

Homework 2

Charles Hewitt Dix

ABSTRACT

This homework has three parts. In the theoretical part, you will find the geometrical amplitude of the acoustic displacement, derive dynamic ray tracing equations, and extend the hyperbolic approximation of reflection moveouts. In the computational part, you will experiment with a field dataset from the Gulf of Mexico. In the programming part, you will implement exact and approximate traveltimes in a $V(z)$ medium.

PREREQUISITES

Completing the computational part of this homework assignment requires

- Madagascar software environment available from <http://www.ahay.org>
- L^AT_EX environment with SEGT_EX 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 this homework assignment from the Madagascar repository by running

```
svn co https://rsf.svn.sourceforge.net/svnroot/rsf/trunk/book/geo384w/hw2
```

The necessary environment is also installed in a computer lab at the Department of Geological Sciences.

THEORETICAL PART

1. In class, we derived the following acoustic wave equation for pressure $P(\mathbf{x}, t)$:

$$\frac{1}{V^2(\mathbf{x})} \frac{\partial^2 P}{\partial t^2} = \nabla^2 P + \rho(\mathbf{x}) \nabla \left(\frac{1}{\rho(\mathbf{x})} \right) \cdot \nabla P. \quad (1)$$

- (a) Using the connection between the pressure and displacement, derive the acoustic wave equation for the displacement vector $\mathbf{u}(\mathbf{x}, t)$:

$$\rho(\mathbf{x}) \frac{\partial^2 \mathbf{u}}{\partial t^2} = \quad (2)$$

- (b) Consider a geometrical wave representation in the vicinity of a wavefront

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{a}(\mathbf{x}) f(t - T(\mathbf{x})) \quad (3)$$

and derive partial differential equations for the traveltime function $T(\mathbf{x})$ and the vector amplitude $\mathbf{a}(\mathbf{x})$.

- (c) Assuming that the geometrical wave propagates in the direction of the traveltime gradient

$$\mathbf{a}(\mathbf{x}) = A(\mathbf{x}) V(\mathbf{x}) \nabla T \quad (4)$$

show that the amplitude continuation along a ray is given by equation

$$|\mathbf{a}_1| = |\mathbf{a}_0| \left(\frac{\rho_0 V_0 J_0}{\rho_1 V_1 J_1} \right)^{1/2}, \quad (5)$$

where J_0 and J_1 are the corresponding geometrical spreading factors.

- (d) (EXTRA CREDIT) Consider the elastic wave equation

$$\rho \ddot{u}_i = C_{ijkl,j} u_{k,l} + C_{ijkl} u_{k,lj} \quad (6)$$

in the case of an isotropic elasticity

$$C_{ijkl} = \lambda \delta_{ij} \delta_{kl} + \mu (\delta_{ik} \delta_{jl} + \delta_{il} \delta_{jk}). \quad (7)$$

Using the geometrical representation (3) with the P-wave polarization given by equation (4), show that the corresponding amplitude equation is similar to equation (5).

2. Consider a medium with a constant gradient of slowness squared

$$S^2(\mathbf{x}) = S_0^2 + 2 \mathbf{g} \cdot (\mathbf{x} - \mathbf{x}_0). \quad (8)$$

- (a) In the 2-D case, the ray-family coordinate system can be specified by $\mathbf{r} = \{\sigma, \theta\}$, where σ goes along the ray, and θ is the initial ray angle. A family of rays starts from the source point \mathbf{x}_0 which each ray traveling in the direction $\mathbf{p}_0 = \{S_0 \cos \theta, S_0 \sin \theta\}$. Show that the coordinate transformation matrix $\mathbf{P} = \partial \mathbf{p} / \partial \mathbf{r}$ changes along the ray as

$$\mathbf{P}(\sigma) = \begin{bmatrix} g_1 & -S_0 \sin \theta \\ g_2 & S_0 \cos \theta \end{bmatrix}, \quad (9)$$

where g_1 and g_2 are the components of \mathbf{g} and find the corresponding transformation of the matrices $\mathbf{X} = \partial \mathbf{x} / \partial \mathbf{r}$ and $\mathbf{K} = \mathbf{P} \mathbf{X}^{-1}$.

$$\mathbf{X}(\sigma) = \quad (10)$$

$$\mathbf{K}(\sigma) = \quad (11)$$

- (b) Find the one-point geometrical spreading J from a point source in the 2-D case as a function of S_0 , \mathbf{g} , the source location \mathbf{x}_0 , the initial ray direction θ , and the ray coordinate σ .
- (c) Using analytical ray tracing solutions, find the two-point geometrical spreading J from a point source in the 2-D case as a function of S_0 , \mathbf{g} , the source location \mathbf{x}_0 and the receiver location \mathbf{x}_1 .
3. In class, we discussed the hyperbolic traveltime approximation for normal move-out

$$T(h) \approx \sqrt{T_0^2 + \frac{h^2}{V_0^2}}. \quad (12)$$

More accurate approximations, involving additional parameters, are possible.

- (a) Consider the following three-parameter approximation

$$T(h) \approx T_0 \left(1 - \frac{1}{S}\right) + \frac{1}{S} \sqrt{T_0^2 + S \frac{h^2}{V_0^2}}, \quad (13)$$

where S is the so-called “heterogeneity” parameter.

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

$$S = \quad (14)$$

by expanding equation (13) 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

$$h = \int_0^{z_0} \frac{p V(z) dz}{\sqrt{1 - p^2 V^2(z)}}, \quad (15)$$

$$T = \int_0^{z_0} \frac{dz}{V(z) \sqrt{1 - p^2 V^2(z)}}. \quad (16)$$

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

$$\tau(p) \approx \tau_0 \left(1 - \frac{1}{S_\tau}\right) + \frac{\tau_0}{S_\tau} \sqrt{1 - S_\tau V_\tau^2 p^2}. \quad (17)$$

Find τ_0 , V_τ , and S_τ .

COMPUTATIONAL PART

In the computational part, we begin working with field data. The left panel in Figure 1 shows a CMP (common midpoint) gather from the Gulf of Mexico (Claerbout, 2006).

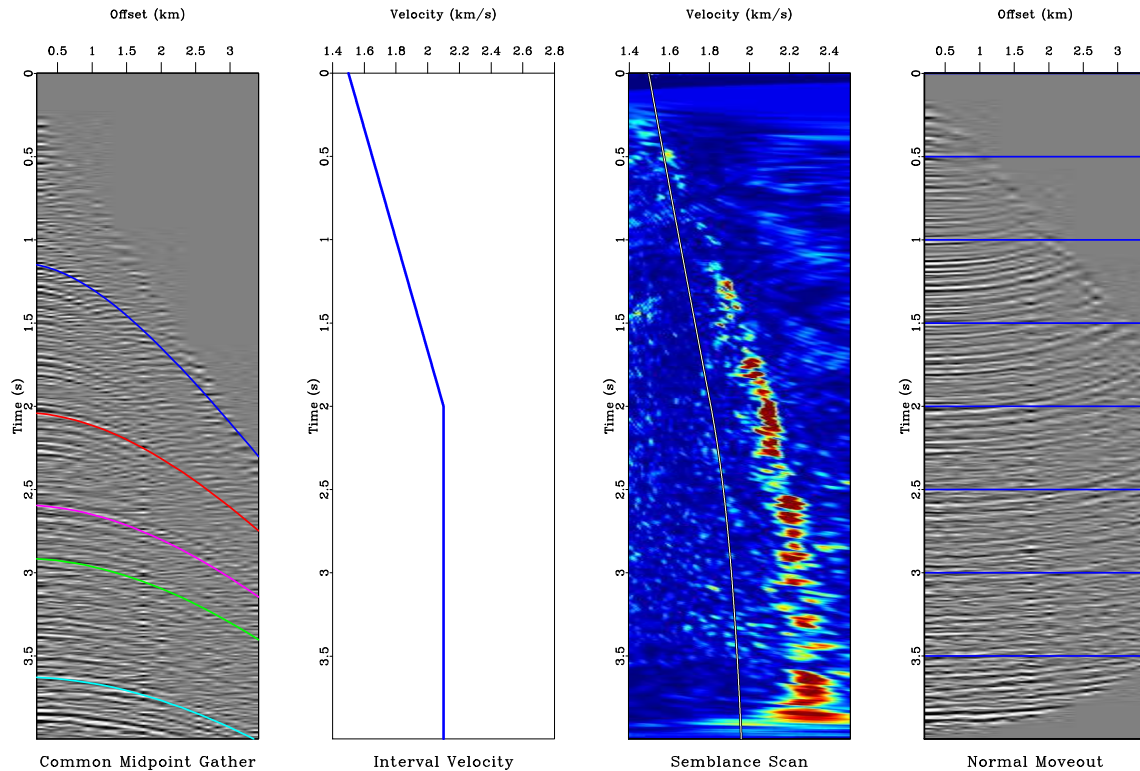


Figure 1: From left to right: (a) CMP (common midpoint) gather with overlaid traveltimes curves. (b) Interval velocity. (c) RMS (root-mean-square) velocity overlaid on the semblance scan. (d) CMP gather after normal moveout.

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 1 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 traveltimes 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
```

```

1 from rsf.proj import *
2
3 # Download data
4 Fetch( 'midpts.hh', 'midpts' )
5
6 # Select a CMP gather, mute
7 Flow( 'cmp', 'midpts.hh',
8       ' ',
9       window n3=1 | dd form=native |
10      mutter half=n v0=1.5 |
11      put labell=Time unit1=s label2=Offset unit2=km
12      ' ')
13 Plot( 'cmp', 'grey title="Common Midpoint Gather" ')
14
15 # Velocity scan
16 Flow( 'vscan', 'cmp',
17       'vscan half=n v0=1.4 nv=111 dv=0.01 semblance=y' )
18 Plot( 'vscan', 'grey color=j allpos=y title="Semblance Scan" ')
19
20 prog = Program( 'traveltime.c', CPPDEFINES='NO_BLAS', LIBS=[ 'rsf', 'm' ] )
21 exe = str( prog[0] )
22
23 #####
24 grad = 0.3 # Velocity gradient
25 #####
26

```

```

27 cmpr = []
28 for iv in range(21):
29     vmax = 1.5+0.2*grad*iv
30
31     # Interval velocity
32     vint = 'vint%d' % iv
33
34     Flow(vint, None,
35         '''
36         math n1=1000 d1=0.004
37         label1=Time unit1=s
38         output="1.5+%g*x1" | clip clip=%g
39         ''' % (grad, vmax))
40     Plot(vint,
41         '''
42         graph yreverse=y transp=y pad=n plotfat=15
43         title="Interval Velocity" min2=1.4 max2=%g
44         wheretitle=b wherexlabel=t
45         label2=Velocity unit2=km/s
46         ''' % (1.6+4*grad))
47
48     # Traveltimes
49     time = 'time%d' % iv
50     Flow(time, [vint, exe],
51         '''
52         ./${SOURCES[1]} nr=5 r=285,509,648,728,906
53         nh=24 dh=0.134 h0=0.264 type=hyperbolic
54         '''
55     )
56     Plot(time+'g', time,
57         '''
58         graph yreverse=y pad=n min2=0 max2=3.996
59         wantaxis=n wanttitle=n plotfat=10
60         '''
61     )
62     Plot(time, ['cmp', time+'g'], 'Overlay')
63
64     # RMS velocity
65     vrms = 'vrms%d' % iv
66
67     Flow(vrms, vint,
68         '''
69         add mode=p $SOURCE | causint |
70         math output="sqrt(input*0.004/(x1+0.004))"
71         '''
72     )
73     Plot(vrms+'w', vrms,
74         '''
75         graph yreverse=y transp=y pad=n
76         wanttitle=n wantaxis=n min2=1.4 max2=2.5

```

```

74         plotcol=7 plotfat=15
75         ' ' ')
76     Plot(vrms+'b',vrms,
77         ' ' ',
78         graph yreverse=y transp=y pad=n
79         wanttitle=n wantaxis=n min2=1.4 max2=2.5
80         plotcol=0 plotfat=3
81         ' ' ')
82     Plot(vrms,[ 'vscan ',vrms+'w',vrms+'b' ], 'Overlay')
83
84     # Normal moveout
85     nmo = 'nmo%d' % iv
86
87     Flow(nmo,[ 'cmp',vrms ], 'nmo velocity=${SOURCES[1]} half=n')
88     Plot(nmo,
89         ' ' ',
90         grey title="Normal Moveout"
91         grid2=y gridcol=6 gridfat=10
92         ' ' ')
93
94     # Display it together
95     cmp = 'cmp%d' % iv
96     Plot(cmp,[ time ,vint ,vrms ,nmo] ,
97         'SideBySideAniso ',vppen='txscale=1.5 ')
98
99     cmps.append(cmp)
100    Plot('cmps',cmps,'Movie',view=1)
101
102    #####
103    frame = 10
104    #####
105    Result('cmp','cmp%d' % frame,'Overlay')
106
107    Flow('time',[ 'vint%d' % frame ,exe] ,
108        ' ' ',
109        ./${SOURCES[1]} nr=1 r=500
110        nh=1001 dh=0.01 h0=0 type=hyperbolic
111        ' ' ')
112    Result('time',
113        ' ' ',
114        graph title=Traveltime
115        label2=Time unit2=s yreverse=y
116        label1=Offset unit1=km
117        ' ' ')
118
119    End()

```

PROGRAMMING PART

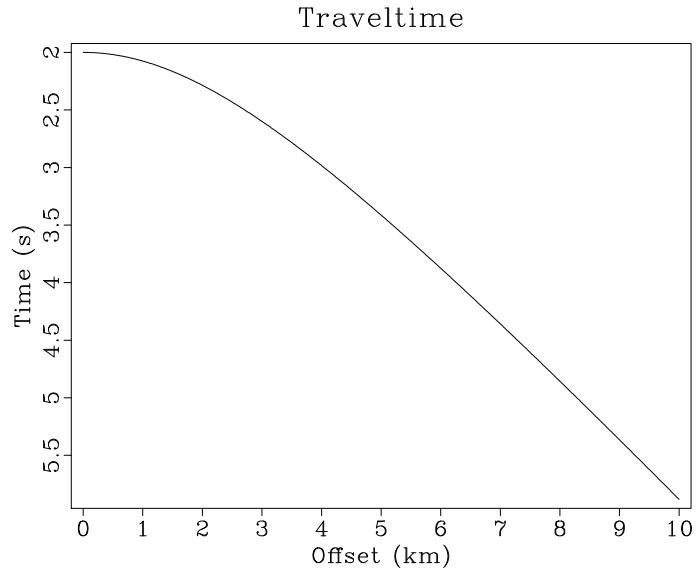


Figure 2: Traveltime in a $V(z)$ medium.

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

1. Modify the program to implement approximation (??) using your equation (14).
2. Modify the program to implement exact traveltime computation by doing shooting iterations with equations (15-16). 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

$$p_{n+1} = p_n - \frac{h(p_n) - h}{h'(p_n)}. \quad (18)$$

3. For the traveltime in Figure 2, find the offset, where the absolute error of the hyperbolic approximation (12) exceeds 0.1 s.
4. For the traveltime in Figure 2, find the offset, where the absolute error of the nonhyperbolic approximation (??) exceeds 0.1 s.

```

1  /* Compute traveltime in a V(z) model. */
2  #include <rsf.h>
3
4  int main(int argc, char* argv[])
5  {
6      char *type;
7      int ih, nh, it, nt, ir, nr, *r, iter, niter;
8      float h, dh, h0, dt, t0, t2, h2, v2, s, p, hp, tp;
9      float *v, *t;

```

```

10     sf_file vel, tim;
11
12     /* initialize */
13     sf_init(argc, argv);
14
15     /* input and output */
16     vel = sf_input("in");
17     tim = sf_output("out");
18
19     /* time axis from input */
20     if (!sf_histint(vel, "n1", &nt)) sf_error("No n1=");
21     if (!sf_histfloat(vel, "d1", &dt)) sf_error("No d1=");
22
23     /* offset axis from command line */
24     if (!sf_getint("nh", &nh)) nh=1;
25     /* number of offsets */
26     if (!sf_getfloat("dh", &dh)) dh=0.01;
27     /* offset sampling */
28     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     /* travelttime 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:
119         sf_error("Unknown type");
120         break;
121     }
122 }
123
124     sf_floatwrite(t,nh,tim);
125 }
126
127     exit(0);
128 }

```

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 Dix'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/>.