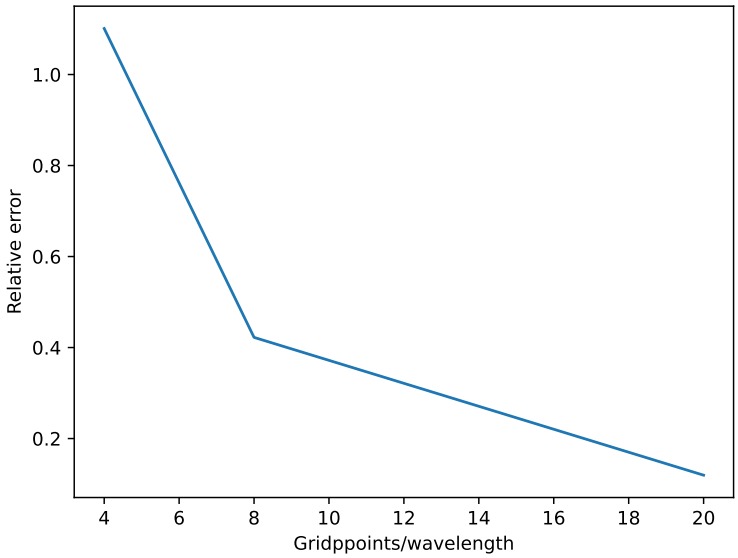

Figure 5: Relative error between pressure at 2000m for $\Delta x = 10, 5, 2.5$ and 1m grid spacing.

```
import numpy as np
import matplotlib.pyplot as pl
from math import *
from wave1d import *

#------------------------------
# Main script
#------------------------------
dt = 0.0001  #Time sampling interval
M = 10000 #No of time samples
t = np.zeros((M))
for i in range(0,M) :
    t[i] = dt*i

diff = np.zeros((M))
err = np.zeros((3))
xerr = np.zeros((3))
rec=np.zeros((M))
recold=np.zeros((M))
fp=30.0
tp=0.1
s=ricker(fp,tp,M,dt)
fs = np.fft.fft(s)
fig=pl.figure()
spec=abs(fs[0:int(M/2)])
deltaf=1.0
pl.xlabel("Frequency (Hz)")
pl.ylabel("Amplitude")
pl.plot(spec[0:100])
pl.savefig('spect.pdf')
pl.show()

fig=pl.figure()
pl.plot(t,s)
```

```
pl.xlim(0.0,1.0)
pl.xlabel("Time (sec)")
pl.ylabel("Amplitude")
pl.savefig('pulse.pdf')
pl.show()
wl=20.0

fig=pl.figure()
N = 300   #Spatial grid size
dx = 10.0    #Spatial time smpling interval
c0 = 2000.0 #Wave velocity
rho=np.zeros((N))
kappa=np.zeros((N))
rho=1000.0
kappa=c0*c0*rho
xs=N/2
out=fd1(N,M,dt,dx,s,xs,kappa,rho)
#Pick receiver at depth of 2000m
rec= out[200,:]
pl.plot(t,rec,label='$\Delta x = 10m$')
#pl.show()
recold[:]=rec[:]

N = 600   #Spatial grid size
dt = 0.0001  #Time sampling interval
dx = 5.0    #Spatial time smpling interval
c0 = 2000.0 #Wave velocity
rho=np.zeros((N))
kappa=np.zeros((N))
rho=1000.0
kappa = c0*c0*rho
fp=30.0
tp=0.1
s=ricker(fp,tp,M,dt)
xs=N/2
out=fd1(N,M,dt,dx,s,xs,kappa,rho)
#Pick receiver at depth of 2000m
rec= out[400,:]
pl.plot(t,rec,label='$\Delta x = 5m$')
#pl.show()
diff = rec-recold
recold[:]=rec[:]
err[0] = sqrt(np.dot(diff,diff))/sqrt(np.dot(rec,rec))
xerr[0] = wl/dx

N = 1200   #Spatial grid size
dt = 0.0001  #Time sampling interval
dx = 2.5    #Spatial time smpling interval
c0 = 2000.0 #Wave velocity
rho=np.zeros((N))
kappa=np.zeros((N))
rho=1000.0
kappa = c0*c0*rho
fp=30.0
tp=0.1
s=ricker(fp,tp,M,dt)
xs=N/2
out=fd1(N,M,dt,dx,s,xs,kappa,rho)
#Pick receiver at depth of 2000m
```

```
rec= out[800,:]
pl.plot(t,rec,label='$\Delta x = 2.5m$')
#pl.show()
diff = rec-recold
recold[:]=rec[:]
err[1] = sqrt(np.dot(diff,diff))/sqrt(np.dot(rec,rec))
xerr[1] = wl/dx

N = 3000   #Spatial grid size
dt = 0.0001  #Time sampling interval
dx = 1.0    #Spatial time smpling interval
c0 = 2000.0 #Wave velocity
rho=np.zeros((N))
kappa=np.zeros((N))
rho=1000.0
kappa = c0*c0*rho
fp=30.0
tp=0.1
s=ricker(fp,tp,M,dt)
xs=N/2
out=fd1(N,M,dt,dx,s,xs,kappa,rho)
#Pick receiver at depth of 2000m
rec= out[2000,:]
pl.plot(t,rec,label='$\Delta x = 1m$')
diff = rec-recold
err[2] = sqrt(np.dot(diff,diff))/sqrt(np.dot(rec,rec))
xerr[2] = wl/dx
pl.xlim(0,1.0)
pl.xlabel("Time (sec)")
pl.ylabel("Amplitude")
pl.savefig('rec.pdf')
pl.legend(loc='upper left')
pl.show()

fig=pl.figure()
pl.xlabel("Gridppoints/wavelength")
pl.ylabel("Relative error")
pl.plot(xerr,err)
pl.savefig('err.pdf')
pl.show()
#================
#Plot figures
#================
fig=pl.figure()
#Set aspect ratio
ar=3000.0
ax=pl.gca()
asr = 1.0/(ax.get_data_ratio()*ar)
im=pl.imshow(out,cmap='gray',extent=(0,1,0,3000))
pl.Axes.set_aspect(ax,asr)
pl.savefig('sect.pdf')
pl.show()
```

See figures 4 and 5 for the output.

Problem 4

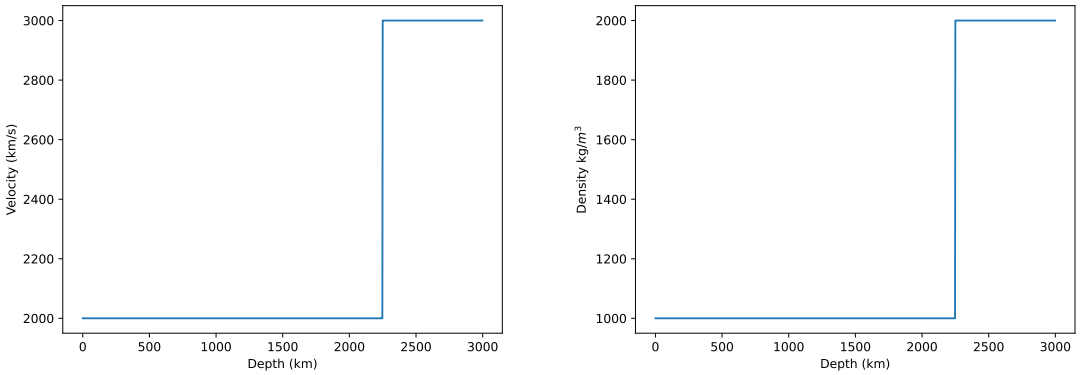

Figure 6: Velocity and density for the model used for computing the solution shown in figure 7

a) Make a model containing reflectors of your own device, compute and plot the solution. Also plot the velocity and density.

Solution:
Edit the code for wavepropagation (wave1d) and change the velocity and density. See the suggested modifications in the listing above. The model and output is shown in figures 6 and 7.

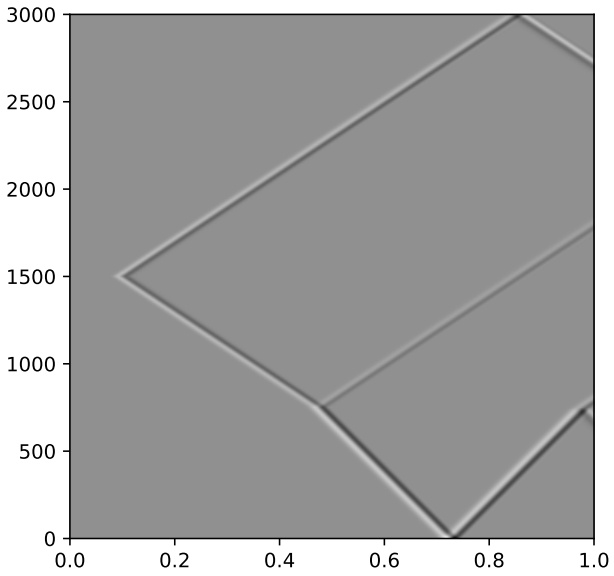

Figure 7: Stress as function of space and time for the model shown in figure 6