Homework 2

Leonhard Euler

ABSTRACT

This homework has three parts. In the theoretical part, you will derive analytical solutions to one-point and two-point ray tracing problems. In the computational part, you will experiment with traveltime computations with an eikonal solver and a field dataset from the Gulf of Mexico. In the programming part, you will implement your analytical solutions and investigate their accuracy.

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/hw2
```

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 `hw2/paper.tex`. Please show all the mathematical derivations that you perform.

1. In class, we derived analytical solutions for one-point and two-point ray tracing problems for the special case of a constant gradient of slowness squared

$$S^2(\mathbf{x}) = S^2(\mathbf{x_0}) + 2\,\mathbf{g}\cdot(\mathbf{x}-\mathbf{x_0})\;. \tag{1}$$

   In this homework, you will consider another special case, that of a constant gradient of velocity

$$V(\mathbf{x}) = \frac{1}{S(\mathbf{x})} = V(\mathbf{x_0}) + \mathbf{G}\cdot(\mathbf{x}-\mathbf{x_0})\;. \tag{2}$$

It is convenient to change the parameterization of the ray tracing system, with parameter ξ defined by equation (3) below:

$$\frac{d\,\mathbf{p}}{d\,\xi} = -\nabla V \;, \tag{3}$$

$$\frac{d\,\mathbf{x}}{d\,\xi} = \mathbf{p}\,V^3(\mathbf{x}) \;, \tag{4}$$

$$\frac{d\,T}{d\,\xi} = V(\mathbf{x}) \;. \tag{5}$$

and consider the one-point ray tracing problem with the initial conditions $\mathbf{x}(0) = \mathbf{x}_0$ and $\mathbf{p}(0) = \mathbf{p}_0$.

(a) Show that the solution of equation (3) for the constant gradient of velocity is

$$\mathbf{p}(\xi) = \mathbf{p}_0 - \mathbf{G}\,\xi \;. \tag{6}$$

and express velocity along the ray as a function of $\mathbf{p}_0$, $\mathbf{G}$, and ξ:

$$V(\mathbf{x}) = \frac{1}{\sqrt{\mathbf{p}\cdot\mathbf{p}}} = \tag{7}$$

(b) Let $a = \mathbf{p}\cdot(\mathbf{x}-\mathbf{x}_0)$. Using the chain rule, find the expression for

$$\frac{d\,a}{d\,\xi} = \tag{8}$$

and solve it to show it that

$$a(\xi) = V(\mathbf{x}_0)\,\xi \tag{9}$$

and

$$a_0(\xi) = \mathbf{p}_0\cdot(\mathbf{x}-\mathbf{x}_0) = V(\mathbf{x})\,\xi \tag{10}$$

(c) One way to seek the solution for the one-point ray tracing problem is to look for scalars α and β in the representation

$$\mathbf{x}(\xi) = \mathbf{x}_0 + \alpha(\xi)\,\mathbf{p}_0 + \beta(\xi)\,\mathbf{G} \;. \tag{11}$$

Under what condition does the linear system of equations

$$V(\mathbf{x})\,\xi = \mathbf{p}_0\cdot(\mathbf{x}-\mathbf{x}_0) = \alpha\,\mathbf{p}_0\cdot\mathbf{p}_0 + \beta\,\mathbf{p}_0\cdot\mathbf{G} \tag{12}$$

$$V(\mathbf{x}) - V(\mathbf{x}_0) = \mathbf{G}\cdot(\mathbf{x}-\mathbf{x}_0) = \alpha\,\mathbf{p}_0\cdot\mathbf{G} + \beta\,\mathbf{G}\cdot\mathbf{G} \tag{13}$$

have a unique solution for α and β? Solve the system to find α and β and obtain an analytical expression for the ray trajectory $\mathbf{x}(\xi)$.

(d) Express the squared distance between the ray end points

$$(\mathbf{x}-\mathbf{x}_0)\cdot(\mathbf{x}-\mathbf{x}_0) = \alpha^2\,\mathbf{p}_0\cdot\mathbf{p}_0 + 2\alpha\,\beta\,\mathbf{p}_0\cdot\mathbf{G} + \beta^2\,\mathbf{G}\cdot\mathbf{G}$$

$$= \tag{14}$$

in terms of $\mathbf{G}$, $\mathbf{p}_0$, and ξ.

(e) In the two-point problem, the unknown parameters are $(\mathbf{p}_0 \cdot \mathbf{G})$ and ξ. Express $(\mathbf{p}_0 \cdot \mathbf{G})$ from your equation (7) and substitute it into your equation (14). Solve for ξ.

(f) Finally, use ξ and $(\mathbf{p}_0 \cdot \mathbf{G})$ expressed in terms of $|\mathbf{x} - \mathbf{x}_0|$, $\mathbf{G}$, $V(\mathbf{x}_0)$, and $V(\mathbf{x})$ and substitute them into the one-point traveltime solution obtained by integrating equation (5)[1]

$$T(\xi) = \frac{1}{|\mathbf{G}|} \operatorname{arccosh}\left(1 + \frac{|\mathbf{G}|^2\, V(\mathbf{x})\, V^2(\mathbf{x}_0)\, \xi^2}{V(\mathbf{x}) + V(\mathbf{x}_0) - (\mathbf{p}_0 \cdot \mathbf{G})\, V(\mathbf{x})\, V^2(\mathbf{x}_0)\, \xi}\right) . \tag{15}$$

Your result will be the analytical two-point traveltime

$$\widehat{T}(\mathbf{x}_0, \mathbf{x}) = \tag{16}$$

2. In class, we discussed the hyperbolic traveltime approximation for normal moveout

$$T(h) \approx \sqrt{T_0^2 + \frac{h^2}{V_0^2}} . \tag{17}$$

More accurate approximations, involving additional parameters, are possible.

(a) Consider the following three-parameter approximation

$$T(h) \approx T_0 \left(1 - \frac{1}{S}\right) + \frac{1}{S} \sqrt{T_0^2 + S\, \frac{h^2}{V_0^2}} , \tag{18}$$

where S is the so-called "heterogeneity" parameter.

Evaluate parameter S in terms of the velocity $V(z)$ and the reflector depth z_0.

$$S = \tag{19}$$

by expanding equation (18) in a Taylor series around the zero offset $h = 0$ and comparing it with the corresponding Taylor series of the exact traveltime. The exact traveltime is given by the parametric equations

$$h(p) = \int_0^{z_0} \frac{p\, V(z)\, dz}{\sqrt{1 - p^2 V^2(z)}} , \tag{20}$$

$$T(p) = \int_0^{z_0} \frac{dz}{V(z) \sqrt{1 - p^2 V^2(z)}} . \tag{21}$$

(b) Let $\tau = T - p\, h$. Show that the function $\tau(p)$ can be approximated to the same accuracy by

$$\tau(p) \approx \tau_0 \left(1 - \frac{1}{S_\tau}\right) + \frac{\tau_0}{S_\tau} \sqrt{1 - S_\tau\, V_\tau^2\, p^2} . \tag{22}$$

Find τ_0, V_τ, and S_τ.

[1] $\operatorname{arccosh}(x)$ is the inverse hyperbolic cosine function defined as $\operatorname{arccosh}(x) = \ln\left(x + \sqrt{x^2 - 1}\right)$.

COMPUTATIONAL PART 1

In the first part of the computational assignment, you will investigate the numerical accuracy of a finite-difference eikonal solver.

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

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

proj = Project()

# COMMENT THE NEXT LINE FOR FORTRAN OR PYTHON
prog = proj.Program('analytical.c')

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


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

exe = str(prog[0])

# Velocity model
################################################
v0 = 0.5          # velocity at the origin
gz = 0.8          # vertical gradient
gx = 0.6          # horizontal gradient
nz = 601          # depth samples
nx = 1201         # lateral samples
dz = 0.5/(nz-1) # depth sampling
dx = 1.0/(nx-1) # lateral sampling

# v(z,x) = v0*/sqrt(1 - 2*v0*v0*(gz*z + gx*x))
################################################

# !!! COMMENT BELOW !!!

```

```
Flow('vel',None,
     '''
     math n1=%d d1=%g n2=%d d2=%g
     output="%g/sqrt(1-%g*(%g*x1+%g*x2))"
     ''' % (nz,dz,nx,dx,v0,2*v0*v0,gz,gx))

# v(z,x) = v0 + gz*z + gx*x
##############################

# !!! UNCOMMENT BELOW !!!

#Flow('vel',None,
#     '''
#     math n1=%d d1=%g n2=%d d2=%g
#     output="%g+%g*x1+%g*x2"
#     ''' % (nz,dz,nx,dx,v0,gz,gx))

Plot('vel',
     '''
     grey color=j allpos=y screenht=6 screenratio=%g
     bias=%g wanttitle=n barreverse=y
     label1=Depth unit1=km label2=Lateral unit2=km
     scalebar=y barlabel=Velocity barunit="km/s"
     ''' % ((nz-1.)/(nx-1.),v0))

# Analytical traveltime
##########################

# !!! CHANGE case=s to case=v BELOW !!!

Flow('time',['vel',exe],
     '''
     ./${SOURCES[1]} v0=%g g1=%g g2=%g case=s
     ''' % (1./(v0*v0),-gz,-gx))
Plot('time',
     '''
     contour scalebar=y plotcol=7 c0=0 dc=0.1 nc=30
     screenht=6 screenratio=%g plotfat=5 wanttitle=n
     ''' % ((nz-1.)/(nx-1.)))

# Plot wavefronts overlayed on the velocity model
#################################################
Result('exact','vel time','Overlay')

errs = []
```

```
spaces = []

for sample in range(1,nz//30):

    # Subsample velocity and analytical traveltime
    ##############################################
    vel = 'vel%d' % sample
    tim = 'tim%d' % sample
    window = 'window j1=%d j2=%d' % (sample,sample)

    Flow(vel,'vel',window)
    Flow(tim,'time',window)

    space = 'spc%d' % sample
    h = sample*math.hypot(dx,dz)
    Flow(space,None,'spike n1=1 mag=%g' % h)
    spaces.append(space)

    # Solve the eikonal equation
    ############################
    eik = 'eik%d' % sample

    # !!!!!!!! CHANGE BELOW !!!!!!!!!

    Flow(eik,vel,
         '''
         eikonal yshot=0 order=1 br1=%g br2=%g
         ''' % (20*dz,20*dx))

    # Compute error
    ###############
    err = 'err%d' % sample
    Flow(err,[eik,tim],
         '''
         math ref=${SOURCES[1]} output="abs(input-ref)" |
         stack axis=2 norm=y | stack axis=1 norm=y
         ''')
    errs.append(err)

# Concatenate results
#####################
cat = 'cat axis=1 ${SOURCES[1:%d]}' % len(spaces)

Flow('space',spaces,cat)
Flow('logspace','space','math output="log(input)" ')
```

```
131
132 Flow('err',errs,cat)
133 Flow('logerr','err','math output="log(input)" ')
134
135 graph = 'cmplx ${SOURCES[0:2]} | graph wanttitle=n '
136 for case in ('0','1'):
137     Plot('err'+case,'space err',graph + '''
138     label1="Grid Spacing" unit1=km
139     label2=Error unit2=s
140     ''',stdin=0)
141     Plot('logerr'+case,'logspace logerr',graph + '''
142     label1="Log(Grid Spacing)" unit1=
143     label2="Log(Error)" unit2= grid=y
144     ''',stdin=0)
145     graph = graph + 'symbol=x plotcol=2 '
146
147 # Plot error versus spacing
148 ##############################
149 Plot('err','err0 err1','Overlay')
150
151 # Plot error versus spacing on a logarithmic scale
152 ##################################################
153 Plot('logerr','logerr0 logerr1','Overlay')
154
155 Result('err','err logerr','SideBySideAniso')
156
157 End()
```

Figure 1 shows wavefronts in a medium with a constant gradient of slowness squared computed with an analytical two-point ray tracing formula from the class.

By computing traveltimes numerically at different sampling intervals and comparing the numerical result with the analytical one, we can get an experimental estimate of the numerical error behavior. The error is shown in Figure 2. The right plot in Figure 2 displays the error in logarithmic coordinates. The slope of the line in these coordinates shows directly the rate of convergence of the numerical method. For example, a first-order accurate method should have a slope of one.

1. Change directory

   ```
   > cd hw2/eikonal
   ```

2. Run

   ```
   > scons view
   ```

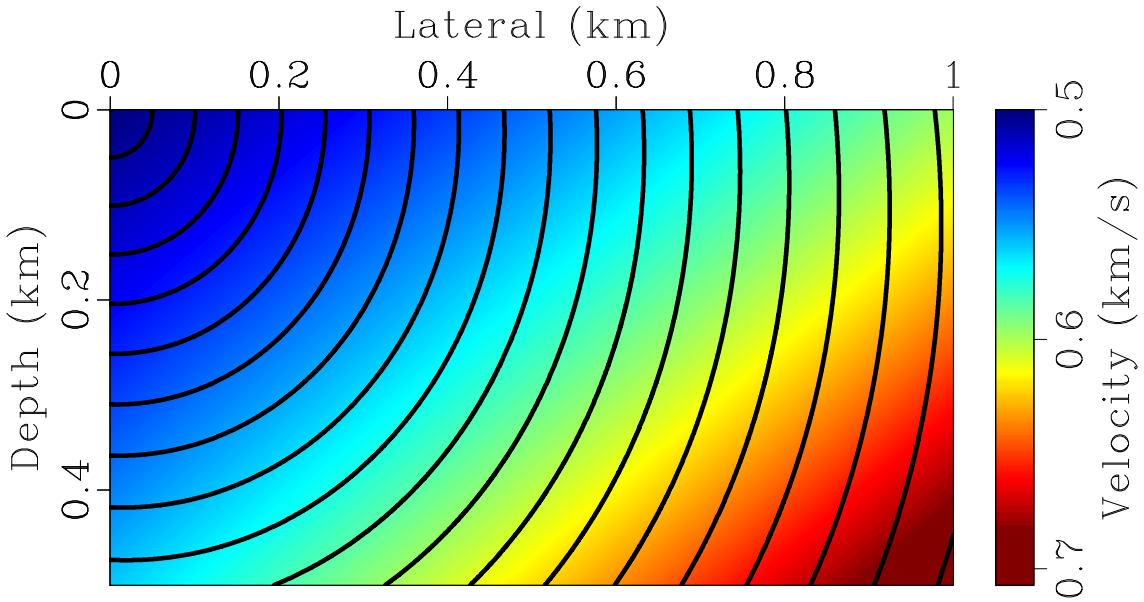

Figure 1: Wavefronts in a constant velocity gradient medium computed with an analytical formula.

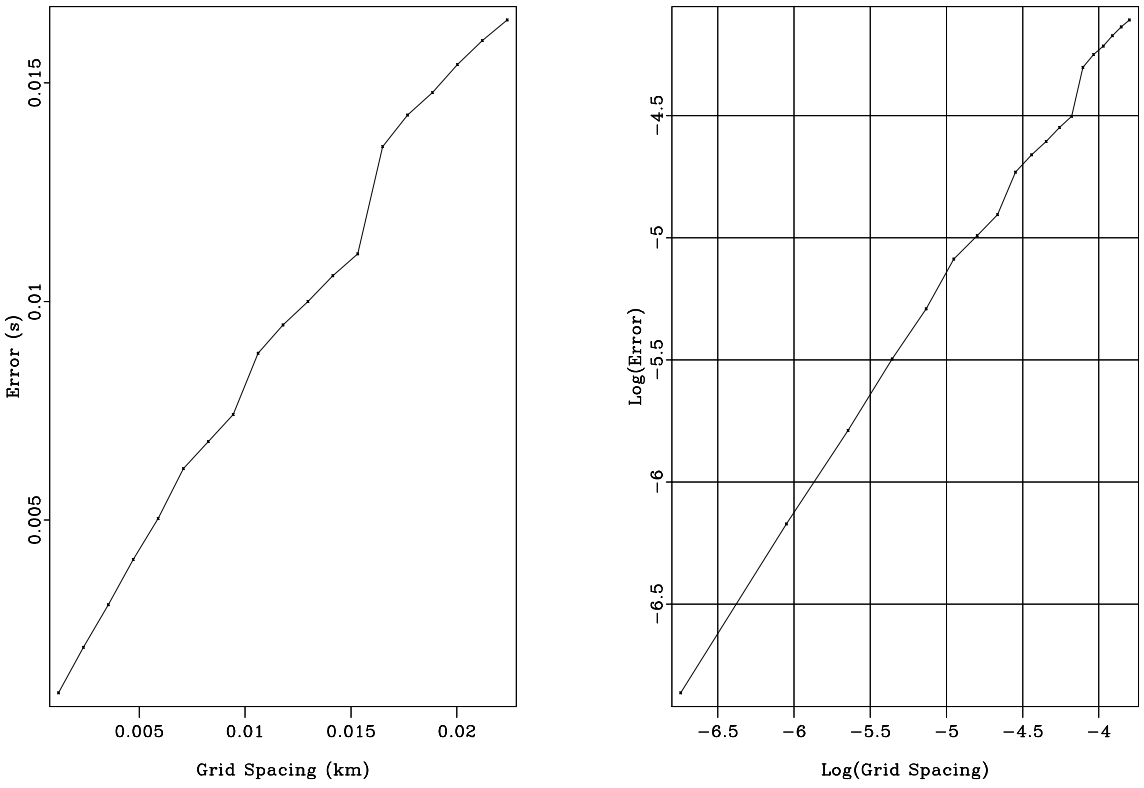

Figure 2: Left: average error of the finite-difference eikonal solver as a function of grid spacing. Right: the same on a log-log plot. The slope of the curve on the log-log plot indicates the order of the numerical accuracy.

to generate figures and display them on your screen.

3. In the `SConstruct` file, find the parameter that defines the order of accuracy for the eikonal solver. Change the order from 1 to 2 and recompute the results. Does the numerical accuracy change? What is the experimental order of accuracy?

PROGRAMMING PART 1

Instead of an analytical solution for a constant gradient of slowness squared, let us try an analytical solution for a constant gradient of velocity.

1. Change directory

   ```
   > cd hw2/eikonal
   ```

   - Uncomment the part of the `SConstruct` file that defines a velocity model with the constant velocity gradient.
   - Modify the program `analytical.c` to implement your equation (16).
   - Recompute the figures and check your results.

```c
/* Analytical first-arrival traveltimes. */
#include <math.h>
#include <rsf.h>

int main (int argc, char* argv[])
{
    char *type;
    int n1, n2, i1, i2;
    float d1,d2, g1,g2, s,v0, x1,x2,gsq,g,s2,z,d;
    float *time;
    sf_file in, out;

    sf_init (argc,argv);
    in = sf_input("in");
    out = sf_output("out");

    /* Get grid dimensions */
    if (!sf_histint(in,"n1",&n1)) sf_error("No n1=");
    if (!sf_histint(in,"n2",&n2)) sf_error("No n1=");
    if (!sf_histfloat(in,"d1",&d1)) sf_error("No d1=");
    if (!sf_histfloat(in,"d2",&d2)) sf_error("No d2=");

    if (!sf_getfloat("g1",&g1)) g1 = 0.;
```

```
24      /* vertical gradient */
25      if (!sf_getfloat("g2",&g2)) g2 = 0.;
26      /* horizontal gradient */
27      gsq = g1*g1+g2*g2;
28      g = sqrtf(gsq);
29
30      if (!sf_getfloat("v0",&v0)) sf_error("Need v0=");
31      /* initial velocity or slowness squared */
32
33      if (!sf_getfloat("s",&s)) s = 0.0;
34      /* shot location at the surface */
35
36      if (NULL == (type = sf_getstring("case"))) type="c";
37      /* case of velocity distribution */
38
39      if (0.0 == g1 && 0.0 == g2) type="const";
40
41      time = sf_floatalloc(n1);
42
43      for (i2 = 0; i2 < n2; i2++) {
44          x2 = i2*d2;
45          for (i1 = 0; i1 < n1; i1++) {
46              x1 = i1*d1;
47              d = x1*x1+(x2-s)*(x2-s);
48              switch (type[0]) {
49                  case 's':
50                      /* slowness squared */
51                      s2 = v0+g1*x1+g2*x2;
52                      z = 2.0*d/(s2+sqrtf(s2*s2-gsq*d));
53                      time[i1] = (s2-gsq*z/6.0)*sqrtf(z);
54                      break;
55                  case 'v':
56                      /* velocity */
57                      s2 = 2.0*v0*(v0+g1*x1+g2*x2);
58                      /* !!! CHANGE BELOW !!! */
59                      time[i1] = hypotf(x2-s,x1)/v0;
60                      break;
61                  case 'c': /* constant velocity */
62                  default:
63                      time[i1] = hypotf(x2-s,x1)/v0;
64                      break;
65              }
66          }
67          sf_floatwrite(time,n1,out);
68      }
```

```
69
70      exit (0);
71  }
```

```
1   program Analytical
2   ! Analytical first-arrival traveltimes. */
3   use rsf
4
5   character(len=FSTRLEN) :: type
6   integer :: n1, n2, i1, i2
7   real    :: d1,d2, g1,g2, s,v0, x1,x2,gsq,g,s2,z,d
8   real, dimension (:), allocatable :: time
9   type (file) :: in, out
10
11  call sf_init() ! initialize Madagascar
12  in = rsf_input("in")
13  out = rsf_output("out")
14
15  ! Get grid dimensions
16  call from_par(in,"n1",n1)
17  call from_par(in,"n2",n2)
18  call from_par(in,"d1",d1)
19  call from_par(in,"d2",d2)
20
21  call from_par("g1",g1,0.) ! vertical gradient
22  call from_par("g2",g2,0.) ! horizontal gradient
23
24  gsq = g1*g1+g2*g2
25  g = sqrt(gsq)
26
27  call from_par("v0",v0)
28  ! initial velocity or slowness squared
29
30  call from_par("s",s,0.)
31  ! shot location at the surface
32
33  call from_par("type",type,"constant")
34  ! case of velocity distribution
35
36  if (0.0 == g1 .and. 0.0 == g2) type="const"
37
38  allocate (time(n1))
39
40  do i2 = 1, n2
41     x2 = (i2-1)*d2
```

```
    do i1 = 1, n1
       x1 = (i1-1)*d1
       d = x1*x1+(x2-s)*(x2-s)

       select case (type(1:1))
       case("s") ! slowness squared
          s2 = v0+g1*x1+g2*x2
          z = 2.0*d/(s2+sqrt(s2*s2-gsq*d))
          time(i1) = (s2-gsq*z/6.0)*sqrt(z)
       case("v") ! velocity
          s2 = 2.0*v0*(v0+g1*x1+g2*x2)

!!! CHANGE BELOW !!!
          time(i1) = hypot(x2-s,x1)/v0
       case("c") ! constant velocity
       case default
          time(i1) = hypot(x2-s,x1)/v0
       end select
    end do
    call rsf_write(out,time)
end do

call exit (0)
end program Analytical
```

```
#!/usr/bin/env python

import numpy
from math import sqrt, hypot
import m8r

# initialize parameters
par = m8r.Par()

# input and output
inp=m8r.Input()
out=m8r.Output()

# get grid dimensions
n1 = inp.int('n1')
n2 = inp.int('n2')
d1 = inp.float('d1')
d2 = inp.float('d2')

g1 = par.float('g1',0.0) # vertical gradient
```

```
21 g2 = par.float('g2',0.0) # horizontal gradient
22
23 gsq = g1*g1+g2*g2
24 g = sqrt(gsq)
25
26 v0 = par.float('v0')
27 # initial velocity or slowness squared
28
29 s = par.float('s',0.0)
30 # shot location at the surface
31
32 type = par.string('case','constant')
33 # case of velocity distribution
34
35 if 0.0 == g1 and 0.0 == g2:
36     type='const'
37
38 time = numpy.zeros(n1,'f')
39
40 for i2 in range(n2):
41     x2 = i2*d2
42     for i1 in range(n1):
43         x1 = i1*d1
44         d = x1*x1+(x2-s)*(x2-s)
45
46         if type[0] == 's':
47         # slowness squared
48             s2 = v0+g1*x1+g2*x2
49             z = 2.0*d/(s2+sqrt(s2*s2-gsq*d))
50             time[i1] = (s2-gsq*z/6.0)*sqrt(z)
51
52         elif type[0] == 'v':
53         # velocity
54             s2 = 2.0*v0*(v0+g1*x1+g2*x2)
55
56 ### CHANGE BELOW ###
57             time[i1] = hypot(x2-s,x1)/v0
58
59         else:
60             # constant velocity
61             time[i1] = hypot(x2-s,x1)/v0
62
63     out.write(time)
```

COMPUTATIONAL PART 2

In the computational part, we begin working with field data. The left panel in Figure 3 shows a CMP (common midpoint) gather from the Gulf of Mexico (Claerbout, 2006).

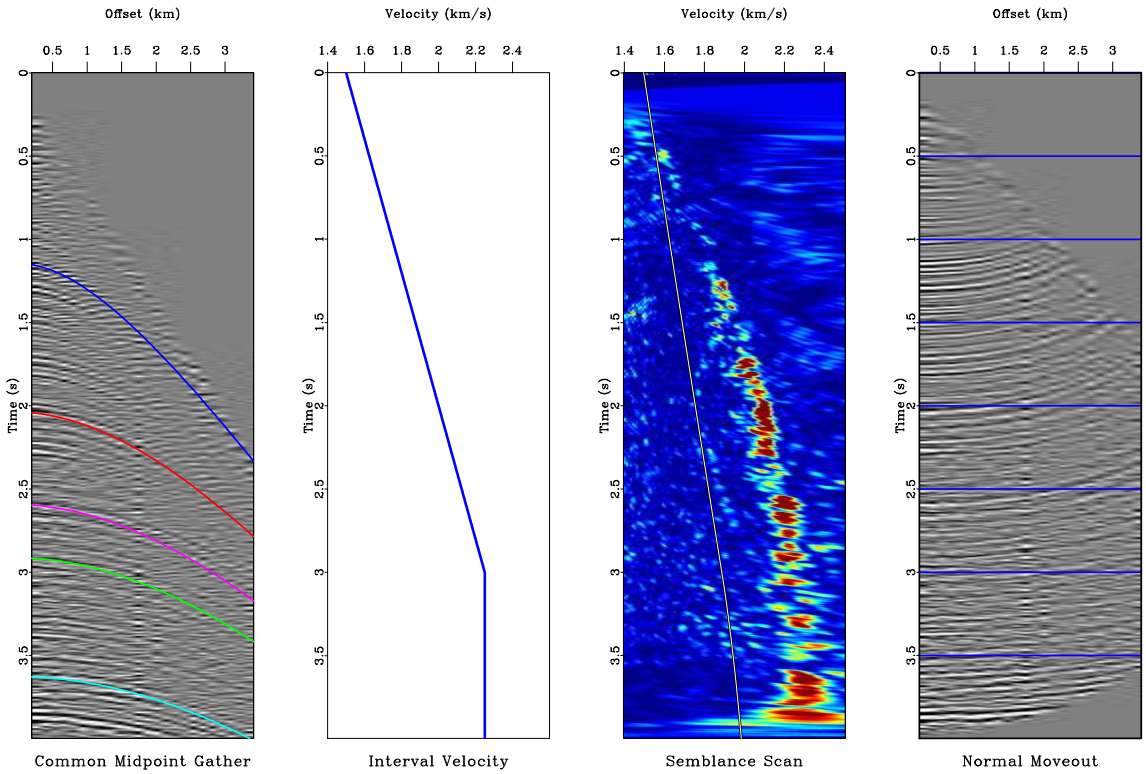

Figure 3: From left to right: (a) CMP (common midpoint) gather with overlaid traveltime curves. (b) Interval velocity. (c) RMS (root-mean-square) velocity overlaid on the semblance scan. (d) CMP gather after normal moveout.

We will assume a $V(z)$ medium and will use a very simple model of the interval velocity to explain the geometry of the observed data. The model involves two parameters: the initial gradient of velocity and the maximum velocity. The velocity function starts at the water velocity of 1.5 km/s and grows linearly with vertical time until it reaches the maximum velocity, after which point it remains flat. The panels in Figure 3 show the interval velocity, the corresponding RMS velocity (overlaid on the semblance scan), and the CMP gather after NMO (normal moveout).

Your task is to find the best values of the two model parameters for optimal prediction of the traveltime geometry and for flattening the CMP gather after NMO.

1. Change directory

   ```
   cd hw2/cmp
   ```

2. Run

```
scons cmps.vpl
```

to generate and display a movie looping through different values of the maximum velocity. If you are on a computer with multiple CPUs, you can also try

```
pscons cmps.vpl
```

to generate different movie frames faster by running computations in parallel.

3. Edit the `SConstruct` file to modify the velocity gradient. Check your result by running

   ```
   pscons cmps.vpl
   ```

   again.

4. Edit the `SConstruct` file to select the best frame of the movie (corresponding to the best maximum velocity). Display it by running

   ```
   scons view
   ```

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

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

proj = Project()

# COMMENT THE NEXT LINE FOR FORTRAN OR PYTHON
prog = proj.Program('traveltime.c')

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

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

exe = str(prog[0])

# Donwload data
```

```
Fetch('midpts.hh','midpts')

# Select a CMP gather, mute
Flow('cmp','midpts.hh',
     '''
     window n3=1 | dd form=native |
     mutter half=n v0=1.5 |
     put label1=Time unit1=s label2=Offset unit2=km
     ''')
Plot('cmp','grey title="Common Midpoint Gather" ')

# Velocity scan
Flow('vscan','cmp',
     'vscan half=n v0=1.4 nv=111 dv=0.01 semblance=y')
Plot('vscan','grey color=j allpos=y title="Semblance Scan" ')

##################################
grad = 0.25 # Velocity gradient
##################################

cmps = []
for iv in range(21):
    vmax = 1.5+0.2*grad*iv

    # Interval velocity
    vint = 'vint%d' % iv

    Flow(vint,None,
         '''
         math n1=1000 d1=0.004
         label1=Time unit1=s
         output="1.5+%g*x1" | clip clip=%g
         ''' % (grad,vmax))
    Plot(vint,
         '''
         graph yreverse=y transp=y pad=n plotfat=15
         title="Interval Velocity" min2=1.4 max2=%g
         wheretitle=b wherexlabel=t
         label2=Velocity unit2=km/s
         ''' % (1.6+4*grad))

    # Traveltimes
    time = 'time%d' % iv
    Flow(time,[vint,exe],
         '''
```

```
70          ./${SOURCES[1]} nr=5 r=285,509,648,728,906
71          nh=24 dh=0.134 h0=0.264 type=hyperbolic
72          ''')
73      Plot(time+'g',time,
74          '''
75          graph yreverse=y pad=n min2=0 max2=3.996
76          wantaxis=n wanttitle=n plotfat=10
77          ''')
78      Plot(time,['cmp',time+'g'],'Overlay')
79
80      # RMS velocity
81      vrms = 'vrms%d' % iv
82
83      Flow(vrms,vint,
84          '''
85          add mode=p $SOURCE | causint |
86          math output="sqrt(input*0.004/(x1+0.004))"
87          ''')
88      Plot(vrms+'w',vrms,
89          '''
90          graph yreverse=y transp=y pad=n
91          wanttitle=n wantaxis=n min2=1.4 max2=2.5
92          plotcol=7 plotfat=15
93          ''')
94      Plot(vrms+'b',vrms,
95          '''
96          graph yreverse=y transp=y pad=n
97          wanttitle=n wantaxis=n min2=1.4 max2=2.5
98          plotcol=0 plotfat=3
99          ''')
100     Plot(vrms,['vscan',vrms+'w',vrms+'b'],'Overlay')
101
102     # Normal moveout
103     nmo = 'nmo%d' % iv
104
105     Flow(nmo,['cmp',vrms],'nmo velocity=${SOURCES[1]} half=n')
106     Plot(nmo,
107         '''
108         grey title="Normal Moveout"
109         grid2=y gridcol=6 gridfat=10
110         ''')
111
112     # Display it together
113     allp = 'cmp%d' % iv
114     Plot(allp,[time,vint,vrms,nmo],
```

```
            'SideBySideAniso',vppen='txscale=1.5')

    cmps.append(allp)
Plot('cmps',cmps,'Movie',view=1)

##################################
frame = 15
##################################
Result('cmp','cmp%d' % frame,'Overlay')

Flow('time',['vint%d' % frame,exe],
     '''
     ./${SOURCES[1]} nr=1 r=500
     nh=1001 dh=0.01 h0=0 type=hyperbolic
     ''')
Result('time',
       '''
       graph title=Traveltime
       label2=Time unit2=s yreverse=y
       label1=Offset unit1=km
       ''')

End()
```

PROGRAMMING PART 2

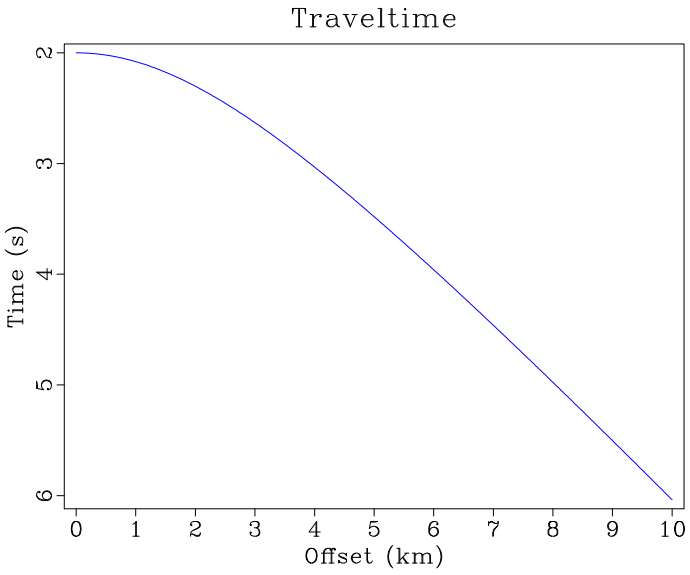

Figure 4: Traveltime in a $V(z)$ medium.

The program `cmp/traveltime.c` computes reflection traveltimes in a $V(z)$ medium

by using different methods.

1. Modify the program to implement approximation (18) using your equation (19).
2. Modify the program to implement exact traveltime computation by doing shooting iterations with equations (20-21). Using Newton's method, you can find the value of p for a given h by solving the non-linear equation $h(p) = h$ with iterations
$$p_{n+1} = p_n - \frac{h(p_n) - h}{h'(p_n)} \; . \tag{23}$$
3. For the traveltime in Figure 4, find the offset, where the absolute error of approximation (17) exceeds 0.05 s.
4. For the traveltime in Figure 4, find the offset, where the absolute error of approximation (18) exceeds 0.05 s.

```
/* Compute traveltime in a V(z) model. */
#include <rsf.h>

int main(int argc, char* argv[])
{
    char *type;
    int ih, nh, it, nt, ir, nr, *r, iter, niter;
    float h, dh, h0, dt, t0, t2, h2, v2, s, p, hp, tp;
    float *v, *t;
    sf_file vel, tim;

    /* initialize */
    sf_init(argc,argv);

    /* input and output */
    vel = sf_input("in");
    tim = sf_output("out");

    /* time axis from input */
    if (!sf_histint(vel,"n1",&nt)) sf_error("No n1=");
    if (!sf_histfloat(vel,"d1",&dt)) sf_error("No d1=");

    /* offset axis from command line */
    if (!sf_getint("nh",&nh)) nh=1;
    /* number of offsets */
    if (!sf_getfloat("dh",&dh)) dh=0.01;
    /* offset sampling */
    if (!sf_getfloat("h0",&h0)) h0=0.0;
```

```
29      /* first offset */
30
31      /* get reflectors */
32      if (!sf_getint("nr",&nr)) nr=1;
33      /* number of reflectors */
34      r = sf_intalloc(nr);
35      if (!sf_getints("r",r,nr)) sf_error("Need r=");
36
37      if (NULL == (type = sf_getstring("type")))
38          type = "hyperbolic";
39      /* traveltime computation type */
40
41      if (!sf_getint("niter",&niter)) niter=10;
42      /* maximum number of shooting iterations */
43
44      /* put dimensions in output */
45      sf_putint(tim,"n1",nh);
46      sf_putfloat(tim,"d1",dh);
47      sf_putfloat(tim,"o1",h0);
48      sf_putint(tim,"n2",nr);
49
50      /* read velocity */
51      v = sf_floatalloc(nt);
52      sf_floatread(v,nt,vel);
53      /* convert to velocity squared */
54      for (it=0; it < nt; it++) {
55          v[it] *= v[it];
56      }
57
58      t = sf_floatalloc(nh);
59
60      for (ir=0; ir<nr; ir++) {
61          nt = r[ir];
62          t0 = nt*dt; /* zero-offset time */
63          t2 = t0*t0;
64
65          p = 0.0;
66
67          for (ih=0; ih<nh; ih++) {
68              h = h0+ih*dh; /* offset */
69              h2 = h*h;
70
71              switch(type[0]) {
72                  case 'h': /* hyperbolic approximation */
73                      v2 = 0.0;
```

```
74                  for (it=0; it < nt; it++) {
75                      v2 += v[it];
76                  }
77                  v2 /= nt;
78
79                  t[ih] = sqrtf(t2+h2/v2);
80                  break;
81              case 's': /* shifted hyperbola */
82
83                  /* !!! MODIFY BELOW !!! */
84
85                  s = 0.0;
86
87                  v2 = 0.0;
88                  for (it=0; it < nt; it++) {
89                      v2 += v[it];
90                  }
91                  v2 /= nt;
92
93                  t[ih] = sqrtf(t2+h2/v2);
94                  break;
95              case 'e': /* exact */
96
97                  /* !!! MODIFY BELOW !!! */
98
99                  for (iter=0; iter < niter; iter++) {
100                     hp = 0.0;
101                     for (it=0; it < nt; it++) {
102                         v2 = v[it];
103                         hp += v2/sqrtf(1.0-p*p*v2);
104                     }
105                     hp *= p*dt;
106
107                     /* !!! SOLVE h(p)=h !!! */
108                 }
109
110                 tp = 0.0;
111                 for (it=0; it < nt; it++) {
112                     v2 = v[it];
113                     tp += dt/sqrtf(1.0-p*p*v2);
114                 }
115
116                 t[ih] = tp;
117                 break;
118             default:
```

```
                    sf_error("Unknown type");
                    break;
                }
            }

            sf_floatwrite(t,nh,tim);
        }

        exit(0);
}
```

```
program Traveltime
! Compute traveltime in a V(z) model.
use rsf

character(len=FSTRLEN) :: type
integer :: ih, nh, it, nt, ir, nr, iter, niter
real    :: h, dh, h0, dt, t0, t2, h2, v2, s, p, hp, tp
integer, allocatable, dimension (:) :: r
real,    allocatable, dimension (:) :: v, t
type (file) :: vel, tim

call sf_init() ! initialize Madagascar

! input and output
vel = rsf_input("in")
tim = rsf_output("out")

! time axis from input
call from_par(vel,"n1",nt)
call from_par(vel,"d1",dt)

! offset axis from command line

call from_par("nh",nh,1)     ! number of offsets
call from_par("dh",dh,0.01) ! offset sampling
call from_par("h0",h0,0.0)   ! first offset

! get reflectors

call from_par("nr",nr,1) ! number of reflectors

allocate (r(nr))

call from_par("r",r)
```

```
35
36 call from_par("type",type,"hyperbolic")
37 ! traveltime computation type
38
39 call from_par("niter",niter,10)
40 ! maximum number of shooting iterations
41
42 ! put dimensions in output
43 call to_par(tim,"n1",nh)
44 call to_par(tim,"d1",dh)
45 call to_par(tim,"o1",h0)
46 call to_par(tim,"n2",nr)
47
48 ! read velocity
49 allocate (v(nt))
50 call rsf_read(vel,v)
51
52 ! convert to velocity squared
53 v = v*v
54
55 allocate(t(nh))
56
57 do ir=1, nr
58     nt = r(ir)
59     t0 = nt*dt ! zero-offset time
60     t2 = t0*t0
61
62     p = 0.0;
63
64     do ih=1, nh
65         h = h0+(ih-1)*dh ! offset
66         h2 = h*h
67
68         select case (type(1:1))
69         case ("h") ! hyperbolic approximation
70             v2 = 0.0
71             do it=1, nt
72                 v2 = v2 + v(it)
73             end do
74             v2 = v2/nt
75
76             t(ih) = sqrt(t2+h2/v2)
77         case("s") ! shifted hyperbola
78
79 !!! MODIFY BELOW !!!
```

```
80
81            s = 0.0
82            v2 = 0.0
83            do it=1, nt
84                v2 = v2 + v(it)
85            end do
86            v2 = v2/nt
87
88            t(ih) = sqrt(t2+h2/v2)
89          case("e") ! exact
90
91  !!! MODIFY BELOW !!!
92
93            do iter=1, niter
94                hp = 0.0
95                do it=1,nt
96                    v2 = v(it)
97                    hp = hp + v2/sqrt(1.0-p*p*v2)
98                end do
99                hp = hp*p*dt
100
101 !!! SOLVE h(p)=h !!!
102
103           end do
104
105           tp = 0.0
106           do it=1, nt
107               v2 = v(it)
108               tp = tp + dt/sqrt(1.0-p*p*v2)
109           end do
110
111           t(ih) = tp
112         case default
113           call sf_error("Unknown type")
114         end select
115     end do
116
117     call rsf_write(tim,t)
118 end do
119
120 call exit(0)
121 end program Traveltime
```

```
1  #!/usr/bin/env python
2
```

```
import numpy
from math import sqrt
import m8r

# initialize parameters
par = m8r.Par()

# input and output
vel=m8r.Input()
tim=m8r.Output()

# time axis from input
nt = vel.int('n1')
dt = vel.float('d1')

# offset axis from command line
nh = par.int('nh',1)       # number of offsets
dh = par.float('dh',0.01) # offset sampling
h0 = par.float('h0',0.0)  # first offset

# get reflectors
nr = par.int('nr',1) # number of reflectors
r = par.ints('r',nr)

type = par.string('type','hyperbolic')
# traveltime computation type

niter = par.int('niter',10)
# maximum number of shooting iterations

# put dimensions in output
tim.put('n1',nh)
tim.put('d1',dh)
tim.put('o1',h0)
tim.put('n2',nr)

# read velocity
v = numpy.zeros(nt,'f')
vel.read(v)

# convert to velocity squared
v = v*v

t = numpy.zeros(nh,'f')

```

```
48 for ir in range(nr):
49     nt = r[ir]
50     t0 = nt*dt # zero-offset time
51     t2 = t0*t0
52
53     p = 0.0
54
55     for ih in range(nh):
56         h = h0+ih*dh # offset
57         h2 = h*h
58
59         if type[0] == 'h':
60             # hyperbolic approximation
61             v2 = numpy.sum(v)/nt
62             t[ih] = sqrt(t2+h2/v2)
63
64         elif type[0] == 's':
65             # shifted hyperbola
66
67             ### MODIFY BELOW ###
68
69             s = 0.0
70             v2 = numpy.sum(v)/nt
71             t[ih] = sqrt(t2+h2/v2)
72
73         elif type[0] == 'e':
74             # exact
75
76             ### MODIFY BELOW ###
77
78             for iter in range(niter):
79                 hp = 0.0
80                 for it in range(nt):
81                     v2 = v[it]
82                     hp += v2/sqrt(1.0-p*p*v2)
83                 hp *= p*dt
84
85             ### SOLVE h(p)=h ###
86
87             tp = 0.0
88             for it in range(nt):
89                 v2 = v[it]
90                 tp += dt/sqrt(1.0-p*p*v2)
91             t[ih] = tp
92         else:
```

```
93              raise RuntimeError('Unknown type')
94
95      tim.write(t)
```

COMPLETING THE ASSIGNMENT

1. Change directory to `hw2`.
2. Edit the file `paper.tex` in your favorite editor and change the first line to have your name instead of Euler'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

Claerbout, J. F., 2006, Basic Earth imaging: Stanford Exploration Project, `http://sepwww.stanford.edu/sep/prof/`. (Madagascar version at `http://www.ahay.org/RSF/book/bei/`).