GEO384W
Seismic Imaging
Homework Assignments

Sergey Fomel

Contents

1 Homework 1 **1**

1.1 Prerequisites . . . 1

1.2 Theoretical part . . . 2

1.3 Computational part . . . 3

1.4 Programming part (extra credit) . . . 7

1.5 Completing the assignment . . . 13

2 Homework 2 **15**

2.1 Prerequisites . . . 15

2.2 Theoretical part . . . 16

2.3 Computational part . . . 18

2.4 Programming part . . . 22

2.5 Completing the assignment . . . 26

3 Homework 3 **27**

3.1 Theoretical part . . . 27

3.2 Computational part . . . 29

3.3 Completing the assignment . . . 42

4 Homework 4 **43**

4.1 Theoretical part . . . 43

4.2 Computational part . . . 44

4.3 Completing the assignment . . . 60

5 Homework 5 **61**

5.1 Computational part . . . 61

5.2 Completing the assignment 77

6 Homework 6 **79**

6.1 Computational part 79

6.2 Completing the assignment 86

Chapter 1

Homework 1

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 can modify a wave modeling program to implement anisotropic wave propagation.

1.1 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://rsf.svn.sourceforge.net/svnroot/rsf/trunk/book/geo384w
```

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

1.2 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

$$\frac{d\mathbf{x}}{d\sigma} = \mathbf{p} \tag{1.1}$$
$$\frac{d\mathbf{p}}{d\sigma} = S(\mathbf{x})\,\nabla S \tag{1.2}$$
$$\frac{dT}{d\sigma} = S^2(\mathbf{x}) \tag{1.3}$$

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

$$\frac{d\mathbf{x}}{d\lambda} = \tag{1.4}$$
$$\frac{d\mathbf{p}}{d\lambda} = \tag{1.5}$$
$$\frac{dT}{d\lambda} = S(\mathbf{x})\;. \tag{1.6}$$

   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 elliptically anisotropic 2-D eikonal equation has the form

$$V_1^2(\mathbf{x}) \left(\frac{\partial T}{\partial x_1}\right)^2 + V_2^2(\mathbf{x}) \left(\frac{\partial T}{\partial x_2}\right)^2 = 1\;, \tag{1.7}$$

   where $\mathbf{x} = \{x_1, x_2\}$ is a point in space, $T(\mathbf{x})$ is the traveltime, $V_1(\mathbf{x})$ is the horizontal velocity, and $V_2(\mathbf{x})$ is the vertical velocity.

   (a) Derive the ray tracing system

$$\frac{dx_1}{dT} = \tag{1.8}$$
$$\frac{dx_2}{dT} = \tag{1.9}$$
$$\frac{dp_1}{dT} = \tag{1.10}$$
$$\frac{dp_2}{dT} = \tag{1.11}$$

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

(b) Assuming constant (but unequal) velocities V_1 and V_2, solve the ray tracing system for a point source at the origin $x_1(0) = 0$, $x_2(0) = 0$ to show that the wavefronts in this case have an elliptical shape.

(c) The isotropic eikonal equation

$$\left(\frac{\partial T}{\partial x_1}\right)^2 + \left(\frac{\partial T}{\partial x_2}\right)^2 = S^2(\mathbf{x}) \tag{1.12}$$

describes wavefronts of the wave equation

$$S^2(\mathbf{x})\,\frac{\partial^2 P}{\partial t^2} = \nabla^2 P + \cdots \tag{1.13}$$

with omitted possible first- and zero-order terms. What wave equation corresponds to equation (1.7)?

$$\frac{\partial^2 P}{\partial t^2} = \tag{1.14}$$

1.3 Computational part

In this part, we will simulate and observe acoustic wave propagation in a simple velocity model shown in Figure 5.1. A wave snapshot is shown in Figure 1.1(b).

1. Change directory to `geo384w/hw1/wave`
2. Run

   ```
   scons model.view
   ```

   to generate and view the velocity model from Figure 5.1.
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
   ```

   to generate and view a wave snapshot selected at 1 s as shown in Figure 1.1(b).
6. Open the file `SConstruct` in your favorite editor. Your task is to make the following modifications in it:

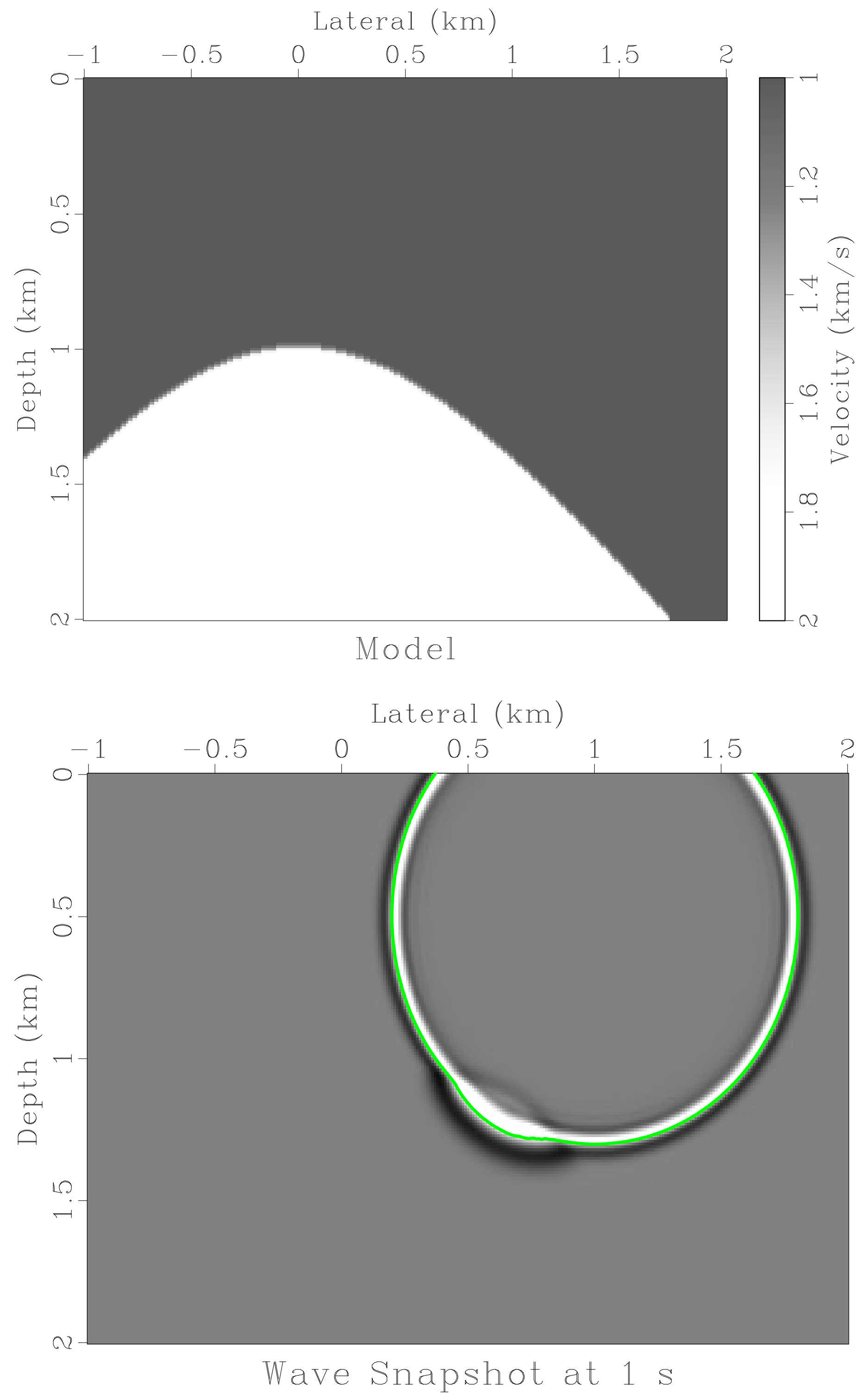


Figure 1.1: (a) Velocity model for simple wave propagation experiments. (b) Wave snapshot with overlaid first-arrival wavefront. $\boxed{\text{hw1/wave model,snap}}$

- Find a parameter responsible for selecting the time frame for the snapshot in Figure 1.1(b). 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 5.1. 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 1.1(b) 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
       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',CPPDEFINES='NO_BLAS')
Flow('wave','source %s wavelet emodel' % exe[0],
        '''
        ./${SOURCES[1]} wav=${SOURCES[2]} vel1=${SOURCES[3]} vel2=${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)

# 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()
```

1.4 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

$$\frac{1}{V^2(\mathbf{x})}\,\frac{\partial^2 P}{\partial t^2} = \nabla^2 P + F(\mathbf{x},t)\;, \tag{1.15}$$

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:

$$\mathbf{P}_{t+\Delta t} = \left[\nabla^2 \mathbf{P}_t + F(\mathbf{x},t)\right] V^2(\mathbf{x})\Delta t^2 + 2\mathbf{P}_t - \mathbf{P}_{t-\Delta t}\;, \tag{1.16}$$

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

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

#include <rsf.h>

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;

    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];
        }
    }
}

int main(int argc, char* argv[])
{
    int it,i1,i2;        /* index variables */
    int nt,n12,ft,jt;
    float dt,d1,d2,dt2;

    float  *ww,**v1,**v2,**rr; /* I/O arrays*/
    float **u0,**u1,**u2,**ud; /* tmp arrays */

    sf_file Fw,Fv1,Fv2,Fr,Fo,Fd; /* I/O files */
    sf_axis at,a1,a2;    /* cube axes */

    sf_init(argc,argv);

    /* setup I/O files */
    Fr = sf_input ("in");  /* source position */
    Fo = sf_output("out"); /* output wavefield */

   Fw = sf_input ("wav"); /* source wavelet */
    Fv1 = sf_input ("vel1"); /* vertical velocity */
    Fv2 = sf_input ("vel2"); /* horizontal velocity */

    /* Read/Write axes */
    at = sf_iaxa(Fw,1); nt = sf_n(at); dt = sf_d(at);
    a1 = sf_iaxa(Fr,1); n1 = sf_n(a1); d1 = sf_d(a1);
    a2 = sf_iaxa(Fr,2); n2 = sf_n(a2); d2 = sf_d(a2);
    n12 = n1*n2;

    if (!sf_getint("ft",&ft)) ft=0;
    /* first recorded time */
    if (!sf_getint("jt",&jt)) jt=1;
    /* time interval */

    sf_putint(Fo,"n3",(nt-ft)/jt);
    sf_putfloat(Fo,"d3",jt*dt);
    sf_putfloat(Fo,"o3",ft*dt);

    dt2 =     dt*dt;

    /* set Laplacian coefficients */
    d1 = 1.0/(d1*d1);
    d2 = 1.0/(d2*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);

    /* read wavelet, velocity & source position */
    rr=sf_floatalloc2(n1,n2); sf_floatread(rr[0],n12,Fr);
    ww=sf_floatalloc(nt);     sf_floatread(ww    ,nt  ,Fw);
    v1=sf_floatalloc2(n1,n2); sf_floatread(v1[0],n12,Fv1);
    v2=sf_floatalloc2(n1,n2); sf_floatread(v2[0],n12,Fv2);

    /* allocate temporary arrays */
    u0=sf_floatalloc2(n1,n2);
    u1=sf_floatalloc2(n1,n2);
    u2=sf_floatalloc2(n1,n2);
    ud=sf_floatalloc2(n1,n2);

    for (i2=0; i2<n2; i2++) {
        for (i1=0; i1<n1; i1++) {
            u0[i2][i1]=0.0;
            u1[i2][i1]=0.0;
            u2[i2][i1]=0.0;
            ud[i2][i1]=0.0;
            v1[i2][i1] *= v1[i2][i1]*dt2;
        }
    }

    /* Time loop */
    for (it=0; it<nt; it++) {
        laplacian(u1,ud);

        for (i2=0; i2<n2; i2++) {
            for (i1=0; i1<n1; i1++) {
                /* inject wavelet */
                ud[i2][i1] += ww[it] * rr[i2][i1];
                /* scale by velocity */
                ud[i2][i1] *= v1[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);
}
```

Your task is to modify the code to implement your anisotropic equation (1.14). You will test your implementation using a constant velocity example shown in Figure 1.2.

1. Change directory to `geo384w/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 (1.14).
4. Run

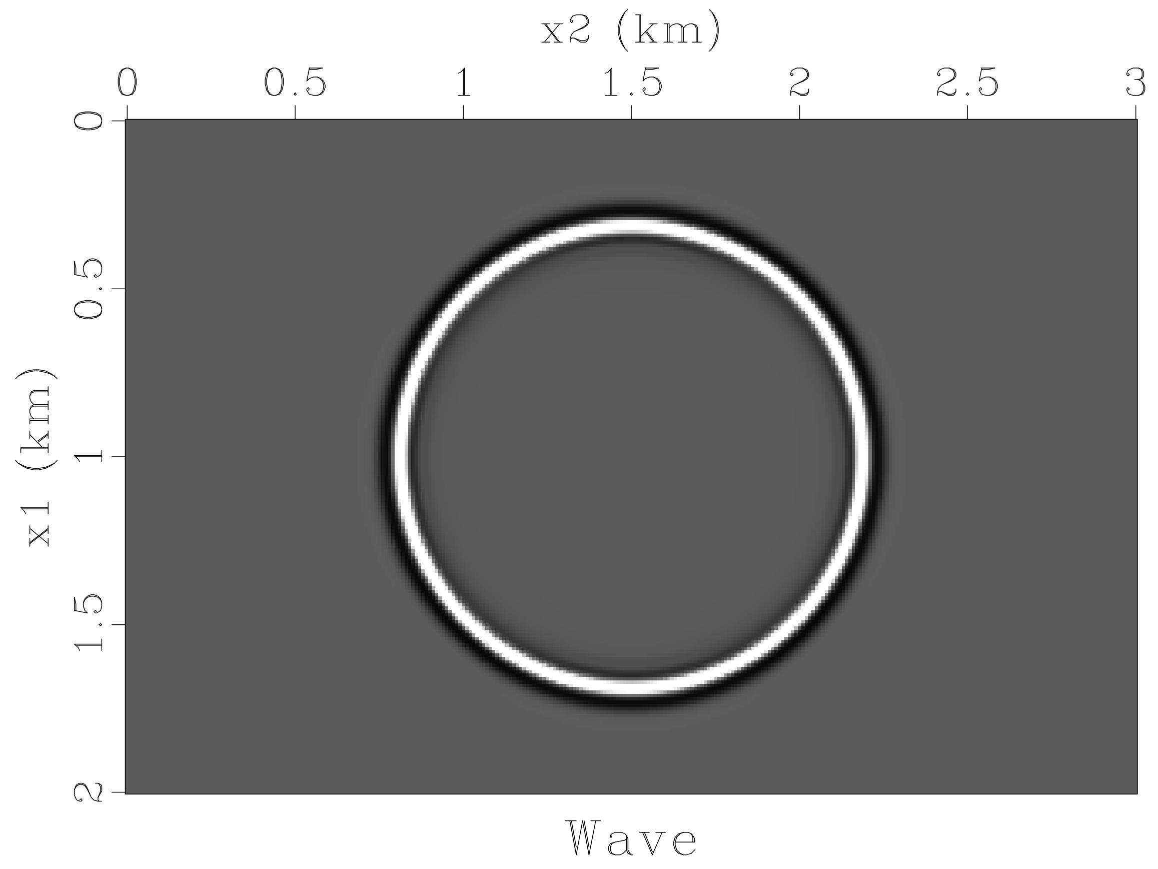


Figure 1.2: Wavefield snapshot for propagation from a point-source in a homogeneous medium. Modify the code to make wave propagation anisotropic. hw1/code wave

```
scons wave.vpl
```

again to compile and test your program. If you want to add additional tests, modify the file `SConstruct`.

```
from rsf.proj import *

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

proj = Project()

exe = proj.Program('wave.c',CPPDEFINES='NO_BLAS')

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

# Source wavelet
Flow('wavelet',None,
     '''
     spike nsp=1 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]}
     vel1=${SOURCES[3]} vel2=${SOURCES[4]}
     ''')

Plot('wave',
     '''
     window j3=5 f3=200 |
     grey gainpanel=all title=Wave
     ''',view=1)
```

```

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

End()
```

1.5 Completing the assignment

1. Change directory to `geo384w/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.

Chapter 2

Homework 2

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 traveltime computations in a $V(z)$ medium.

2.1 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 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.

2.2 Theoretical part

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

$$\frac{1}{V^2(\mathbf{x})}\,\frac{\partial^2 P}{\partial t^2} = \nabla^2 P + \rho(\mathbf{x})\,\nabla\left(\frac{1}{\rho(\mathbf{x})}\right)\cdot\nabla P\;. \tag{2.1}$$

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

$$\rho(\mathbf{x})\,\frac{\partial^2 \mathbf{u}}{\partial t^2} = \tag{2.2}$$

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

$$\mathbf{u}(\mathbf{x},t) = \mathbf{a}(\mathbf{x})\,f\left(t - T(\mathbf{x})\right) \tag{2.3}$$

   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

$$\mathbf{a}(\mathbf{x}) = A(\mathbf{x})\,V(\mathbf{x})\,\nabla T \tag{2.4}$$

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

$$|\mathbf{a}_1| = |\mathbf{a}_0|\left(\frac{\rho_0\,V_0\,J_0}{\rho_1\,V_1\,J_1}\right)^{1/2}\;, \tag{2.5}$$

   where J_0 and J_1 are the corresponding geometrical spreading factors.

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

$$\rho\,\ddot{u}_i = C_{ijkl,j}\,u_{k,l} + C_{ijkl}\,u_{k,lj} \tag{2.6}$$

   in the case of an isotropic elasticity

$$C_{ijkl} = \lambda\,\delta_{ij}\,\delta_{kl} + \mu\left(\delta_{ik}\,\delta_{jl} + \delta_{il}\,\delta_{jk}\right)\;. \tag{2.7}$$

   Using the geometrical representation (2.3) with the P-wave polarization given by equation (2.4), show that the corresponding amplitude equation is similar to equation (2.5).

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

$$S^2(\mathbf{x}) = S_0^2 + 2\,\mathbf{g}\cdot(\mathbf{x}-\mathbf{x}_0)\;. \tag{2.8}$$

   (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

$$\mathbf{P}(\sigma) = \left[\begin{array}{cc} g_1 & -S_0\,\sin\theta \\ g_2 & S_0\,\cos\theta \end{array}\right]\;, \tag{2.9}$$

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 $\mathbf{K} = \mathbf{P}\,\mathbf{X}^{-1}$.

$$\mathbf{X}(\sigma) = \tag{2.10}$$

$$\mathbf{K}(\sigma) = \tag{2.11}$$

(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 moveout

$$T(h) \approx \sqrt{T_0^2 + \frac{h^2}{V_0^2}}\,. \tag{2.12}$$

More accurate approximations, involving additional parameters, are possible.

(a) Consider the following three-parameter approximation

$$T(h) \approx T_0\left(1 - \frac{1}{S}\right) + \frac{1}{S}\sqrt{T_0^2 + S\,\frac{h^2}{V_0^2}}\,, \tag{2.13}$$

where S is the so-called "heterogeneity" parameter.
Evaluate parameter S in terms of the velocity $V(z)$ and the reflector depth z_0.

$$S = \tag{2.14}$$

by expanding equation (2.13) 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

$$h = \int\limits_0^{z_0} \frac{p\,V(z)\,dz}{\sqrt{1 - p^2 V^2(z)}}\,, \tag{2.15}$$

$$T = \int\limits_0^{z_0} \frac{dz}{V(z)\,\sqrt{1 - p^2 V^2(z)}}\,. \tag{2.16}$$

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

$$\tau(p) \approx \tau_0\left(1 - \frac{1}{S_\tau}\right) + \frac{\tau_0}{S_\tau}\sqrt{1 - S_\tau\,V_\tau^2\,p^2}\,. \tag{2.17}$$

Find τ_0, V_τ, and S_τ.

2.3 Computational part

In the computational part, we begin working with field data. The left panel in Figure 2.1 shows a CMP (common midpoint) gather from the Gulf of Mexico (Claerbout, 2006).

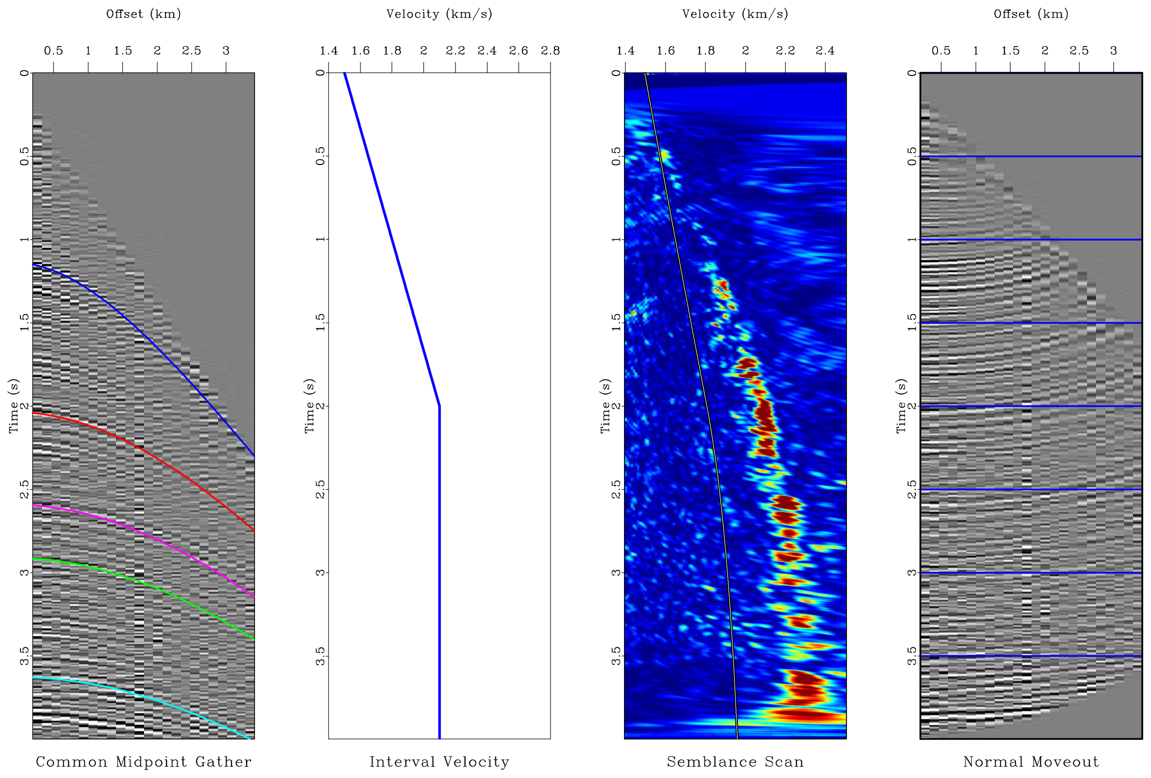


Figure 2.1: From left to right: (a) CMP (common midpoint) gather with overlaid traveltime curves. (b) Interval velocity. (c) RMS (root-mean-square) velocity overlaid on the semblance scan. (d) CMP gather after normal moveout. $\boxed{\text{hw2/cmp cmp}}$

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 2.1 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 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
   ```

```python
from rsf.proj import *

# 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" ')

prog = Program('traveltime.c',CPPDEFINES='NO_BLAS',LIBS=['rsf','m'])
exe = str(prog[0])

###############################
grad = 0.3 # 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],
         '''
         ./${SOURCES[1]} nr=5 r=285,509,648,728,906
         nh=24 dh=0.134 h0=0.264 type=hyperbolic
         ''')
    Plot(time+'g',time,
         '''
         graph yreverse=y pad=n min2=0 max2=3.996
         wantaxis=n wanttitle=n plotfat=10
         ''')
    Plot(time,['cmp',time+'g'],'Overlay')

    # RMS velocity
    vrms = 'vrms%d' % iv

    Flow(vrms,vint,
         '''
         add mode=p $SOURCE | causint |
         math output="sqrt(input*0.004/(x1+0.004))"
         ''')
    Plot(vrms+'w',vrms,
         '''
         graph yreverse=y transp=y pad=n
         wanttitle=n wantaxis=n min2=1.4 max2=2.5
         plotcol=7 plotfat=15
         ''')
    Plot(vrms+'b',vrms,
         '''
         graph yreverse=y transp=y pad=n
         wanttitle=n wantaxis=n min2=1.4 max2=2.5
         plotcol=0 plotfat=3
         ''')
    Plot(vrms,['vscan',vrms+'w',vrms+'b'],'Overlay')

    # Normal moveout
    nmo = 'nmo%d' % iv

    Flow(nmo,['cmp',vrms],'nmo velocity=${SOURCES[1]} half=n')
    Plot(nmo,
         '''
         grey title="Normal Moveout"
         grid2=y gridcol=6 gridfat=10
         ''')

    # Display it together
    cmp = 'cmp%d' % iv
    Plot(cmp,[time,vint,vrms,nmo],
         'SideBySideAniso',vppen='txscale=1.5')

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

##################################
frame = 10
##################################
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()
```

2.4 Programming part

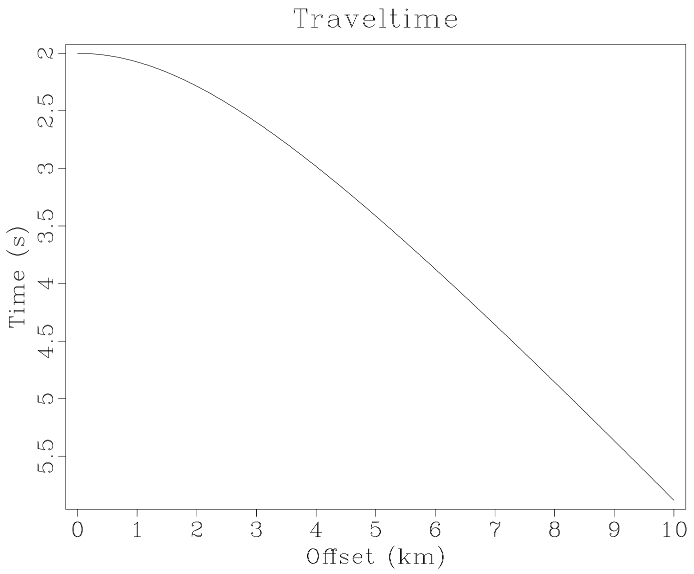


Figure 2.2: Traveltime in a $V(z)$ medium. hw2/cmp time

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 (2.14).
2. Modify the program to implement exact traveltime computation by doing shooting iterations with equations (2.15-2.16). 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
$$p_{n+1} = p_n - \frac{h(p_n) - h}{h'(p_n)} \; . \tag{2.18}$$
3. For the traveltime in Figure 2.2, find the offset, where the absolute error of the hyperbolic approximation (2.12) exceeds 0.1 s.
4. For the traveltime in Figure 2.2, find the offset, where the absolute error of the non-hyperbolic approximation (**??**) exceeds 0.1 s.

```c
/* 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;
    /* first offset */

    /* get reflectors */
    if (!sf_getint("nr",&nr)) nr=1;
    /* number of reflectors */
    r = sf_intalloc(nr);
    if (!sf_getints("r",r,nr)) sf_error("Need r=");

    if (NULL == (type = sf_getstring("type")))
        type = "hyperbolic";
    /* traveltime computation type */

    if (!sf_getint("niter",&niter)) niter=10;
    /* maximum number of shooting iterations */

    /* put dimensions in output */
    sf_putint(tim,"n1",nh);
    sf_putfloat(tim,"d1",dh);
    sf_putfloat(tim,"o1",h0);
    sf_putint(tim,"n2",nr);

    /* read velocity */
    v = sf_floatalloc(nt);
    sf_floatread(v,nt,vel);
    /* convert to velocity squared */
    for (it=0; it < nt; it++) {
        v[it] *= v[it];
    }

    t = sf_floatalloc(nh);

    for (ir=0; ir<nr; ir++) {
        nt = r[ir];
        t0 = nt*dt; /* zero-offset time */
        t2 = t0*t0;

        p = 0.0;

        for (ih=0; ih<nh; ih++) {
            h = h0+ih*dh; /* offset */
            h2 = h*h;

            switch(type[0]) {
                case 'h': /* hyperbolic approximation */
                    v2 = 0.0;
                    for (it=0; it < nt; it++) {
                        v2 += v[it];
                    }
                    v2 /= nt;

                    t[ih] = sqrtf(t2+h2/v2);
                    break;
                case 's': /* shifted hyperbola */

                    /* !!! MODIFY BELOW !!! */

                    s = 0.0;

                    v2 = 0.0;
                    for (it=0; it < nt; it++) {
                        v2 += v[it];
                    }
                    v2 /= nt;

                    t[ih] = sqrtf(t2+h2/v2);
                    break;
                case 'e': /* exact */

                    /* !!! MODIFY BELOW !!! */

                    for (iter=0; iter < niter; iter++) {
                        hp = 0.0;
                        for (it=0; it < nt; it++) {
                            v2 = v[it];
                            hp += v2/sqrtf(1.0-p*p*v2);
                        }
                        hp *= p*dt;

                        /* !!! SOLVE h(p)=h !!! */
                    }

                    tp = 0.0;
                    for (it=0; it < nt; it++) {
                        v2 = v[it];
                        tp += dt/sqrtf(1.0-p*p*v2);
                    }

                    t[ih] = tp;
                    break;
                default:
                    sf_error("Unknown type");
                    break;
            }
        }

        sf_floatwrite(t,nh,tim);
    }

    exit(0);
}
```

2.5 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/`.

Chapter 3

Homework 3

ABSTRACT

This homework has two parts. In the theoretical part, you will perform several analytical derivations related to geometrical integration and reflections from elliptical reflections. In the computational part, you will experiment with imaging a synthetic dataset and a field dataset from the Gulf of Mexico.

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://rsf.svn.sourceforge.net/svnroot/rsf/trunk/book/geo384w/hw3
```

3.1 Theoretical part

1. Consider a 2-D common-midpoint gather $G(t,x)$, which contains a geometrical event $A_0\,f\,(t - T(x))$ with a constant amplitude A_0 along a parabolic shape

   $$T(x) = t_0 + a\,x^2\ . \tag{3.1}$$

   The gather gets transformed by the slant-stack (Radon transform) operator

   $$R(\tau,p) = \mathbf{D}_t^{1/2} \int G(\tau + px, x)dx\ . \tag{3.2}$$

where $\mathbf{D}_t^{1/2}$ is a waveform-correcting half-order derivative operator.

Using the theory of geometrical integration, show that $R(\tau, p)$ will contain a geometrical event $A_1(p)\, f\,(\tau - T_1(p))$. Determine $T_1(p)$ and $A_1(p)$.

2. Consider source and receiver coordinates s and r on the surface of a 2-D constant-velocity medium with velocity V.

   (a) Assuming that the reflection traveltime is T, show that the reflection point $\{x, z\}$ must belong to an ellipse (*migration impulse response*)

   $$z(x) = \sqrt{R^2 - \alpha\,(x-m)^2}\;, \tag{3.3}$$

   where $R = V\,T_n/2$, $T_n = \sqrt{T^2 - \frac{4\,h^2}{V^2}}$, $h = (r-s)/2$, and $m = (r+s)/2$.

   (b) Find α.

   (c) Consider that the ellipse in equation 3.3 as a reflection surface and apply Fermat's principle to find the reflection traveltime $T_0(x_0)$ for observations with sources and receivers coincident at x_0.

3.2 Computational part

1. In the first part, you will experiment with creating and imaging a synthetic seismic reflection dataset.

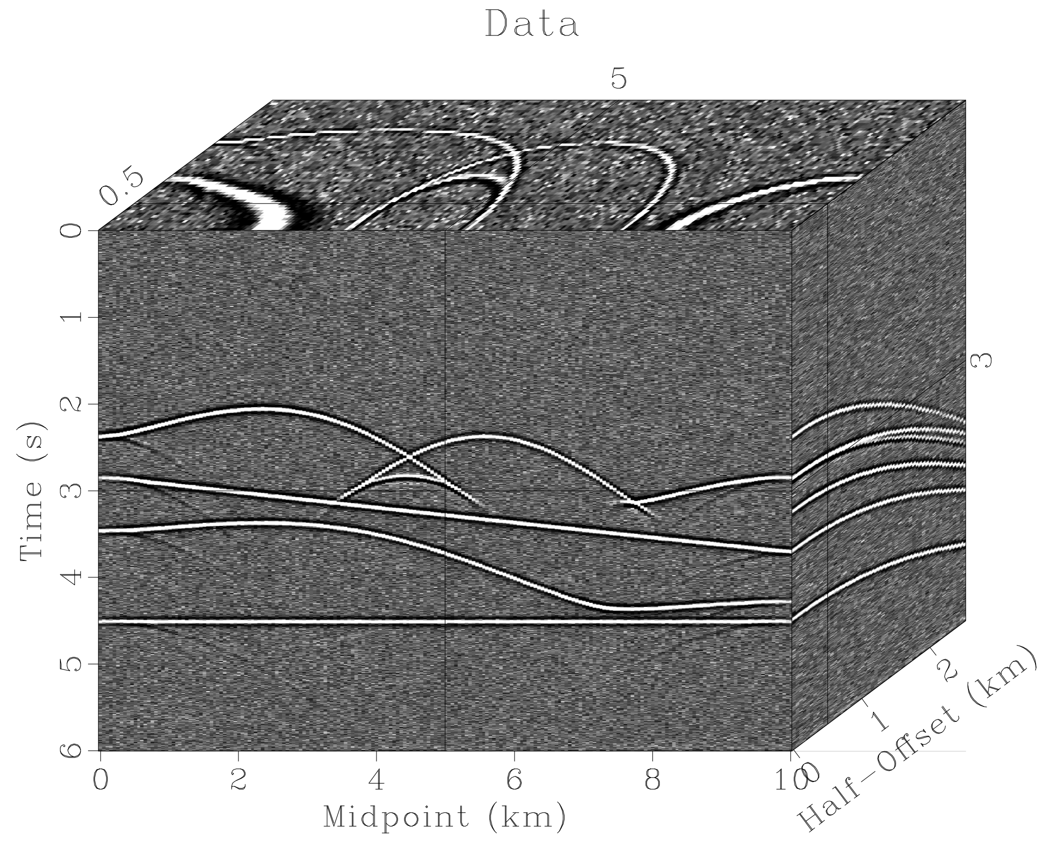


Figure 3.1: 2-D synthetic data. $\boxed{\text{hw3/synth data}}$

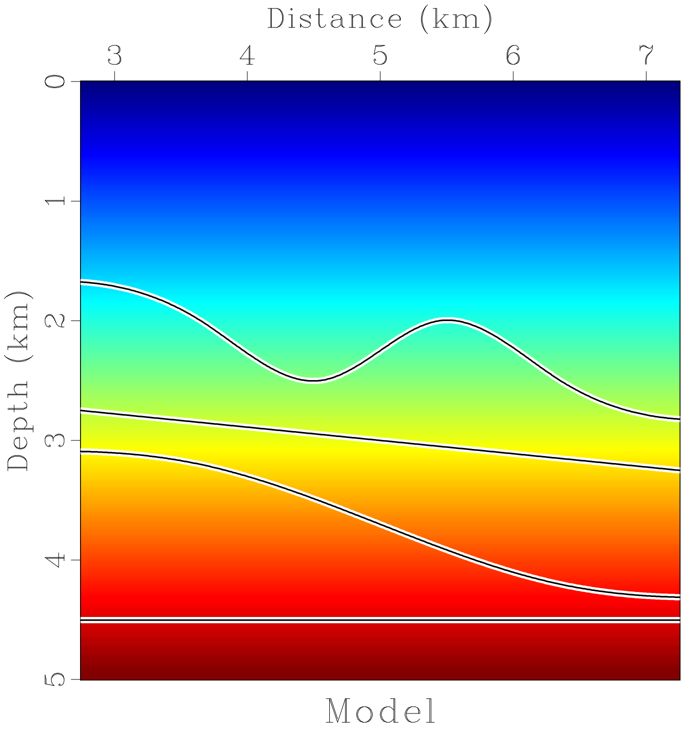


Figure 3.2: (a) Synthetic model: curved reflectors in a $V(z)$ velocity. hw3/synth model

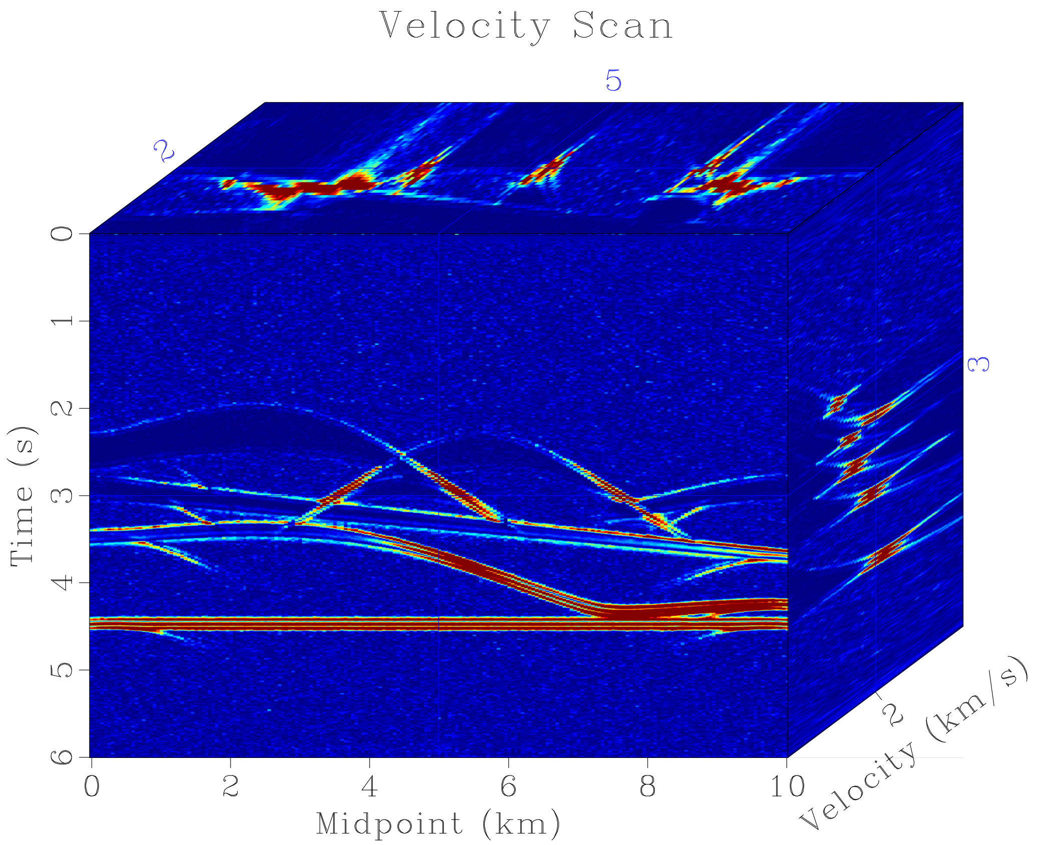


Figure 3.3: Velocity semblance scan. hw3/synth vscan

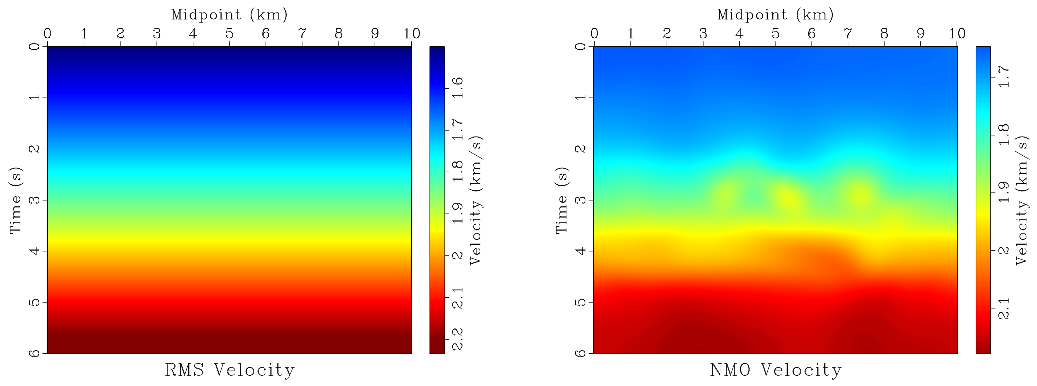


Figure 3.4: RMS velocity (a) and picked NMO velocity (b). hw3/synth vnmo

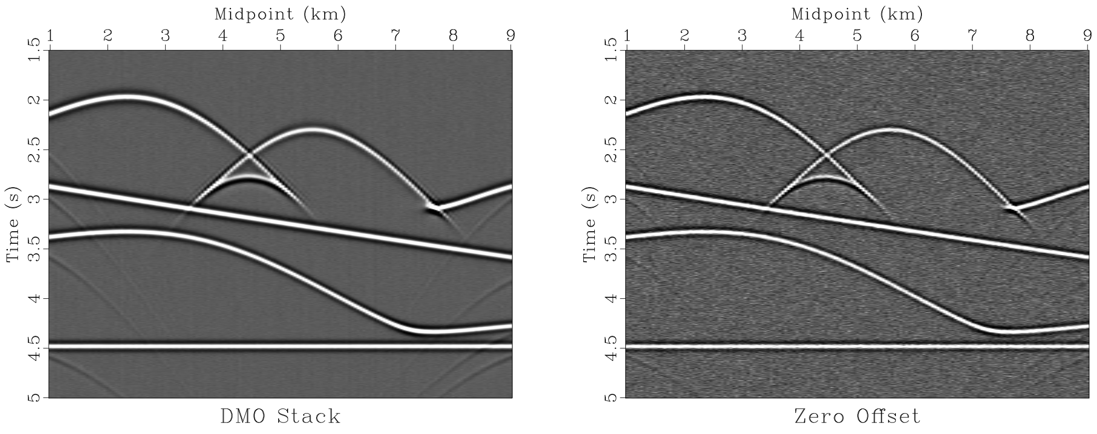


Figure 3.5: (a) DMO stack. (b) Zero-offset section. hw3/synth dstack,zoff

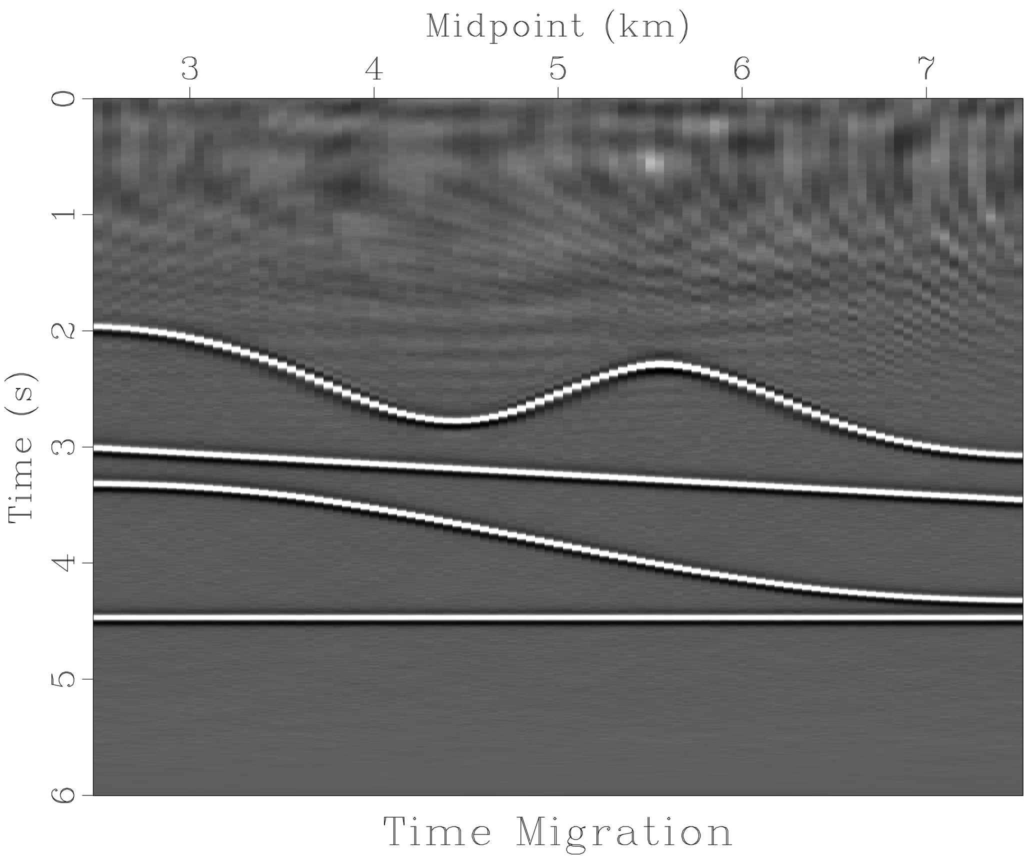


Figure 3.6: Kirchhoff poststack time migration. hw3/synth tmig

Figure 3.7: Time migration converted to depth, with reflectors overlaid. hw3/synth dmig2

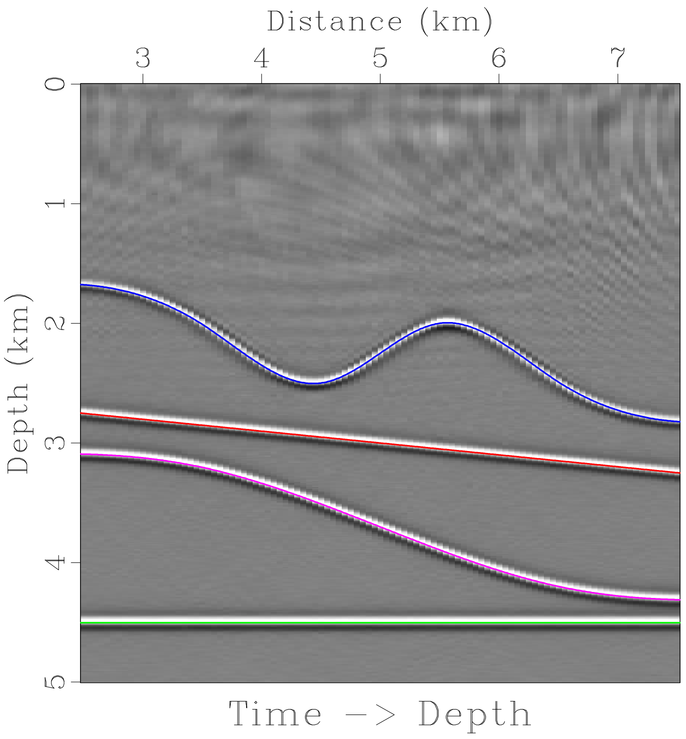


Figure 4.6 shows a synthetic reflection dataset computed from a reflector model shown in Figure 5.1 and assuming a velocity model with a constant vertical gradient $V(z) = 1.5 + 0.25\,z$. A small amount of random noise is added to the synthetic data.

Figure 3.3 shows a conventional velocity semblance scan. Figure 3.4 compares the RMS velocity with the NMO velocity picked from the scan. Figure 3.5 compares a DMO stack and the zero-offset section from the data. Finally, Figure **??** shows the result of Kirchhoff time migration before and after conversion from time to depth.

(a) Change directory

```
cd hw3/synth
```

(b) Run

```
scons view
```

to generate the figures and display them on your screen. If you are on a computer with multiple CPUs, you can also try

```
pscons view
```

to run certain computations in parallel.

(c) Explain the cause of the difference between the RMS velocity and the NMO velocity in Figure 3.4.

(d) Edit the `SConstruct` file to switch on the antialiasing parameter in Kirchhoff migration (by changing it from 0 to 1). Generate the migration figures again. What differences do you observe?

(e) Edit the `kirchhoff.c` file to improve the result of migration. This task is open-ended. The change is up to you, as long as you can achieve an improvement. Possible options:

- Use a more accurate traveltime computation.
- Introduce an amplitude weight.
- Change from time to depth migration (you can assume a locally $V(z)$ medium and use results from previous homeworks.)
- Change from poststack to prestack migration.

(f) The processing flow in the `SConstruct` file involves some cheating: the exact depth velocity and the exact RMS velocity are used without being estimated from the data. Modify the processing flow so that only properties estimated from the data get used. This task is open-ended as well, different data processing strategies are possible.

```
from rsf.proj import *

# Generate a reflector model

layers = (
    ((0,2),(3.5,2),(4.5,2.5),(5,2.25),
      (5.5,2),(6.5,2.5),(10,2.5)),
    ((0,2.5),(10,3.5)),
    ((0,3.2),(3.5,3.2),(5,3.7),
      (6.5,4.2),(10,4.2)),
    ((0,4.5),(10,4.5))
)

nlays = len(layers)
for i in range(nlays):
    inp = 'inp%d' % i
    Flow(inp+'.asc',None,
         '''
         echo %s in=$TARGET
         data_format=ascii_float n1=2 n2=%d
         ''' % \
         (' '.join(map(lambda x: ' '.join(map(str,x)),
                       layers[i])),len(layers[i])))

dim1 = 'o1=0 d1=0.01 n1=1001'
Flow('lay1','inp0.asc',
    'dd form=native | spline %s fp=0,0' % dim1)
Flow('lay2',None  ,
    'math %s output="2.5+x1*0.1" '      % dim1)
Flow('lay3','inp2.asc',
    'dd form=native | spline %s fp=0,0' % dim1)
Flow('lay4',None  ,'math %s output=4.5'  % dim1)

Flow('lays','lay1 lay2 lay3 lay4',
    'cat axis=2 ${SOURCES[1:4]}')

graph = '''
graph min1=2.5 max1=7.5 min2=0 max2=5
yreverse=y wantaxis=n wanttitle=n screenratio=1
'''
Plot('lays0','lays',graph + ' plotfat=10 plotcol=0')
Plot('lays1','lays',graph + ' plotfat=2 plotcol=7')
Plot('lays2','lays',graph + ' plotfat=2')

# Velocity

Flow('vofz',None,
     '''
     math output="1.5+0.25*x1"
     d2=0.01 n2=1001 d1=0.01 n1=501
     label1=Depth unit1=km
     label2=Distance unit2=km
     label=Velocity unit=km/s
     ''')
Plot('vofz',
     '''
```

```
        window min2=2.75 max2=7.25 |
        grey color=j allpos=y bias=1.5
        title=Model screenratio=1
        ''')

Result('model','vofz lays0 lays1','Overlay')

# Model data

Flow('dips','lays','deriv | scale dscale=100')
Flow('modl','lays dips',
        '''
        kirmod cmp=y dip=${SOURCES[1]}
        nh=51  dh=0.1  h0=0
        ns=201 ds=0.05 s0=0
        freq=10 dt=0.004 nt=1501
        vel=1.5 gradz=0.25 verb=y |
        tpow tpow=1 |
        put d2=0.05 label3=Midpoint unit3=km
        ''',split=[1,1001], reduce='add')

# Add random noise
Flow('data','modl','noise var=1e-6 seed=101811')

Result('data',
          '''
          byte |
          transp plane=23 |
          grey3 flat=n frame1=750 frame2=100 frame3=10
          label1=Time unit1=s
          label3=Half-Offset unit3=km
          title=Data point1=0.8 point2=0.8
          ''')

# Velocity estimation
######################
Flow('voft','vofz',
        'depth2time velocity=$SOURCE dt=0.004 nt=1501')
Flow('vrms','voft',
        '''
        add mode=p $SOURCE | causint |
        math output="sqrt(input*0.004/(x1+0.004))"
        ''')

# Velocity scan
Flow('vscan','data',
        'vscan v0=1.5 dv=0.02 nv=51 semblance=y',
        split=[3,201], reduce='cat')
```

```

Result('vscan',
        '''
        byte allpos=y gainpanel=all |
        transp plane=23 |
        grey3 flat=n frame1=750 frame2=100 frame3=25
        label1=Time unit1=s color=j
        label3=Velocity unit3=km/s
        label2=Midpoint unit2=km
        title="Velocity Scan" point1=0.8 point2=0.8
        ''')

# Velocity picking
Flow('vnmo','vscan','pick rect1=100 rect2=10 | window')

for vel in ('vrms','vnmo'):
    Plot(vel,
      '''
      grey color=j allpos=y bias=1.5 clip=0.7
      scalebar=y barreverse=y barunit=km/s
      label2=Midpoint unit2=km label1=Time unit1=s
      title="%s Velocity"
      ''' % vel[1:].upper())
Result('vnmo','vrms vnmo','SideBySideIso')

# Stacking
##########

Flow('nmo','data vnmo','nmo velocity=${SOURCES[1]}')
Flow('stack','nmo','stack')

# Using vrms is CHEATING
########################
Flow('nmo0','data vrms','nmo velocity=${SOURCES[1]}')
Flow('dstack','nmo0',
      '''
      window f1=250 |
      logstretch | fft1 |
      transp plane=13 memsize=1000 |
      finstack |
      transp memsize=1000 |
      fft1 inv=y | logstretch inv=y |
      pad beg1=250 | put unit1=s
      ''')

Flow('zoff','data','window n2=1')

stacks = {
    'stack': 'Stack with NMO Velocity',
    'dstack': 'DMO Stack',
    'zoff': 'Zero Offset'
    }

for stack in stacks.keys():
    Result(stack,
           '''
           window min1=1.5 max1=5 min2=1 max2=9 |
           grey title="%s"
           ''' % stacks[stack])

# Kirchhoff Migration
#######################

prog = Program('kirchhoff.c',CPPDEFINES='NO_BLAS',LIBS=['rsf','m'])
exe = str(prog[0])

# Using vrms is CHEATING
##########################
Flow('tmig','dstack %s vrms' % prog[0],
     './${SOURCES[1]} vel=${SOURCES[2]} antialias=0')

Result('tmig',
       '''
       window min2=2.5 max2=7.5 |
       grey title="Time Migration"
       ''')

# Using vofz is CHEATING
##########################
Flow('dmig','tmig vofz',
     'time2depth velocity=${SOURCES[1]}')

Plot('dmig',
     '''
     window max1=5 min2=2.5 max2=7.5 |
     grey title="Time -> Depth" screenratio=1
     label2=Distance label1=Depth unit1=km
     ''')

Result('dmig','Overlay')
Result('dmig2','dmig lays2','Overlay')

End()
```

```
/* 2-D Poststack Kirchhoff time migration. */
#include <rsf.h>
```

```

static void doubint(int nt, float *trace)
/* causal and anticausal integratio */
{
    int it;
    float tt;

    tt = trace[0];
    for (it=1; it < nt; it++) {
        tt += trace[it];
        trace[it] = tt;
    }
    tt = trace[nt-1];
    for (it=nt-2; it >=0; it--) {
        tt += trace[it];
        trace[it] = tt;
    }
}

static float pick(float ti, float deltat,
                  const float *trace,
                  int nt, float dt, float t0)
/* pick a traveltime sample from a trace */
{
    int it, itm, itp;
    float ft, tm, tp, ftm, ftp, imp;

    ft = (ti-t0)/dt; it = floorf(ft); ft -= it;
    if ( it < 0 || it >= nt-1) return 0.0;

    tm = ti-deltat-dt;
    ftm = (tm-t0)/dt; itm = floorf(ftm); ftm -= itm;
    if (itm < 0) return 0.0;

    tp = ti+deltat+dt;
    ftp = (tp-t0)/dt; itp = floorf(ftp); ftp -= itp;
    if (itp >= nt-1) return 0.0;

    imp = dt/(dt+tp-tm);
    imp *= imp;

    return imp*(
        2.*(1.-ft)*trace[it] + 2.*ft*trace[it+1] -
        (1.-ftm)*trace[itm] - ftm*trace[itm+1]     -
        (1.-ftp)*trace[itp] - ftp*trace[itp+1]);
}

int main(int argc, char* argv[])
```

```
{
    int nt, nx, nz, ix, iz, iy, i;
    float *trace, **out, **v;
    float x,z, dx, ti, tx, t0,dt, z0,dz, vi,aal;
    sf_file inp, mig, vel;

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

    if (!sf_histint(inp,"n1",&nt)) sf_error("No n1=");
    if (!sf_histint(inp,"n2",&nx)) sf_error("No n2=");

    if (!sf_histfloat(inp,"o1",&t0)) sf_error("No o1=");
    if (!sf_histfloat(inp,"d1",&dt)) sf_error("No d1=");
    if (!sf_histfloat(inp,"d2",&dx)) sf_error("No d2=");

    if (!sf_getint("nz",&nz)) nz=nt;
    if (!sf_getfloat("dz",&dz)) dz=dt;
    if (!sf_getfloat("z0",&z0)) z0=t0;

    if (!sf_getfloat("antialias",&aal)) aal=1.0;
    /* antialiasing */

    v = sf_floatalloc2(nz,nx);
    sf_floatread(v[0],nz*nx,vel);

    trace = sf_floatalloc(nt);
    out = sf_floatalloc2(nz,nx);

    for (i=0; i < nz*nx; i++) {
        out[0][i] = 0.;
    }

    for (iy=0; iy < nx; iy++) {
        sf_floatread (trace,nt,inp);
        doubint(nt,trace);

        for (ix=0; ix < nx; ix++) {
            x = (ix-iy)*dx;

            for (iz=0; iz < nz; iz++) {
                z = z0 + iz*dz;
                vi = v[ix][iz];

                /* hypot(a,b) = sqrt(a*a+b*b) */
                ti = hypotf(z,2.0*x/vi);

                /* tx = |dt/dx| */
                tx = 4.0*fabsf(x)/(vi*vi*(ti+dt));

                out[ix][iz] +=
                    pick(ti,tx*dx*aal,trace,nt,dt,t0);
            }
        }
    }

    sf_floatwrite(out[0],nz*nx,mig);

    exit(0);
}
```

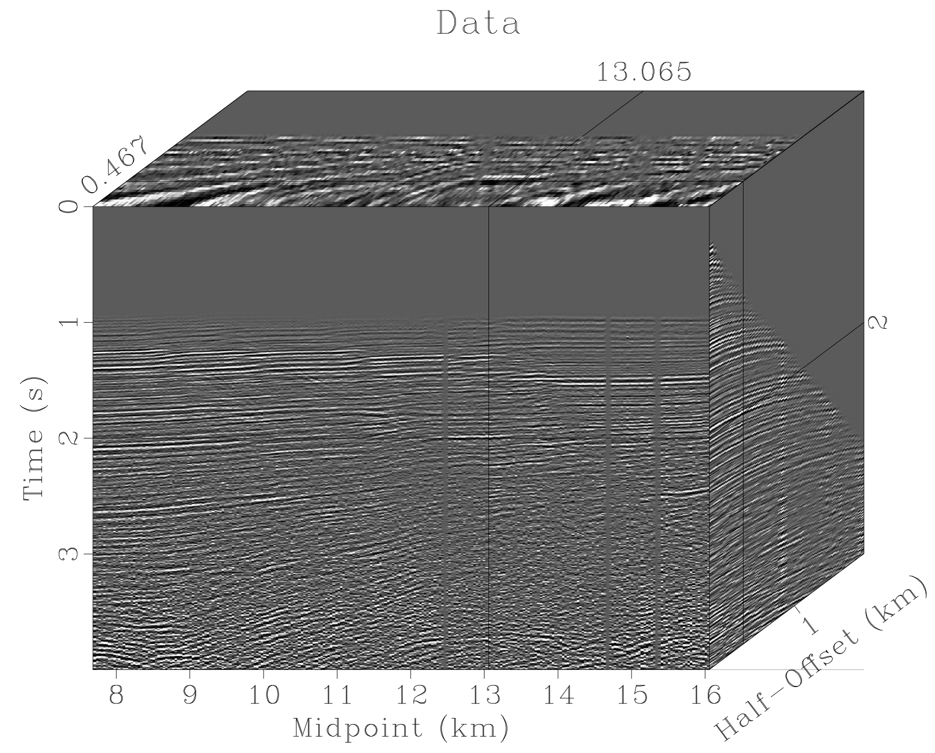


Figure 3.8: 2-D field dataset from the Gulf of Mexico. hw3/gulf gulf

2. In the second part of the computational assignment, you will use the processing strategy developed for synthetic data, to process a 2-D field dataset from the Gulf of Mexico, shown in Figure 3.8.

   (a) Change directory

       `cd hw3/gulf`

   (b) Run

       `scons view`

       to generate the figures and display them on your screen.

   (c) Edit the `SConstruct` file to implement your processing strategy. Make sure to select appropriate processing parameters.

```
from rsf.proj import *

Fetch('beinew.HH','midpts')
Flow('gulf','beinew.HH',
     '''
     dd form=native |
     put
     label1=Time unit1=s
     label2=Half-Offset unit2=km
     label3=Midpoint unit3=km
```

```
        ''')

Result('gulf',
         '''
         byte |
         transp plane=23 |
         grey3 flat=n frame1=500 frame2=160 frame3=10
         title=Data point1=0.8 point2=0.8
         ''')

End()
```

3.3 Completing the assignment

1. Change directory to `hw3`.
2. Edit the file `paper.tex` in your favorite editor and change the first line to have your name instead of Kirchhoff'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.

Chapter 4

Homework 4

ABSTRACT

This homework has two parts. In the theoretical part, you will perform analytical derivations related to the shifted hyperbola approximation. In the computational part, you will experiment with imaging two synthetic datasets and a field dataset from the Gulf of Mexico using velocity continuation.

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://rsf.svn.sourceforge.net/svnroot/rsf/trunk/book/geo384w/hw4
```

4.1 Theoretical part

Using the hyperbolic traveltime approximation

$$T = \sqrt{\left(T_0 - \frac{z}{V(x_0,T_0)}\right)^2 + \frac{(x_0 - x)^2}{V^2(x_0,T_0)}} \tag{4.1}$$

makes the geometrical imaging analysis equivalent to analyzing wave propagation in a constant-velocity medium. In particular, we can easily verify that the traveltime satisfies the isotropic eikonal equation

$$\left(\frac{\partial T}{\partial x}\right)^2 + \left(\frac{\partial T}{\partial z}\right)^2 = \frac{1}{V^2}\ . \tag{4.2}$$

Suppose that you switch to the more accurate shifted hyperbola approximation

$$T = \left(T_0 - \frac{z}{V}\right)\,(1 - \frac{1}{S}) + \frac{1}{S}\sqrt{\left(T_0 - \frac{z}{V}\right)^2 + S\,\frac{(x_0 - x)^2}{V^2}} \tag{4.3}$$

1. How would you need to modify the eikonal equation?
2. How would you need to modify the following expressions for the escape time and location for use in the angle-domain time migration?

$$\hat{T} = \frac{T_0 - z/V}{\cos\theta} \tag{4.4}$$

$$\hat{x} = x_0 + V\,\left(T_0 - \frac{z}{V}\right)\,\tan\theta \tag{4.5}$$

4.2 Computational part

1. In the first part of the computational assignment, you will experiment with imaging a synthetic seismic reflection dataset from Homework 3 using prestack velocity continuation.

   Figure 4.6 shows a synthetic reflection dataset computed from a reflector model shown in Figure **??** and assuming a velocity model with a constant vertical gradient $V(z) = 1.5 + 0.25\,z$. A small amount of random noise is added to the data.

   Figure 4.3 shows an initial prestack common-offset time migration using a constant velocity of 1.5 km/s. Figure 4.4 shows the result of prestack time migration after velocity continuation, extraction of a velocity slice, and conversion from time to depth.

   (a) Change directory

   ```
   cd hw4/synth
   ```

   (b) Run

   ```
   scons view
   ```

   to generate the figures and display them on your screen. If you are on a computer with multiple CPUs, you can also try

   ```
   pscons view
   ```

   to run certain computations in parallel.

   (c) Run

   ```
   pscons velcon.vpl
   ```

   to display a movie of the velocity continuation process.

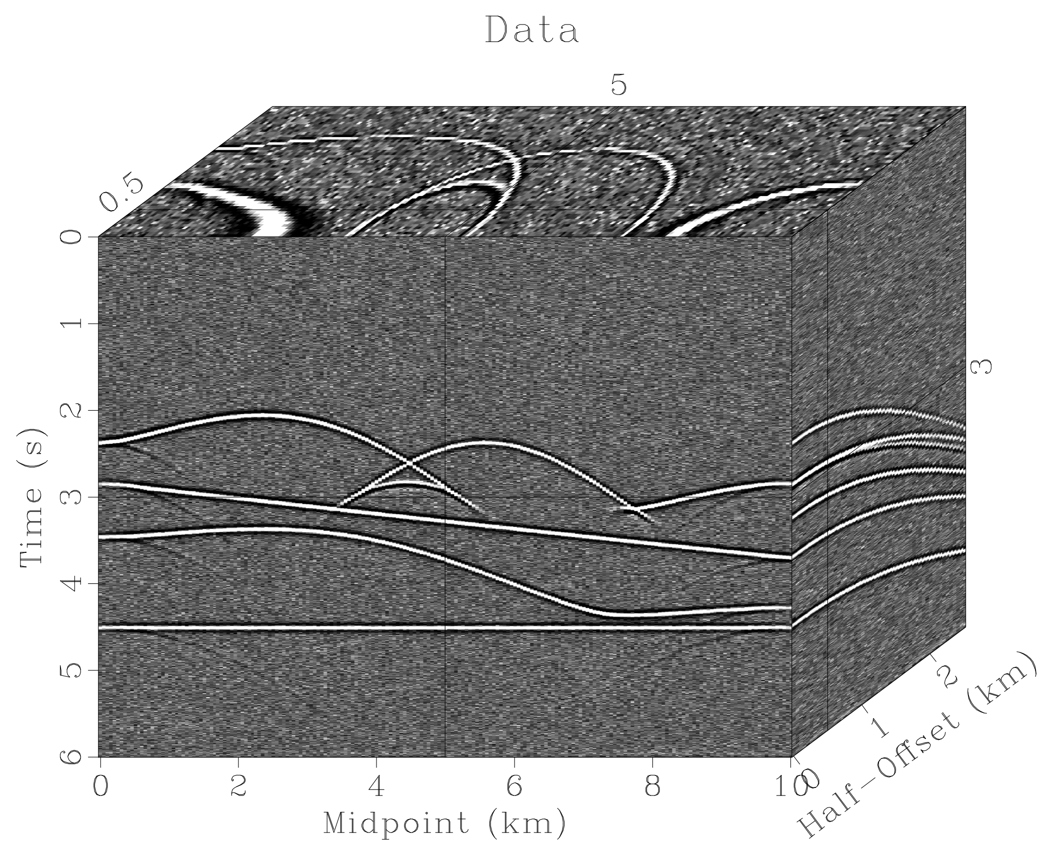


Figure 4.1: 2-D synthetic data. $\boxed{\text{hw4/synth data}}$

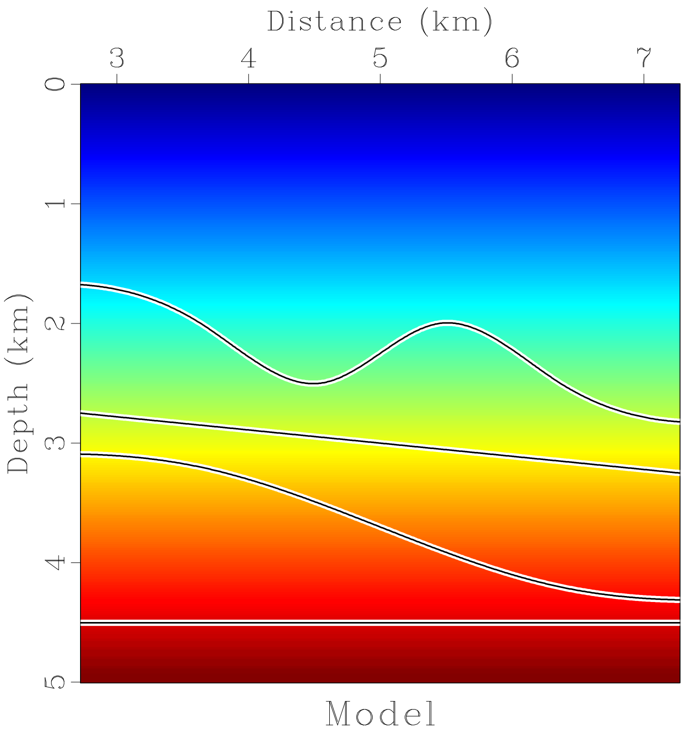


Figure 4.2: (a) Synthetic model: curved reflectors in a $V(z)$ velocity. $\boxed{\text{hw4/synth model}}$

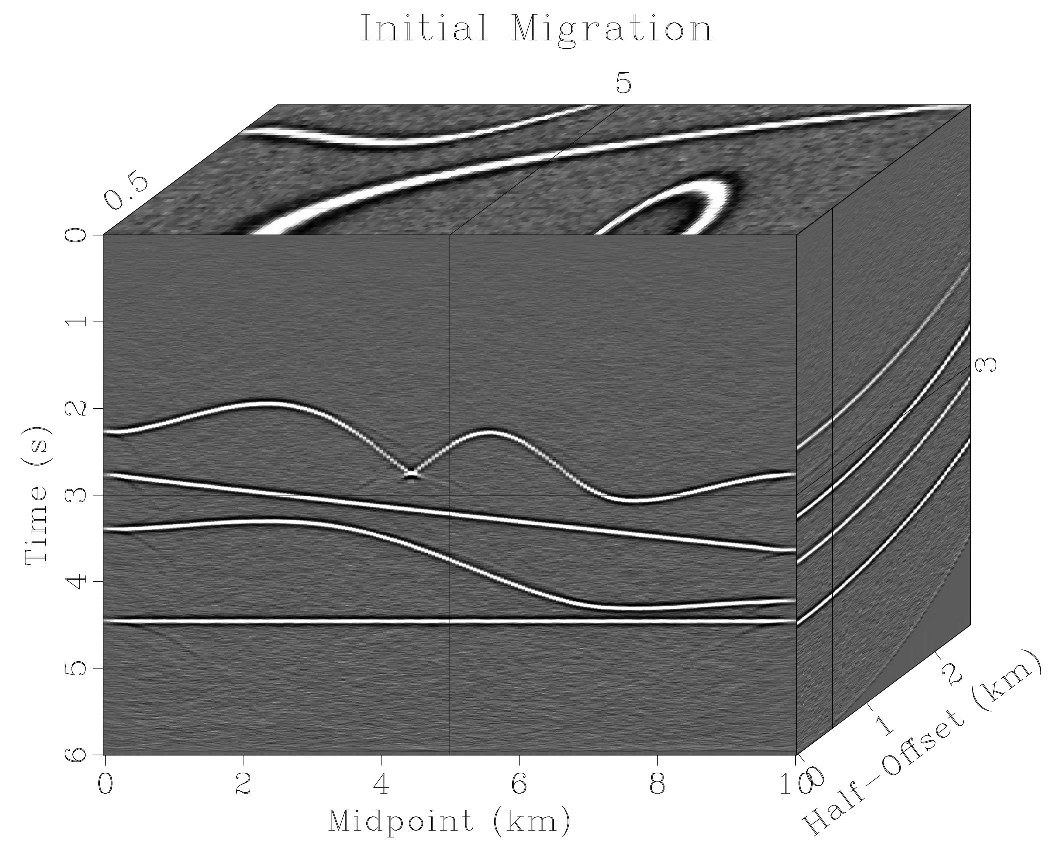


Figure 4.3: Initial constant-velocity migration. hw4/synth mig

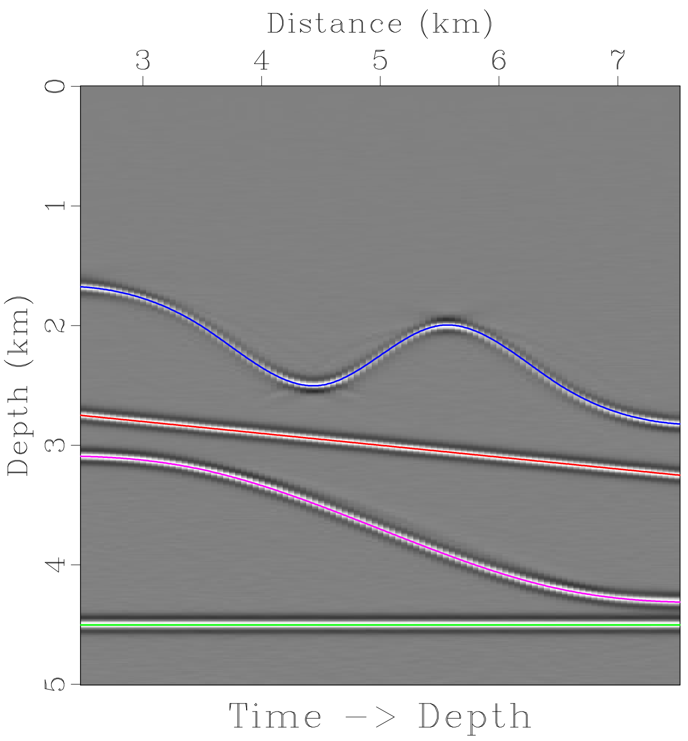


Figure 4.4: Time migration converted to depth, with reflectors overlaid. hw4/synth dmig2

(d) Run

```
pscons semb.vpl
```

to display a movie slicing through a semblance cube computed from velocity continuation.

(e) The processing flow in the `SConstruct` file involves some cheating: the exact RMS velocity is used to extract the final image. Modify the processing flow so that only properties estimated from the data get used.

```
from rsf.proj import *

# Generate a reflector model

layers = (
        ((0,2),(3.5,2),(4.5,2.5),(5,2.25),
           (5.5,2),(6.5,2.5),(10,2.5)),
        ((0,2.5),(10,3.5)),
        ((0,3.2),(3.5,3.2),(5,3.7),
           (6.5,4.2),(10,4.2)),
        ((0,4.5),(10,4.5))
)

nlays = len(layers)
for i in range(nlays):
        inp = 'inp%d' % i
        Flow(inp+'.asc',None,
                '''
                echo %s in=$TARGET
                data_format=ascii_float n1=2 n2=%d
                ''' % \
                (' '.join(map(lambda x: ' '.join(map(str,x)),
                                        layers[i])),len(layers[i])))

dim1 = 'o1=0 d1=0.01 n1=1001'
Flow('lay1','inp0.asc',
        'dd form=native | spline %s fp=0,0' % dim1)
Flow('lay2',None   ,
        'math %s output="2.5+x1*0.1"  '       % dim1)
Flow('lay3','inp2.asc',
        'dd form=native | spline %s fp=0,0' % dim1)
Flow('lay4',None   ,'math %s output=4.5'  % dim1)

Flow('lays','lay1 lay2 lay3 lay4',
        'cat axis=2 ${SOURCES[1:4]}')

graph = '''
graph min1=2.5 max1=7.5 min2=0 max2=5
yreverse=y wantaxis=n wanttitle=n screenratio=1
```

```
'''
Plot('lays0','lays',graph + ' plotfat=10 plotcol=0')
Plot('lays1','lays',graph + ' plotfat=2 plotcol=7')
Plot('lays2','lays',graph + ' plotfat=2')

# Velocity

Flow('vofz',None,
     '''
     math output="1.5+0.25*x1"
     d2=0.05 n2=201 d1=0.01 n1=501
     label1=Depth unit1=km
     label2=Distance unit2=km
     label=Velocity unit=km/s
     ''')
Plot('vofz',
     '''
     window min2=2.75 max2=7.25 |
     grey color=j allpos=y bias=1.5
     title=Model screenratio=1
     ''')

Result('model','vofz lays0 lays1','Overlay')

# Model data

Flow('dips','lays','deriv | scale dscale=100')
Flow('modl','lays dips',
     '''
     kirmod cmp=y dip=${SOURCES[1]}
     nh=51  dh=0.1  h0=0
     ns=201 ds=0.05 s0=0
     freq=10 dt=0.004 nt=1501
     vel=1.5 gradz=0.25 verb=y |
     tpow tpow=1 |
     put d2=0.05 label3=Midpoint unit3=km
     ''',split=[1,1001], reduce='add')

# Add random noise
Flow('data','modl','noise var=1e-6 seed=101811')

Result('data',
       '''
       byte |
       transp plane=23 |
       grey3 flat=n frame1=750 frame2=100 frame3=10
       label1=Time unit1=s
       label3=Half-Offset unit3=km
```

```
         title=Data point1=0.8 point2=0.8
         ''')

# Initial constant-velocity migration
########################################
Flow('mig','data',
       '''
       transp plane=23 |
       spray axis=3 n=1 d=0.1 o=0 |
       preconstkirch vel=1.5 |
       halfint inv=1 adj=1
       ''',split=[2,51],reduce='cat axis=4')

Result('mig',
         '''
         byte | window |
         grey3 flat=n frame1=750 frame2=100 frame3=10
         label1=Time unit1=s
         label3=Half-Offset unit3=km
         title="Initial Migration" point1=0.8 point2=0.8
         ''')

# Velocity continuation
########################

Flow('thk','mig',
       'window | transp plane=23 | cosft sign3=1')
Flow('velconk','thk',
       'fourvc nv=81 dv=0.01 v0=1.5 verb=y',
       split=[3,201])
Flow('velcon','velconk','cosft sign3=-1')

Plot('velcon',
       '''
       transp plane=23 memsize=1000 |
       window min2=2.5 max2=7.5 |
       grey title="Velocity Continuation"
       ''',view=1)

# Continue data squared
Flow('thk2','mig',
       '''
       mul $SOURCE |
       window | transp plane=23 | cosft sign3=1
       ''')
Flow('velconk2','thk2',
       'fourvc nv=81 dv=0.01 v0=1.5 verb=y',
       split=[3,201])
```

```
Flow('velcon2','velconk2','cosft sign3=-1')

# Compute semblance
Flow('semb','velcon velcon2',
     '''
     mul $SOURCE | divn den=${SOURCES[1]} rect1=25
     ''',split=[3,201])

Plot('semb',
     '''
     byte gainpanel=all allpos=y |
     transp plane=23 |
     grey3 flat=n frame1=750 frame2=0 frame3=48
     label1=Time unit1=s color=j
     label3=Velocity unit3=km/s movie=2 dframe=5
     title=Semblance point1=0.8 point2=0.8
     ''',view=1)

# Extracting images
###################
Flow('voft','vofz',
     'depth2time velocity=$SOURCE dt=0.004 nt=1501')
Flow('vrms','voft',
     '''
     add mode=p $SOURCE | causint |
     math output="sqrt(input*0.004/(x1+0.004))"
     ''')

# Using vrms is CHEATING
########################
Flow('slice','velcon vrms','slice pick=${SOURCES[1]}')

# Using vofz is CHEATING
########################
Flow('dmig','slice vofz',
     'time2depth velocity=${SOURCES[1]}')

Plot('dmig',
     '''
     window max1=5 min2=2.5 max2=7.5 |
     grey title="Time -> Depth" screenratio=1
     label2=Distance label1=Depth unit1=km
     ''')

Result('dmig','Overlay')
Result('dmig2','dmig lays2','Overlay')

End()
```

2. In the second part of the computational assignment, we will use velocity continuation again but this time on a synthetic zero-offset section containing diffraction events.

   Figure 6.3(b) shows a famous Sigsbee synthetic velocity model. We will focus on the left part of the model, which is appropriate for time-domain imaging. A synthetically generated zero-offset section is shown in Figure 4.6.

   Our processing strategy is to extract diffractions from the data (Figure 4.7) and to image them using zero-offset velocity continuation (Figure 4.8). In addition, we are going to analyze the image by expanding it in dip angles by using dip-angle migration (Figure 4.9).

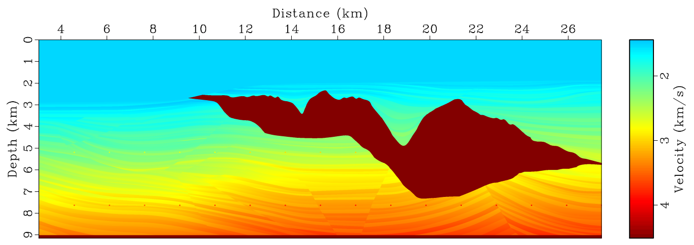


Figure 4.5: Sigsbee velocity model. hw4/sigsbee vel

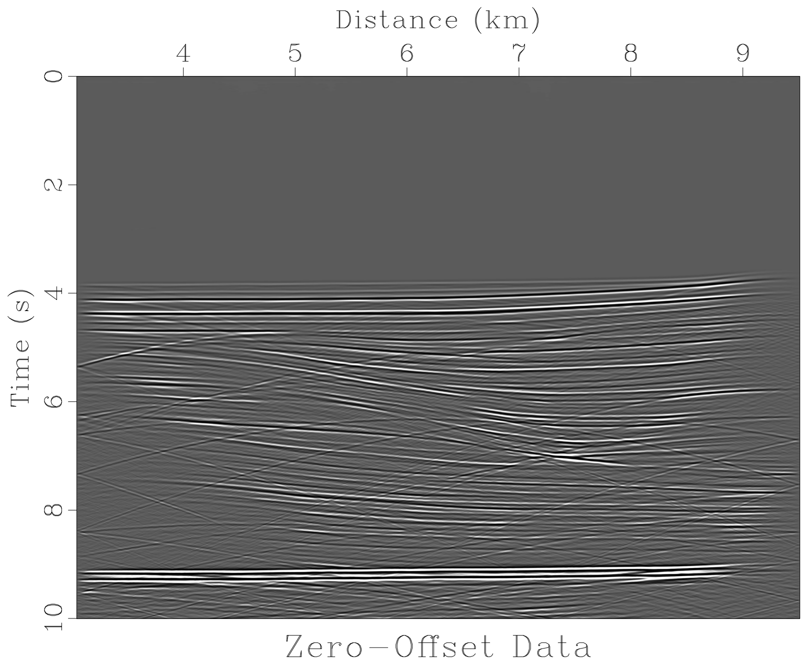


Figure 4.6: Zero-offset synthetic data. hw4/sigsbee data

   (a) Change directory

   ```
   cd hw4/sigsbee
   ```

   (b) Run

   ```
   scons view
   ```

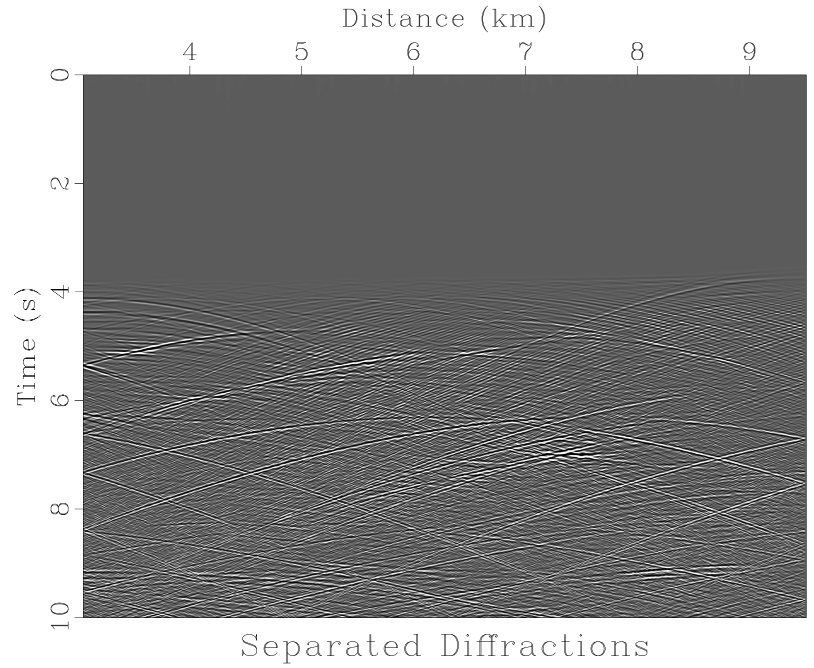


Figure 4.7: Diffractions extracted from the data by plane-wave destruction. hw4/sigsbee dif

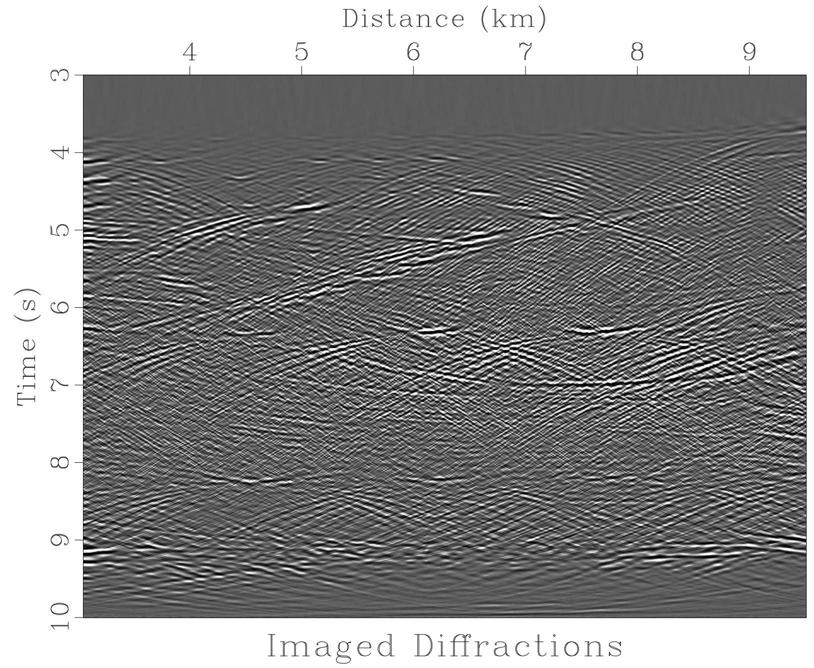


Figure 4.8: Time-migrated image of diffractions. hw4/sigsbee dimage

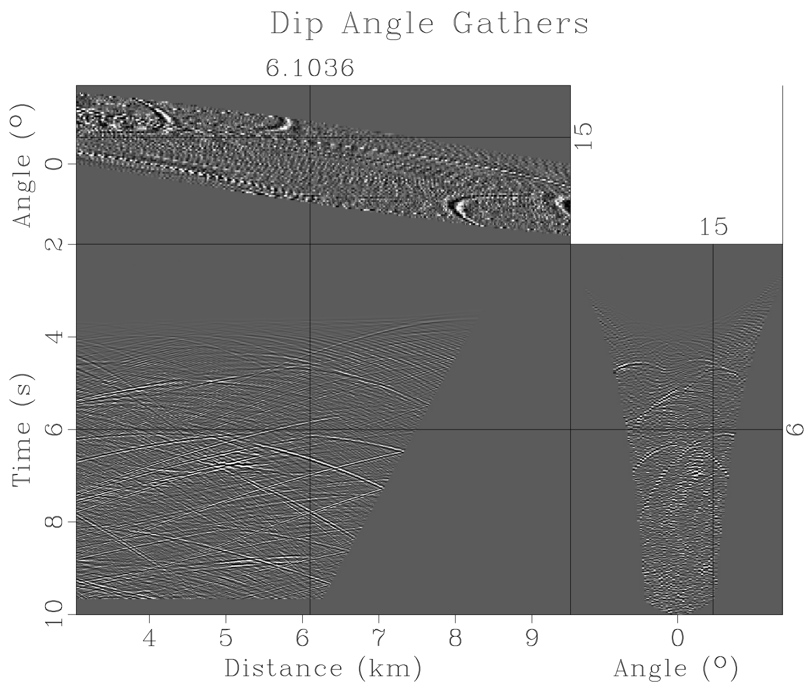


Figure 4.9: Dip angle gathers from constant-velocity angle-domain migration. hw4/sigsbee anglemig

to generate the figures and display them on your screen. If you are on a computer with multiple CPUs, you can also try

```
pscons view
```

to run certain computations in parallel.

(c) Generate a movie displaying the velocity continuation process. Is it possible to detect velocities from focusing zero-offset diffractions?

(d) Modify the program in the `anglemig.c` file to input a variable migration velocity instead of using a constant velocity. Regenerate Figure 4.9 using a variable velocity

```
pscons anglemig.view
```

Do you notice a difference?

(e) For EXTRA CREDIT, find a way to estimate migration velocity from the data.

```
from rsf.proj import *

# Download velocity model from the data server
###################################################
vstr = 'sigsbee2a_stratigraphy.sgy'
Fetch(vstr,'sigsbee')
Flow('zvstr',vstr,'segyread read=data')

Flow('vel','zvstr',
     '''
     put d1=0.00762 o2=3.05562 d2=0.00762
     label1=Depth unit1=km label2=Distance unit2=km
     label=Velocity unit=km/s |
     scale dscale=0.0003048
     ''')
Result('vel',
       '''
       grey wanttitle=n color=j allpos=y
       screenratio=0.3125 screenht=4 labelsz=4
       scalebar=y barreverse=y
       ''')

# Window a portion
Flow('vel2','vel','window max2=9.5')

dt = 0.002
nt = 5001

# Convert to RMS
Flow('voft','vel2',
     'depth2time velocity=$SOURCE dt=%g nt=%d' % (dt,nt))
Flow('vrms','voft',
     '''
     add mode=p $SOURCE | causint |
     math output="sqrt(input*%g/(x1+%g))" |
     window j1=2 j2=2
     ''' % (dt,dt))

# Download zero-offset from the data server
###############################################
Fetch('data0.rsf','sigsbee')
Flow('data','data0','dd form=native')

Result('data',
       'grey title="Zero-Offset Data" ')

# Slope estimation
Flow('dip','data','dip rect1=100 rect2=10')
Result('dip',
       '''
       grey color=j scalebar=y
       title="Dominant Slope"
       barlabel=Slope barunit=samples
       ''')

# Plane-wave destruction
```

```
Flow('dif','data dip','pwd dip=${SOURCES[1]}')
Result('dif','grey title="Separated Diffractions" ')

# Velocity continuation
Flow('fourier','dif','cosft sign2=1')
Flow('velconf','fourier',
     '''
     stolt vel=1.5 | spray axis=2 n=1 o=0 d=1 |
     fourvc pad2=4096 nv=61 dv=0.02 v0=1.4 verb=y
     ''',split=[2,424],reduce='cat axis=3')
Flow('velcon','velconf',
     '''
     transp plane=23 memsize=1000 |
     cosft sign2=-1 |
     transp plane=23 memsize=1000
     ''')

# Picking a slice
##################
Flow('dimage','velcon vrms',
     'slice pick=${SOURCES[1]}')
Result('dimage',
       '''
       window min1=3 |
       grey title="Imaged Diffractions"
       ''')

# Angle-gather migration
##########################
prog = Program('anglemig.c',
               CPPDEFINES='NO_BLAS',LIBS=['rsf','m'])

Flow('anglemig','dif %s' % prog[0],
     './${SOURCES[1]} vel=2.5 na=90 a0=-45 da=1')

Result('anglemig',
       '''
       window min2=2 |
       transp | transp plane=23 memsize=1000 |
       byte gainpanel=all | grey3
       frame1=1000 frame2=200 frame3=60 unit3="\^o\_"
       title="Dip Angle Gathers" point1=0.7 point2=0.7
       ''')

End()
```

```
/* 2-D angle-domain zero-offset migration. */
#include <rsf.h>
```

```c

static float get_sample (float **dat,
                         float t, float y,
                         float t0, float y0,
                         float dt, float dy,
                         int nt, int ny)
/* extract data sample by linear interpolation */
{
    int it, iy;

    y = (y - y0)/dy; iy = floorf (y);
    y -= (float)iy;
    if (iy < 0 || iy >= (ny - 1)) return 0.0;
    t = (t - t0)/dt; it = floorf (t);
    t -= (float)it;
    if (it < 0 || it >= (nt - 1)) return 0.0;

    return (dat[iy][it]*(1.0 - y)*(1.0 - t) +
            dat[iy][it + 1]*(1.0 - y)*t +
            dat[iy + 1][it]*y*(1.0 - t) +
            dat[iy + 1][it + 1]*y*t);
}

int main (int argc, char* argv[])
{
    int iz, ix, nx, iy, ia, na, nt;
    float dt, dy, vel, da, a0, dx, z, t, y, x, a;
    float **dat, *img;
    sf_file data, imag;

    sf_init (argc, argv);

    data = sf_input ("in");
    imag = sf_output ("out");

    /* get dimensions */
    if (!sf_histint (data, "n1", &nt))   sf_error ("n1");
    if (!sf_histint (data, "n2", &nx))   sf_error ("n2");
    if (!sf_histfloat (data, "d1", &dt)) sf_error ("d1");
    if (!sf_histfloat (data, "d2", &dx)) sf_error ("d2");

    if (!sf_getint("na",&na)) sf_error("Need na=");
    /* number of angles */
    if (!sf_getfloat("da",&da)) sf_error("Need da=");
    /* angle increment */
    if (!sf_getfloat("a0",&a0)) sf_error("Need a0=");
    /* initial angle */

    sf_shiftdim(data, imag, 1);

    sf_putint(imag,"n1",na);
    sf_putfloat(imag,"d1",da);
    sf_putfloat(imag,"o1",a0);
    sf_putstring(imag,"label1","Angle");

    /* degrees to radians */
    a0 *= SF_PI/180.;
    da *= SF_PI/180.;

    if (!sf_getfloat("vel",&vel)) vel=1.5;
    /* constant velocity */

    dat = sf_floatalloc2(nt,nx);
    sf_floatread (dat[0],nt*nx, data);

    img = sf_floatalloc (na);


    for (ix = 0; ix < nx; ix++) {
        x = ix*dx;
        sf_warning ("CMP %d of %d;", ix, nx);

        for (iz = 0; iz < nt; iz++) {
            z = iz*dt;

            for (ia = 0; ia < na; ia++) {
                a = a0+ia*da;

                t = z/cosf(a);
                /* escape time */
                y = x+0.5*vel*t*sinf(a);
                /* escape location */

                img[ia] = get_sample (dat,t,y,0.,0.,
                                      dt,dx,nt,nx);
            }

            sf_floatwrite (img, na, imag);
        } /* iz */
    } /* ix */

    exit(0);
}
```

3. In the final part of the computational assignment, we return to the 2-D field dataset

from the Gulf of Mexico. The zero-offset data after a DMO stack are shown in Figure 6.2(b).

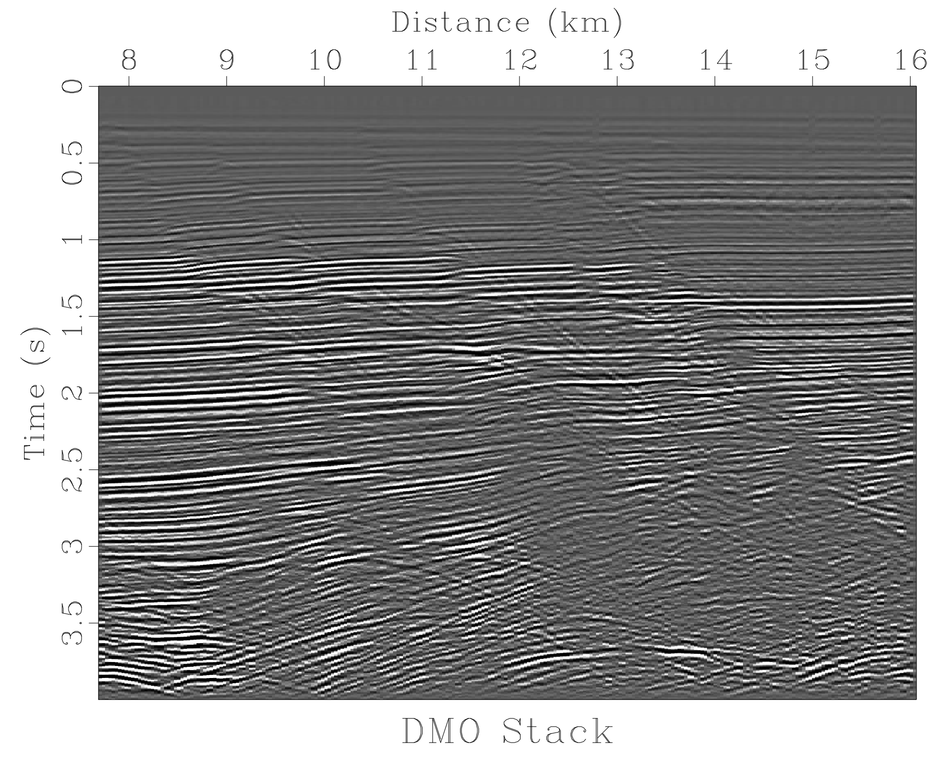


Figure 4.10: 2-D field dataset from the Gulf of Mexico after DMO stack. hw4/gulf stack

(a) Change directory

```
cd hw4/gulf
```

(b) Run

```
scons view
```

to generate Figure 6.2(b) and display it on your screen.

(c) Edit the `SConstruct` file to implement a processing flow involving velocity continuation and angle-gather migration. Make sure to select appropriate processing parameters.

```
from rsf.proj import *

Fetch('bei-stack.rsf','midpts')
Flow('stack','bei-stack',
     '''
     dd form=native |
     put label2=Distance unit2=km label1=Time unit1=s
     ''')

Result('stack','grey title="DMO Stack" ')
```

```

End ()
```

4.3 Completing the assignment

1. Change directory to `hw4`.
2. Edit the file `paper.tex` in your favorite editor and change the first line to have your name instead of Malovichko'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.

Chapter 5

Homework 5

ABSTRACT

This homework has only a computational part but it will require you to make some theoretical developments as well. The theme of the homework is wavefield extrapolation. You will develop efficient approximations for one-way and two-way wave extrapolators for a constant velocity and for a highly variable velocity.

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://rsf.svn.sourceforge.net/svnroot/rsf/trunk/book/geo384w/hw5
```

5.1 Computational part

1. The first example is the model from Homework 1 with a hyperbolic reflector under a constant velocity layer. The model is shown in Figure 5.1. Figure 5.2 shows a shot gather modeled at the surface and extrapolated to a level of 1 km in depth using two extrapolation operators:

   (a) The exact phase shift filter

$$\hat{U}(z,k,\omega) = \hat{U}(0,k,\omega)\, e^{i\sqrt{S^2\,\omega^2-k^2}\,z} \;. \tag{5.1}$$

(b) Its approximation

$$\hat{U}(z,k,\omega) \approx \hat{U}(0,k,\omega)\, e^{i\,S\,\omega\,z\, \frac{S\,\omega + \frac{i\,(\cos(k\,\Delta x)-1)\,z}{2\,(\Delta x)^2}}{S\,\omega - \frac{i\,(\cos(k\,\Delta x)-1)\,z}{2\,(\Delta x)^2}}}\,. \tag{5.2}$$

Approximation (5.2) is suitable for an implementation in the space domain with a digital recursive filter. However, its accuracy is limited, which is evident both from Figure 5.2 and from Figure 5.3, which compares the phases of the exact and the approximate extrapolators. We can see that approximation (5.2) is accurate only for small angles from the vertical θ (defined by $\sin\theta = k\,S/\omega$).

Your task: Design an approximation that would be more accurate than approximation (5.2). Your approximation should be suitable for a digital filter implementation in the space domain. Therefore, it can involve k only through $\cos(k\,\Delta x)$ functions.

(a) Change directory

```
cd hw5/hyper
```

(b) Run

```
scons view
```

to generate figures and display them on your screen.

(c) Edit the `SConstruct` file to change the approximate extrapolator.

(d) Run

```
scons view
```

again to observe the differences.

```
from rsf.proj import *
from math import pi

# Make a reflector model
Flow('refl',None,
     '''
     math n1=10001 o1=-4 d1=0.001 output="sqrt(1+x1*x1)"
     ''')
Flow('model','refl',
     '''
     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',
       '''
       window min2=-1 max2=2 j2=10 |
       grey allpos=y title=Model
```

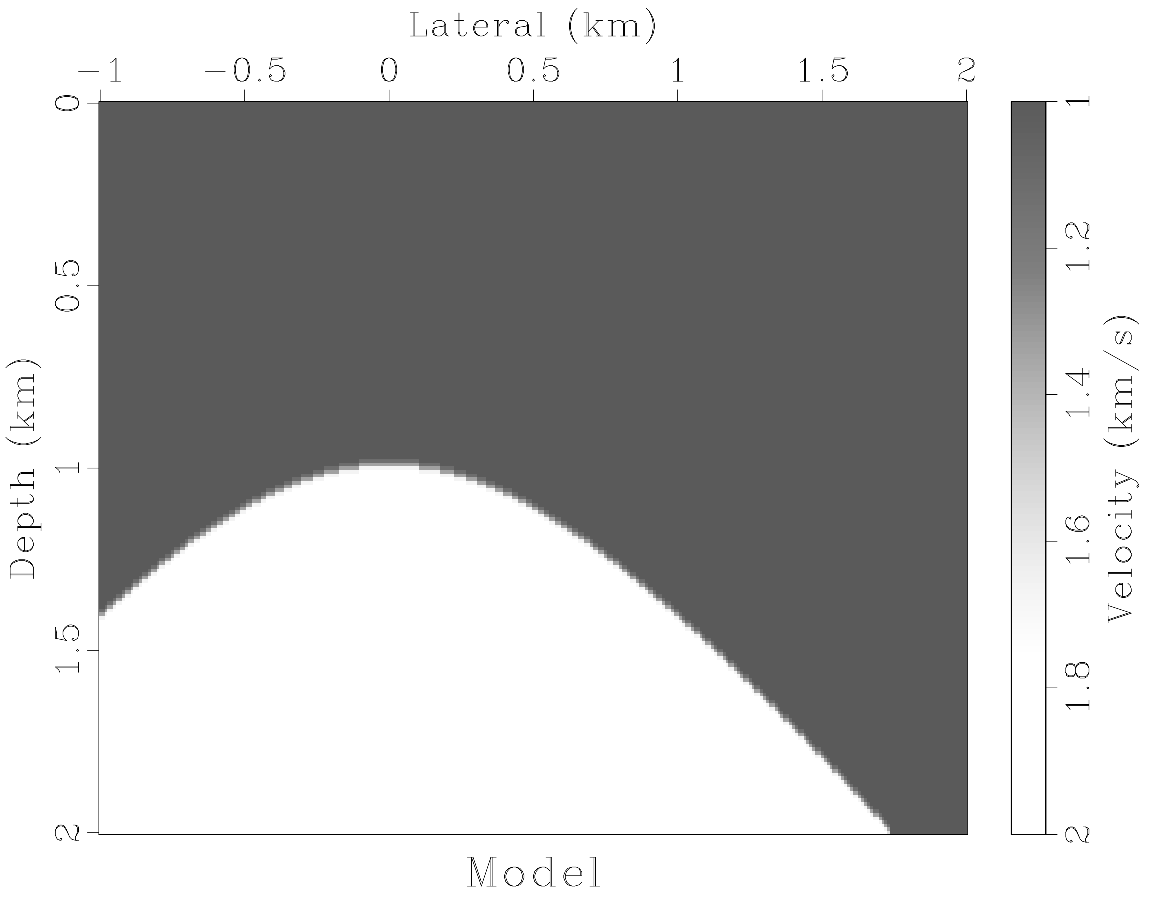


Figure 5.1: Synthetic velocity model with a hyperbolic reflector. $\boxed{\text{hw5/hyper model}}$

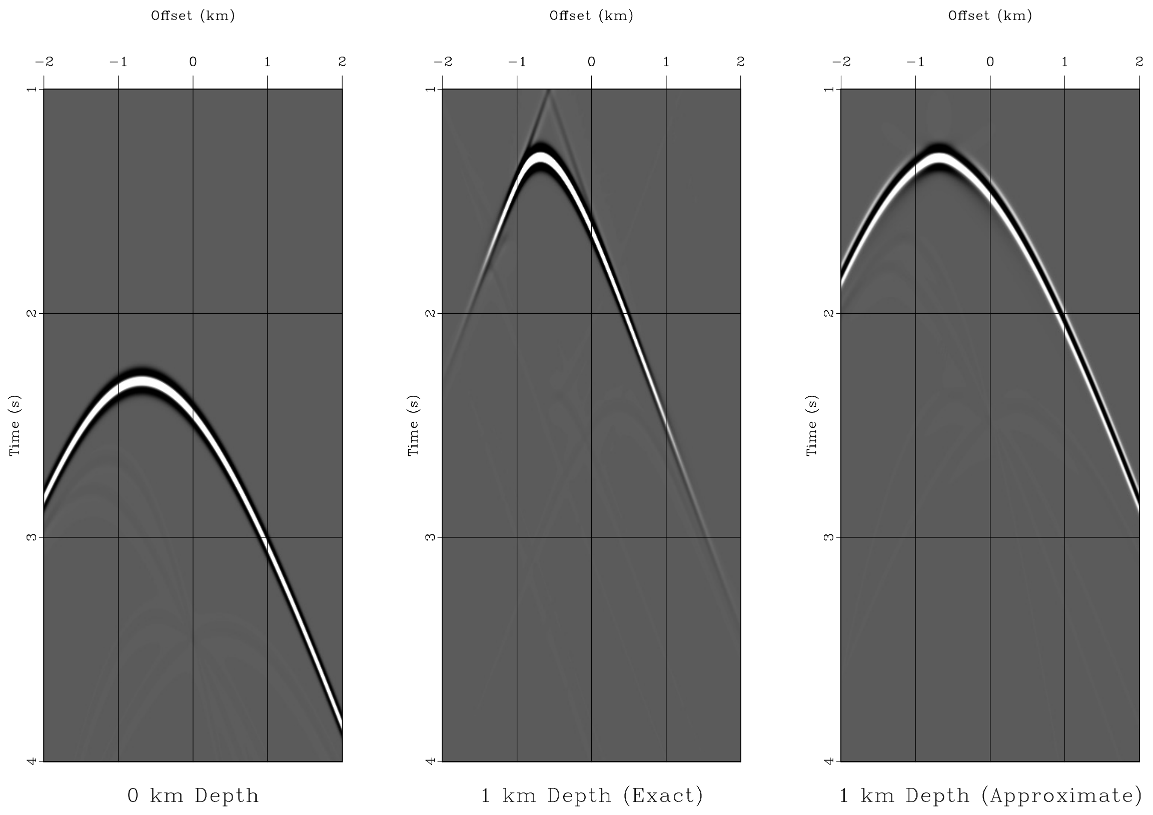


Figure 5.2: Synthetic shot gather. Left: Modeled for receivers at the surface. Middle: Receivers extrapolated to 1 km in depth with an exact phase-shift extrapolation operator. Right: Receivers extrapolated to 1 km in depth with an approximate extrapolation operator. hw5/hyper shot

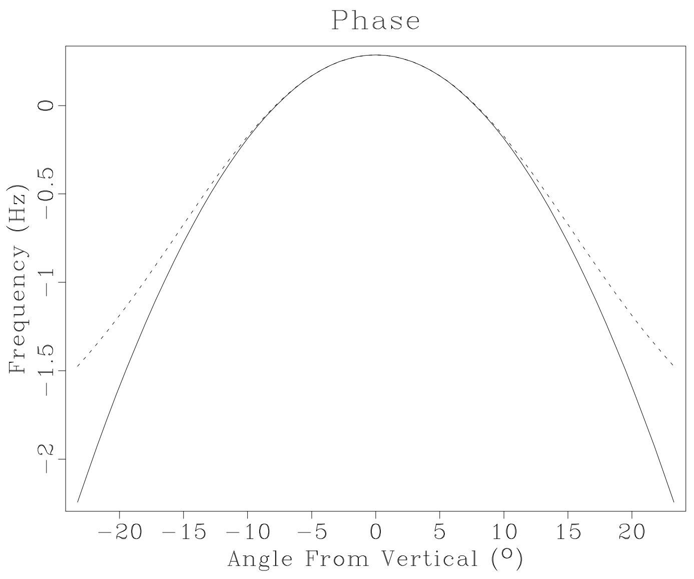


Figure 5.3: Phase of the extrapolation operator at 5 Hz frequency as a function of the wave propagation angle. Solid line: exact extrapolator. Dashed line: approximate extrapolator. hw5/hyper phase

```
          scalebar=y barreverse=y
          ''')

# Reflector dip
Flow('dip','refl','math output="x1/input" ')

# Kirchoff modeling
Flow('shot','refl dip',
       '''
       kirmod nt=1501 ns=1 s0=1 ds=0.01 nh=801 dh=0.01 h0=-4
       twod=y vel=1 freq=10 dip=${SOURCES[1]} verb=y |
       costaper nw2=200
       ''')
Plot('shot',
       '''
       window min2=-2 max2=2 min1=1 max1=4 |
       grey title="0 km Depth"
       label1=Time unit1=s label2=Offset unit2=km
       labelsz=10 titlesz=15 grid=y
       ''')

# Double Fourier transform

Flow('kw','shot','fft1 | fft3 axis=2')

# Extrapolation filters

dx = 0.01
dz = 1

dx2 = 2*pi*dx
dz2 = 2*pi*dz
a = 0.5*dz2/(dx2*dx2)

Flow('exact','kw',
       'math output="exp(I*sqrt(x1*x1-x2*x2)*%g)" ' % dz2)

Flow('approximate','kw',
       '''
       math output="exp(I*x1*%g)*
       (x1+0.01+I*(cos(x2*%g)-1)*%g)/
       (x1+0.01-I*(cos(x2*%g)-1)*%g)"
       ''' % (dz2,dx2,a,dx2,a))

# Extrapolation

for case in ('exact','approximate'):
    Flow('phase-'+case,case,
           '''
           window n1=1 min1=5 min2=-2 max2=2 |
           math output="log(input)" | sfimag
           ''')
    Flow('angle-'+case,'phase-'+case,
           '''
```

```
            math output="%g*asin(x1/5)" |
            cmplx $SOURCE
            ''' % (180/pi))

      Flow('shot-'+case,['kw',case],
            '''
            mul ${SOURCES[1]} |
            fft3 axis=2 inv=y | fft1 inv=y
            ''')
      Plot('shot-'+case,
            '''
            window min2=-2 max2=2 min1=1 max1=4 |
            grey title="%g km Depth (%s)"
            label1=Time unit1=s label2=Offset unit2=km
            labelsz=10 titlesz=15 grid=y
            ''' % (dz, case.capitalize()))

Result('shot','shot shot-exact shot-approximate',
        'SideBySideAniso')

Result('phase','angle-exact angle-approximate',
        '''
        cat axis=2 ${SOURCES[1]} |
        graph title=Phase dash=0,1
        label1="Angle From Vertical" unit1="\^o\_"
        ''')

## TWO-WAY WAVE EXTRAPOLATION
###############################



End()
```

2. Next, we will approach the imaging task using reverse-time migration with a two-way wave extrapolation. Figure 5.4(a) shows synthetic zero-offset data generated by Kirchhoff modeling. Figure 5.4(b) shows an image generated by zero-offset reverse-time migration using an explicit finite-difference wave extrapolation in time.

Your task: Change the program for reverse-time migration to implement forward-time modeling using the "exploding reflector" approach.

(a) Change directory

```
cd hw5/hyper2
```

(b) Run

```
scons view
```

to generate figures and display them on your screen.

(c) Run

```
scons wave0.vpl
```

to observe a movie of reverse-time wave extrapolation.

(d) Edit the program in the `rtm.c` file to implement a process opposite to migration: starting from the reflectivity image like the one in Figure 5.4(b) and generating zero-offset data like the one in Figure 5.4(a).

(e) Run

```
scons view
```

again to observe the differences.

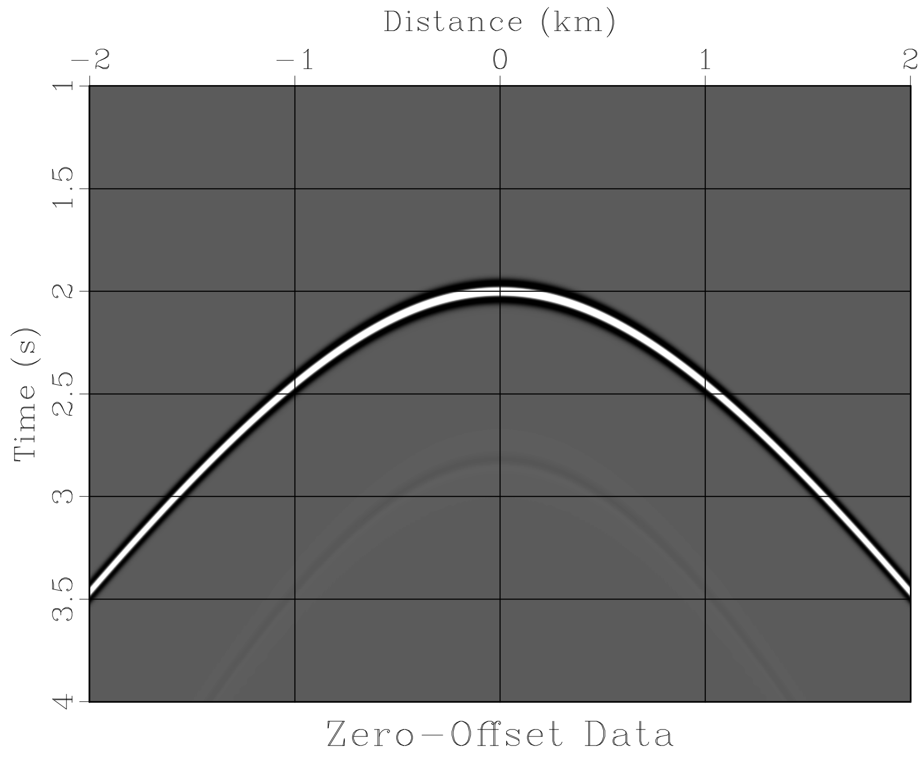
```
from rsf.proj import *

# Make a reflector model
Flow('refl',None,
     '''
     math n1=10001 o1=-4 d1=0.001 output="sqrt(1+x1*x1)"
     ''')
Flow('model0','refl',
     '''
     window min1=-3 max1=3 j1=5 |
     unif2 n1=401 d1=0.005 v00=1,2 |
     put label1=Depth unit1=km label2=Lateral unit2=km |
     smooth rect1=3
     ''')

Plot('model0',
     '''
     window min2=-1 max2=2 |
     grey allpos=y bias=1 title=Model
     ''')
Plot('refl',
     '''
     graph wanttitle=n wantaxis=n yreverse=y
     min1=-1 max1=2 min2=0 max2=2
     plotfat=3 plotcol=4
     ''')
Result('model0','model0 refl','Overlay')

# Kirchoff zero-offset modeling
Flow('dip','refl','math output="x1/input" ')

```



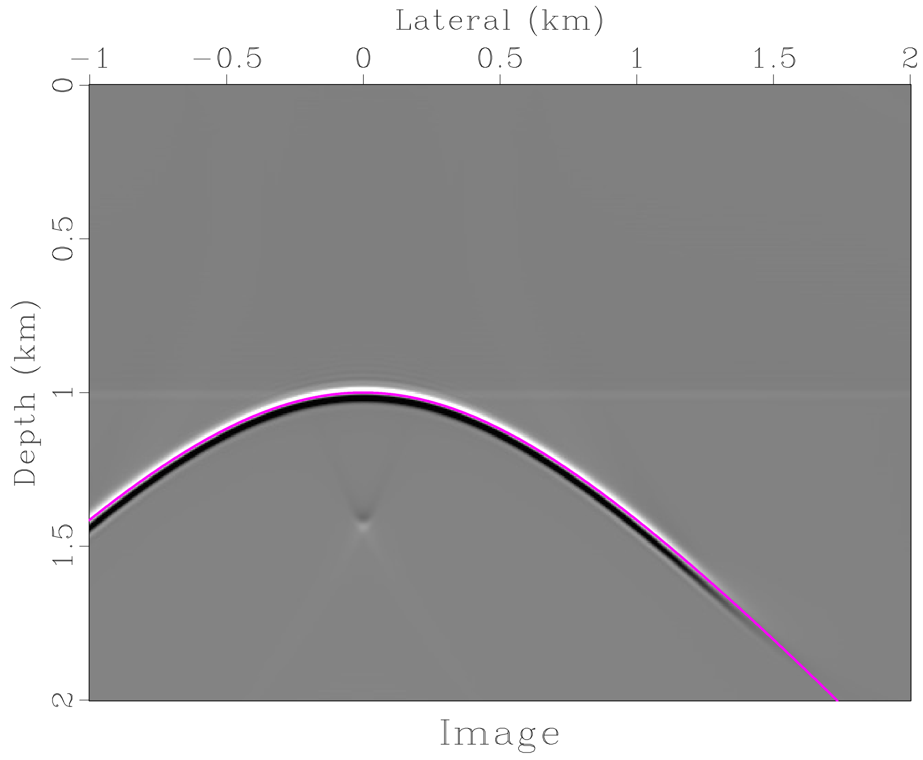


Figure 5.4: (a) Synthetic zero-offset data corresponding to the model in Figure 5.1. (b) Image generated by reverse-time exploding-reflector migration. The location of the exact reflector is indicated by a curve. hw5/hyper2 data0,image0

```
Flow('data0','refl dip',
     '''
     kirmod nt=1501 dt=0.004 freq=10
     ns=1201 s0=-3 ds=0.005 nh=1 dh=0.01 h0=0
     twod=y vel=1 dip=${SOURCES[1]} verb=y |
     window | costaper nw2=200
     ''')
Result('data0',
     '''
     window min2=-2 max2=2 min1=1 max1=4 |
     grey title="Zero-Offset Data" grid=y
     label1=Time unit1=s label2=Distance unit2=km
     ''')

# Reverse-time migration
exe = Program('rtm.c',CPPDEFINES='NO_BLAS')
Flow('image0 wave0','model0 %s data0' % exe[0],
     '''
     pad beg1=200 end1=200 | math output=0.5 |
     ./${SOURCES[1]} data=${SOURCES[2]}
     wave=${TARGETS[1]} jt=20 n0=200
     ''')

# Display wave propagation movie
Plot('wave0',
     '''
     window min2=-1 max2=2 min1=0 max1=2 n3=1 f3=35 |
     grey wanttitle=n gainpanel=all
     ''',view=1)
Result('wave0','wave0 refl','Overlay')
# Display image
Plot('image0',
     '''
     window min2=-1 max2=2 min1=0 max1=2 |
     grey title=Image
     ''')
Result('image0','image0 refl','Overlay')

End()
```

```c
/* 2-D zero-offset reverse-time migration */
#include <stdio.h>

#include <rsf.h>

static int nz, nx;
static float c0, c11, c21, c12, c22;

static void laplacian(float **uin  /* [nx][nz] */,
                      float **uout /* [nx][nz] */)
/* Laplacian operator, 4th-order finite-difference */
{
    int iz, ix;

    for (ix=2; ix < nx-2; ix++) {
        for (iz=2; iz < nz-2; iz++) {
            uout[ix][iz] =
                c11*(uin[ix][iz-1]+uin[ix][iz+1]) +
                c12*(uin[ix][iz-2]+uin[ix][iz+2]) +
                c21*(uin[ix-1][iz]+uin[ix+1][iz]) +
                c22*(uin[ix-2][iz]+uin[ix+2][iz]) +
                c0*uin[ix][iz];
        }
    }
}

int main(int argc, char* argv[])
{
    int it,ix,iz;          /* index variables */
    int nt,n0,n2, jt;
    float dt,dx,dz, dt2,d1,d2;

    float  **vv, **dd;
    float  **u0,**u1,u2,**ud; /* tmp arrays */

    sf_file data, imag, modl, wave; /* I/O files */

    sf_init(argc,argv);

    /* setup I/O files */
    modl = sf_input ("in");    /* velocity model */
    imag = sf_output("out");   /* output image */

    data = sf_input ("data"); /* seismic data */
    wave = sf_output("wave"); /* wavefield */

    /* Dimensions */
    if (!sf_histint(modl,"n1",&nz)) sf_error("n1");
    if (!sf_histint(modl,"n2",&nx)) sf_error("n2");

    if (!sf_histfloat(modl,"d1",&dz)) sf_error("d1");
    if (!sf_histfloat(modl,"d2",&dx)) sf_error("d2");

    if (!sf_histint  (data,"n1",&nt)) sf_error("n1");
    if (!sf_histfloat(data,"d1",&dt)) sf_error("d1");

    if (!sf_histint(data,"n2",&n2) || n2 != nx)
        sf_error("Need n2=%d in data",nx);

    if (!sf_getint("n0",&n0)) n0=0;
    /* surface */
    if (!sf_getint("jt",&jt)) jt=1;
    /* time interval */

    sf_putint(wave,"n3",1+(nt-1)/jt);
    sf_putfloat(wave,"d3",-jt*dt);
    sf_putfloat(wave,"o3",(nt-1)*dt);

    dt2 = dt*dt;

    /* set Laplacian coefficients */
    d1 = 1.0/(dz*dz);
    d2 = 1.0/(dx*dx);

    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);

    /* read data and velocity */
    dd = sf_floatalloc2(nt,nx);
    sf_floatread(dd[0],nt*nx,data);

    vv = sf_floatalloc2(nz,nx);
    sf_floatread(vv[0],nz*nx,modl);

    /* allocate temporary arrays */
    u0=sf_floatalloc2(nz,nx);
    u1=sf_floatalloc2(nz,nx);
    ud=sf_floatalloc2(nz,nx);

    for (ix=0; ix<nx; ix++) {
        for (iz=0; iz<nz; iz++) {
            u0[ix][iz]=0.0;
            u1[ix][iz]=0.0;
            ud[ix][iz]=0.0;
            vv[ix][iz] *= vv[ix][iz]*dt2;
        }
    }

    /* Time loop */
    for (it=nt-1; it >= 0; it--) {
        sf_warning("%d;",it);

        laplacian(u1,ud);

        for (ix=0; ix<nx; ix++) {
            for (iz=0; iz<nz; iz++) {
                /* scale by velocity */
                ud[ix][iz] *= vv[ix][iz];

                /* time step */
                u2 =
                    2*u1[ix][iz]
                    - u0[ix][iz]
                    + ud[ix][iz];

                u0[ix][iz] = u1[ix][iz];
                u1[ix][iz] = u2;
            }

            /* inject data */
            u1[ix][n0] += dd[ix][it];
        }

        if (0 == it%jt)
            sf_floatwrite(u1[0],nx*nz,wave);
    }
    sf_warning(".");

    /* output image */
    sf_floatwrite(u1[0],nx*nz,imag);

    exit (0);
}
```

3. The second example is the Sigsbee synthetic velocity model which you encountered in Homework 4. The model is shown in Figure 6.3(b). We will select one slice of the model (at 15 kft depth) to analyze different one-way wavefield extrapolators. Figure 5.6 compares the phases of two one-way wave extrapolation operators:

   (a) The exact non-stationary phase shift filter

   $$E(k,x) = e^{i\,\sqrt{S^2(x)\,\omega^2 - k^2}\,\Delta z} \tag{5.3}$$

   (b) Its split-step approximation

   $$E(k,x) \approx e^{i\,S(x)\,\omega\,\Delta z}\;e^{i\,\sqrt{S_0^2\,\omega^2 - k^2}\,\Delta z - i\,S_0\,\omega\,\Delta z} \tag{5.4}$$

   Approximation (5.4) can be implemented efficiently, because it involves only diagonal matrix multiplications in space and wavenumber domains. However, its accuracy is limited.

   Your task: Design an approximation that would be more accurate than approximation (5.4). Your approximation should be suitable for an efficient implementation. It can involve products of functions of x and functions of k but not any functions that mix x and k.

   (a) Change directory

   ```
   cd hw5/sigsbee
   ```

   (b) Run

   ```
   scons view
   ```

   to generate figures and display them on your screen.

   (c) Edit the `SConstruct` file to change the approximate extrapolator.

   (d) Run

   ```
   scons view
   ```

   again to observe the differences.

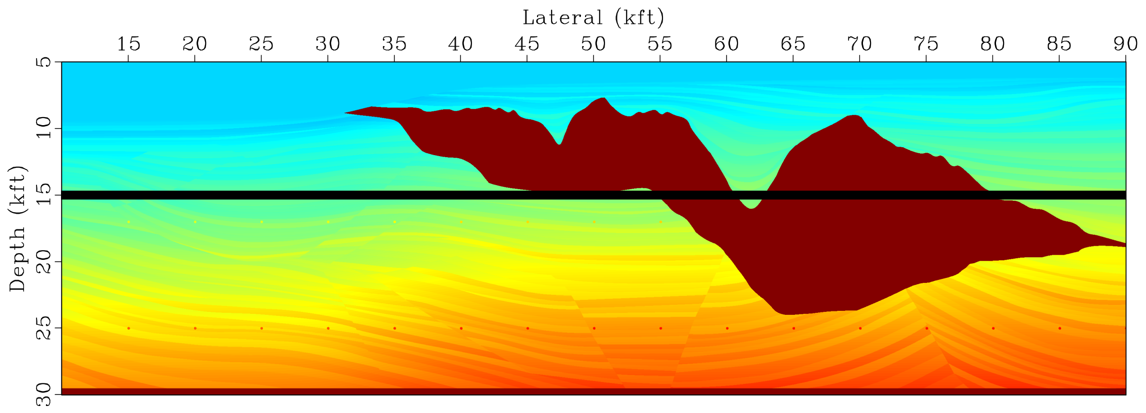


Figure 5.5: Sigsbee velocity model. A slice of the model at 15 kft depth is selected for analyzing wavefield extrapolation operators. $\boxed{\text{hw5/sigsbee vel}}$

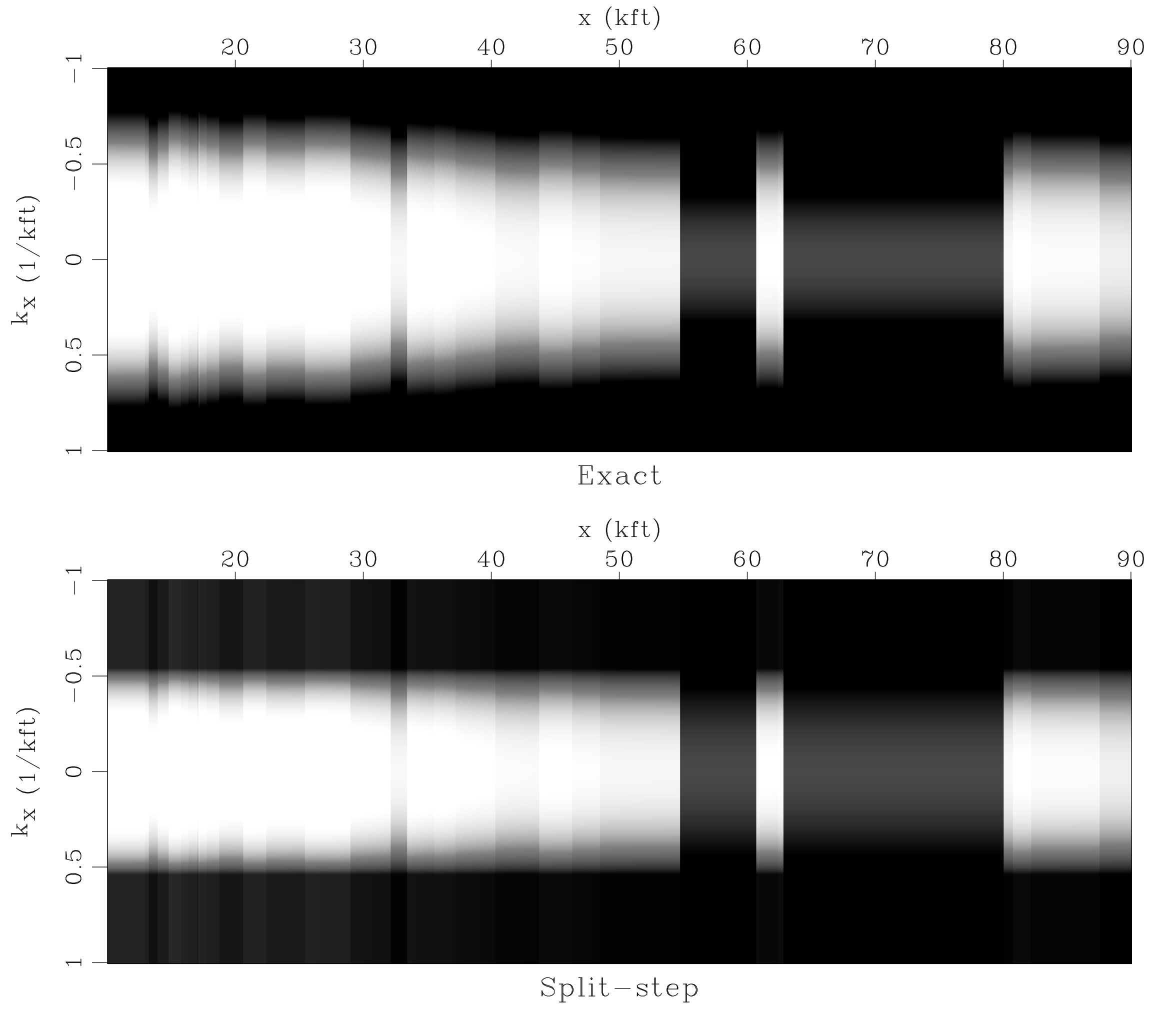


Figure 5.6: Phase of the wavefield extrapolation operators for a depth slice at 15 kft and frequency of 5 Hz as a function of space x and wavenumber k. Top: Exact mixed-domain extrapolator. Bottom: Split-step approximation. hw5/sigsbee phase2

```
from rsf.proj import *
from math import pi

# Download velocity model from the data server
###################################################
vstr = 'sigsbee2a_stratigraphy.sgy'
Fetch(vstr,'sigsbee')
Flow('zvstr zvstr.asc zvstr.bin',vstr,
     '''
     segyread read=data
     hfile=${TARGETS[1]}
     bfile=${TARGETS[2]}
     ''')

Flow('vel','zvstr',
     '''
     put d1=0.025 o2=10.025 d2=0.025 |
     window f1=200 | scale dscale=0.001
     ''')
Plot('vel',
     '''
     grey wanttitle=n color=j allpos=y
     label1=Depth unit1=kft label2=Lateral unit2=kft
     screenratio=0.3125 screenht=4 labelsz=4
     ''')

# Take a slice at 15 kft
##########################

Flow('slice','vel','window n1=1 min1=15')

Plot('line','slice',
     '''
     math output=15 | graph min2=5 max2=30 yreverse=y
     pad=n wanttitle=n wantaxis=n plotcol=7 plotfat=10
     screenratio=0.3125 screenht=4
     ''')
Result('vel','vel line','Overlay')

# Compute extrapolation matrix
#################################

w = 5       # frequency
dz = 0.025  # depth step
v0 = 9.38   # mean velocity

# x-k plane
Flow('xk','slice',
     '''
     spray axis=2 n=201 d=0.01 o=-1 | rtoc |
     put label1=x unit1=kft label2="k\_x" unit2=1/kft
     ''')

```

```
Flow('Exact','xk',
     '''
     math output="exp(I*(sqrt((%g/input)^2-x2^2)*%g))"
     ''' % (w,2*pi*dz))

Flow('Split-step','xk',
     '''
     math output="exp(I*((%g/input+sqrt(%g^2-x2^2)-%g)*%g))"
     ''' % (w,w/v0,w/v0,2*pi*dz))

for case in ('Exact','Split-step'):
    Plot(case,
         '''
         math output="log(input)" | imag |
         grey transp=n title=%s allpos=y
         labelsz=10 titlesz=12
         ''' % case)
Result('phase2','Exact Split-step','OverUnderAniso')


End()
```

4. Figure 5.7 shows an approximate filtered reflectivity of the Sigsbee model.

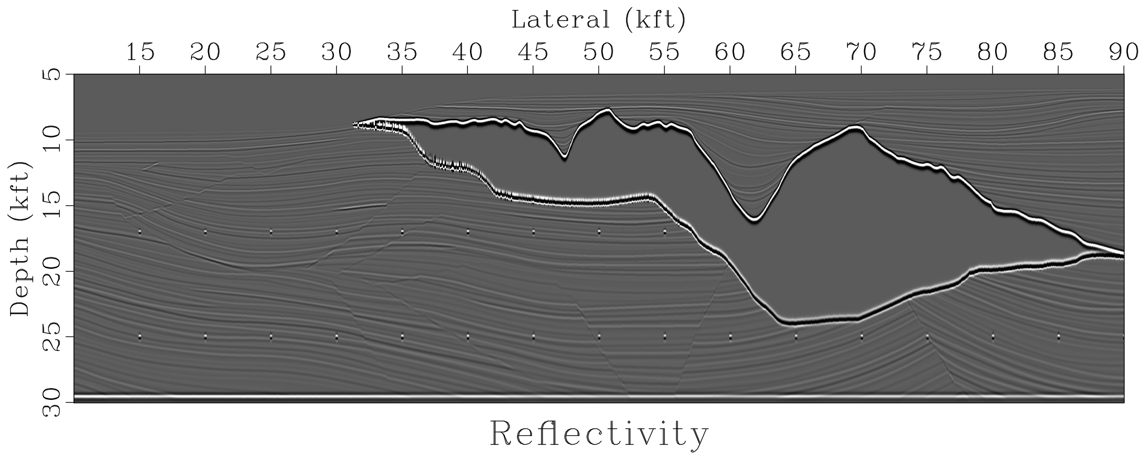


Figure 5.7: Approximate reflectivity of the Sigsbee velocity model (an ideal image). hw5/sigsbee2 zref

Your task: Apply your exploding-reflector modeling and migration program to generate zero-offset data for Sigsbee and image it.

(a) Change directory

`cd hw5/sigsbee2`

(b) Run

`scons view`

to generate figures and display them on your screen.

(c) Edit the `SConstruct` file to implement the modeling and migration experiment.

(d) Include your results in the paper by editing the `hw5/paper.tex` file.

```
from rsf.proj import *

# Download velocity model from the data server
###################################################
vstr = 'sigsbee2a_stratigraphy.sgy'
Fetch(vstr,'sigsbee')
Flow('zvstr',vstr,'segyread read=data')

Flow('zvel','zvstr',
     '''
     put d1=0.025 o2=10.025 d2=0.025
     label1=Depth unit1=kft label2=Lateral unit2=kft |
     scale dscale=0.001
     ''')

Result('zvel',
       '''
       window f1=200 |
       grey title=Velocity titlesz=7 color=j
       screenratio=0.3125 screenht=4 labelsz=5
       mean=y
       ''')


# Compute approximate reflectivity
####################################
Flow('zref','zvel',
     '''
     depth2time velocity=$SOURCE nt=2501 dt=0.004 |
     ai2refl | ricker1 frequency=10 |
     time2depth velocity=$SOURCE
     ''')

Result('zref',
       '''
       window f1=200 |
       grey title=Reflectivity titlesz=7
       screenratio=0.3125 screenht=4 labelsz=5
       ''')


End()
```

5.2 Completing the assignment

1. Change directory to `hw5`.
2. Edit the file `paper.tex` in your favorite editor and change the first line to have your name instead of Fourier'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.

Chapter 6

Homework 6

ABSTRACT

In this (after class) homework we return to the Blake Outer Ridge dataset and process it to create an image of the subsurface. Your task is open-ended: find a way to improve the image by modifying the data processing flow.

6.1 Computational part

The dataset is a 2-D line from the Blake Outer Ridge area offshore Florida. It was collected by USGS in order to study the occurrence of methane hydrates. The dataset and its analysis for gas hydrate detection are described by Ecker et al. (1998, 2000).

The following figures show the dataset at different stages of seismic data processing: from initial data to an image in depth. **Your task:** Modify the data processing sequence to create a justifiably better image.

```
from rsf.proj import *

# get data
Fetch('cmps-tp.HH','blake')

# CMP (common midpoint) gathers
Flow('cmps','cmps-tp.HH',
     'dd form=native | reverse which=2')
# one CMP
Flow('cmp','cmps',
     '''
     window f3=950 n3=1 max1=6 |
     put o2=0.0 d2=1
     ''')
Plot('cmp',
     '''
     grey title="CMP gather"
```

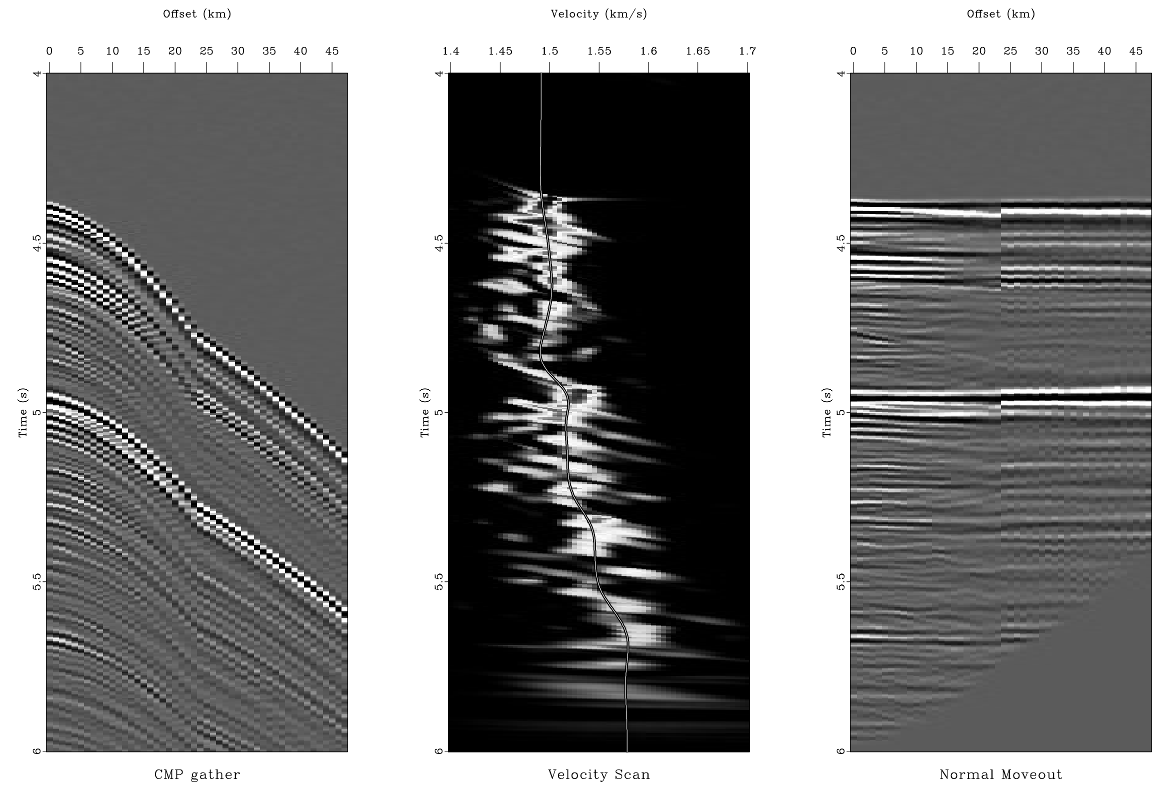
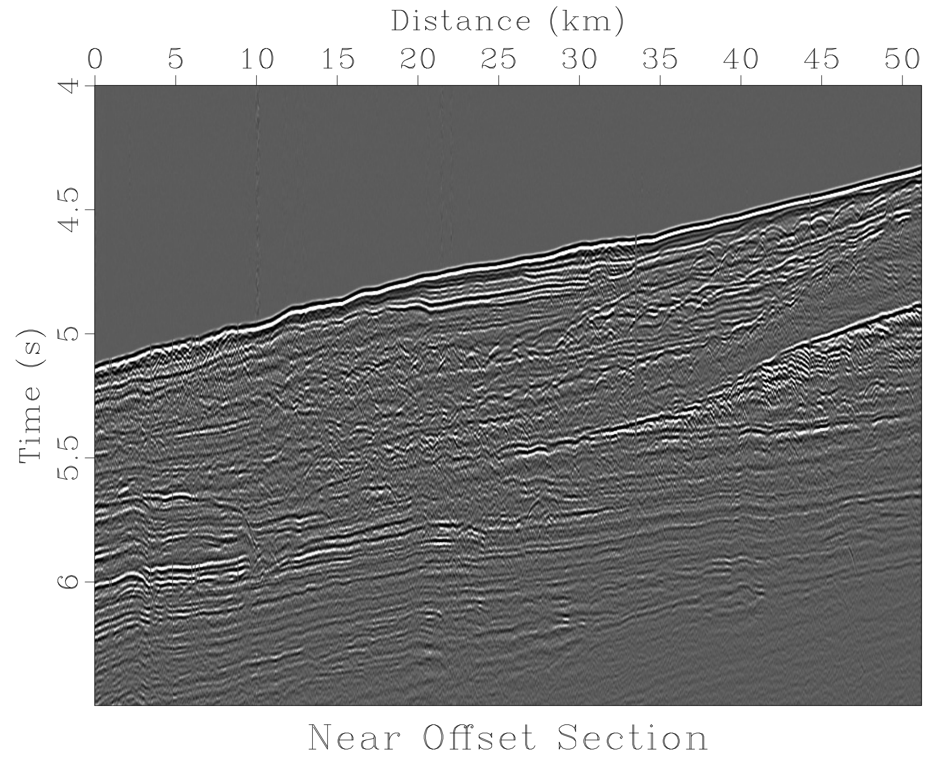


Figure 6.1: Common midpoint gather (left), velocity analysis panel using normal moveout (middle), and common midpoint gather after normal moveout (right). A curve in the middle plot indicates an automatically picked velocity trend. hw6/blake nmo



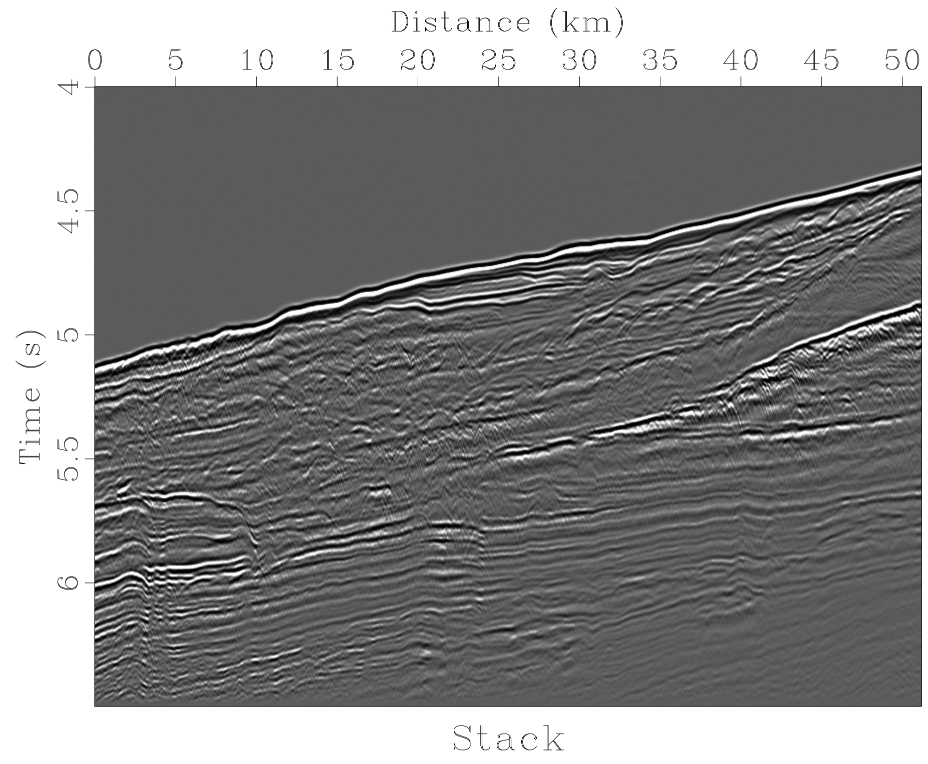
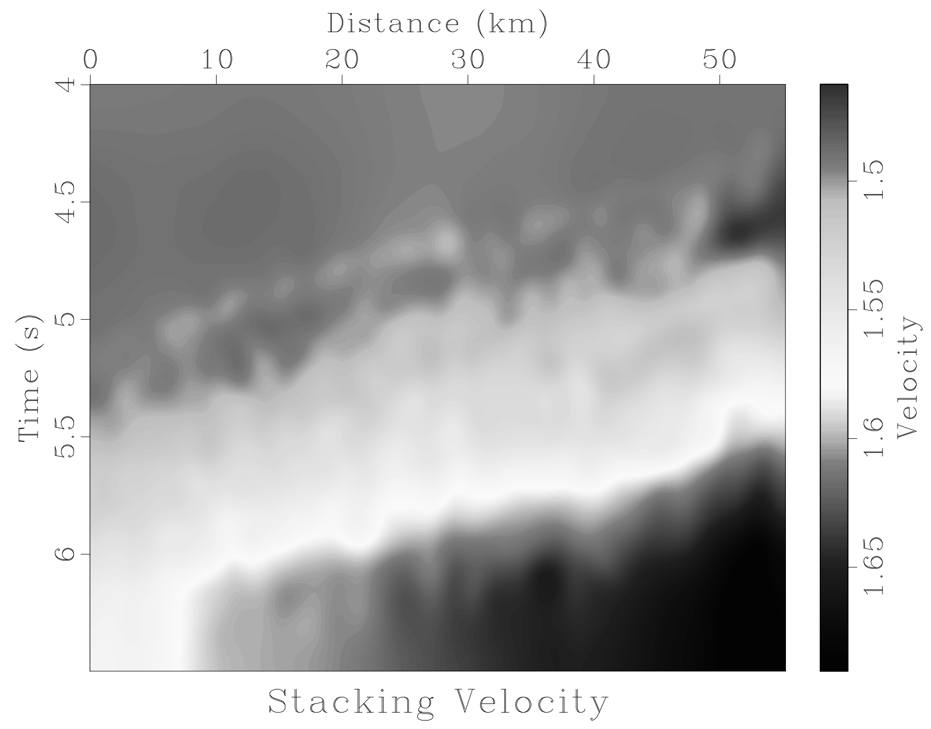


Figure 6.2: (a) Near-offset section. (b) Normal moveout stack. hw6/blake noff,stack



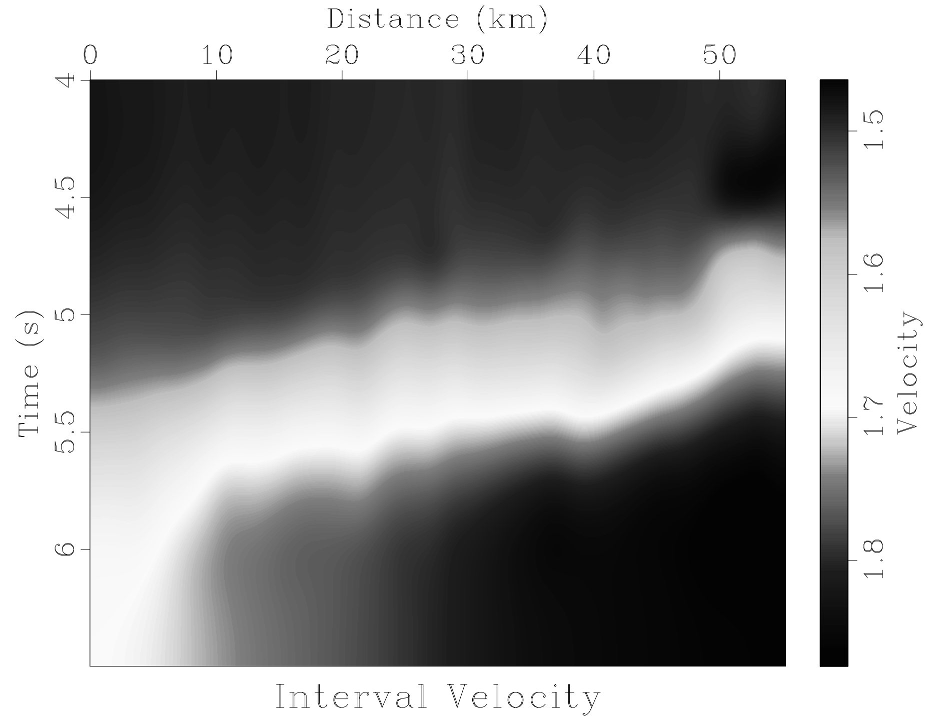


Figure 6.3: (a) Picked stacking velocity. (b) Interval velocity estimated by Dix inversion. hw6/blake picks,vel

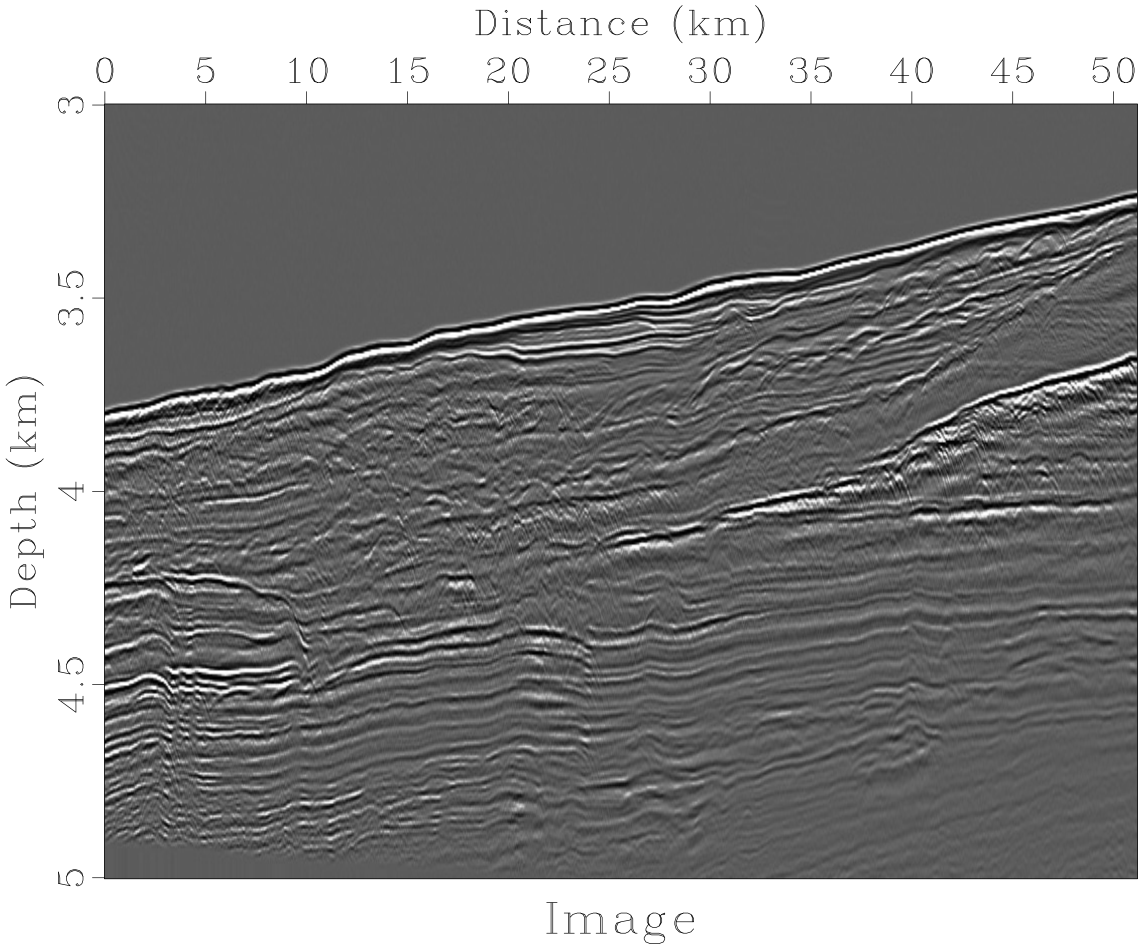


Figure 6.4: Seismic image created by converting stacked data from time to depth. Can you identify geological features that are not properly imaged? hw6/blake image

```
     unit1=s label2=Offset unit2=km
     ''')

# Near-offset section
Result('noff','cmps',
       '''
       window n2=1 n3=1024 |
       grey title="Near Offset Section"
       unit1=s label2=Distance unit2=km
       ''')

# offset maps
Flow('off1',None,'math n1=24 o1=0.4 d1=0.1  output=x1')
Flow('off2',None,'math n1=24 o1=2.8 d1=0.05 output=x1')
Flow('off','off1 off2','cat axis=1 ${SOURCES[1]}')
Flow('offs','off','spray n=111')

# Velocity analysis
###################

vscan = '''
vscan
offset=${SOURCES[1]} v0=1.4 nv=61 dv=0.005 half=n semblance=y
'''
pick = 'pick rect1=20 rect2=3 vel0=1.5'

# compute semblance for one CMP
Flow('vscan','cmp off',vscan)
Plot('vscan',
     '''
     grey color=j allpos=y title="Velocity Scan"
     unit1=s label2=Velocity unit2=km/s pclip=100
     ''')

# pick maximum semblance for one CMP
Flow('pick','vscan',pick)
Plot('pick0','pick',
     '''
     graph transp=y yreverse=y min2=1.4 max2=1.7
     plotcol=7 plotfat=10 pad=n wanttitle=n wantaxis=n
     ''')
Plot('pick1','pick',
     '''
     graph transp=y yreverse=y min2=1.4 max2=1.7
     plotcol=0 plotfat=1 pad=n wanttitle=n wantaxis=n
     ''')
Plot('vscan2','vscan pick0 pick1','Overlay')

# compute semblance for every 10th CMP
Flow('vscans','cmps offs','window j3=10 | ' + vscan)

# pick max semblance for every 10th CMP
Flow('picks0','vscans',pick)

```

```
# interpolate picks on the original grid
Flow('picks','picks0',
     'window | transp | remap1 n1=1105 d1=0.05 o1=0 | transp')
Result('picks',
        '''
        grey color=j scalebar=y barreverse=y allpos=y bias=1.4
        title="Stacking Velocity" unit1=s label2=Distance unit2=km
        ''')

# Normal moveout and stack
###########################

nmo = 'nmo offset=${SOURCES[1]} velocity=${SOURCES[2]} half=n'

Flow('nmo','cmp off pick',nmo)
Plot('nmo',
      '''
      grey title="Normal Moveout"
      unit1=s label2=Offset unit2=km
      ''')
Result('nmo','cmp vscan2 nmo','SideBySideAniso')

Flow('nmos','cmps off picks',nmo)
Flow('stack','nmos','stack')
Result('stack',
        '''
        window n2=1024 |
        grey title=Stack
        unit1=s label2=Distance unit2=km
        ''')

# Convert from time to depth
#############################

# Dix inversion
Flow('semb','vscans picks0','slice pick=${SOURCES[1]}')
Flow('vel0','picks0 semb',
     'window | dix rect1=20 rect2=2 weight=${SOURCES[1]}')

# interpolate on the original grid
Flow('vel','vel0','transp | remap1 n1=1105 d1=0.05 o1=0 | transp')
Result('vel',
        '''
        grey color=j scalebar=y barreverse=y allpos=y bias=1.4
        title="Interval Velocity" unit1=s label2=Distance unit2=km
        ''')

Flow('image','stack vel',
     'time2depth velocity=${SOURCES[1]} intime=y dz=0.005 nz=1001')
Result('image',
        '''
        window n2=1024 min1=3 | grey title=Image
        label1=Depth unit1=km label2=Distance unit2=km
        ''')
```

```

End()
```

1. Change directory

   `> cd ~/geo384w/hw6/blake`

2. Run

   `> scons view`

   to generate figures and display them on your screen.

3. Edit the `SConstruct` file. Check your result by running

   `> scons view`

   again.

6.2 Completing the assignment

1. Change directory to `~/geo384w/hw6`.
2. Edit the file `paper.tex` in your favorite editor and change the first line to have your name instead of Huygens'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. If you do your assignment on one of the computers in the Unix lab, you can simply leave the file in your directory.

REFERENCES

Ecker, C., J. Dvorkin, and A. Nur, 1998, Sediments with gas hydrates: Internal structure from seismic AvO: Geophysics, **63**, 1659–1669.

Ecker, C., J. Dvorkin, and A. M. Nur, 2000, Estimating the amount of gas hydrate and free gas from marine seismic data: Geophysics, **65**, 565–573.