Homework 1

Isaac Newton

ABSTRACT

This homework has three parts. In the theoretical part, you will derive some new forms of ray tracing equations and their solutions. In the computational part, you will experiment with wave propagation in a simple synthetic model. In the programming part, you will modify a finite-difference wave modeling program.

PREREQUISITES

Completing the computational part of this homework assignment requires

- `Madagascar` software environment available from `http://www.ahay.org`
- LaTeX environment with `SEGTeX` available from `http://www.ahay.org/wiki/SEGTeX`

You are welcome to do the assignment on your personal computer by installing the required environments. In this case, you can obtain all homework assignments from the `Madagascar` repository by running

```
svn co https://github.com/ahay/src/trunk/book/geo384w/hw1
```

You can also do this assignment in the computer lab at the Department of Geological Sciences (JGB 3.216B).

THEORETICAL PART

You can either write your answers on paper or edit them in the file `hw1/paper.tex`. Please show all the mathematical derivations that you perform.

1. In class, we used a mysterious parameter σ to represent a variable continuously increasing along a ray. There are other variables that can play a similar role.

(a) Transform the isotropic ray tracing system

$$\frac{d\mathbf{x}}{d\sigma} = \mathbf{p} \tag{1}$$
$$\frac{d\mathbf{p}}{d\sigma} = S(\mathbf{x})\,\nabla S \tag{2}$$
$$\frac{dT}{d\sigma} = S^2(\mathbf{x}) \tag{3}$$

into an equivalent system that uses λ instead of σ, where λ represents the length of the ray trajectory:

$$\frac{d\mathbf{x}}{d\lambda} = \tag{4}$$
$$\frac{d\mathbf{p}}{d\lambda} = \tag{5}$$
$$\frac{dT}{d\lambda} = S(\mathbf{x})\ . \tag{6}$$

Remember to check physical dimensions.

(b) Suppose you are given $T(\mathbf{x})$ – the traveltime from the source to all points $\mathbf{x}$ in the domain of interest. Your task is to find $\lambda(\mathbf{x})$ - the length of the ray trajectory at all $\mathbf{x}$. Derive a first-order partial differential equation that connects $\nabla\lambda$ and ∇T.

2. The so-called "parabolic" or 15° eikonal equation (Tappert, 1977; Claerbout, 1985; Bamberger et al., 1988) has the form

$$\frac{\partial T}{\partial x_1} + \frac{1}{2\,S(\mathbf{x})}\left(\frac{\partial T}{\partial x_2}\right)^2 = S(\mathbf{x}) \tag{7}$$

where $\mathbf{x} = \{x_1, x_2\}$ is a point in space, $T(\mathbf{x})$ is the traveltime, and $S(\mathbf{x})$ is slowness.

(a) Derive the ray tracing system for equation (7)

$$\frac{dx_2}{dx_1} = \tag{8}$$
$$\frac{dp_1}{dx_1} = \tag{9}$$
$$\frac{dp_2}{dx_1} = \tag{10}$$
$$\frac{dT}{dx_1} = \tag{11}$$

where p_1 represents $\partial T/\partial x_1$ and p_2 represents $\partial T/\partial x_2$.

(b) Assuming a constant slowness $S(\mathbf{x}) \equiv S_0$, solve the ray tracing system for a point source at the origin $\{x_1, x_2\} = \{0, 0\}$.

(c) Using the ray tracing solution, find the shape of the wavefronts defined by equation (7) in the case of a constant slowness.

(d) The isotropic eikonal equation

$$\left(\frac{\partial T}{\partial x_1}\right)^2 + \left(\frac{\partial T}{\partial x_2}\right)^2 = S^2(\mathbf{x}) \tag{12}$$

describes wavefronts of the wave equation

$$\nabla^2 P = S^2(\mathbf{x})\,\frac{\partial^2 P}{\partial t^2} + \cdots \tag{13}$$

with omitted possible first- and zero-order terms. What wave equation corresponds to equation (7)?

$$S(\mathbf{x})\,\frac{\partial^2 P}{\partial t^2} = \tag{14}$$

COMPUTATIONAL PART

In this part, we will simulate and observe acoustic wave propagation in a simple velocity model shown in Figure 1a. A wave snapshot is shown in Figure 1b.

1. Change directory to `hw1/wave`.
2. Run

   ```
   scons model.view
   ```

   to generate and view the velocity model from Figure 1a.
3. Run

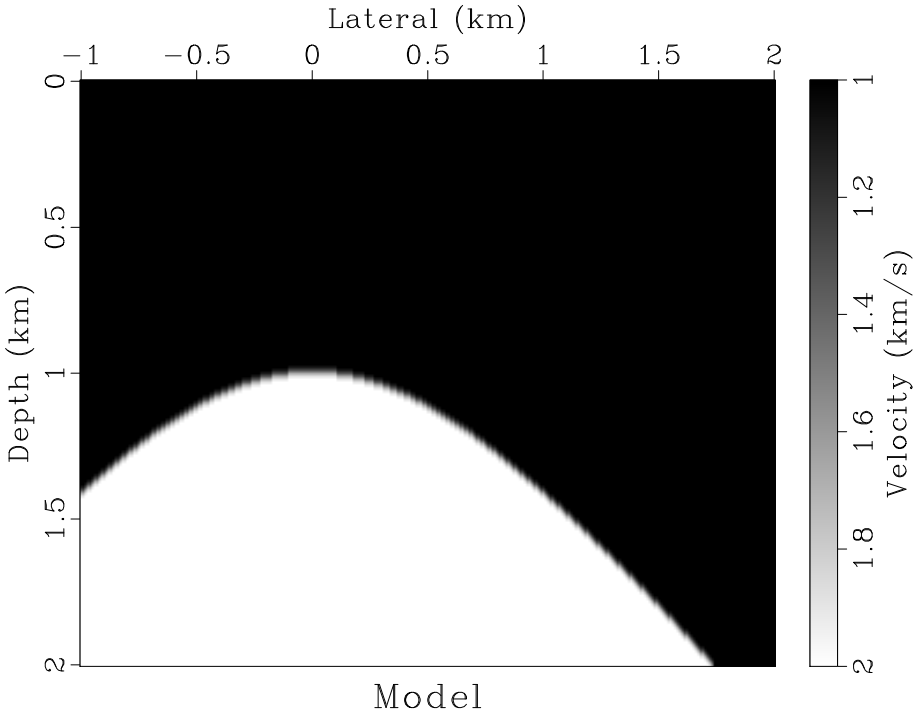
   ```
   scons wave.vpl
   ```

   to generate and observe a propagating wave on your screen.
4. Run

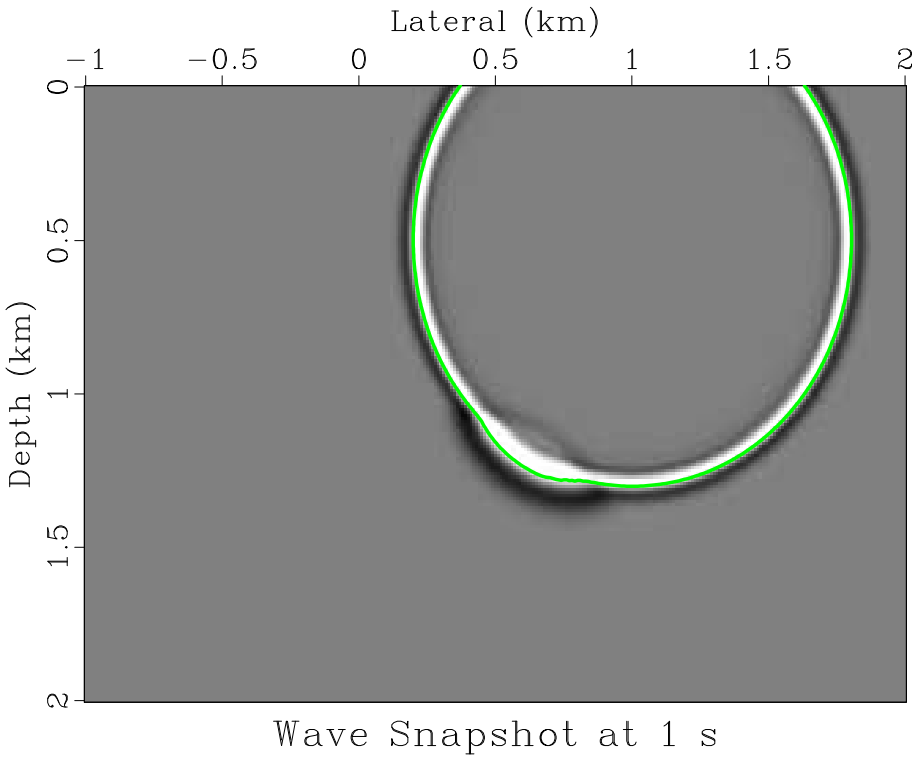
   ```
   scons fronts.vpl
   ```

   to generate and observe a propagating first-arrival wavefront on your screen.
5. Run

   ```
   scons snap.view
   ```



(a)



(b)

Figure 1: (a) Velocity model for simple wave propagation experiments. (b) Wave snapshot with overlaid first-arrival wavefront.

to generate and view a wave snapshot selected at 1 s as shown in Figure 1b.

6. Open the file `SConstruct` in your favorite editor. Your task is to make the following modifications in it:

   - Find a parameter responsible for selecting the time frame for the snapshot in Figure 1b. Modify it to increase the time from 1 s to your favorite point in the movie. Run `scons snap.view` again to verify your change.
   - Can you observe a geometrical part of the wave that is not captured by the first-arrival wavefront? What is its physical meaning?
   - Find a parameter in the `SConstruct` file responsible for the vertical smoothness of the model in Figure 1a. Modify it to increase the smoothness of the model in such a way that the first-arrival wavefront follows the wave geometry exactly. Run `scons snap.view` again to verify your change.
   - (EXTRA CREDIT) For extra credit, modify `SConstruct` to generate a movie of pictures like Figure 1b for a gradually increasing smoothness.

```
from rsf.proj import *

# Make a velocity model with a hyperbolic reflector
Flow('model',None,
     '''
     math n1=301 o1=-1 d1=0.01 output="sqrt(1+x1*x1)" |
     unif2 n1=201 d1=0.01 v00=1,2 |
     put label1=Depth unit1=km label2=Lateral unit2=km
     label=Velocity unit=km/s |
     smooth rect1=3
     ''')

# Plot model
Result('model',
       '''
       grey allpos=y title=Model bias=1
       scalebar=y barreverse=y
       ''')

# Source wavelet
Flow('wavelet',None,
     '''
     spike nsp=1 n1=2000 d1=0.001 k1=201 |
     ricker1 frequency=10
     ''')

# Extended model (for absorbing boundaries)
```

```
Flow('left','model',
     '''
     window n2=1 | spray axis=2 n=50 o=-1.5 d=0.01 |
     math output="input*exp(-4*(-1-x2)^2)"
     ''')
Flow('right','model',
     '''
     window n2=1 f2=300 | spray axis=2 n=50 o=3.01 d=0.01 |
     math output="input*exp(-4*(3-x2)^2)"
     ''')
Flow('emodel2','left model right',
     'cat axis=2 ${SOURCES[1:3]}')

Flow('top','emodel2',
     '''
     window n1=1 | spray axis=1 n=50 o=-0.5 d=0.01 |
     math output="input*exp(-4*x1^2)"
     ''')
Flow('bottom','emodel2',
     '''
     window n1=1 f1=200 | spray axis=1 n=50 o=2.01 d=0.01 |
     math output="input*exp(-4*(2-x1)^2)"
     ''')
Flow('emodel','top emodel2 bottom',
     'cat axis=1 ${SOURCES[1:3]}')

# Source location
Flow('source',None,
     '''
     spike k1=101 k2=251
     n2=401 o2=-1.5 d2=0.01 label1=Depth    unit1=km
     n1=301 o1=-0.5 d1=0.01 label2=Lateral unit2=km
     ''')

# Modeling
exe = Program('wave.c')
Flow('wave','source %s wavelet emodel' % exe[0],
     '''
     ./${SOURCES[1]} wav=${SOURCES[2]} v=${SOURCES[3]}
     jt=5 ft=200
     ''')

# Movie of wave snapshots
Plot('wave',
     '''
```

```
      window f1=50 f2=50 n1=201 n2=301 |
      grey gainpanel=all title=Wave
      ''',view=1)

# Your favorite time moment
##############################
time = 1.0 # !!! MODIFY ME
##############################

# Wavefield snapshot
Plot('snap','wave',
      '''
      window f1=50 f2=50 n1=201 n2=301 n3=1 min3=%g |
      grey title="Wave Snapshot at %g s"
      label1=Depth unit1=km label2=Lateral unit2=km
      ''' % (time,time))

# First-arrival traveltime
Flow('first','model',
      'eikonal yshot=1 zshot=0.5 | add add=0.2')

# Movie of first-arrival wavefronts
fronts = []
for snap in range(180):
    front = 'front%d' % snap
    fronts.append(front)
    tsnap = 0.2+snap*0.01
    Plot(front,'first',
         'contour nc=1 c0=%g title="%g s" ' % (tsnap,tsnap))
Plot('fronts',fronts,'Movie',view=1)

# First-arrival wavefront
Plot('front','first',
      '''
      contour nc=1 c0=%g wanttitle=n wantaxis=n
      plotcol=3 plotfat=5
      ''' % time)

# Overlay wavefront and traveltime
Result('snap','snap front','Overlay')

End()
```

PROGRAMMING PART (EXTRA CREDIT)

For extra credit, you can modify the wave modeling program to include anisotropic wave propagation effects. The program below (slightly modified from the original version by Paul Sava) implements wave modeling with equation

$$\frac{\partial^2 P}{\partial t^2} = V^2(\mathbf{x})\,\nabla^2 P + F(\mathbf{x},t) = V^2(\mathbf{x})\,\left(\frac{\partial^2 P}{\partial x_1^2} + \frac{\partial^2 P}{\partial x_2^2}\right) + F(\mathbf{x},t)\;, \tag{15}$$

where $F(\mathbf{x},t)$ is the source term. The implementation uses finite-difference discretization (second-order in time and fourth-order in space). Stepping in time involves the following computations:

$$\mathbf{P}_{t+\Delta t} = \left[V^2(\mathbf{x})\,\nabla^2 \mathbf{P}_t + F(\mathbf{x},t)\right]\,\Delta t^2 + 2\mathbf{P}_t - \mathbf{P}_{t-\Delta t}\;, \tag{16}$$

where $\mathbf{P}$ represents the propagating wavefield discretized at different time steps.

```
/* 2-D finite-difference acoustic wave propagation */
#include <rsf.h>

#ifdef _OPENMP
#include <omp.h>
#endif

static int n1, n2;
static float c0, c11, c21, c12, c22;

static void laplacian(float **uin  /* [n2][n1] */,
                      float **uout /* [n2][n1] */)
/* Laplacian operator, 4th-order finite-difference */
{
    int i1, i2;

#ifdef _OPENMP
#pragma omp parallel for                                \
    private(i2,i1)                                      \
    shared(n2,n1,uout,uin,c11,c12,c21,c22,c0)
#endif
    for (i2=2; i2 < n2-2; i2++) {
        for (i1=2; i1 < n1-2; i1++) {
            uout[i2][i1] =
                c11*(uin[i2][i1-1]+uin[i2][i1+1]) +
                c12*(uin[i2][i1-2]+uin[i2][i1+2]) +
                c21*(uin[i2-1][i1]+uin[i2+1][i1]) +
                c22*(uin[i2-2][i1]+uin[i2+2][i1]) +
                c0*uin[i2][i1];
        }
```

```
31      }
32  }
33
34  int main(int argc, char* argv[])
35  {
36      int it,i1,i2;          /* index variables */
37      int nt,n12,ft,jt;
38      float dt,d1,d2,dt2;
39
40      float  *ww,**vv,**rr;
41      float **u0,**u1,**u2,**ud;
42
43      sf_file Fw,Fv,Fr,Fo;   /* I/O files */
44
45      /* initialize Madagascar */
46      sf_init(argc,argv);
47
48      /* initialize OpenMP support */
49      omp_init();
50
51      /* setup I/O files */
52      Fr = sf_input ("in");     /* source position */
53      Fo = sf_output("out");    /* output wavefield */
54
55      Fw = sf_input ("wav");    /* source wavelet */
56      Fv = sf_input ("v");      /* velocity */
57
58      /* Read/Write axes */
59      if (!sf_histint(Fr,"n1",&n1))    sf_error("No n1= in inp");
60      if (!sf_histint(Fr,"n2",&n2))    sf_error("No n2= in inp");
61      if (!sf_histfloat(Fr,"d1",&d1)) sf_error("No d1= in inp");
62      if (!sf_histfloat(Fr,"d2",&d2)) sf_error("No d2= in inp");
63
64      if (!sf_histint(Fw,"n1",&nt))    sf_error("No n1= in wav");
65      if (!sf_histfloat(Fw,"d1",&dt)) sf_error("No d1= in wav");
66
67      n12 = n1*n2;
68
69      if (!sf_getint("ft",&ft)) ft=0;
70      /* first recorded time */
71      if (!sf_getint("jt",&jt)) jt=1;
72      /* time interval */
73
74      sf_putint(Fo,"n3",(nt-ft)/jt);
75      sf_putfloat(Fo,"d3",jt*dt);
```

```
76      sf_putfloat(Fo,"o3",ft*dt);
77
78      dt2 =     dt*dt;
79
80      /* set Laplacian coefficients */
81      d1 = 1.0/(d1*d1);
82      d2 = 1.0/(d2*d2);
83
84      c11 = 4.0*d1/3.0;
85      c12= -d1/12.0;
86      c21 = 4.0*d2/3.0;
87      c22= -d2/12.0;
88      c0  = -2.0 * (c11+c12+c21+c22);
89
90      /* read wavelet, velocity & source position */
91      ww=sf_floatalloc(nt);       sf_floatread(ww    ,nt ,Fw);
92      vv=sf_floatalloc2(n1,n2); sf_floatread(vv[0],n12,Fv);
93      rr=sf_floatalloc2(n1,n2); sf_floatread(rr[0],n12,Fr);
94
95      /* allocate temporary arrays */
96      u0=sf_floatalloc2(n1,n2);
97      u1=sf_floatalloc2(n1,n2);
98      u2=sf_floatalloc2(n1,n2);
99      ud=sf_floatalloc2(n1,n2);
100
101     for (i2=0; i2<n2; i2++) {
102         for (i1=0; i1<n1; i1++) {
103             u0[i2][i1]=0.0;
104             u1[i2][i1]=0.0;
105             u2[i2][i1]=0.0;
106             ud[i2][i1]=0.0;
107             vv[i2][i1] *= vv[i2][i1]*dt2;
108         }
109     }
110
111     /* Time loop */
112     for (it=0; it<nt; it++) {
113         laplacian(u1,ud);
114
115 #ifdef _OPENMP
116 #pragma omp parallel for              \
117     private(i2,i1)                    \
118     shared(ud,vv,ww,it,rr,u2,u1,u0)
119 #endif
120         for (i2=0; i2<n2; i2++) {
```

```
            for (i1=0; i1<n1; i1++) {
                /* scale by velocity */
                ud[i2][i1] *= vv[i2][i1];
                /* inject wavelet */
                ud[i2][i1] += ww[it] * rr[i2][i1];
                /* time step */
                u2[i2][i1] =
                    2*u1[i2][i1]
                    - u0[i2][i1]
                    + ud[i2][i1];
                u0[i2][i1] = u1[i2][i1];
                u1[i2][i1] = u2[i2][i1];
            }
        }

        /* write wavefield to output */
        if (it >= ft && 0 == (it-ft)%jt) {
            sf_warning("%d;",it+1);
            sf_floatwrite(u1[0],n12,Fo);
        }
    }
    sf_warning(".");

    exit (0);
}
```

```
! 2-D finite-difference acoustic wave propagation
module laplace
  ! Laplacian operator, 4th-order finite-difference
  implicit none
  real :: c0, c11, c21, c12, c22
contains
  subroutine laplacian_set(d1,d2)
    real, intent (in) :: d1,d2

    c11 = 4.0*d1/3.0
    c12= -d1/12.0
    c21 = 4.0*d2/3.0
    c22= -d2/12.0
    c0  = -2.0 * (c11+c12+c21+c22)
  end subroutine laplacian_set

  subroutine laplacian(uin, uout)
    real, dimension (:,:), intent (in)  :: uin
    real, dimension (:,:), intent (out) :: uout
```

```
20      integer n1, n2
21
22      n1 = size(uin,1)
23      n2 = size(uin,2)
24
25      uout(3:n1-2,3:n2-2) = &
26            c11*(uin(2:n1-3,3:n2-2) + uin(4:n1-1,3:n2-2)) + &
27            c12*(uin(1:n1-4,3:n2-2) + uin(5:n1,   3:n2-2)) + &
28            c21*(uin(3:n1-2,2:n2-3) + uin(3:n1-2,4:n2-1)) + &
29            c22*(uin(3:n1-2,1:n2-4) + uin(3:n1-2,5:n2  )) + &
30            c0*uin(3:n1-2,3:n2-2)
31    end subroutine laplacian
32  end module laplace
33
34  program Wave
35    use rsf
36    use laplace
37    implicit none
38
39    integer :: it, nt, ft, jt, n1, n2
40    real    :: dt, d1, d2, dt2
41
42    real, dimension (:),   allocatable :: ww
43    real, dimension (:,:), allocatable :: vv, rr
44    real, dimension (:,:), allocatable :: u0,u1,u2,ud
45
46    type(file) :: Fw,Fv,Fr,Fo   ! I/O files
47
48    call sf_init() ! initialize Madagascar
49
50    ! setup I/O files
51    Fr = rsf_input ("in")    ! source position
52    Fo = rsf_output("out")   ! output wavefield
53
54    Fw = rsf_input ("wav")   ! source wavelet
55    Fv = rsf_input ("v")     ! velocity
56
57    ! Read/Write axes
58    call from_par(Fr,"n1",n1)
59    call from_par(Fr,"n2",n2)
60    call from_par(Fr,"d1",d1)
61    call from_par(Fr,"d2",d2)
62
63    call from_par(Fw,"n1",nt)
64    call from_par(Fw,"d1",dt)
```

```
65
66    call from_par("ft",ft,0) ! first recorded time
67    call from_par("jt",jt,0) ! time interval
68
69    call to_par(Fo,"n3",(nt-ft)/jt)
70    call to_par(Fo,"d3",jt*dt)
71    call to_par(Fo,"o3",ft*dt)
72
73    dt2 = dt*dt
74    ft  = ft+1
75
76    ! set Laplacian coefficients
77    call laplacian_set(1.0/(d1*d1),1.0/(d2*d2))
78
79    ! read wavelet, velocity & source position
80    allocate (ww(nt), vv(n1,n2), rr(n1,n2))
81    call rsf_read(Fw,ww)
82    call rsf_read(Fv,vv)
83    call rsf_read(Fr,rr)
84
85    ! allocate temporary arrays */
86    allocate (u0(n1,n2), u1(n1,n2), u2(n1,n2), ud(n1,n2))
87
88    u0=0.0
89    u1=0.0
90    u2=0.0
91    ud=0.0
92    vv = vv*vv*dt2
93
94    ! Time loop
95    do it=1,nt
96       call laplacian(u1,ud)
97
98       ! scale by velocity
99       ud = ud*vv
100      ! inject wavelet
101      ud = ud + ww(it) * rr
102      ! time step
103      u2 = 2*u1 - u0 + ud
104      u0 = u1
105      u1 = u2
106
107      ! write wavefield to output
108      if (it >= ft .and. 0 == mod(it-ft,jt)) then
109         write(0,'(a,i4)',advance='no') achar(13), it
```

```
110         call rsf_write(Fo,u1)
111       end if
112     end do
113     write (0,*)
114
115     call exit(0)
116 end program Wave
```

```
1  #!/usr/bin/env python
2
3  import sys
4  import numpy
5  import m8r
6
7  class Laplacian:
8      'Laplacian operator, 4th-order finite-difference'
9
10     def __init__(self,d1,d2):
11         self.c11 = 4.0*d1/3.0
12         self.c12= -d1/12.0
13         self.c21 = 4.0*d2/3.0
14         self.c22= -d2/12.0
15         self.c0  = -2.0*(self.c11+self.c12+self.c21+self.c22)
16
17     def apply(self,uin,uout):
18         n1,n2 = uin.shape
19
20         uout[2:n1-2,2:n2-2] = \
21           self.c11*(uin[1:n1-3,2:n2-2]+uin[3:n1-1,2:n2-2]) + \
22           self.c12*(uin[0:n1-4,2:n2-2]+uin[4:n1  ,2:n2-2]) + \
23           self.c21*(uin[2:n1-2,1:n2-3]+uin[2:n1-2,3:n2-1]) + \
24           self.c22*(uin[2:n1-2,0:n2-4]+uin[2:n1-2,4:n2  ]) + \
25           self.c0*uin[2:n1-2,2:n2-2]
26
27 par = m8r.Par()
28
29 # setup I/O files
30 Fr=m8r.Input()        # source position
31 Fo=m8r.Output()       # output wavefield
32
33 Fv=m8r.Input("v")     # velocity
34 Fw=m8r.Input("wav")   # source wavefield
35
36
37 # Read/Write axes
```

```
a1 = Fr.axis(1); n1 = a1['n']; d1 = a1['d']
a2 = Fr.axis(2); n2 = a2['n']; d2 = a2['d']
at = Fw.axis(1); nt = at['n']; dt = at['d']

ft = par.int('ft',0)
jt = par.int('jt',0)

Fo.put('n3',(nt-ft)/jt)
Fo.put('d3',jt*dt)
Fo.put('o3',ft*dt)

dt2 =     dt*dt

# set Laplacian coefficients
laplace = Laplacian(1.0/(d1*d1),1.0/(d2*d2))

# read wavelet, velocity & source position
ww = numpy.zeros(nt,'f');       Fw.read(ww)
vv = numpy.zeros([n2,n1],'f'); Fv.read(vv)
rr = numpy.zeros([n2,n1],'f'); Fr.read(rr)

# allocate temporary arrays
u0 = numpy.zeros([n2,n1],'f')
u1 = numpy.zeros([n2,n1],'f')
u2 = numpy.zeros([n2,n1],'f')
ud = numpy.zeros([n2,n1],'f')

vv = vv*vv*dt2

# Time loop
for it in range(nt):
    laplace.apply(u1,ud)

    # scale by velocity
    ud = ud*vv
    # inject wavelet
    ud = ud + ww[it] * rr
    # time step
    u2 = 2*u1 - u0 + ud
    u0 = u1
    u1 = u2

    # write wavefield to output
    if it >= ft and 0 == (it-ft)%jt:
        sys.stderr.write("\b\b\b\b\b %d" % it)
```

```
83          Fo.write(u1)
```

Your task is to modify the code to implement elliptically-anisotropic wave propagation according to equation

$$\frac{\partial^2 P}{\partial t^2} = V_1^2(\mathbf{x})\,\frac{\partial^2 P}{\partial x_1^2} + V_2^2(\mathbf{x})\,\frac{\partial^2 P}{\partial x_2^2} + F(\mathbf{x},t) \tag{17}$$

You can test your implementation using a constant velocity example shown in Figure 2.

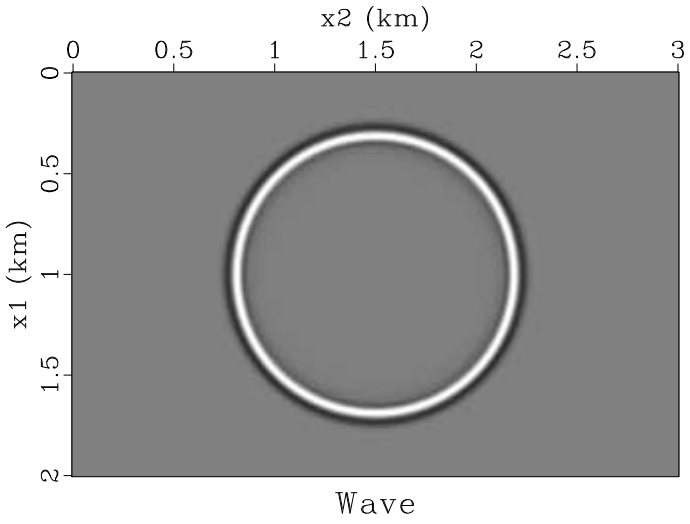


Figure 2: Wavefield snapshot for propagation from a point-source in a homogeneous medium. Modify the code to make wave propagation anisotropic.

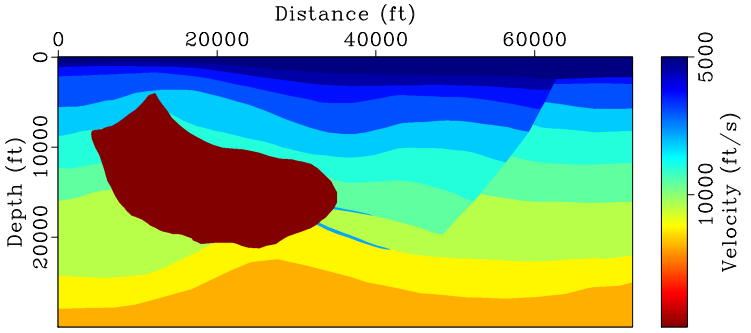
1. Change directory to `hw1/code`
2. Run

   ```
   scons wave.vpl
   ```

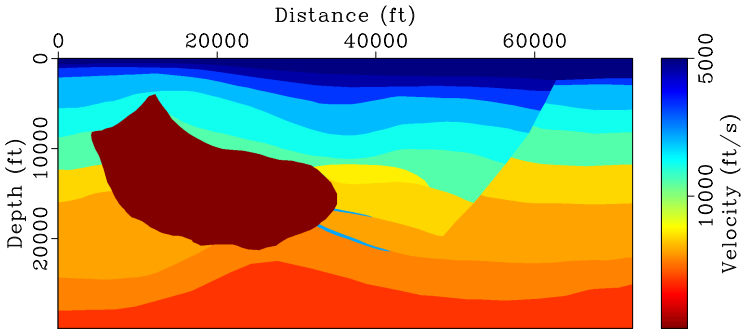
   to compile and run the program and to observe a propagating wave on your screen.
3. Open the file `wave.c` in your favorite editor and modify it to implement the wave operator from equation (17).
4. Run

   ```
   scons wave.vpl
   ```

   again to compile and test your program.
5. (EXTRA EXTRA CREDIT) For an additional test, modify the file `SConstruct` and run appropriate commands to output a snapshot of wave propagation in the Hess VTI model, shown in Figure 3[1]. Modify the file `paper.tex` to include your figure.



(a)



(b)

Figure 3: Vertical velocity (a) and horizontal velocity (b) in the Hess VTI model.

```
from rsf.proj import *
from rsf.prog import RSFROOT

# Program compilation
########################

proj = Project()

# To do the coding assignment in Fortran,
# comment the next line and uncomment the lines below
exe = proj.Program('wave.c')

# UNCOMMENT BELOW IF YOU WANT TO USE FORTRAN
#exe = proj.Program('wave.f90',
#                   F90PATH=os.path.join(RSFROOT,'include'),
#                   LIBS=['rsff90']+proj.get('LIBS'))

```

[1]The Hess VTI model was generated at Hess Corporation and released at `http://software.seg.org`. For this exercise, we are going to approximate it with an elliptically anisotropic model

```
# UNCOMMENT BELOW IF YOU WANT TO USE PYTHON
#exe = proj.Command('wave.exe','wave.py','cp $SOURCE $TARGET')
#AddPostAction(exe,Chmod(exe,0o755))


# Constant velocity test
##########################

# Source wavelet
Flow('wavelet',None,
     '''
     spike n1=1000 d1=0.001 k1=201 |
     ricker1 frequency=10
     ''')

# Source location
Flow('source',None,
     '''
     spike n1=201 n2=301 d1=0.01 d2=0.01
     label1=x1 unit1=km label2=x2 unit2=km
     k1=101 k2=151
     ''')

# Velocity model
Flow('v1','source','math output=1')
Flow('v2','source','math output=1.5')

# Modeling
Flow('wave','source %s wavelet v1 v2' % exe[0],
     '''
     ./${SOURCES[1]} wav=${SOURCES[2]}
     v=${SOURCES[3]}   vx=${SOURCES[4]}
     ft=200 jt=5
     ''')

Plot('wave','grey gainpanel=all title=Wave',view=1)

Result('wave',
       '''
       window n3=1 min3=0.9 |
       grey title=Wave screenht=8 screenwd=12
       ''')

# Download Hess VTI model
###########################
```

```
zcat = WhereIs('gzcat') or WhereIs('zcat')
for case in ('vp','epsilon'):
    sgy = 'timodel_%s.segy' % case
    sgyz = sgy + '.gz'
    Fetch(sgyz,dir='hessvti',
          server='https://s3.amazonaws.com',
          top='open.source.geoscience/open_data')

    # Uncompress
    Flow(sgy,sgyz,zcat + ' $SOURCE',stdin=0)
    # Convert to RSF format
    Flow(case,sgy,
         '''
         segyread read=data |
         window j1=2 j2=2 | put d1=40 d2=40
         unit1=ft label1=Depth unit2=ft label2=Distance
         ''')

# Horizontal velocity
Flow('vx','vp epsilon',
     'math e=${SOURCES[1]} output="input*sqrt(1+2*e)"')

for case in ('vp','vx'):
    Result(case,
           '''
           grey color=j pclip=100 allpos=y bias=5000
           scalebar=y barreverse=y wanttitle=n
           barlabel=Velocity barunit=ft/s
           screenht=5 screenwd=12 labelsz=6
           ''')

Flow('hsource','vp','spike k1=300 k2=900')
Flow('hess','hsource %s wavelet vp vx' % exe[0],
     '''
     ./${SOURCES[1]} wav=${SOURCES[2]}
     v=${SOURCES[3]}  vx=${SOURCES[4]}
     ft=200 jt=5
     ''')

End()
```

COMPLETING THE ASSIGNMENT

1. Change directory to `hw1`.
2. Edit the file `paper.tex` in your favorite editor and change the first line to have your name instead of Newton's.
3. Run

   ```
   sftour scons lock
   ```

   to update all figures.
4. Run

   ```
   sftour scons -c
   ```

   to remove intermediate files.
5. Run

   ```
   scons pdf
   ```

   to create the final document.
6. Submit your result (file `paper.pdf`) on paper or by e-mail.

REFERENCES

Bamberger, A., B. Engquist, L. Halpern, and P. Joly, 1988, Parabolic wave equation approximation in heterogenous media: SIAM Journal on Appl. Math., **48**, 99–128.

Claerbout, J. F., 1985, Imaging the Earth's interior: Blackwell Scientific Publications.

Tappert, F. D., 1977, The parabolic approximation method, *in* Wave Propagation and Underwater Acoustics: Springer, **70**, 224–287.