Madagascar tutorial

Maurice the Aye-Aye

ABSTRACT

This tutorial takes you through different steps required for writing a report paper with reproducible examples. You will experiment with running Madagascar programs on the command line, using SCons to conduct numerical experiments, solving a simple research problem, and including experimental results in the written report.

PREREQUISITES

Completing this tutorial requires

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

To do the assignment on your personal computer, you need to install the required environments.

INTRODUCTION

In this tutorial, you will be asked to run commands from the Unix shell (identified by `bash$`) and to edit files in a text editor. Different editors are available in a typical Unix environment (`vi`, `emacs`, `gedit`, `nedit`, etc.)

Your first assignment:

1. Open a Unix shell.
2. Change directory to the tutorial directory

```
bash$ cd $RSFSRC/book/rsf/school2016
```

3. Open the `paper.tex` file in your favorite editor, for example by running

```
bash$ nedit paper.tex &
```

4. Look at the first line in the file and change the author name from Maurice the Aye-Aye to your name (first things first).

FIRST STEPS

You can skip this section if you are already familiar with Madagascar.

If these are your first experience, let us start with simple steps.

1. Create a new directory and change to it.

   ```
   bash$ mkdir first
   bash$ cd first
   ```

2. Let us create some data. Run

   ```
   bash$ sfmath n1=501 d1=0.004 output=1.5+x1 > vel.rsf
   ```

3. Run

   ```
   bash$ file vel.rsf
   ```

   to verify that `vel.rsf` is a ASCII text file. Examine the contents of the file by running

   ```
   bash$ cat vel.rsf
   ```

   Find a string starting with `in=`. It should point to the location of the binary file containing the actual data.

4. A more convenient way to check the file parameters is running

   ```
   bash$ sfin vel.rsf
   ```

   Verify that the reported binary file size in bytes corresponds to the number of elements multiplied by the size of one element (4 bytes for a single-precision floating-point number).

5. To inspect data elements one by one, run

   ```
   bash$ < vel.rsf sfdisfil | more
   ```

   What is the minimum and maximum value?

6. A more convenient way to check the data statistics is by running

   ```
   bash$ < vel.rsf sfattr
   ```

   Note that you can run any of the Madagascar programs (`sfmath`, `sfin`, `sfdisfil`, `sfattr`) on the command line without parameters to display their brief documentation. For examine, run

   ```
   bash$ sfattr
   ```

   to figure out what parameter to give to this program to display only the maximum value of the input data.

7. Now let us display the data on the screen. Run

   ```
   bash$ < vel.rsf sfgraph wanttitle=n \
   label1=Time unit1=s label2=Velocity unit2=km/s | sfpen
   ```

8. Suppose we want the time axis in the display to run vertically. Run

   ```
   bash$ sfgraph
   ```

   without parameters to figure out which parameter transposes the axes. Generate a new graph.

9. Running all commands on the command line can be tedious, especially for complex processing flows. The preferred way to control processing flows in Madagascar is by using the SCons utility. Create a file called `SConstruct` using your favorite editor, for example

   ```
   bash$ gedit SConstruct &
   ```

   Your first `SConstruct` may contain the following lines:

```
1 from rsf.proj import *
2
3 Flow('vel',None,'math n1=501 d1=0.004 output=1.5+x1')
4 Plot('vel','''
5 graph wanttitle=n
6 label1=Time unit1=s label2=Velocity unit2=km/s
7 ''')
```

   The `SConstruct` file is written in the Python language. The first line tells SCons to load all commands from the `rsf.proj` module, which contains the definitions of `Flow` and `Plot` commands. Note that Python uses triple quotes for strings that span multiple lines.

10. To see what commands SCons is going to execute without actually executing them, run

    ```
    bash$ scons -nQ
    ```

    Note that you can save the output of this command in a Shell script. Running processing flows with SCons scripts rather than Shell scripts is, however, a more powerful method, as we shall see next.

11. Run

    ```
    bash$ scons
    ```

    to build the targets specified in `SConstruct`: files `vel.rsf` and `vel.vpl`. The second file is a figure in Vplot binary format. You can display it on the screen by running

    ```
    bash$ sfpen vel.vpl
    ```

12. Edit the `SConstruct` file to modify the number of samples (`n1=` parameter.) Then run

    ```
    bash$ scons
    ```

    again to rebuild the targets.

13. Now edit the `SConstruct` file to change `wanttitle=n` parameter to give your figure a title, for example `title="My Velocity"`. Run

    ```
    bash$ scons
    ```

    and observe that, this time, SCons does not rebuild all the targets but only the target (`vel.vpl` file) affected by the parameter change. This behavior makes SCons particularly convenient for working with complex workflows and conducting experiments, which require repeated changes of parameters.

MAKING SYNTHETIC DATA

1. Change directory to `../synth`.

2. We will start by generating a synthetic CMP (common midpoint) gather using a linearly increasing RMS (root-mean-square) velocity, a random reflectivity, and a Ricker wavelet (Figure 1). To display this figure on your screen, run

    ```
    bash$ scons cmp.view
    ```

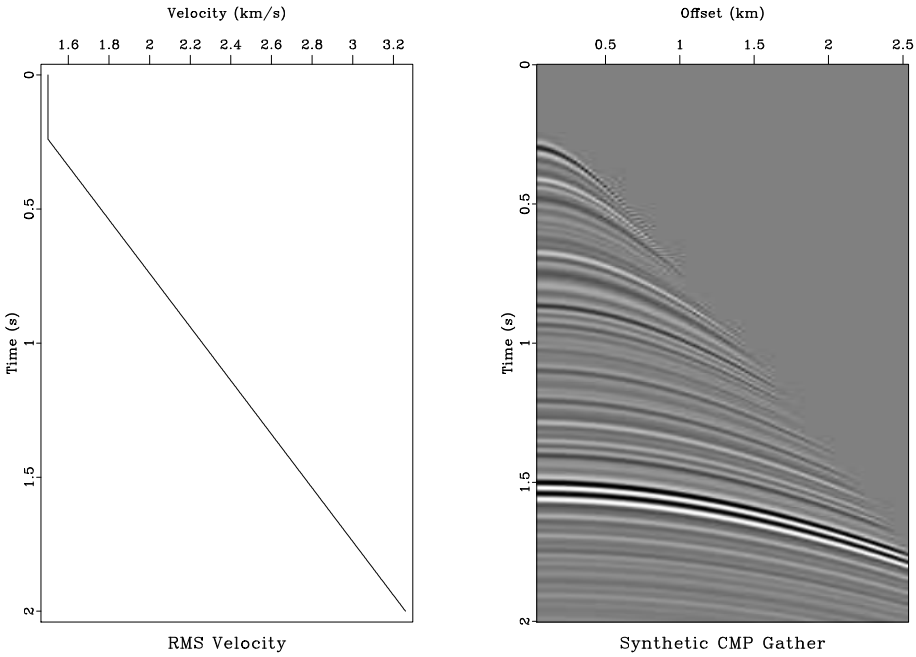


Figure 1: RMS velocity profile (left) and synthetic CMP gather (right).

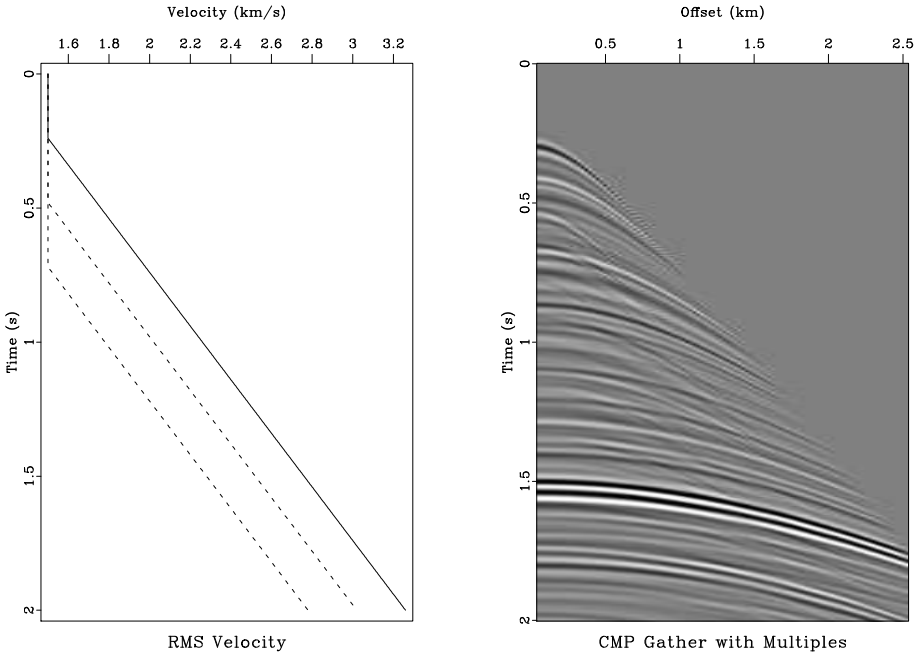


Figure 2: RMS velocity profiles for primaries and multiples (left) and synthetic multiple-infected CMP gather (right).

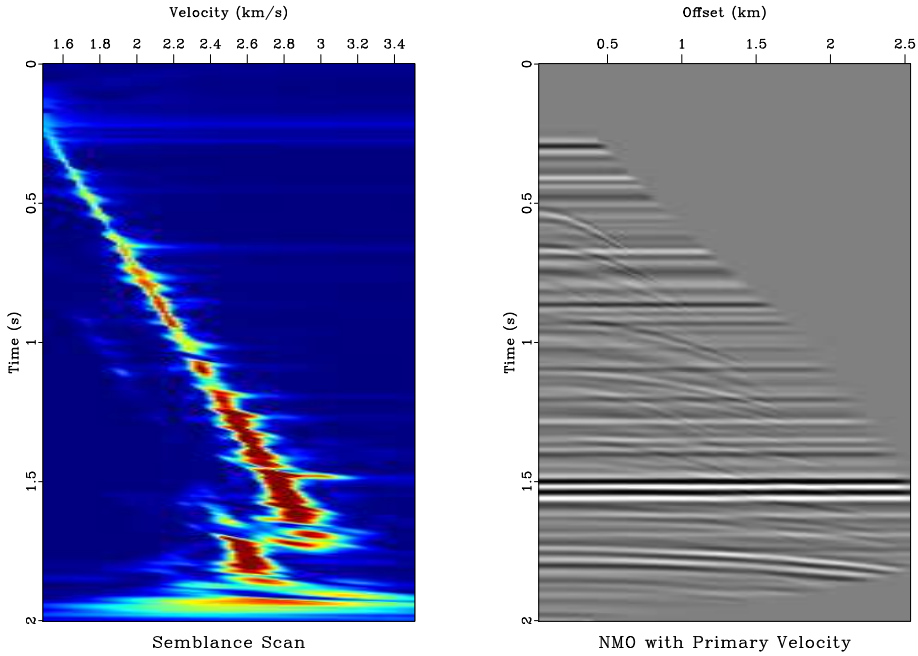


Figure 3: Semblance scan (left) and NMO with the primary velocity (right) applied to the synthetic CMP gather from Figure 2.

You can examine the `SConstruct` file to see the programs and parameter selections for generating these data. What does `sfinmo` do?

3. Next, we add some regular noise to the gather in the form of multiple reflections. In a rough approximation, the velocity trend of surface-related pegleg multiples follows that of the primary reflections but with a delay by the time of the water-bottom reflection (Figure 2.) To reproduce the figure on your screen, run

   ```
   bash$ scons cmp2.view
   ```

4. To better understand the effect of multiples on the data, we can examine the result of velocity analysis by semblance scanning and NMO (normal moveout) correction using the velocity of the primary reflection (Figure 3.) NMO flattens primary reflections and highlights multiple reflections as curved events. In processing field data, we may not know the primary velocity precisely and would need to estimate it.

5. **CHALLENGE 1:** The modeled synthetic data uses rough approximations. Your first challenge is to try making it more realistic or more challenging. You can modify the `SConstruct` file to change parameters (such as the water depth), to add noise (check out `sfnoise`), amplitude variations with offset, other kinds of multiples, etc.

synth/SConstruct

```
from rsf.proj import *

t0 = 0.24 # water depth in two-way time

# RMS velocity profile
Flow('vel',None,
     '''
     math n1=501 d1=0.004 output=1.5+x1-%g
     label1=Time unit1=s | clip2 lower=1.5
     ''' % t0)
Plot('vel',
     '''
     graph title="RMS Velocity" transp=y yreverse=y
     label2=Velocity unit2=km/s wheretitle=b wherexlabel=t
     ''')

# Synthetic CMP gather
Flow('trace','vel',
     '''
     noise rep=y seed=2016 | math output=input^3 |
     cut max1=%g | ricker1 frequency=20
     ''' % t0)
Flow('cmp','trace vel',
     '''
     spray axis=2 n=100 o=0.05 d=0.025 label=Offset unit=km |
     inmo half=n velocity=${SOURCES[1]} | mutter half=n v0=1.5
     ''')
Plot('cmp','grey title="Synthetic CMP Gather" ')

Result('cmp','vel cmp','SideBySideAniso')

# First pegleg multiple
Flow('vel1','vel',
     '''
     math n1=501 d1=0.004 output=1.5+x1-%g
     label1=Time unit1=s | clip2 lower=1.5
     ''' % (2*t0))
Flow('trace1','trace',
     '''
     pad beg1=%d | window n1=501 |
     scale dscale=-0.5
     ''' % int(t0/0.004))
Flow('mult1','trace1 vel1',
```

```
44        '''
45        spray axis=2 n=100 o=0.05 d=0.025 label=Offset unit=km |
46        inmo half=n velocity=${SOURCES[1]} | mutter half=n v0=1.5
47        ''')
48
49  # Second pegleg multiple
50  Flow('vel2','vel1',
51        '''
52       math n1=501 d1=0.004 output=1.5+x1-%g
53       label1=Time unit1=s | clip2 lower=1.5
54       ''' % (3*t0))
55  Flow('trace2','trace1',
56        '''
57        pad beg1=%d | window n1=501 |
58        scale dscale=-0.5
59        ''' % int(t0/0.004))
60  Flow('mult2','trace2 vel2',
61        '''
62        spray axis=2 n=100 o=0.05 d=0.025 label=Offset unit=km |
63        inmo half=n velocity=${SOURCES[1]} | mutter half=n v0=1.5
64        ''')
65  Plot('vel2','vel vel1 vel2',
66        '''
67        cat axis=2 ${SOURCES[1:3]} |
68        graph title="RMS Velocity" transp=y yreverse=y dash=0,1,1
69        label2=Velocity unit2=km/s wheretitle=b wherexlabel=t
70        ''')
71
72  Flow('cmp2','cmp mult1 mult2','add ${SOURCES[1:3]}')
73  Plot('cmp2','grey title="CMP Gather with Multiples" ')
74  Result('cmp2','vel2 cmp2','SideBySideAniso')
75
76  # Velocity analysis
77  Flow('vscan','cmp2',
78        'vscan half=n v0=1.5 nv=101 dv=0.02 semblance=y')
79  Plot('vscan','grey color=j allpos=y title="Semblance Scan" ')
80
81  Flow('nmo','cmp2 vel','nmo half=n velocity=${SOURCES[1]}')
82  Plot('nmo','grey title="NMO with Primary Velocity" ')
83
84  Result('nmo','vscan nmo','SideBySideAniso')
85
86  End()
```

SEPARATING PRIMARIES AND MULTIPLES USING HYPERBOLIC RADON TRANSFORM

Next, we will apply the hyperbolic Radon transform (also known as the velocity stack or the velocity transform) to our modeled CMP gather in order to separate primaries and multiples. As suggested by Thorson and Claerbout (1985), the hyperbolic Radon transform can be made invertible by employing an iterative least-squares inversion.

1. Change directory to `../radon`.

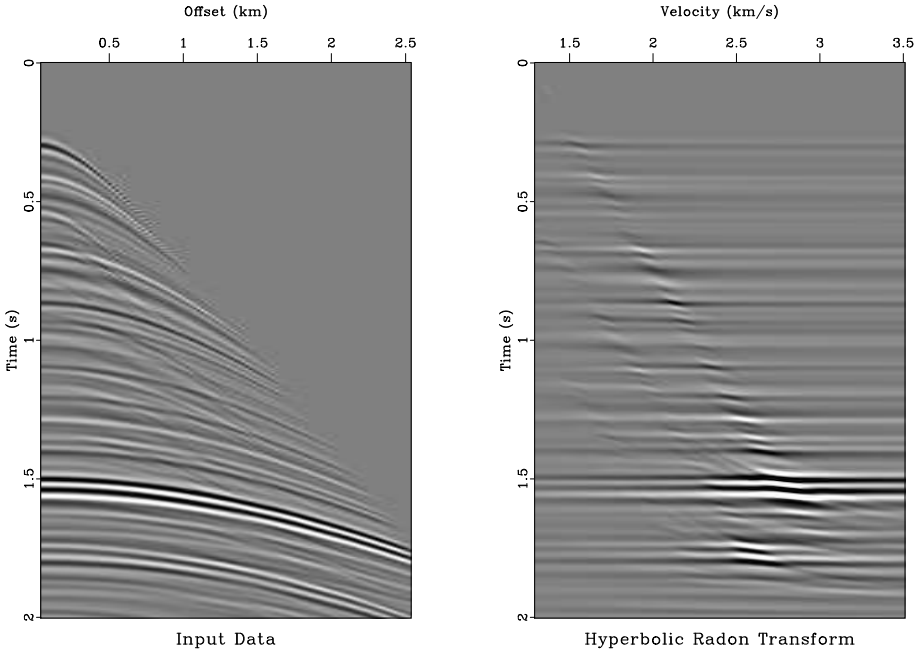


Figure 4: Synthetic CMP gather (left) its hyperbolic Radon transform (right).

2. We will use the same CMP gather as in the previous section. Figure 4 shows the selected gather and its hyperbolic Radon transform. To display it on your screen, run

   ```
   bash$ scons radon.view
   ```

3. The program for implementing the hyperbolic Radon transform is included in this directory and is called `hradon.c`. The program has an adjoint flag (`adj=`) to switch between the forward and adjoint operations. The forward operation (`adj=no`) transforms from the Radon time-velocity domain to the CMP time-offset domain. The adjoint operation (`adj=yes`) transforms in the opposite direction.

   To test if the adjoint operator is implemented correctly in this program, you can run the dot-product test

```
bash$ sfdottest ./hradon.exe mod=radon.rsf dat=cmp2.rsf \
ov=1.3 dv=0.02 nv=111 ox=0.05 dx=0.025 nx=100
```

The output should display a pair of numbers equal to six digits of accuracy. You can run the test multiple times.

The result of going back from the Radon domain to the CMP domain using the adjoint operation is shown in Figure 5a. To display it on your screen, run

```
bash$ scons adj.view
```

4. As is evident from the right plot in Figure 5a (the difference between the predicted and the original data), the adjoint operation does not reconstruct the data exactly. To achieve a better reconstruction, we can run iterative least-squares inversion using the method of conjugate gradients. The result is shown in Figure 5b. To display it on your screen, run

   ```
   bash$ scons inv.view
   ```

   Notice that only 10 conjugate-gradient iterations are sufficient to achieve a reasonable reconstruction.

5. **CHALLENGE 2:** Can you make this process faster? The computational cost of the method is proportional to the number of conjugate-gradient iterations. Can we accelerate the convergence to achieve the same result with less iterations?

   Open the `hradon.c` program in an editor and uncomment the lines containing calls to `halfint_lop` (the half-order derivative filter for correcting the wavelet shape). Replace `XXXX` with `true` or `false` as appropriate and comment out extra calls to `sf_floatread` and `sf_floatwrite`. To check if you did it correctly, run the dot-product test

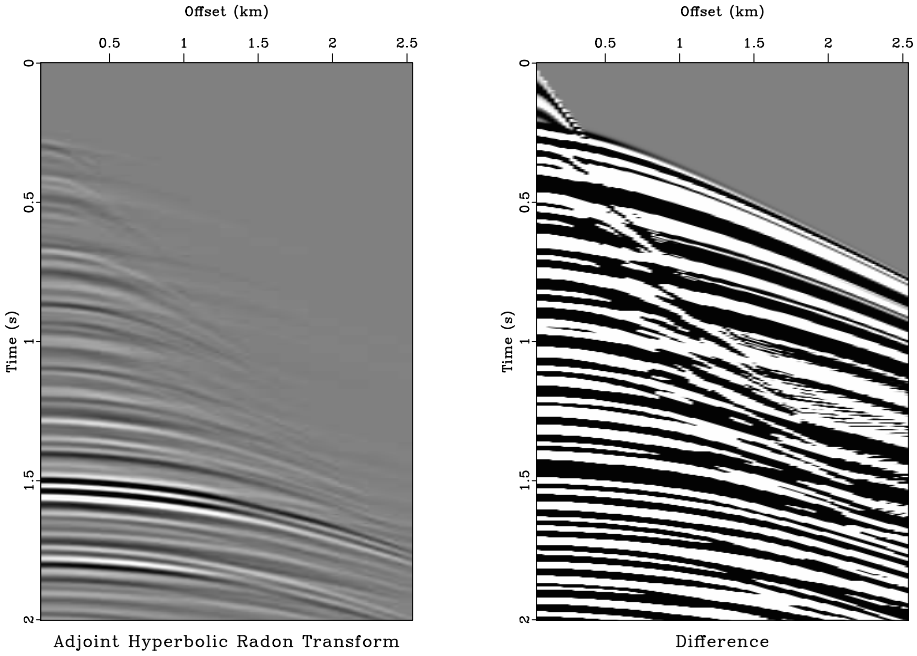
   ```
   bash$ sfdottest ./hradon.exe mod=radon.rsf dat=cmp2.rsf \
   ov=1.3 dv=0.02 nv=111 ox=0.05 dx=0.025 nx=100
   ```

   Did including the half-order derivative filter improve the convergence of conjugate gradients?

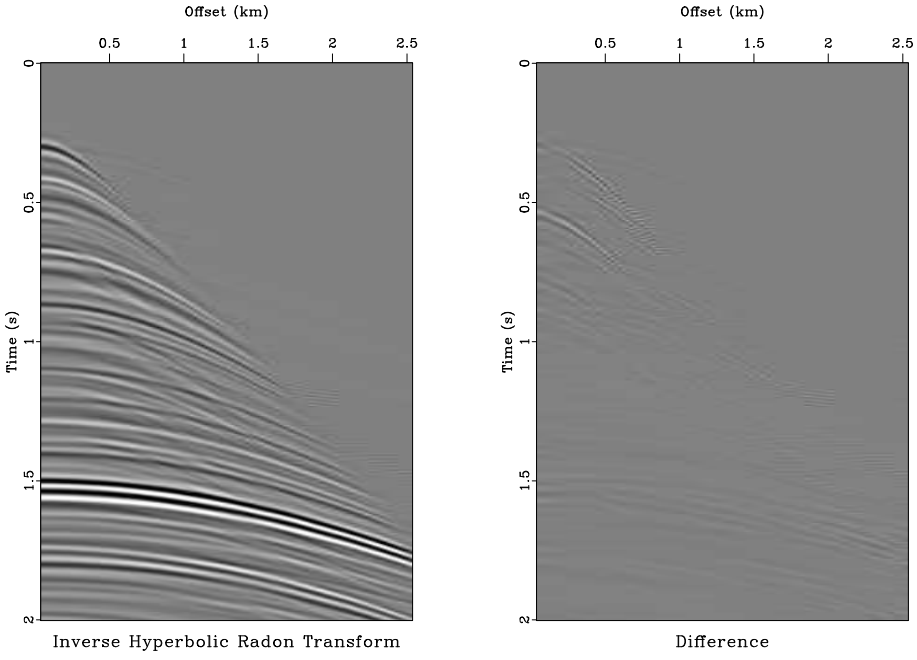
   You can try to achieve a further acceleration using preconditioning by diagonal weighting. Appropriate weights can be found by experimentation or by using the asymptotic theory (Fomel, 2003).

6. Separating primaries and multiples in the Radon domain amounts to muting (Figure 6.) To display the muting result and its inverse transform to the CMP space, run

   ```
   scons mute.view mult.view
   ```



(a)



(b)

Figure 5: Synthetic CMP gather reconstructed from the inverse transform (left) and its difference wit the original gather (right). Top: using adjoint transform, bottom: using 10 iterations of iterative least-squares inversion.

Verify that the separated primaries and multiples sum to the original CMP gather (check out the `sfadd` program.)

To confirm the separation, we can also run the semblance scan on the separated results (Figure 7b.) To display it on your screen, run

```
scons vscan.view
```

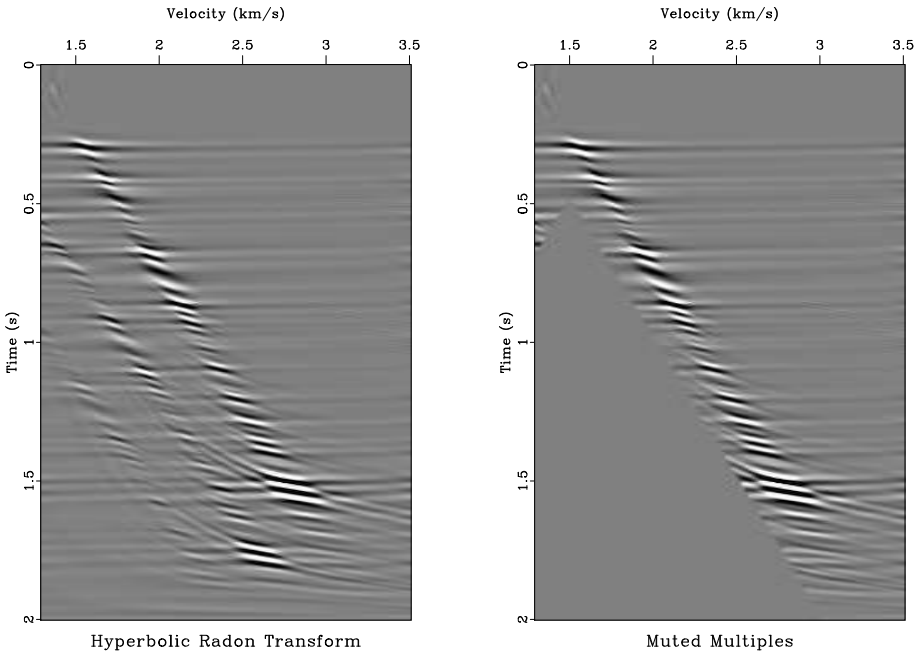


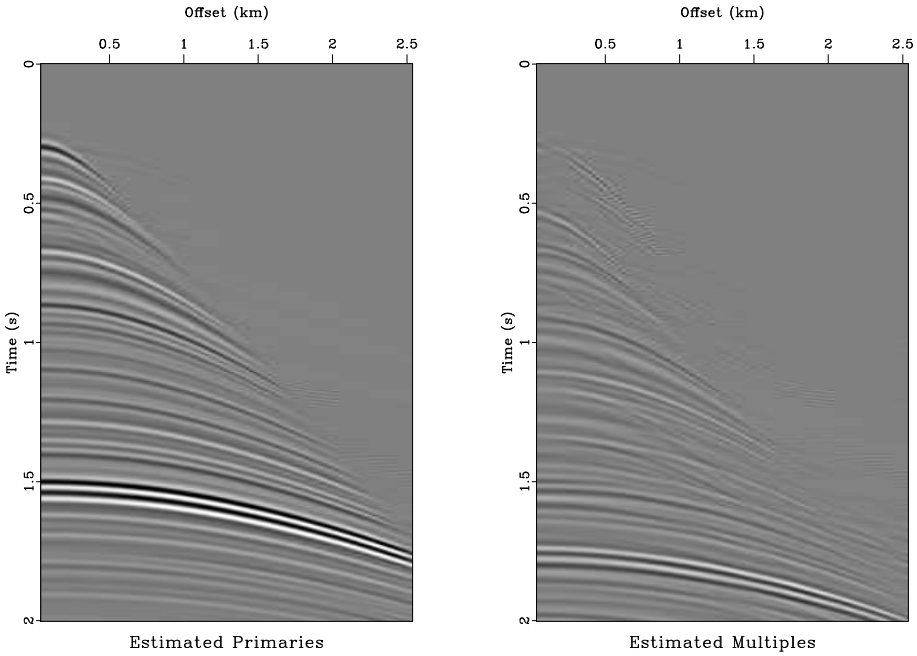
Figure 6: Isolating signal in the hyperbolic Radon domain (left) by muting (right).

7. **CHALLENGE 3:** Figure 7 shows some energy leakage between the estimated primaries and multiples. Can you improve the signal localization in the Radon domain for better separation? Modify the `SConstruct` file to implement iterative inversion using sparsity regularization to achieve a sparser output.

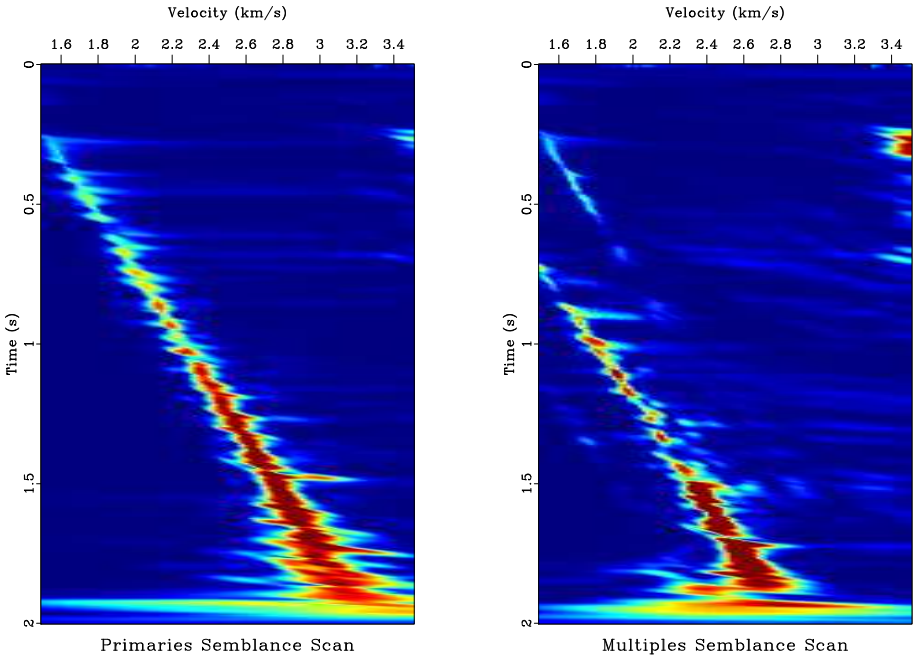
radon/hradon.c

```
/* Hyperbolic Radon transform */
#include <rsf.h>

int main(int argc, char* argv[])
{
    sf_map4 map;
    int it,nt, ix,nx, iv,nv, i3,n3;
    bool adj;
    float t0,dt,t, x0,dx,x, v0,dv,v;
```



(a)



(b)

Figure 7: (a) Separated primaries and multiples. (b) Velocity analysis using semblance scan applied to separated primaries and multiples.

```
10      float **cmp, **rad, *trace;
11      sf_file in, out;
12
13      /* initialize Madagascae */
14      sf_init(argc,argv);
15
16      /* input and output files */
17      in = sf_input("in");
18      out = sf_output("out");
19
20      /* Time axis parameters from the input */
21      if (!sf_histint(in,"n1",&nt)) sf_error("No n1=");
22      if (!sf_histfloat(in,"o1",&t0)) sf_error("No o1=");
23      if (!sf_histfloat(in,"d1",&dt)) sf_error("No d1=");
24
25      /* number of CMPS */
26      n3 = sf_leftsize(in,2);
27
28      if (!sf_getbool("adj",&adj)) adj=false;
29      /* adjoint flag */
30
31      if (adj) {
32          if (!sf_histint(in,"n2",&nx))    sf_error("No n2=");
33          if (!sf_histfloat(in,"o2",&x0)) sf_error("No o2=");
34          if (!sf_histfloat(in,"d2",&dx)) sf_error("No d2=");
35
36          if (!sf_getint("nv",&nv))    sf_error("Need nv=");
37          if (!sf_getfloat("ov",&v0)) sf_error("need ov=");
38          if (!sf_getfloat("dv",&dv)) sf_error("need dv=");
39
40          sf_putint(out,"n2",nv);
41          sf_putfloat(out,"o2",v0);
42          sf_putfloat(out,"d2",dv);
43          sf_putstring(out,"label2","Velocity");
44          sf_putstring(out,"unit2","km/s");
45      } else {
46          if (!sf_histint(in,"n2",&nv))    sf_error("No n2=");
47          if (!sf_histfloat(in,"o2",&v0)) sf_error("No o2=");
48          if (!sf_histfloat(in,"d2",&dv)) sf_error("No d2=");
49
50          if (!sf_getint("nx",&nx))    sf_error("Need nx=");
51          if (!sf_getfloat("ox",&x0)) sf_error("need ox=");
52          if (!sf_getfloat("dx",&dx)) sf_error("need dx=");
53
54          sf_putint(out,"n2",nx);
```

```
55          sf_putfloat(out,"o2",x0);
56          sf_putfloat(out,"d2",dx);
57          sf_putstring(out,"label2","Offset");
58          sf_putstring(out,"unit2","km");
59      }
60
61      /* allocate storage */
62      trace = sf_floatalloc(nt);
63      rad = sf_floatalloc2(nt,nv);
64      cmp = sf_floatalloc2(nt,nx);
65
66
67      /* initialize half-order differentiation */
68      sf_halfint_init (true,nt,1.0f-1.0f/nt);
69
70      /* initialize spline interpolation */
71      map = sf_stretch4_init (nt, t0, dt, nt, 0.01);
72
73      for (i3=0; i3 < n3; i3++) {
74          if( adj) {
75  /*
76              for (ix=0; ix < nx; ix++) {
77                  sf_floatread(trace,nt,in);
78                  sf_halfint_lop(XXXX,false,nt,nt,cmp[ix],trace);
79              }
80  */
81              sf_floatread(cmp[0],nt*nx,in);
82          } else {
83              sf_floatread(rad[0],nt*nv,in);
84          }
85
86          /* zero output */
87          sf_adjnull(adj,false,nt*nv,nt*nx,rad[0],cmp[0]);
88
89          for (iv=0; iv < nv; iv++) {
90              v = v0 + iv*dv;
91              for (ix=0; ix < nx; ix++) {
92                  x = (x0 + ix*dx)/v;
93
94                  for (it=0; it < nt; it++) {
95                      t = t0 + it*dt;
96                      trace[it] = hypotf(t,x);
97                      /* hypot(a,b)=sqrt(a*a+b*b) */
98                  }
99
```

```
100                     /* define mapping */
101                     sf_stretch4_define(map,trace);
102
103                     if (adj) {
104                         sf_stretch4_apply_adj(true,map,rad[iv],cmp[ix]);
105                     } else {
106                         sf_stretch4_apply     (true,map,rad[iv],cmp[ix]);
107                     }
108                 }
109             }
110
111             if (adj) {
112                 sf_floatwrite(rad[0],nt*nv,out);
113             } else {
114                 sf_floatwrite(cmp[0],nt*nx,out);
115 /*
116                 for (ix=0; ix < nx; ix++) {
117                     sf_halfint_lop(XXXX,false,nt,nt,cmp[ix],trace);
118                     sf_floatwrite(trace,nt,out);
119                 }
120 */
121             }
122         }
123
124         exit(0);
125 }
```

radon/SConstruct

```
1  from rsf.proj import *
2
3  # Use cmp2 data from the previous project
4  Fetch('cmp2.rsf','synth',top='..',server='local')
5  Plot('cmp2','grey title="Input Data" clip=6')
6
7  # Compile hyperbolic Radon program
8  prog = Program('hradon.c')
9  hradon = str(prog[0])
10
11 # Hyperbolic Radon using adjoint
12 ###################################
13 Flow('radon',['cmp2',hradon],
14       '${SOURCES[1].abspath} adj=y ov=1.3 dv=0.02 nv=111')
15 Plot('radon',
16       'grey title="Hyperbolic Radon Transform" ')
```

```
17
18 Result('radon','cmp2 radon','SideBySideAniso')
19
20 Flow('adj',['radon',hradon],
21       '${SOURCES[1].abspath} ox=0.05 dx=0.025 nx=100')
22 Plot('adj',
23       'grey title="Adjoint Hyperbolic Radon Transform" ')
24
25 Flow('adiff','cmp2 adj','add scale=1,-1 ${SOURCES[1]}')
26 Plot('adiff','grey title=Difference clip=6')
27
28 Result('adj','adj adiff','SideBySideAniso')
29
30 # Hyperbolic Radon using least-squares inversion
31 ################################################
32 Flow('radon2',['cmp2',hradon,'radon'],
33       '''
34       conjgrad ${SOURCES[1].abspath} mod=${SOURCES[2]}
35       ov=1.3 dv=0.02 nv=111 ox=0.05 dx=0.025 nx=100 niter=10
36       ''')
37 Plot('radon2',
38       'grey title="Hyperbolic Radon Transform" ')
39
40 Flow('inv',['radon2',hradon],
41       '${SOURCES[1].abspath} ox=0.05 dx=0.025 nx=100')
42 Plot('inv',
43       'grey title="Inverse Hyperbolic Radon Transform" clip=6')
44
45 Flow('idiff','cmp2 inv','add scale=1,-1 ${SOURCES[1]}')
46 Plot('idiff','grey title=Difference clip=6')
47
48 Result('inv','inv idiff','SideBySideAniso')
49
50 # Separating primaries and multiples
51 ####################################
52 Flow('mute','radon2',
53       'mutter half=n t0=0.48 x0=1.5 v0=1 inner=y')
54 Plot('mute','grey title="Muted Multiples" ')
55
56 Result('mute','radon2 mute','SideBySideAniso')
57
58 Flow('prim',['mute',hradon],
59       '${SOURCES[1].abspath} ox=0.05 dx=0.025 nx=100')
60 Plot('prim',
61       'grey title="Estimated Primaries" clip=6')
```

```
62
63 Flow('mult','cmp2 prim','add scale=1,-1 ${SOURCES[1]}')
64 Plot('mult','grey title="Estimated Multiples" clip=6')
65
66 Result('mult','prim mult','SideBySideAniso')
67
68 # Velocity analysis
69 ####################
70 Flow('pvscan','prim',
71      'vscan half=n v0=1.5 nv=101 dv=0.02 semblance=y')
72 Plot('pvscan',
73      '''
74      grey color=j allpos=y title="Primaries Semblance Scan"
75      ''')
76
77 Flow('mvscan','mult',
78      'vscan half=n v0=1.5 nv=101 dv=0.02 semblance=y')
79 Plot('mvscan',
80      '''
81      grey color=j allpos=y title="Multiples Semblance Scan"
82      ''')
83
84 Result('vscan','pvscan mvscan','SideBySideAniso')
85
86 End()
```

WRITING A REPORT

1. Change directory to the parent directory

   ```
   bash$ cd ..
   ```

   This should be the directory which contains `paper.tex`.

2. Run

   ```
   bash$ sftour scons lock
   ```

   The `sftour` command visits all subdirectories and runs `scons lock`, which copies result files to a different location so that they will not get modified until further notice.

3. You can also run

   ```
   bash$ sftour scons -c
   ```

to clean intermediate results.

4. Edit the file `paper.tex` to include your additional results. If you have not used LaTeX before, no worries. LaTeX is a descriptive language. Study the file, and it should become evident by example how to include figures.

5. Run

   ```
   bash$ scons read
   ```

   to generate a PDF report `paper.pdf` and open it with a PDF viewing program.

6. Want to submit your paper to *Geophysics*? Edit `SConstruct` in the paper directory to add `options=manuscript` to the `End` statement. Then run

   ```
   bash$ scons read
   ```

   again.

7. If you have LaTeX2HTML installed, you can also generate an HTML version of your paper by running

   ```
   bash$ scons html
   ```

   and opening `paper_html/index.html` in a web browser.

REFERENCES

Fomel, S., 2003, Asymptotic pseudounitary stacking operators: Geophysics, **68**, 1032–1042.

Thorson, J. R., and J. F. Claerbout, 1985, Velocity stack and slant stochastic inversion: Geophysics, **50**, 2727–2741.