

TPG4155 Computer methods in Engineering

Exercise 4

Problem 1

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

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

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

Here $f_k = f(k\Delta x)$ where $k = 0, 1, 2, \dots, N-1$, 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 Δx . The function Df should compute the derivatives of the input array at points $k\Delta x + \Delta x/2$ for $k = 0, 2, \dots, N-2$ while Db should compute the derivative at points $k\Delta x - \Delta x/2$ for $k = 1, 2, \dots, N-1$.

Solution:

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

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

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:

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

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

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

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 $f(x - \Delta x)$:

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

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

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

Solving for $f'(x)$ we get

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

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

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

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

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

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

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

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

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

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

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

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.

```

"""Solve 1D elastic equations using stress-acceleration"""
import sys
import re
import struct
import numpy as np
import matplotlib.pyplot as pl
from math import *

#-----
# functions
#-----

def ricker(fp,tp,M,dt) :
    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 df(u,v,dx) :
    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) :
    nx=u.shape[0]
    for i in range(1,nx):
        v[i] = (u[i]-u[i-1])/dx

```

The output is given in figure

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.

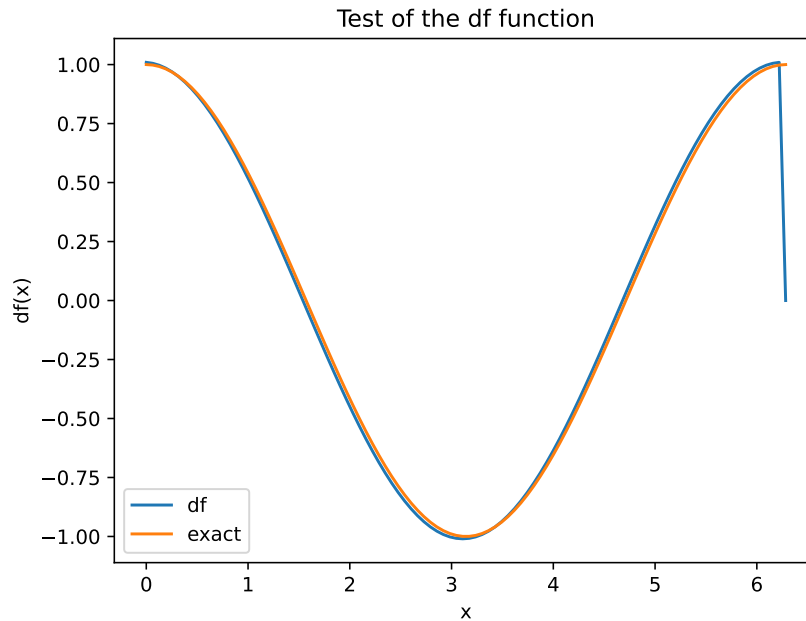


Figure 1: Test of differentiator function df

Problem 2

The one-dimensional acoustic equations of motion are given as

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

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

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

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

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

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

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

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

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 the 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

$$c\Delta t/\Delta x \leq 1, \quad (18)$$

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

Solution:

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 plt
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) :
```

```

''' fdl 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) gridpoints.
        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=plt.figure()
spec=abs(fs[0:int(M/2)])
Δf=1.0
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude")
plt.plot(spec[0:100])
plt.savefig('spect.pdf')
plt.show()

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

fig=plt.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=fdl(N,M,dt,dx,s,xs,kappa,rho)
#=====
#Plot figures
#=====
fig=plt.figure()
#Set aspect ratio
ar=3000.0
ax=plt.gca()
asr = 1.0/(ax.get_data_ratio()*ar)
im=plt.imshow(out,cmap='gray',extent=(0,1,0,3000))
plt.Axes.set_aspect(ax,asr)

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

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

```

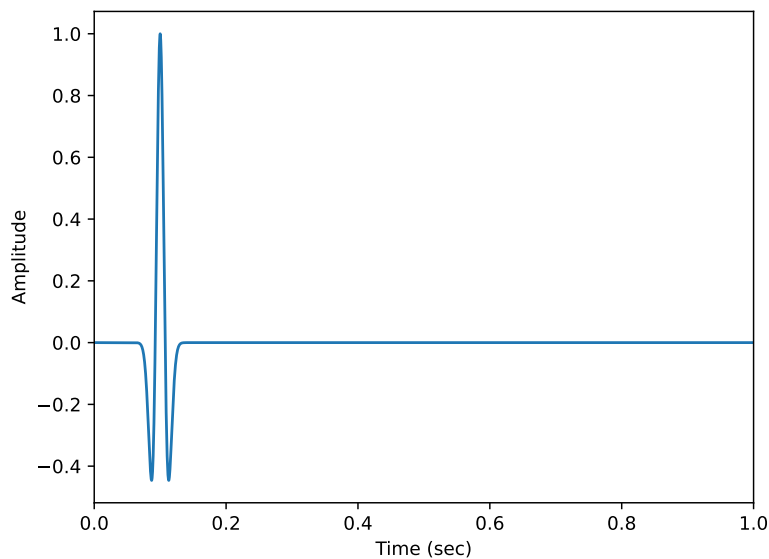


Figure 2: Ricker pulse with peak frequency 30Hz

```

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 velocity should be equal to 2000 m/s and the density set to 1000kg/m³. The source should have a peak frequency of 30Hz. Select suitable $\Delta x = 5.0\text{m}$ 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:

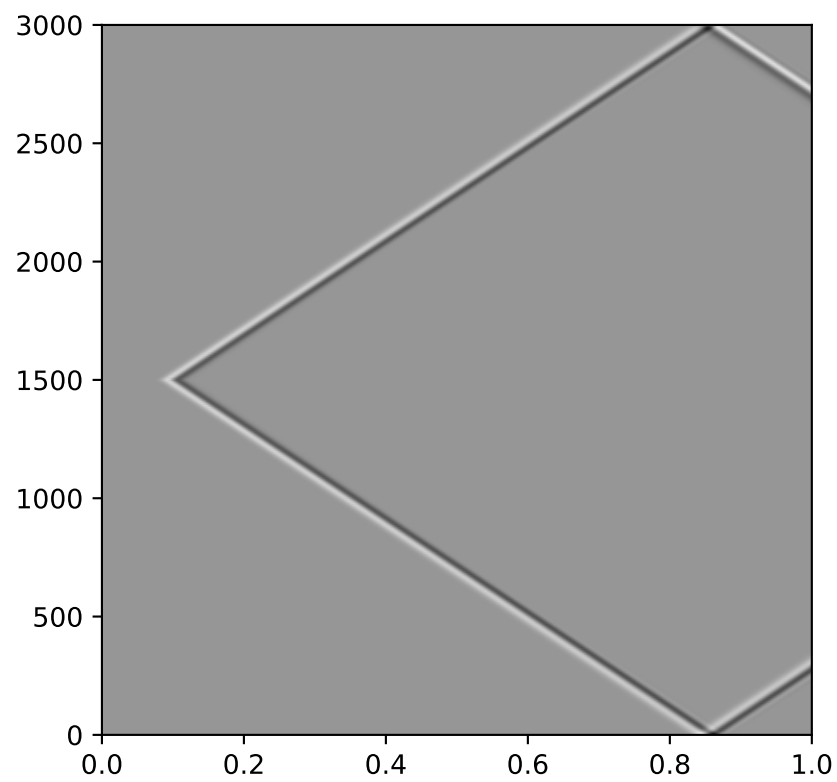


Figure 3: Stress for homogeneous model with $c = 2000$ m/s.

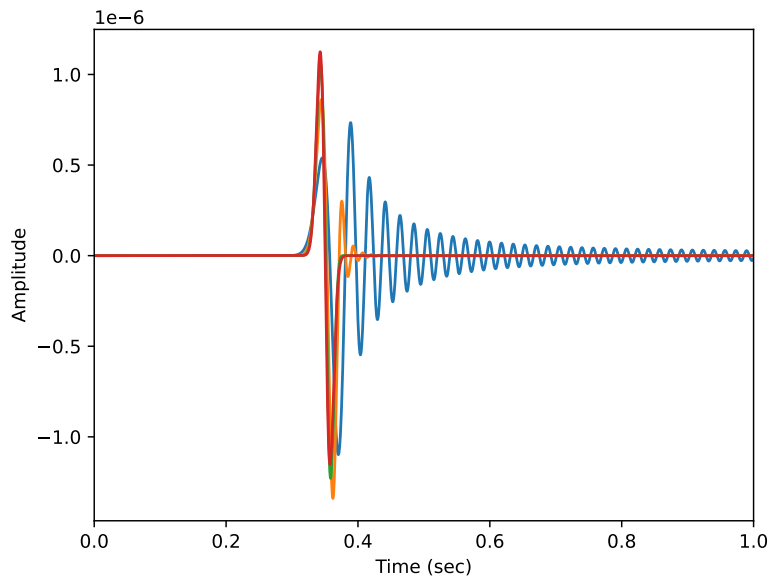


Figure 4: Pressure recorded at position of 2000m for $\Delta x = 10, 5, 2.5$ and 1m grid spacing.

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

$$\epsilon = \frac{\sqrt{\sum_{i=0}^N [p_1(i\Delta t) - p_2(i\Delta t)]^2}}{\sqrt{\sum_{i=0}^{N-1} p_1^2(i\Delta t)}} \quad (19)$$

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

Solution:

See figures 4 and 5.

Problem 4

- 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.

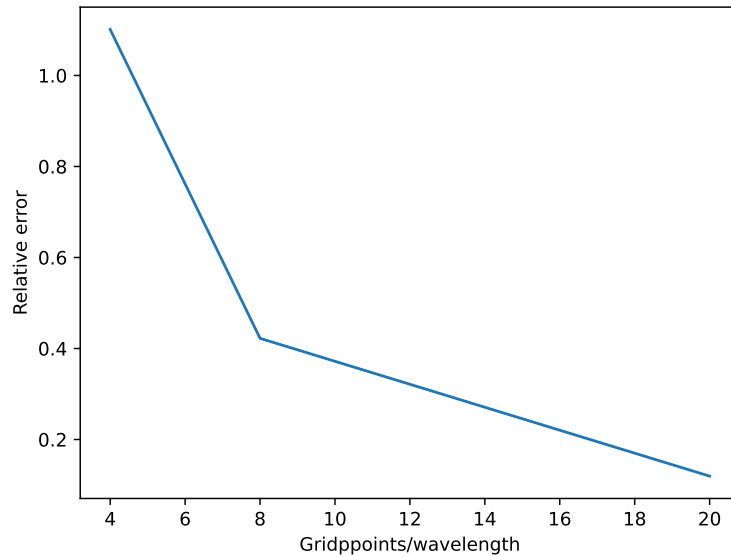


Figure 5: Relative error between pressure at 2000m for $\Delta x = 10, 5, 2.5$ and 1m grid spacing.

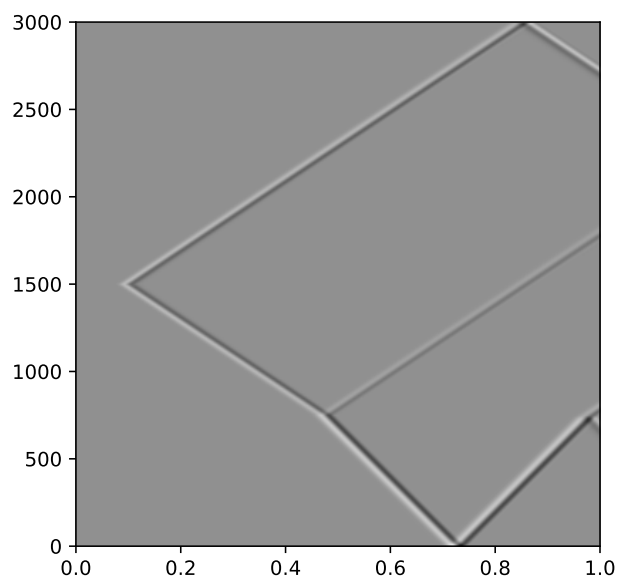


Figure 6: Stress as function of space and time

