Homework 5

Jean Baptiste Joseph Fourier

ABSTRACT

This homework has only a computational part but it will require you to make some theoretical developments as well. You will develop efficient approximations for one-way extrapolators and experiment with wave-equation migration by least-squares inversion and reverse-time migration.

Completing the computational part of this homework assignment requires

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

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

```
svn co https://github.com/ahay/src/trunk/book/geo384w/hw5
```

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 1. Figure 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

   $$\hat{U}(z,k,\omega) = \hat{U}(0,k,\omega)\, e^{i\sqrt{S^2\,\omega^2 - k^2}\,z}\;. \tag{1}$$

   (b) Its approximation

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

Approximation (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 2 and from Figure 3, which compares the phases of the exact and the approximate extrapolators. We can see that approximation (2) is accurate only for small angles from the vertical θ, defined by

$$\theta = \arcsin\left(\frac{k}{S\,\omega}\right) .$$

Your task: Design an approximation that would be more accurate than approximation (2). Your approximation should be suitable for a digital filter implementation in the space domain. Therefore, it can involve k only through $\cos(n\,k\,\Delta x)$ functions with integer n.

(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
     ''')
Result('model',
       '''
       window min2=-1 max2=2 j2=10 |
```

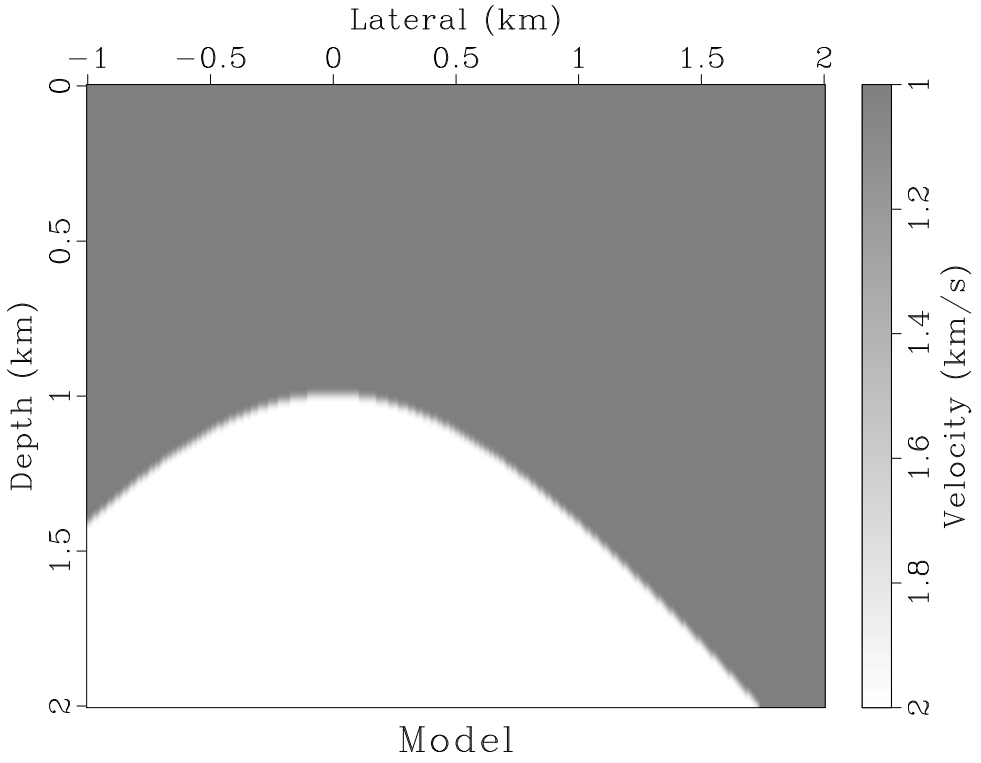


Figure 1: Synthetic velocity model with a hyperbolic reflector.

```
        grey allpos=y title=Model
        scalebar=y barreverse=y
        ''')

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

# Kirchoff modeling
Flow('shot','refl dip',
     '''
     kirmod nt=1501 twod=y vel=1 freq=10
     ns=1 s0=1 ds=0.01 nh=801 dh=0.01 h0=-4
     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
```

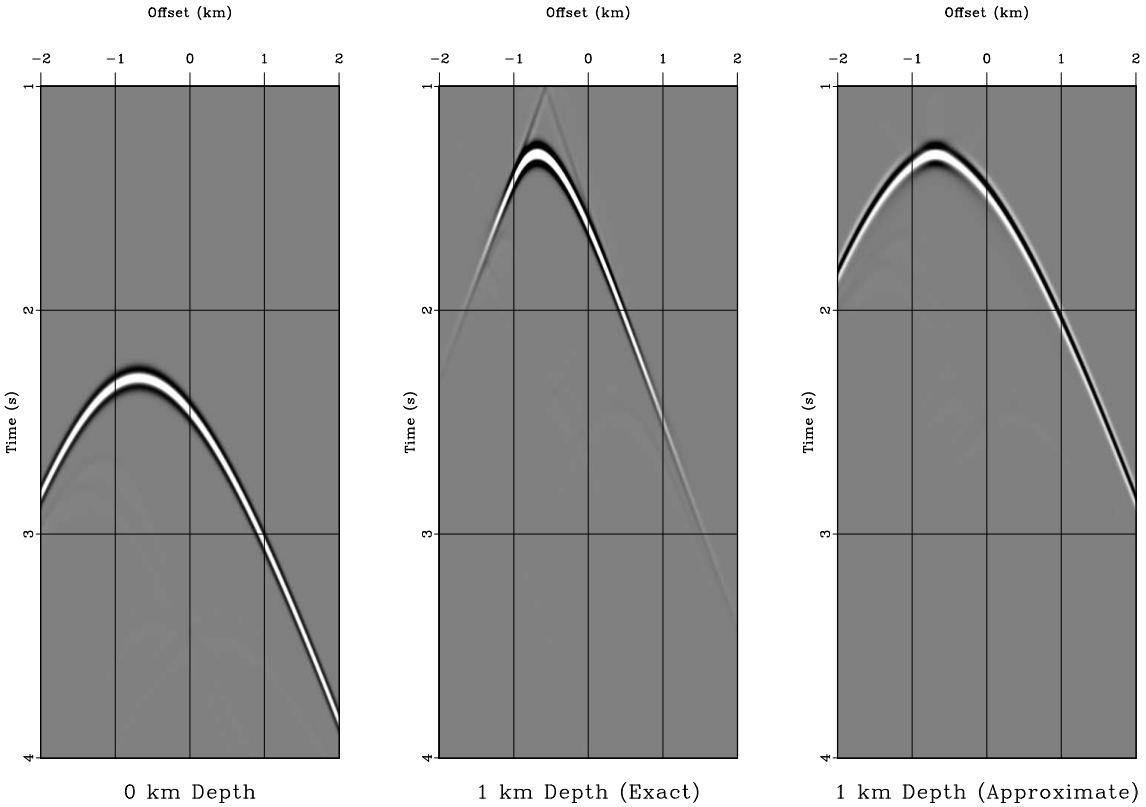


Figure 2: Synthetic shot gather. Left: Modeled for receivers at the surface. Middle: Receivers extrapolated to 1 km in depth with an exact phase-shift extrapolation operator. Right: Receivers extrapolated to 1 km in depth with an approximate extrapolation operator.

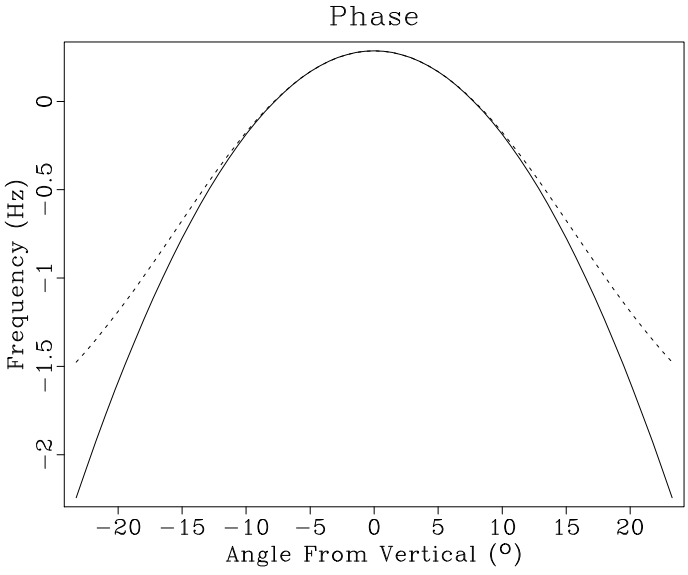


Figure 3: Phase of the extrapolation operator at 5 Hz frequency as a function of the wave propagation angle. Solid line: exact extrapolator. Dashed line: approximate extrapolator.

```
40      ''')
41
42  # Double Fourier transform
43
44  Flow('kw','shot','fft1 | fft3 axis=2')
45
46  # Extrapolation filters
47
48  dx = 0.01
49  dz = 1
50
51  dx2 = 2*pi*dx
52  dz2 = 2*pi*dz
53  a = 0.5*dz2/(dx2*dx2)
54
55  # In the expressions below,
56  # x1 refers to omega and x2 refers to k
57
58  Flow('exact','kw',
59       'math output="exp(I*sqrt(x1*x1-x2*x2)*%g)" ' % dz2)
60
61  Flow('approximate','kw',
62       '''
63       window f1=1 |
64       math output="exp(I*x1*%g)*
65       (x1+I*(cos(x2*%g)-1)*%g)/
66       (x1-I*(cos(x2*%g)-1)*%g)" |
67       pad beg1=1
68       ''' % (dz2,dx2,a,dx2,a))
69
70  # Extrapolation
71
72  for case in ('exact','approximate'):
73      Flow('phase-'+case,case,
74           '''
75           window n1=1 min1=5 min2=-2 max2=2 |
76           math output="log(input)" | sfimag
77           ''')
78      Flow('angle-'+case,'phase-'+case,
79           '''
80           math output="%g*asin(x1/5)" |
81           cmplx $SOURCE
82           ''' % (180/pi))
83
84      Flow('shot-'+case,['kw',case],
```

```
85              '''
86              mul ${SOURCES[1]} |
87              fft3 axis=2 inv=y | fft1 inv=y
88              ''')
89      Plot('shot-'+case,
90              '''
91              window min2=-2 max2=2 min1=1 max1=4 |
92              grey title="%g km Depth (%s)"
93              label1=Time unit1=s label2=Offset unit2=km
94              labelsz=10 titlesz=15 grid=y
95              ''' % (dz, case.capitalize()))
96
97  Result('shot','shot shot-exact shot-approximate',
98          'SideBySideAniso')
99
100 Result('phase','angle-exact angle-approximate',
101         '''
102         cat axis=2 ${SOURCES[1]} |
103         graph title=Phase dash=0,1
104         label1="Angle From Vertical" unit1="\^o\_"
105         ''')
106
107 End()
```

2. Now we will approach the imaging task using reverse-time migration with a two-way wave extrapolation. Figure 4a shows synthetic zero-offset data generated by Kirchhoff modeling. Figure 4b 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 hw6/hyper2
   ```

   (b) Run

   ```
   scons view
   ```

   to generate figures and display them on your screen.

   (c) Run

   ```
   scons wave.vpl
   ```

   to observe a movie of reverse-time wave extrapolation.

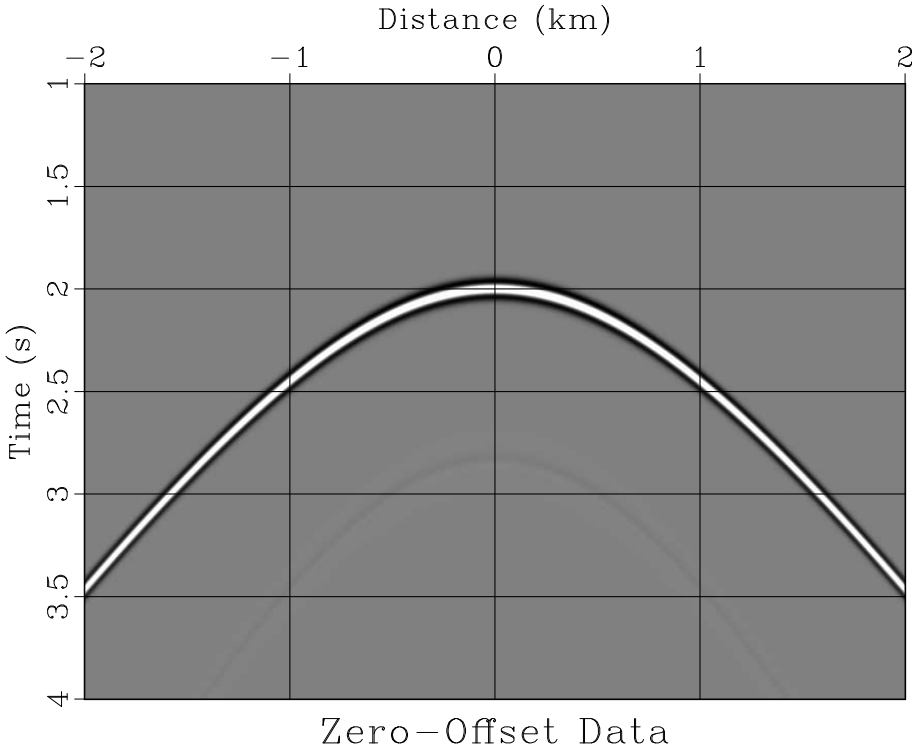
(d) Edit the program in the `rtm.c` file (or `rtm.py`) to implement a process opposite to migration: starting from the reflectivity image like the one in Figure 4b and generating zero-offset data like the one in Figure 4a.

(e) Run

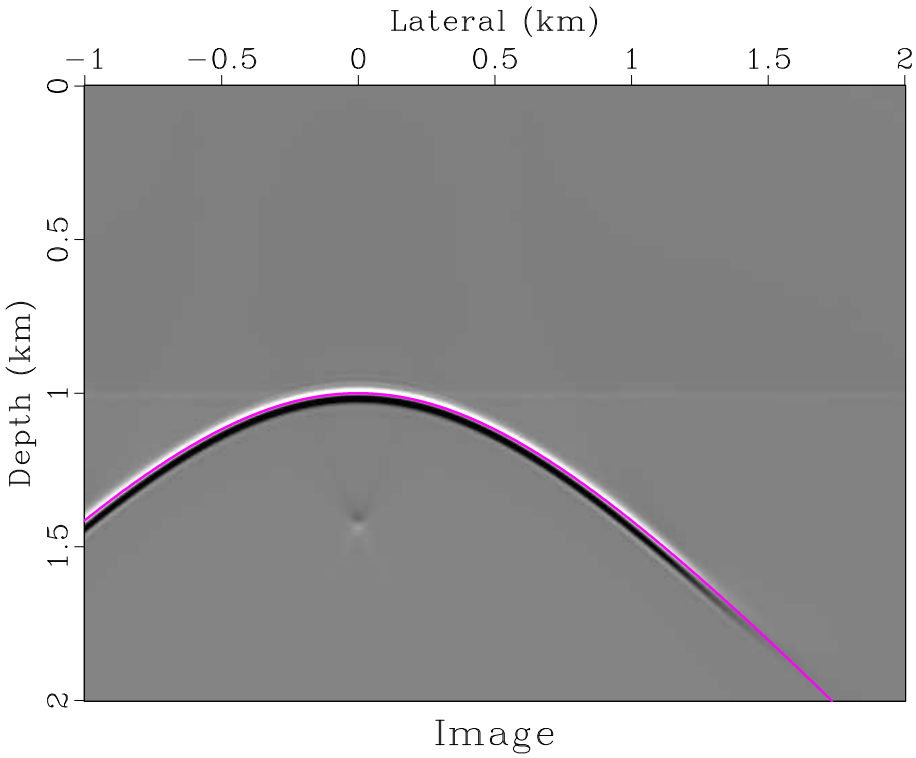
```
scons view
```

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('model','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('model',
     '''
     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('model','model refl','Overlay')

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

Flow('data','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
```



(a)



(b)

Figure 4: (a) Synthetic zero-offset data corresponding to the model in Figure 1. (b) Image generated by reverse-time exploding-reflector migration. The location of the exact reflector is indicated by a curve.

```
38      ''')
39  Result('data',
40      '''
41      window min2=-2 max2=2 min1=1 max1=4 |
42      grey title="Zero-Offset Data" grid=y
43      label1=Time unit1=s label2=Distance unit2=km
44      ''')
45
46  # Reverse-time migration
47  proj = Project()
48
49  # To do the coding assignment in Python,
50  # comment the next line and uncomment the lines below
51  prog = proj.Program('rtm.c')
52
53  #prog = proj.Command('rtm.exe','rtm.py','cp $SOURCE $TARGET')
54  #AddPostAction(prog,Chmod(prog,0o755))
55
56  Flow('image wave','model %s data' % prog[0],
57      '''
58      pad beg1=200 end1=200 | math output=0.5 |
59      ./${SOURCES[1]} data=${SOURCES[2]}
60      wave=${TARGETS[1]} jt=20 n0=200
61      ''')
62
63  # Display wave propagation movie
64  Plot('wave',
65      '''
66      window min2=-1 max2=2 min1=0 max1=2 f3=33 |
67      grey wanttitle=n gainpanel=all
68      ''',view=1)
69
70  # Display image
71  Plot('image',
72      '''
73      window min2=-1 max2=2 min1=0 max1=2 |
74      grey title=Image
75      ''')
76  Result('image','image refl','Overlay')
77
78  End()
```

```
1  /* 2-D zero-offset reverse-time migration */
2  #include <rsf.h>
3
```

```
4  #ifdef _OPENMP
5  #include <omp.h>
6  #endif
7
8  static int n1, n2;
9  static float c0, c11, c21, c12, c22;
10
11 static void laplacian(float **uin  /* [n2][n1] */,
12                        float **uout /* [n2][n1] */)
13 /* Laplacian operator, 4th-order finite-difference */
14 {
15     int i1, i2;
16
17 #ifdef _OPENMP
18 #pragma omp parallel for                                \
19     private(i2,i1)                                      \
20     shared(n2,n1,uout,uin,c11,c12,c21,c22,c0)
21 #endif
22     for (i2=2; i2 < n2-2; i2++) {
23         for (i1=2; i1 < n1-2; i1++) {
24             uout[i2][i1] =
25                 c11*(uin[i2][i1-1]+uin[i2][i1+1]) +
26                 c12*(uin[i2][i1-2]+uin[i2][i1+2]) +
27                 c21*(uin[i2-1][i1]+uin[i2+1][i1]) +
28                 c22*(uin[i2-2][i1]+uin[i2+2][i1]) +
29                 c0*uin[i2][i1];
30         }
31     }
32 }
33
34
35 int main(int argc, char* argv[])
36 {
37     int it,i1,i2;          /* index variables */
38     int nt,n12,n0,nx, jt;
39     float dt,dx,dz, dt2,d1,d2;
40
41     float  **vv, **dd;
42     float  **u0,**u1,u2,**ud; /* tmp arrays */
43
44     sf_file data, imag, modl, wave; /* I/O files */
45
46     /* initialize Madagascar */
47     sf_init(argc,argv);
48
```

```
49      /* initialize OpenMP support */
50      omp_init();
51
52      /* setup I/O files */
53      modl = sf_input ("in");    /* velocity model */
54      imag = sf_output("out");   /* output image */
55
56      data = sf_input ("data"); /* seismic data */
57      wave = sf_output("wave"); /* wavefield */
58
59      /* Dimensions */
60      if (!sf_histint(modl,"n1",&n1)) sf_error("n1");
61      if (!sf_histint(modl,"n2",&n2)) sf_error("n2");
62
63      if (!sf_histfloat(modl,"d1",&dz)) sf_error("d1");
64      if (!sf_histfloat(modl,"d2",&dx)) sf_error("d2");
65
66      if (!sf_histint   (data,"n1",&nt)) sf_error("n1");
67      if (!sf_histfloat(data,"d1",&dt)) sf_error("d1");
68
69      if (!sf_histint(data,"n2",&nx) || nx != n2)
70          sf_error("Need n2=%d in data",n2);
71
72      n12 = n1*n2;
73
74      if (!sf_getint("n0",&n0)) n0=0;
75      /* surface */
76      if (!sf_getint("jt",&jt)) jt=1;
77      /* time interval */
78
79      sf_putint(wave,"n3",1+(nt-1)/jt);
80      sf_putfloat(wave,"d3",-jt*dt);
81      sf_putfloat(wave,"o3",(nt-1)*dt);
82
83      dt2 = dt*dt;
84
85      /* set Laplacian coefficients */
86      d1 = 1.0/(dz*dz);
87      d2 = 1.0/(dx*dx);
88
89      c11 = 4.0*d1/3.0;
90      c12= -d1/12.0;
91      c21 = 4.0*d2/3.0;
92      c22= -d2/12.0;
93      c0  = -2.0 * (c11+c12+c21+c22);
```

```
94
95      /* read data and velocity */
96      dd = sf_floatalloc2(nt,n2);
97      sf_floatread(dd[0],nt*n2,data);
98
99      vv = sf_floatalloc2(n1,n2);
100     sf_floatread(vv[0],n12,modl);
101
102     /* allocate temporary arrays */
103     u0=sf_floatalloc2(n1,n2);
104     u1=sf_floatalloc2(n1,n2);
105     ud=sf_floatalloc2(n1,n2);
106
107     for (i2=0; i2<n2; i2++) {
108         for (i1=0; i1<n1; i1++) {
109             u0[i2][i1]=0.0;
110             u1[i2][i1]=0.0;
111             ud[i2][i1]=0.0;
112             vv[i2][i1] *= vv[i2][i1]*dt2;
113         }
114     }
115
116     /* Time loop */
117     for (it=nt-1; it >= 0; it--) {
118         sf_warning("%d;",it);
119
120         laplacian(u1,ud);
121
122 #ifdef _OPENMP
123 #pragma omp parallel for              \
124     private(i2,i1,u2)                 \
125     shared(ud,vv,it,u1,u0,dd,n0)
126 #endif
127         for (i2=0; i2<n2; i2++) {
128             for (i1=0; i1<n1; i1++) {
129                 /* scale by velocity */
130                 ud[i2][i1] *= vv[i2][i1];
131
132                 /* time step */
133                 u2 =
134                     2*u1[i2][i1]
135                     - u0[i2][i1]
136                     + ud[i2][i1];
137
138                 u0[i2][i1] = u1[i2][i1];
```

```
                    u1[i2][i1] = u2;
                }

                /* inject data */
                u1[i2][n0] += dd[i2][it];
            }

            if (0 == it%jt)
                sf_floatwrite(u1[0],n12,wave);
        }
        sf_warning(".");

        /* output image */
        sf_floatwrite(u1[0],n12,imag);

        exit (0);
}
```

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

import sys
import numpy
import m8r

class Laplacian:
    'Laplacian operator, 4th-order finite-difference'

    def __init__(self,d1,d2):
        self.c11 = 4.0*d1/3.0
        self.c12= -d1/12.0
        self.c21 = 4.0*d2/3.0
        self.c22= -d2/12.0
        self.c1  = -2.0*(self.c11+self.c12)
        self.c2  = -2.0*(self.c21+self.c22)

    def apply(self,uin,uout):
        n1,n2 = uin.shape

        uout[2:n1-2,2:n2-2] = \
          self.c11*(uin[1:n1-3,2:n2-2]+uin[3:n1-1,2:n2-2]) + \
          self.c12*(uin[0:n1-4,2:n2-2]+uin[4:n1  ,2:n2-2]) + \
          self.c1*uin[2:n1-2,2:n2-2] + \
          self.c21*(uin[2:n1-2,1:n2-3]+uin[2:n1-2,3:n2-1]) + \
          self.c22*(uin[2:n1-2,0:n2-4]+uin[2:n1-2,4:n2  ]) + \
          self.c2*uin[2:n1-2,2:n2-2]
```

```
28
29 par = m8r.Par()
30
31 # setup I/O files
32 modl=m8r.Input()         # velocity model
33 imag=m8r.Output()        # output image
34
35 data=m8r.Input ('data') # seismic data
36 wave=m8r.Output('wave') # wavefield
37
38 # Dimensions
39 n1 = modl.int('n1')
40 n2 = modl.int('n2')
41
42 dz = modl.float('d1')
43 dx = modl.float('d2')
44
45 nt = data.int('n1')
46 dt = data.float('d1')
47
48 nx = data.int('n2')
49 if nx != n2:
50     raise RuntimeError('Need n2=%d in data',n2)
51
52 n0 = par.int('n0',0) # surface
53 jt = par.int('jt',1) # time interval
54
55 wave.put('n3',1+(nt-1)/jt)
56 wave.put('d3',-jt*dt)
57 wave.put('o3',(nt-1)*dt)
58
59 dt2 =     dt*dt
60
61 # set Laplacian coefficients
62 laplace = Laplacian(1.0/(dz*dz),1.0/(dx*dx))
63
64 # read data and velocity
65 dd = numpy.zeros([n2,nt],'f')
66 data.read(dd)
67 vv = numpy.zeros([n2,n1],'f')
68 modl.read(vv)
69
70 # allocate temporary arrays
71 u0 = numpy.zeros([n2,n1],'f')
72 u1 = numpy.zeros([n2,n1],'f')
```

```
ud = numpy.zeros([n2,n1],'f')

vv *= vv*dt2

# Time loop
for it in range(nt-1,-1,-1):
    sys.stderr.write("\b\b\b\b %d" % it)

    laplace.apply(u1,ud)

    # scale by velocity
    ud *= vv

    # time step
    u2 = 2*u1 - u0 + ud
    u0 = u1
    u1 = u2

    # inject data
    u1[:,n0] += dd[:,it]

    if 0 == it%jt:
        wave.write(u1)

# output image
imag.write(u1)
```

3. Figure 5 shows the Sigsbee velocity model and its an approximate filtered reflectivity.

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

   (a) Change directory

   ```
   cd hw6/sigsbee
   ```

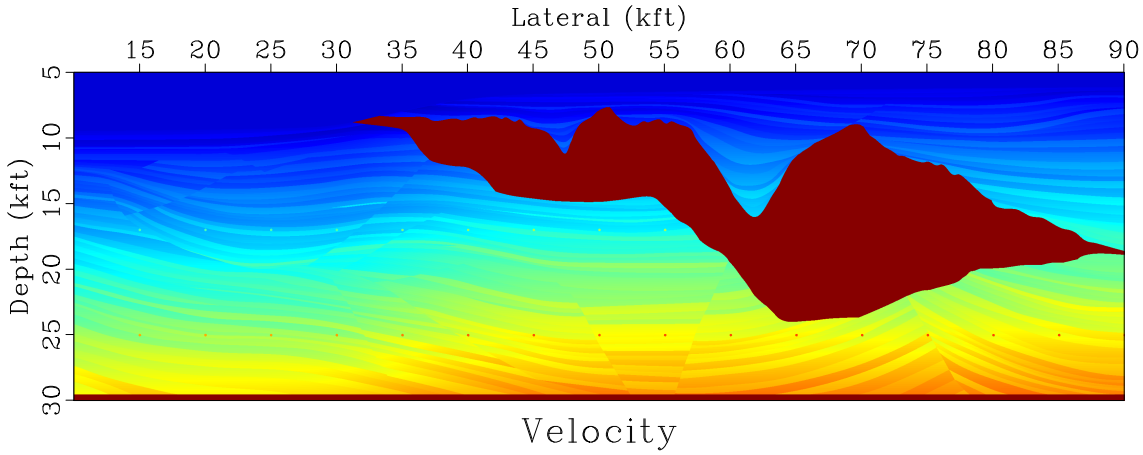
   (b) Run

   ```
   scons view
   ```

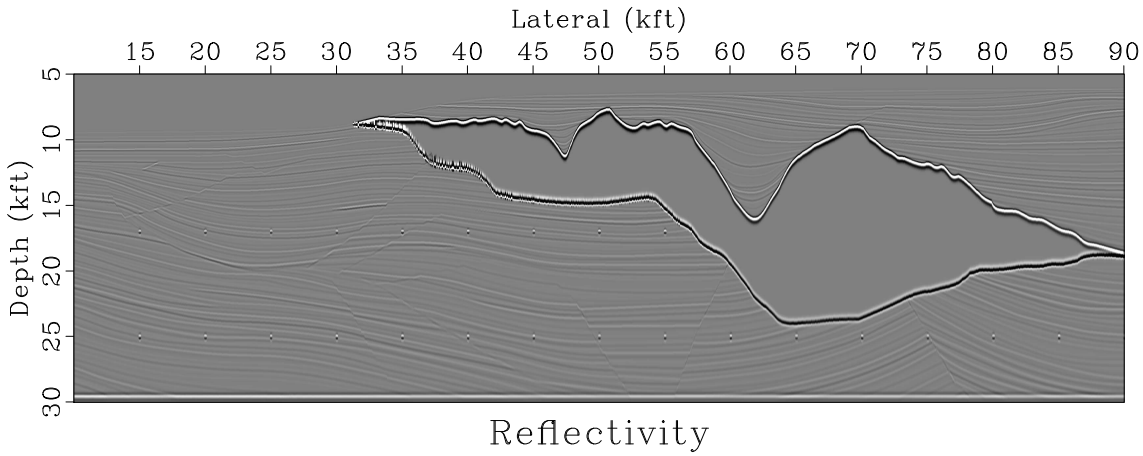
   to generate figures and display them on your screen.

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

   (d) Include your results in the paper by editing the `hw6/paper.tex` file.



(a)



(b)

Figure 5: (a) Sigsbee velocity model. (b) Approximate reflectivity of the Sigsbee model (an ideal image).

```
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()
```

4. We return to the synthetic model shown in Figure 6a to experiment with mod-

eling and migration by the phase-shift method in a $V(z)$ medium.

Figure 6b shows an idealized image (band-passed reflectivity) for the synthetic model. In "exploding-reflector" modeling, this image is assumed to be the seismic wavefield frozen at zero time. The modeling program extrapolates the wavefield to the surface and is implemented in `phaseshift.c`. The program operates in the frequency-wavenumber domain and establishes a linear relationship between reflectivity as a function of depth and zero-offset data as a function of frequency for one space wavenumber. If we think of this linear relationship as a matrix multiplication, we can associate forward modeling with multiplication

$$\mathbf{d} = \mathbf{A}\,\mathbf{m} \tag{3}$$

and migration with the adjoint multiplication

$$\widehat{\mathbf{m}} = \mathbf{A}^T\,\mathbf{d}\ , \tag{4}$$

where $\mathbf{A}^T$ is the conjugate transpose of $\mathbf{A}$.

Figure 7 shows the result of forward modeling (multiplication by $\mathbf{A}$) after inverse Fourier transform to time and space. Your task is to implement the corresponding migration (multiplication by $\mathbf{A}^T$).

Instead of simply applying the adjoint operator, we can also try to compute the *least-squares inverse*

$$\widehat{\mathbf{m}} = \left(\mathbf{A}^T\,\mathbf{A}\right)^{-1}\,\mathbf{A}^T\,\mathbf{d}\ , \tag{5}$$

which corresponds to the minimum of the least-squares misfit function $|\mathbf{A}\,\mathbf{m} - \mathbf{d}|^2$. In practice, inversion in equation (5) is implemented with an iterative *conjugate-gradient* algorithm, which applies $\mathbf{A}$ and $\mathbf{A}^T$ (modeling and migration) at each iteration without the need to form any matrices explicitly.

(a) Change directory

```
cd hw5/lsmig
```

(b) Run

```
scons view
```

to generate the figures and display them on your screen.

(c) Modify the program in the `phaseshift.c` file to fill the missing part and to implement phase-shift migration as the adjoint of phase-shift modeling.

(d) Modify the `SConstruct` file to uncomment the part related to migration. Check your result by running

```
scons migr.view
```

(e) Test if your migration code is truly the adjoint of modeling by running the dot-product test

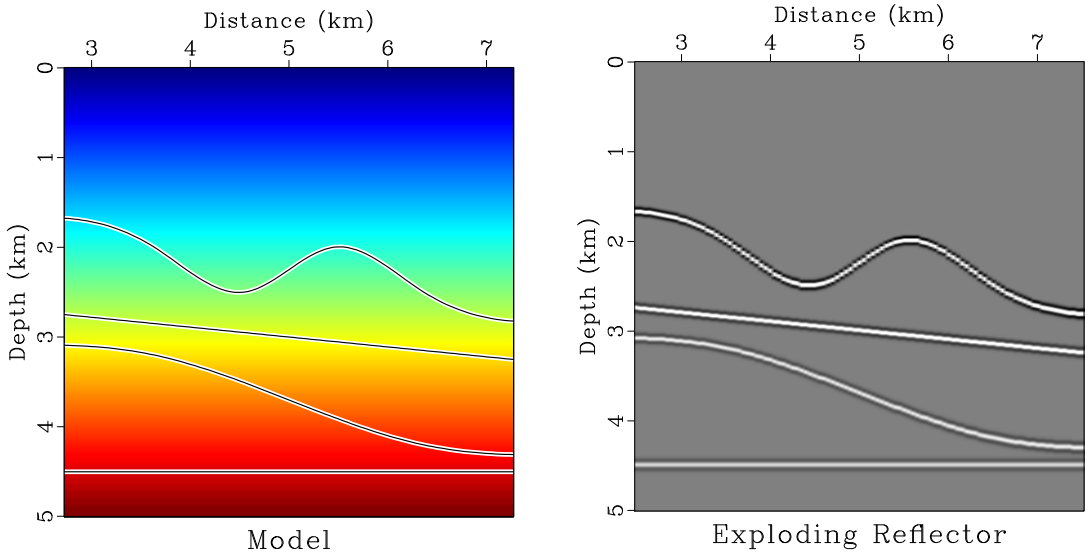


Figure 6: (a) Synthetic model: curved reflectors in a $V(z)$ velocity. (b) "Exploding reflector", an ideal image of band-passed reflectivity.

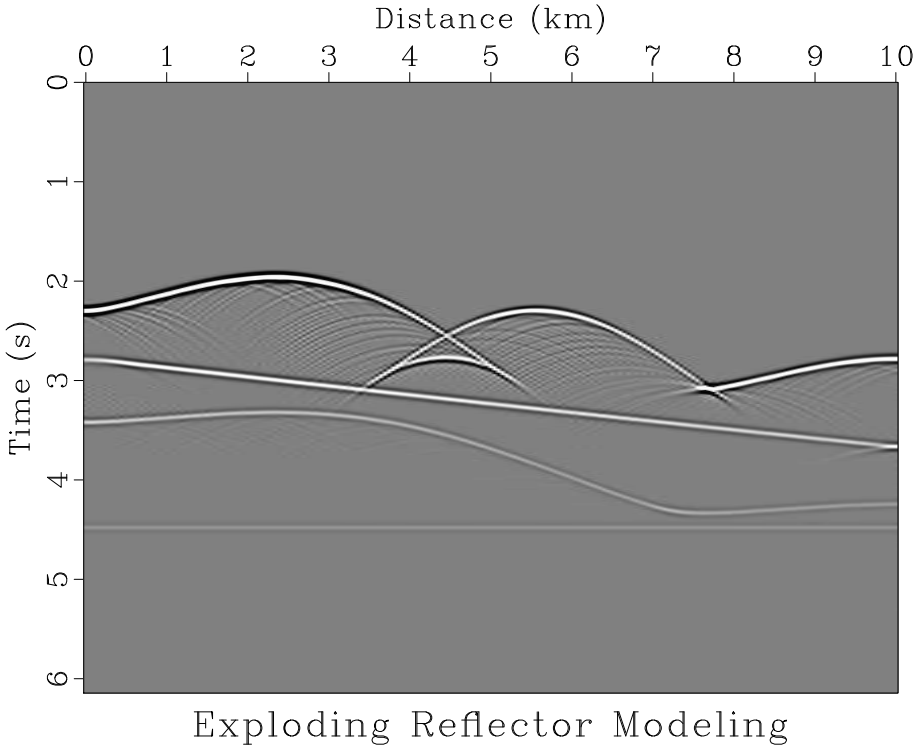


Figure 7: Zero-offset data generated by exploding-reflector phase-shift modeling.

```
sfcdottest ./phaseshift.exe mod=kexpl.rsf dat=kmodl.rsf \
vel=vofz.rsf nw=247 dw=0.16276
```

On a machine with multiple CPUs, you can also try

```
sfcdottest sfomp ./phaseshift.exe split=2 \
mod=kexpl.rsf dat=kmodl.rsf vel=vofz.rsf nw=247 dw=0.16276
```

If your adjoint is correct, you should see two identical sets of numbers, such as

```
sfcdottest:  L[m]*d=(25264,-20273.9)
sfcdottest: L'[d]*m=(25264,-20273.9)
```

Your actual numbers will be different because of random input vectors but they should be the same between `L[m]*d` and `L'[d]*m`.

(f) Now you are ready for testing least-squares migration. Uncomment the corresponding lines in `SConstruct` and run

```
scons invs.view
```

Do you notice any difference with the previous result? Increase the number of iterations from 10 to 100 and repeat the experiment.

(g) Include your figures in this document by editing `hw5/paper.tex`.

(h) For EXTRA CREDIT, use two-way wave extrapolation from the previous part to implement least-squares exploding-reflector reverse-time migration and apply it to the Sigsbee model.

```
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
```

```
20              data_format=ascii_float n1=2 n2=%d
21              ''' % \
22              (' '.join(map(lambda x: ' '.join(map(str,x)),
23                            layers[i])),len(layers[i])))
24
25  dim1 = 'o1=0 d1=0.05 n1=201'
26  Flow('lay1','inp0.asc',
27       'dd form=native | spline %s fp=0,0' % dim1)
28  Flow('lay2',None  ,
29       'math %s output="2.5+x1*0.1" '      % dim1)
30  Flow('lay3','inp2.asc',
31       'dd form=native | spline %s fp=0,0' % dim1)
32  Flow('lay4',None  ,'math %s output=4.5' % dim1)
33
34  Flow('lays','lay1 lay2 lay3 lay4',
35       'cat axis=2 ${SOURCES[1:4]}')
36
37  graph = '''
38  graph min1=2.5 max1=7.5 min2=0 max2=5
39  yreverse=y wantaxis=n wanttitle=n screenratio=1
40  '''
41  Plot('lays0','lays',graph + ' plotfat=10 plotcol=0')
42  Plot('lays1','lays',graph + ' plotfat=2 plotcol=7')
43  Plot('lays2','lays',graph + ' plotfat=2')
44
45  # Velocity
46
47  Flow('vofz',None,
48       '''
49       math output="1.5+0.25*x1"
50       d2=0.05 n2=201 d1=0.01 n1=501
51       label1=Depth unit1=km
52       label2=Distance unit2=km
53       label=Velocity unit=km/s
54       ''')
55  Plot('vofz',
56       '''
57       window min2=2.75 max2=7.25 |
58       grey color=j allpos=y bias=1.5
59       title=Model screenratio=1
60       ''')
61
62  Result('lmodel','vofz lays0 lays1','Overlay')
63
64  # Exploding reflector
```

```
65
66  Flow('expl','lays vofz',
67       '''
68       unif2 d1=0.01 n1=501 v00=1,2,3,4,5 |
69       depth2time velocity=${SOURCES[1]} |
70       ai2refl | ricker1 frequency=10 |
71       time2depth velocity=${SOURCES[1]} |
72       put label1=Depth unit1=km
73       label2=Distance unit2=km
74       ''')
75
76  Result('expl',
77         '''
78         window max1=5 min2=2.5 max2=7.5 |
79         grey title="Exploding Reflector" screenratio=1
80         ''')
81
82  # Modeling
83  ##########
84
85  proj = Project()
86  prog = proj.Program('phaseshift.c')
87
88  # Cosine Fourier transform
89  Flow('kexpl','expl','cosft sign2=1 | rtoc')
90
91  Flow('kmodl','kexpl %s vofz' % prog[0],
92       '''
93       ./${SOURCES[1]} vel=${SOURCES[2]}
94       nw=247 dw=0.16276
95       ''')
96
97  Flow('modl','kmodl',
98       'pad n1=769 | fft1 inv=y | cosft sign2=-1')
99
100 Result('modl',
101        '''
102        grey title="Exploding Reflector Modeling"
103        label1=Time unit1=s
104        ''')
105
106 # Migration
107 ###########
108
109 Flow('kdata','modl',
```

```
110       'cosft sign2=1 | fft1 | window n1=247')
111
112 Flow('kmigr','kdata %s vofz' % prog[0],
113       './${SOURCES[1]} vel=${SOURCES[2]} adj=1')
114
115 Flow('migr','kmigr','real | cosft sign2=-1')
116
117 # !!! UNCOMMENT BELOW !!!
118
119 #Result('migr',
120 #         '''
121 #         window max1=5 min2=2.5 max2=7.5 |
122 #         grey title="Exploding Reflector Migration"
123 #         screenratio=1 label1=Depth unit1=km
124 #         ''')
125
126 # Least-Squares Migration
127 ##########################
128
129 Flow('kinvs','kdata %s vofz kmigr' % prog[0],
130       '''
131       cconjgrad ./${SOURCES[1]} split=2
132       nw=247 dw=0.16276
133       vel=${SOURCES[2]} mod=${SOURCES[3]} niter=10
134       ''')
135
136 Flow('invs','kinvs','real | cosft sign2=-1')
137
138 # !!! UNCOMMENT BELOW !!!
139
140 #Result('invs',
141 #         '''
142 #         window max1=5 min2=2.5 max2=7.5 |
143 #         grey title="Exploding Reflector LS Migration"
144 #         screenratio=1 label1=Depth unit1=km
145 #         ''')
146
147
148 End()
```

```
1 /* Phase-shift modeling and migration */
2
3 #include <rsf.h>
4
5 int main(int argc, char* argv[])
```

```
6  {
7      bool adj;
8      int ik, iw, nk, nw, iz, nz;
9      float k, dk, k0, dw, dz, z0, *v, eps;
10     sf_complex *dat, *mod, w2, ps, wave;
11     sf_file inp, out, vel;
12
13     sf_init(argc,argv);
14
15     if (!sf_getbool("adj",&adj)) adj=false;
16     /* adjoint flag, 0: modeling, 1: migration */
17
18     inp = sf_input("in");
19     out = sf_output("out");
20     vel = sf_input("vel"); /* velocity file */
21
22     if (!sf_histint(vel,"n1",&nz))
23         sf_error("No n1= in vel");
24     if (!sf_histfloat(vel,"d1",&dz))
25         sf_error("No d1= in vel");
26     if (!sf_histfloat(vel,"o1",&z0)) z0=0.0;
27
28     if (!sf_histint(inp,"n2",&nk))
29         sf_error("No n2= in input");
30     if (!sf_histfloat(inp,"d2",&dk))
31         sf_error("No d2= in input");
32     if (!sf_histfloat(inp,"o2",&k0))
33         sf_error("No o2= in input");
34
35     dk *= 2*SF_PI;
36     k0 *= 2*SF_PI;
37
38     if (adj) { /* migration */
39         if (!sf_histint(inp,"n1",&nw))
40             sf_error("No n1= in input");
41         if (!sf_histfloat(inp,"d1",&dw))
42             sf_error("No d1= in input");
43
44         sf_putint(out,"n1",nz);
45         sf_putfloat(out,"d1",dz);
46         sf_putfloat(out,"o1",z0);
47     } else {  /* modeling */
48         if (!sf_getint("nw",&nw)) sf_error("No nw=");
49         if (!sf_getfloat("dw",&dw)) sf_error("No dw=");
50
```

```
51          sf_putint(out,"n1",nw);
52          sf_putfloat(out,"d1",dw);
53          sf_putfloat(out,"o1",0.0);
54      }
55
56      if (!sf_getfloat("eps",&eps)) eps = 1.0f;
57      /* stabilization parameter */
58
59      dw *= 2*SF_PI;
60
61      dat = sf_complexalloc(nw);
62      mod = sf_complexalloc(nz);
63
64      /* read velocity, convert to slowness squared */
65      v = sf_floatalloc(nz);
66      sf_floatread(v,nz,vel);
67      for (iz=0; iz < nz; iz++) {
68          v[iz] = 2.0f/v[iz];
69          v[iz] *= v[iz];
70      }
71
72      for (ik=0; ik<nk; ik++) { /* wavenumber */
73          sf_warning("wavenumber %d of %d;",ik+1,nk);
74          k=k0+ik*dk;
75          k *= k;
76
77          if (adj) {
78              sf_complexread(dat,nw,inp);
79              for (iz=0; iz < nz; iz++) {
80                  mod[iz]=sf_cmplx(0.0,0.0);
81              }
82          } else {
83              sf_complexread(mod,nz,inp);
84          }
85
86          for (iw=0; iw<nw; iw++) { /* frequency */
87              w2 = sf_cmplx(eps*dw,iw*dw);
88              w2 *= w2;
89
90              if (adj) { /* migration */
91                  wave = dat[iw];
92                  for (iz=0; iz < nz; iz++) {
93                      /* !!! FILL MISSING LINES !!! */
94                  }
95              } else {  /* modeling */
```

```
96                      wave = mod[nz-1];
97                      for (iz=nz-2; iz >=0; iz--) {
98                          ps = cexpf(-csqrt(w2*v[iz]+k)*dz);
99                          wave = wave*ps+mod[iz];
100                     }
101                     dat[iw] = wave;
102                 }
103             }
104
105             if (adj) {
106                 sf_complexwrite(mod,nz,out);
107             } else {
108                 sf_complexwrite(dat,nw,out);
109             }
110     }
111     sf_warning(".");
112
113     exit(0);
114 }
```

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.