GEO 365N/384S Seismic Data Processing
Computational Assignment 6

Team: Longhorns

ABSTRACT

In this assignment, you will experiment with different kinds of depth migration:

1. Post-stack phase-shift (Gazdag) migration and its approximation by Stolt stretch.
2. Prestack Kirchhoff migration.
3. Post-stack reverse-time migration (RTM).

Migration will be applied to previously processed datasets: Viking Graben and Teapot Dome data.

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`

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

To setup the Madagascar environment in the JGB 3.216B computer lab, run the following commands:

```
module load madagascar-devel
source $RSFROOT/share/madagascar/etc/env.csh
setenv DATAPATH $HOME/data/
setenv RSFBOOK $HOME/data/book
setenv RSFFIGS $HOME/data/figs
```

You can put these commands in your `$HOME/.cshrc` file to run them automatically at login time.

To setup the LaTeX environment, run the following commands:

```
cd
git clone https://github.com/SEGTeX/texmf.git
texhash
```

You only need to do it once.

The homework code is available from the class repository by running

```
svn co https://github.com/GEO384S/geo384s/trunk/hw6
```

You can also download it from your team's private repository.

GENERATING THIS DOCUMENT

At any point of doing this computational assignment, you can regenerate this document and display it on your screen.

1. Change directory to `hw6`:

   ```
   cd hw6
   ```

2. Run

   ```
   sftour scons lock
   scons read &
   ```

As the first step, open `hw6/paper.tex` file in your favorite editor and edit the first line to enter the name of your team. Then run `scons read` again.

PHASE-SHIFT MIGRATION

In the first part of the assignment, we will return to processing 3D land data, the Teapot Dome dataset.

1. Change directory to `hw6/gazdag`.

2. Run

   ```
   scons -c
   ```

   to remove (clean) previously generated files.

3. To simplify the processing, we will use a single velocity function assuming a laterally homogeneous medium. This velocity can be picked from a supergather combining traces from different locations. Figure 1 shows the semblance scan using every 500th trace in the data and the corresponding picked NMO velocity trend. To reproduce it on your screen, run

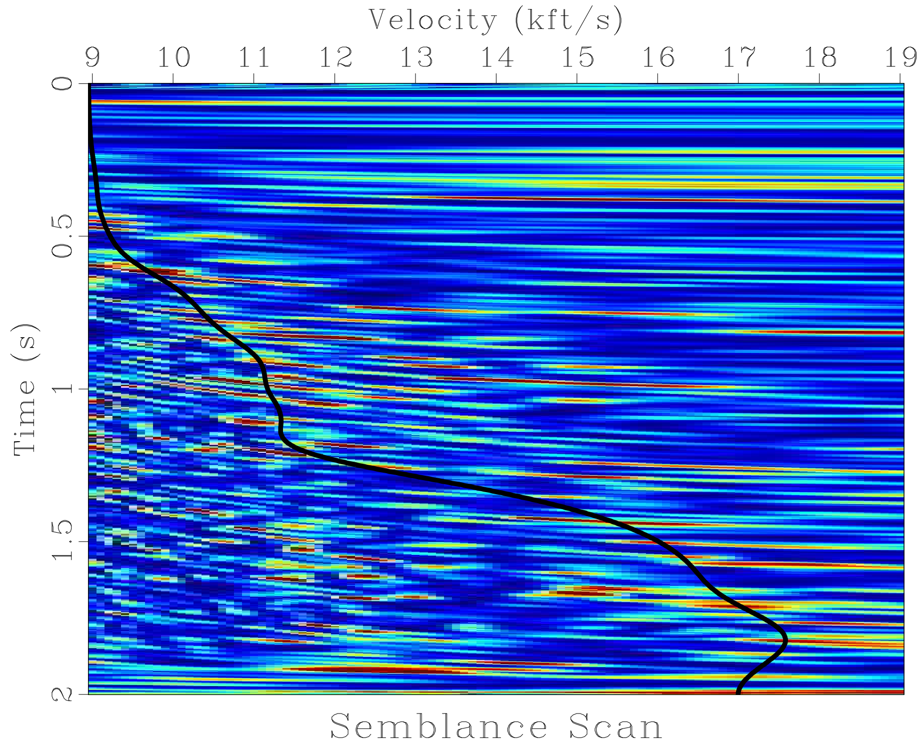


Figure 1: Semblance scan of a supergather (every 500th trace) from the Teapot Dome dataset. The curve shows the picked velocity trend. gazdag/ vscan

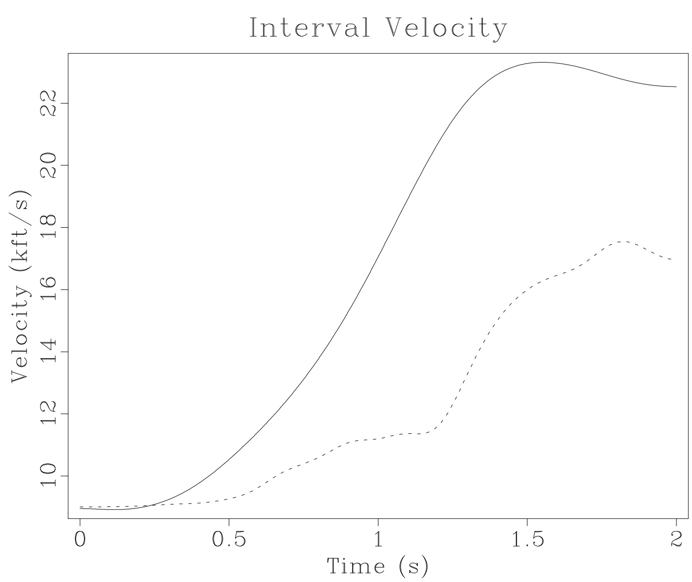


Figure 2: Interval velocity in the Teapot Dome dataset estimated by Dix inversion (regularized by smoothing). The dashed curve shows the corresponding picked RMS velocity. gazdag/ velocity

```
scons vscan.view
```

4. In the next step, we will treat the NMO velocity as the RMS velocity and convert it to interval velocity using Dix inversion (Dix, 1955). To display the result (Figure 2), run

   ```
   scons velocity.view
   ```

   To avoid instabilities, Dix inversion is assisted by smoothing regularization.

5. Using the picked NMO velocity, we can apply NMO stacking to create a 3D stacked cube (Figure 3). To display it on your screen, run

   ```
   scons stack.view
   ```

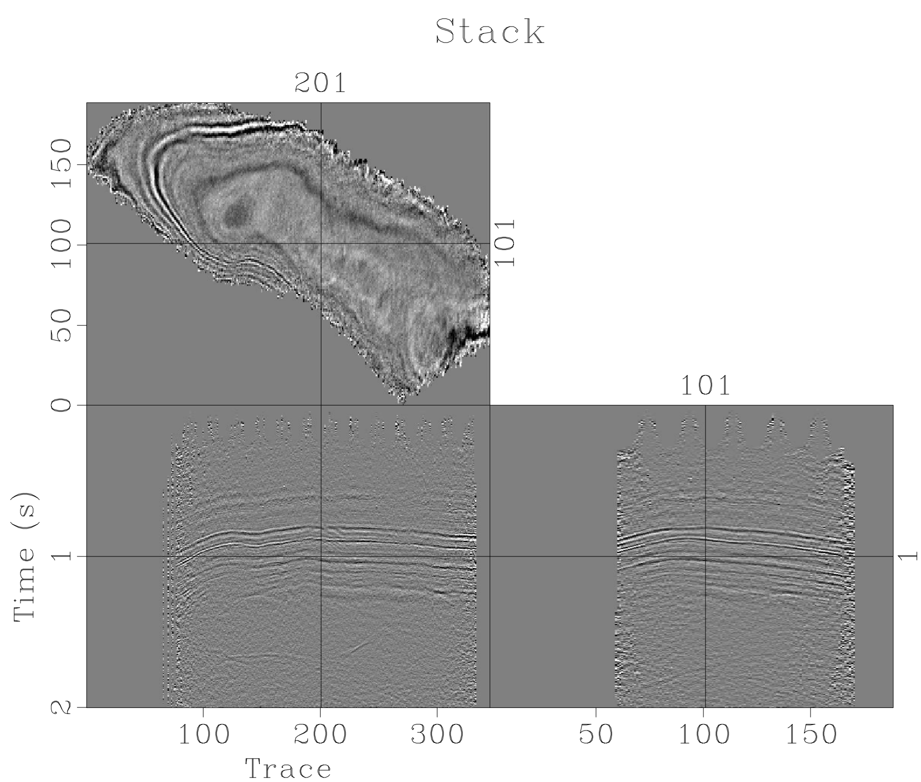


Figure 3: Stack of the Teapot Dome dataset. gazdag/ stack

6. The stack in Figure 3 covers a region with an irregular shape. For further processing, we will select a portion of the data and rotate it to align with the axes (Figure 4a). To perform windowing and rotation, run

   ```
   scons stack2.view
   ```

7. A depth migration method designed specifically for laterally-invariant $V(z)$ velocity distributions is phase-shift migration, also known as Gazdag migration (Gazdag, 1978). The phase-shift method extrapolates the recorded wavefield in depth.

Applying the Fourier transform in time and in lateral spatial directions to the wave equation turns it into an ordinary differential equation

$$\frac{d^2\hat{U}}{dz^2} + \lambda^2\,\hat{U} = 0\;, \tag{1}$$

where $U(\omega, \mathbf{k}, z)$ is the wavefield corresponding to temporal frequency ω, lateral wavenumber $\mathbf{k}$, and depth z, and

$$\lambda^2 = \frac{\omega^2}{V^2(z)} - \mathbf{k}\cdot\mathbf{k}$$

Equation (1) admits a depth-stepping solution

$$\hat{U}(z+\Delta z, \mathbf{k}, \omega) = \hat{U}(z, \mathbf{k}, \omega)\,e^{\pm i\,\lambda\,\Delta z}\;, \tag{2}$$

which is the basis of the phase-shift method.

To generate a 3D phase-shift migration of the stack in Figure 3, run

```
scons image.view
```

The computation proceeds in the vertical time coordinates, with the result displayed on the same grid as the input. To compare the data before and after migration, run

```
sfpen Fig/stack2.vpl Fig/image.vpl
```

Do you observe notable differences?

8. Rewrite equations (1) and (2) using the vertical time coordinate

$$\tau = \int\limits_0^z \frac{2\,d\zeta}{V(\zeta)}$$

instead of z.

Answer:

Which sign (plus or minus) should be used in equation (2) for post-stack migration?

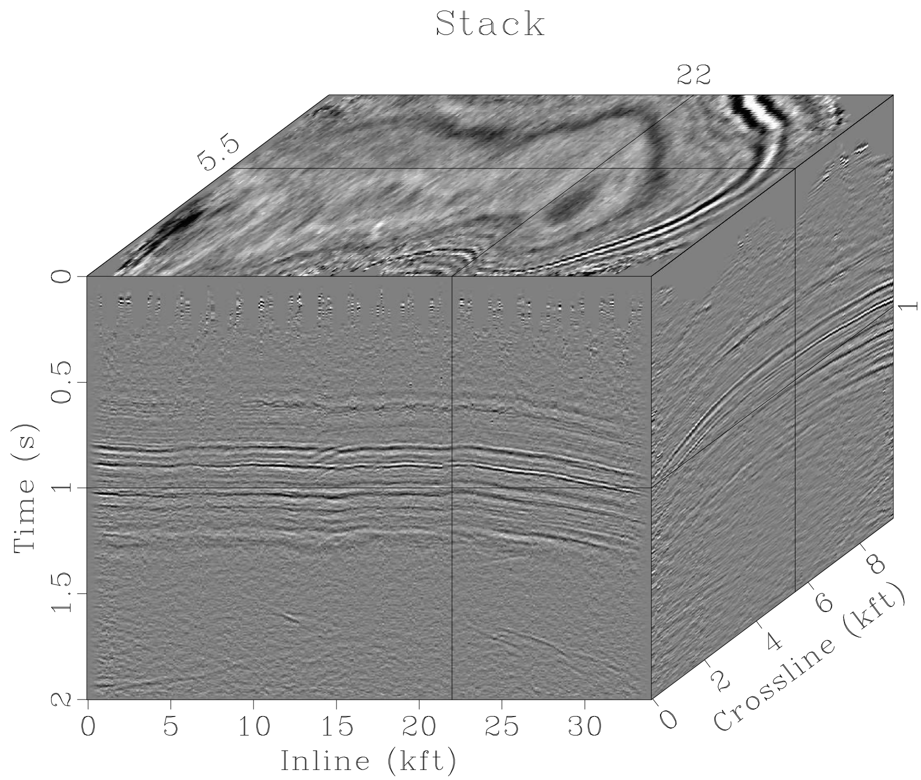
9. In the classic paper, Stolt (1985) proposed not only a Fourier-domain method for a constant-velocity migration but also an effective method for approximating the output of phase-shift migration in a $V(z)$ medium. The method became known as *Stolt stretch*. It consists of three steps:

   (a) Stretching the data in time by transforming from time t to stretched time t_s according to

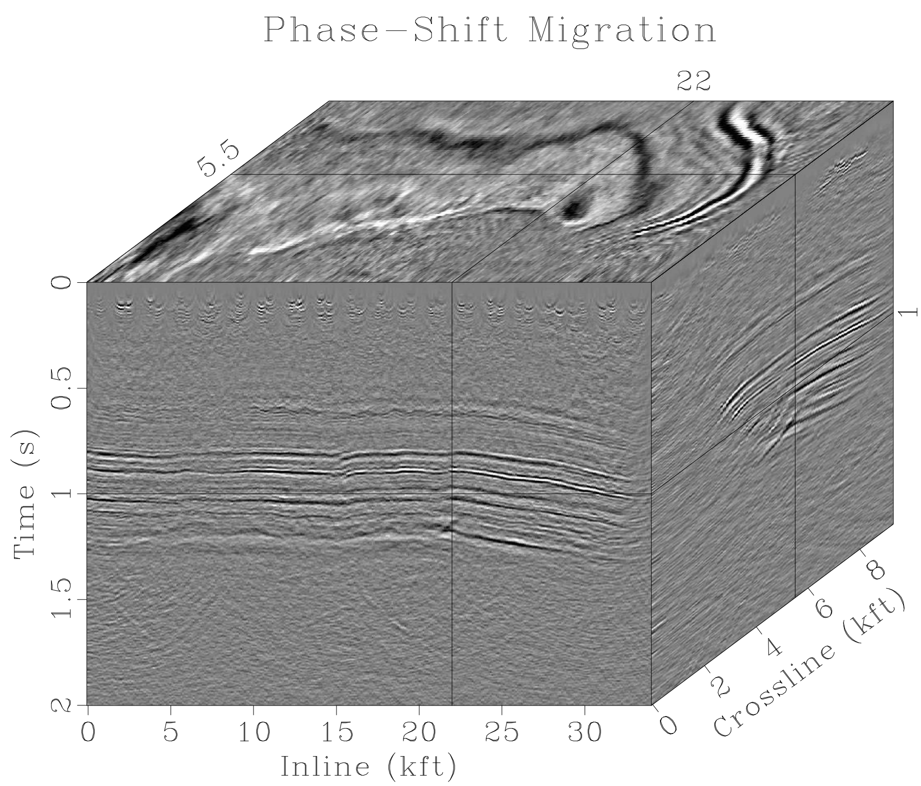
$$t_s(t) = \left(\frac{2}{V_0^2}\int\limits_0^t \tau\,V_r(\tau)d\tau\right)^{1/2}\;, \tag{3}$$

   where V_0 is a reference constant velocity, and $V_r(t)$ is the RMS velocity:

$$V_r(t) = \frac{1}{t}\int\limits_0^t V^2(\tau)d\tau\;.$$

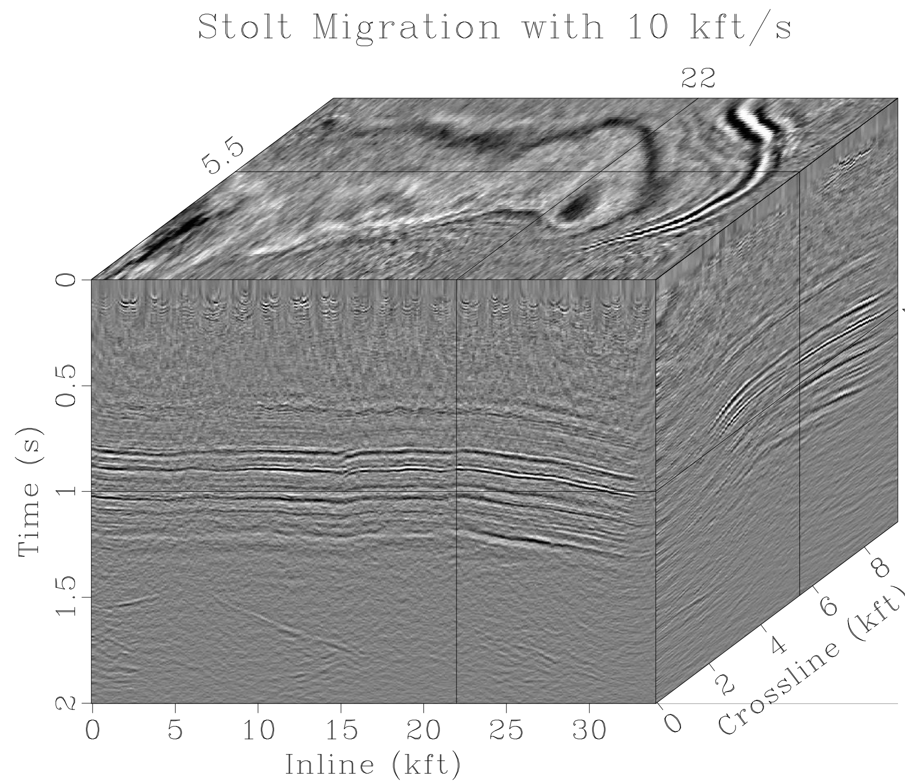


(a)

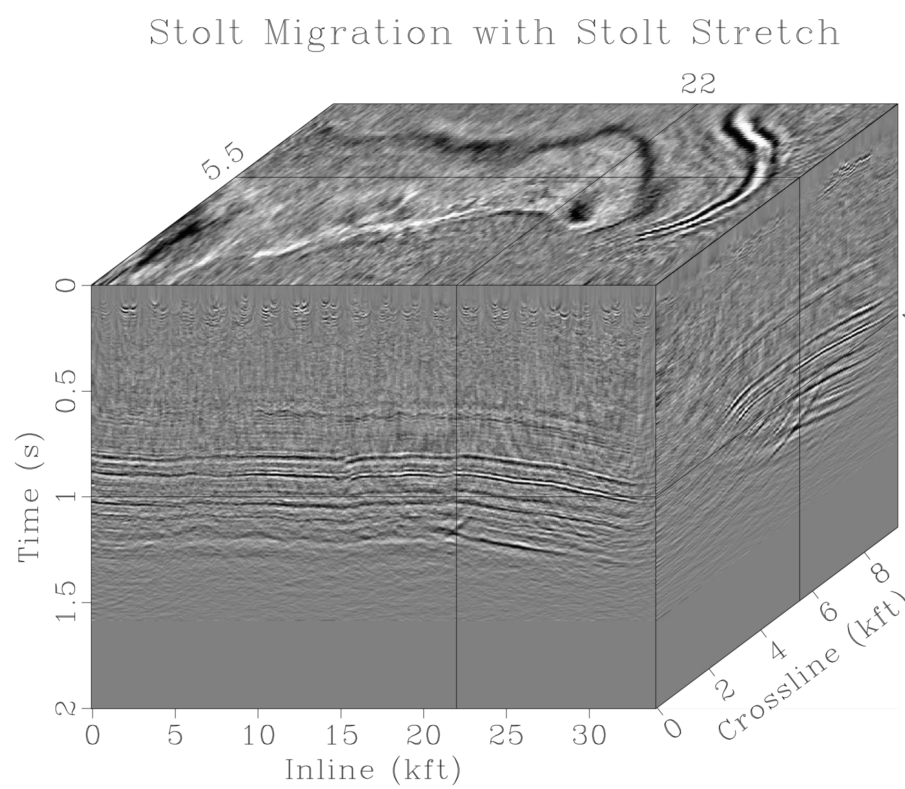


(b)

Figure 4: Portion of the rotated Teapot Dome stack (a) and its migration by the phase-shift method displayed in vertical time (b). gazdag/ stack2,image



(a)



(b)

Figure 5: Portion of the rotated Teapot Dome stack migrated using Stolt migration. (a) Using a constant velocity of 10 kft/s. (b) Using the Stolt stretch method. gazdag/ mig10,migst

(b) Performing Stolt migration according to mapping

$$\omega = \left(1 - \frac{1}{W}\right) \omega_0 + \frac{\omega_0}{W} \sqrt{1 + W \frac{V_0^2 k^2}{4\,\omega_0^2}}\;, \tag{4}$$

where W is a constant (typically between 1 and 2).

(c) Inverse stretch (squeeze) from t_s to t.

Figure 5a shows the result of Stolt migration without stretch and using velocity of 10 kft/s. Figure 5b is the result of Stolt migration using the stretch approach. To display these figures on your screen, run

```
scons mig10.view migst.view
```

10. How much faster is Stolt migration compared to Gazdag migration?

    You can compare the CPU time of different methods experimentally by running `scons TIMER=y` instead of `scons`.

Answer:

11. How accurate is Stolt stretch in approximating the output of Gazdag migration? You can compare the results by running

    ```
    sfpen Fig/image.vpl Fig/migst.vpl
    ```

    Try improving the match by modifying the value of the Stolt-stretch parameter W from the value of $W = 1.5$. You can find a better value by experimentation or by using the appropriate theory (Fomel and Vaillant, 2001).

12. (**EXTRA CREDIT**) For an extra credit, try improving the imaging result by using a more detailed velocity analysis or a more accurate migration method.

gazdag/SConstruct

```
from rsf.proj import *

# Download Teapot Dome field data
Fetch('npr3_gathers.sgy','teapot',
      server='http://s3.amazonaws.com',top='')
#Fetch('npr3_gathers.sgy','TeapotDome3D',
#      top='/home/p1/seismic_datasets/SeismicProcessingClass',
#      server='local')

# Convert from SEGY to RSF
Flow('traces header header.asc','npr3_gathers.sgy',
     '''
     segyread tfile=${TARGETS[1]} hfile=${TARGETS[2]} |
     window max1=2
     ''')
```

```

# Seismic data corresponds to trid=1
Flow('trid','header','headermath output=trid | mask min=1 max=1')

Flow('tcmp','header trid','headerwindow mask=${SOURCES[1]}')
Flow('cmps','traces trid','headerwindow mask=${SOURCES[1]}')

# Extract offset, convert from ft to kft
Flow('offset','tcmp',
     '''
     headermath output=offset |
     dd type=float | scale dscale=0.001
     ''')

# Velocity analysis using a supergather
#########################################

# take every 500th trace
Flow('subcmps','cmps','window j2=500')
Flow('suboffset','offset','window j2=500')

Flow('vscan','subcmps suboffset',
     '''
     vscan offset=${SOURCES[1]} half=n semblance=y
     v0=9 nv=101 dv=0.1
     ''')
Plot('vscan',
     '''
     grey color=j allpos=y title="Semblance Scan"
     unit2=kft/s
     ''')

Flow('vpick','vscan',
     '''
     mutter inner=y x0=9 half=n t0=0.5 v0=3 |
     scale axis=2 | pick rect1=50
     ''')
Plot('vpick',
     '''
     graph yreverse=y transp=y plotcol=7 plotfat=7
     pad=n min2=9 max2=19 wantaxis=n wanttitle=n
     ''')

Result('vscan','vscan vpick','Overlay')

# Dix conversion to interval velocity
Flow('semb','vscan vpick','slice pick=${SOURCES[1]}')
Flow('vdix','vpick semb','dix weight=${SOURCES[1]} rect1=50')
```

```

Result('velocity','vdix vpick',
        '''
        cat axis=2 ${SOURCES[1]} |
        graph dash=0,1 title="Interval Velocity" unit2=kft/s
        ''')

# NMO and stack
###############

Flow('binorder','tcmp','headermath output="345*xline+iline" ')
Flow('tcmp2','tcmp binorder','headersort head=${SOURCES[1]}')
Flow('cmps2','cmps binorder tcmp2',
      '''
      headersort head=${SOURCES[1]} |
      intbin3 head=${SOURCES[2]} xkey=-1 yk=iline zk=xline
      ''')
Flow('offset2 mask','offset binorder tcmp2',
      '''
      headersort head=${SOURCES[1]} |
      intbin3 head=${SOURCES[2]} xkey=-1 yk=iline zk=xline
      mask=${TARGETS[1]}
      ''')

Flow('vpick3','vpick',
     '''
     spray axis=2 n=1 | spray axis=3 n=345 | spray axis=4 n=188
     ''')
Flow('mask3','mask','spray axis=1 n=1')

Flow('nmo','cmps2 offset2 mask3 vpick3',
      '''
      nmo offset=${SOURCES[1]} half=n
      mask=${SOURCES[2]} velocity=${SOURCES[3]}
      ''',split=[4,'omp'])

Flow('stack','nmo','stack')

Result('stack',
        '''
        byte gainpanel=all |
        grey3 frame1=500 frame2=200 frame3=100 title=Stack
        ''')

# Rotate and window

import math
az = 70*math.pi/180 # azimuth angle
```

```

nx=345
ny=188
nxy=nx*ny

Flow('x','stack',
     '''
     window n1=1 |
     math output="%g+(%g)*x1+(%g)*x2"
     ''' % (nx*0.5, math.cos(az),math.sin(az)))
Flow('y','x',
     '''
     math output="%g+(%g)*x1+(%g)*x2"
     ''' % (ny*0.5,-math.sin(az),math.cos(az)))

Flow('xy','x y',
     '''
     cat axis=3 ${SOURCES[1]} | put n1=%d n2=1 |
     window | transp
     ''' % nxy)

Flow('stack2','stack xy',
     '''
     put n2=%d n3=1 |
     transp memsize=5000 |
     bin xkey=0 ykey=1 head=${SOURCES[1]}
     nx=85 x0=295 dx=1 ny=310 y0=-200 dy=1 interp=2 |
     costaper nw2=10 nw3=10 |
     transp plane=13 memsize=5000 |
     put o2=0 d2=0.11 o3=0 d3=0.11
     unit2=kft label2=Inline unit3=kft label3=Crossline
     ''' % nxy)

Result('stack2',
       '''
       byte gainpanel=all |
       grey3 frame1=500 frame2=200 frame3=50
       title=Stack point1=0.7 point2=0.7 flat=n
       ''')

# Fourier transform in space

Flow('cosft','stack2','cosft sign2=1 sign3=1')

# Phase-shift migration
########################

Flow('gazdag','cosft vdix',
```

```
     'gazdag velocity=${SOURCES[1]} verb=y',
     split=[3,'omp',[0]])

Flow('image','gazdag','cosft sign2=-1 sign3=-1')

Result('image',
       '''
       byte gainpanel=all |
       grey3 frame1=500 frame2=200 frame3=50
       title="Phase-Shift Migration" point1=0.7 point2=0.7 flat=n
       ''')

# Stolt migration with 10 kft/s
Flow('cosft3','cosft','cosft sign1=1')

Flow('map10','cosft3',
     'math output="sqrt(x1*x1+25*(x2*x2+x3*x3))" ')

Flow('stolt10','cosft3 map10',
     'iwarp warp=${SOURCES[1]} inv=n',split=[3,'omp'])

Flow('mig10','stolt10','cosft sign1=-1 sign2=-1 sign3=-1')

Result('mig10',
       '''
       byte gainpanel=all |
       grey3 frame1=500 frame2=200 frame3=50
       title="Stolt Migration with 10 kft/s"
       point1=0.7 point2=0.7 flat=n
       ''')

# Apply Stolt stretch
Flow('cosftst','cosft vdix',
     'stoltstretch velocity=${SOURCES[1]} vel=10 | cosft sign1=1')

# Stolt stretch parameter
###########################
st=1.5 # !!! MODIFY ME !!!

Flow('mapst','cosft3',
     '''
     math output="(1-1/%g)*x1+sqrt(x1*x1+%g*(x2*x2+x3*x3))/%g"
     ''' % (st,st*25,st))

Flow('stoltst','cosftst mapst',
     'iwarp warp=${SOURCES[1]} inv=n',split=[3,'omp'])

Flow('migst','stoltst vdix',
```

```
      '''
      cosft sign1=-1 sign2=-1 sign3=-1 |
      stoltstretch velocity=${SOURCES[1]} vel=10 inv=y
      ''')

Result('migst',
        '''
        byte gainpanel=all |
        grey3 frame1=500 frame2=200 frame3=50
        title="Stolt Migration with Stolt Stretch"
        point1=0.7 point2=0.7 flat=n
        ''')

End()
```

REVERSE-TIME MIGRATION

In this part of the assignment, we will create a depth image of the Viking Graben data using post-stack reverse-time migration (RTM).

1. Change directory to `hw6/rtm`.

2. Run

   ```
   scons -c
   ```

   to remove (clean) previously generated files.

3. We start by reproducing the DMO processing workflow from Assignment 3 using the recipe from *Radii Interceptor*. A picked DMO (time-migration) velocity distribution is shown in Figure 7a. To reproduce it on your screen, run

   ```
   scons vdix.view
   ```

4. The next step is converting the time-migration velocity to depth for creating an initial velocity model for depth migration. To display the Dix-converted velocity in time and depth (Figure 7), run

   ```
   scons vdix.view vofz.view
   ```

5. The DMO stack (Figure 8a) can be used as the input to post-stack depth migration. We will use reverse-time migration by the lowrank approximation method (Fomel et al., 2013). The method employs several (in this case, three) spatial Fourier transforms per time step. To generate the result (Figure 8b), run

   ```
   scons rtm.view
   ```

To watch a movie of reverse-time wave propagation which produced the image in Figure 8b, run

```
scons snaps.vpl
```

6. Your task: try to improve the quality of the image by
   (a) using the results from your own time-domain imaging (Assignments 1–5);
   (b) modifying the time-to-depth conversion step.

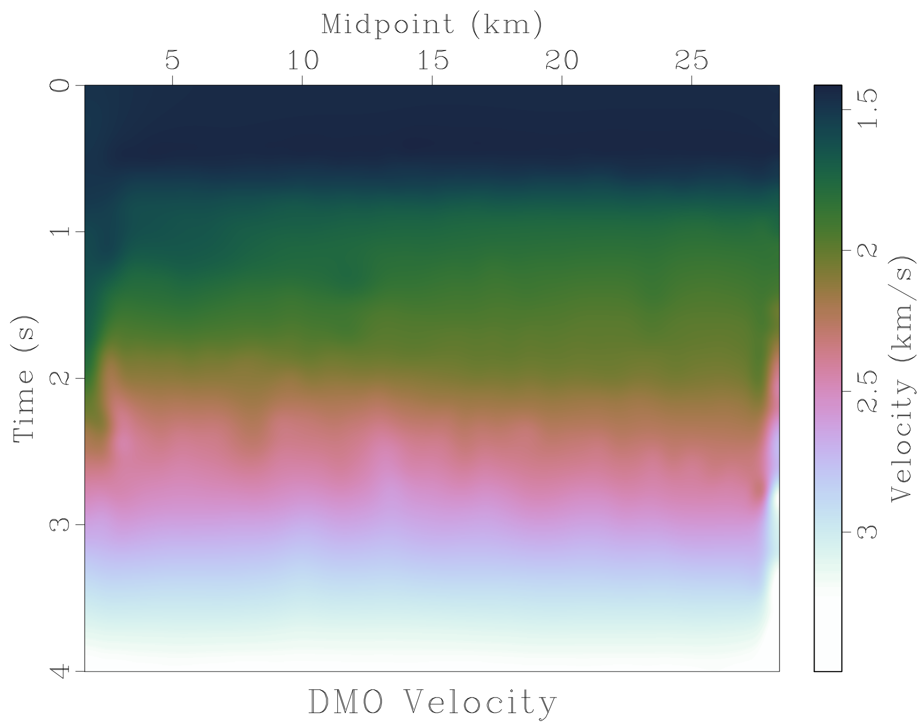
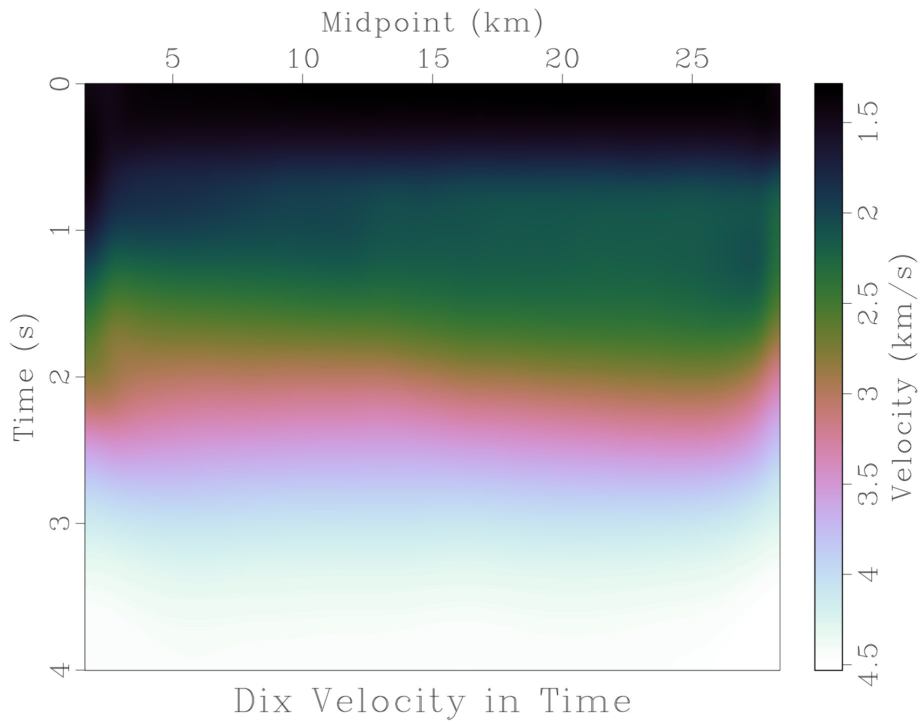


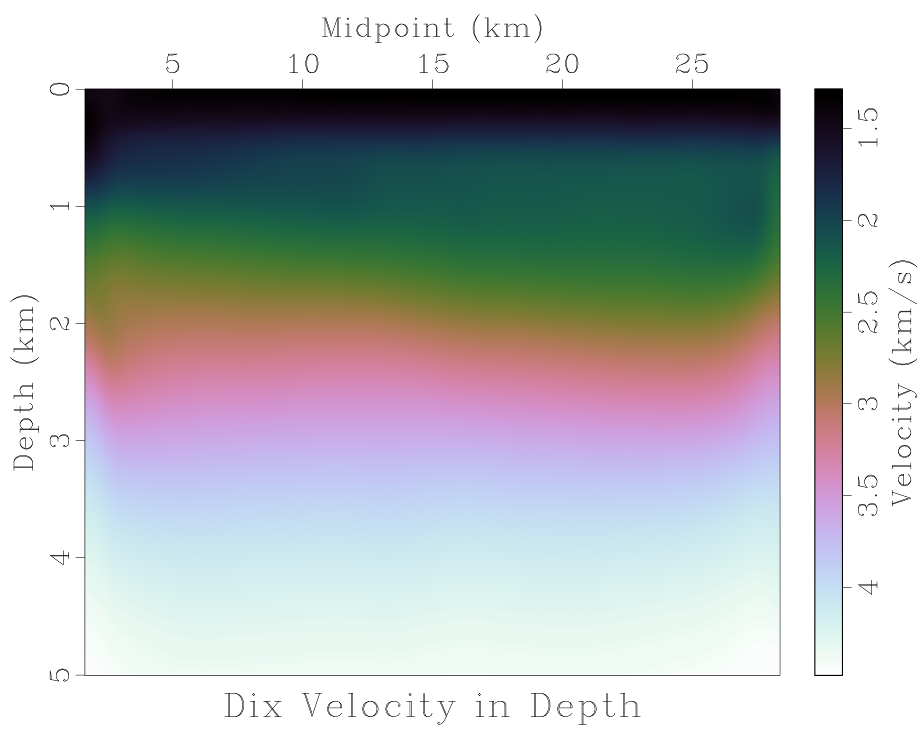
Figure 6: Picked DMO (time-migration) velocity in the Viking Graben dataset. rtm/ vpick

rtm/SConstruct

```
from rsf.proj import *

# Download pre-processed CMP gathers
# from the Viking Graben dataset
Fetch('paracdp.segy','viking')

# Convert to RSF
Flow('paracdp tparacdp','paracdp.segy',
     'segyread tfile=${TARGETS[1]}')

# Convert to CDP gathers, time-power gain and high-pass filter
```

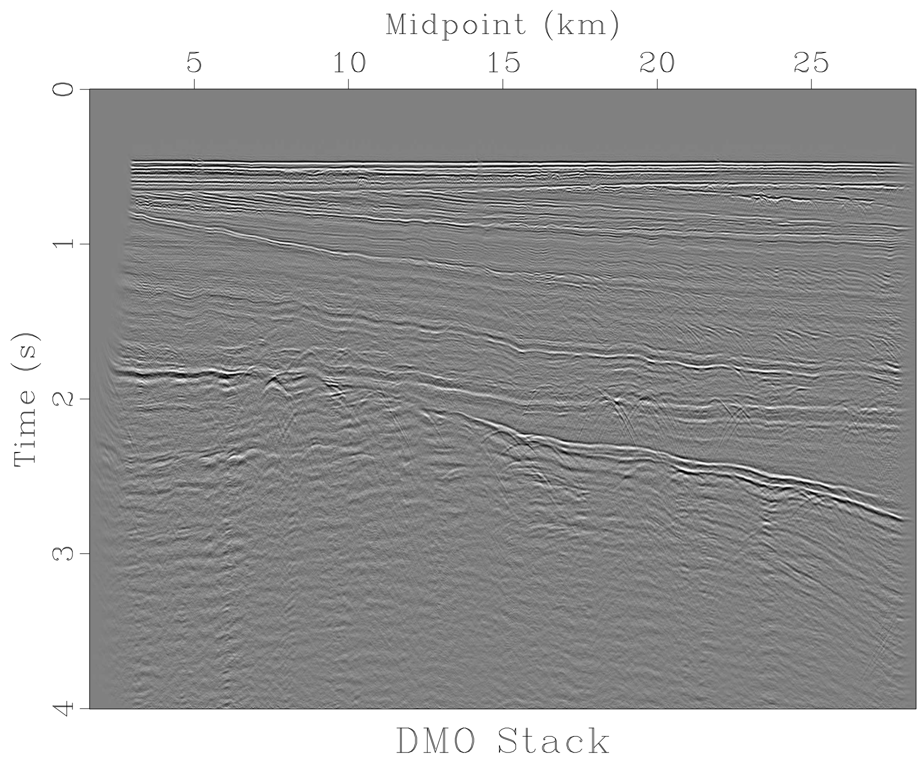


(a)

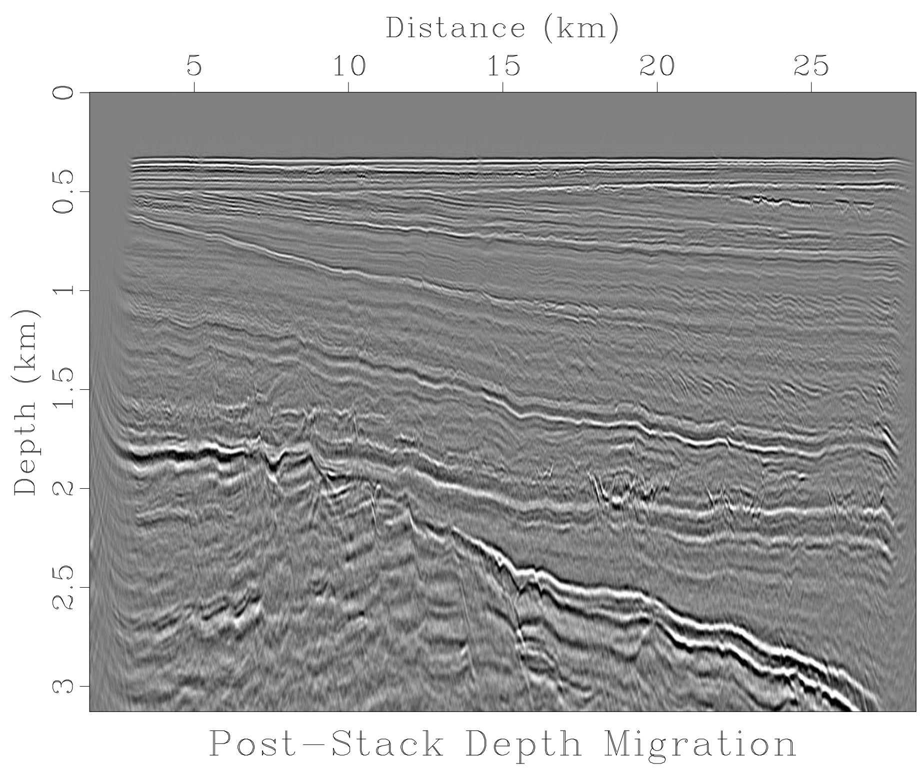


(b)

Figure 7: Interval velocity estimated by Dix inversion in vertical time (a) and depth (b). rtm/ vdix,vofz



(a)



(b)

Figure 8: DMO stack of the Viking Graben dataset (a) and its depth migration by RTM (b). rtm/ slice,rtm

```
Flow('cmps','paracdp',
     '''intbin xk=cdpt yk=cdp | window max1=4 |
     pow pow1=2 | bandpass flo=5 |
     put label3=Midpoint unit3=km o3=1.619 d3=0.0125''')

# Extract offsets
Flow('offsets mask','tparacdp',
     '''headermath output=offset |
     intbin head=$SOURCE xk=cdpt yk=cdp mask=${TARGETS[1]} |
     dd type=float | scale dscale=0.001''')

# Window bad traces
Flow('maskbad','cmps',
     'mul $SOURCE | stack axis=1 | mask min=1e-20')
Flow('mask2','maskbad mask',
     'spray axis=1 n=1 | mul ${SOURCES[1]}')

# NMO stack with an ensemble of constant velocities
Flow('stacks','cmps offsets mask2',
     '''stacks half=n v0=1.4 nv=121 dv=0.02
     offset=${SOURCES[1]} mask=${SOURCES[2]}''',split=[3,'omp'])

# Taper midpoint
Flow('stackst','stacks','costaper nw3=100')

# Apply double Fourier transform (cosine transform)
Flow('cosft','stackst','pad n3=2401 | cosft sign1=1 sign3=1')
# Transpose f-v-k to v-f-k
Flow('transp','cosft','transp',split=[3,'omp'])

# Fowler DMO: mapping velocities
Flow('map','transp',
     '''math output="x1/sqrt(1+0.25*x3*x3*x1*x1/(x2*x2))" |
     cut n2=1''')
Flow('fowler','transp map',
     'iwarp warp=${SOURCES[1]} | transp',split=[3,'omp'])

# Inverse Fourier transform
Flow('dmo','fowler','cosft sign1=-1 sign3=-1 | window n3=2142')

# Compute envelope for picking
Flow('envelope','dmo','envelope | scale axis=2',split=[3,'omp'])

###########################################################################
# Improved Automatic Picking from Radii Interceptor
###########################################################################
mute = Program('mute.c')
Flow('envmute','envelope %s'%(mute[0]),
```

```
      './${SOURCES[1]} t1=1.4 v1=3.2')
Flow('vpick','envmute','pick vel0=1.45 rect1=25 rect2=50')

Result('vpick',
       '''
       grey mean=y color=x scalebar=y title="DMO Velocity"
       barreverse=y barlabel=Velocity barunit=km/s
       ''')

# Take a slice from mutting envelope
Flow('slice','dmo vpick','slice pick=${SOURCES[1]}')
Result('slice','grey title="DMO Stack"')

###################################################################
# Dix conversion to interval velocity
###################################################################

Flow('weight','envmute vpick','slice pick=${SOURCES[1]}')
Flow('vdix','vpick weight',
     'dix rect1=25 rect2=50 weight=${SOURCES[1]}')

Result('vdix',
       '''
       grey allpos=y bias=1.3 clip=3.2
       color=x scalebar=y title="Dix Velocity in Time"
       barreverse=y barlabel=Velocity barunit=km/s
       ''')

Flow('vofz','vdix',
     '''
     time2depth velocity=$SOURCE intime=y nz=1001 z0=0 dz=0.005 |
     put label1=Depth unit1=km
     ''')

Result('vofz',
       '''
       grey allpos=y bias=1.3 clip=3.2
       color=x scalebar=y title="Dix Velocity in Depth"
       barreverse=y barlabel=Velocity barunit=km/s
       ''')

###################################################################
# Zero-offset reverse-time migration
###################################################################

Flow('fft','vofz','transp | fft1 | fft3 axis=2 pad=1')
Flow('right left','vofz fft',
     '''
```

```
    transp | scale dscale=0.5 |
    isolr2 seed=2016 dt=0.002 npk=50
    fft=${SOURCES[1]} left=${TARGETS[1]}
    ''')

Flow('rtm snaps','slice left right',
    '''
    spline n1=2000 o1=0 d1=0.002 |
    reverse which=1 |
    transp |
    fftexp0 mig=y snap=10 snaps=${TARGETS[1]}
    left=${SOURCES[1]} right=${SOURCES[2]}
    nz=1001 dz=0.005
    ''')

Result('rtm',
       '''
       window max1=3.125 |
       grey title="Post-Stack Depth Migration" unit2=km
       ''')

Plot('snaps','grey title=Snapshots gainpanel=all unit2=km',view=1)

End()
```

KIRCHHOFF MIGRATION

How accurate is the image in Figure 8b? We can evaluate it by performing prestack depth migration (PSDM). In this part of the assignment, we will image the same dataset using prestack Kirchhoff depth migration method.

1. Change directory to `hw6/kirchhoff`.
2. Run

   ```
   scons -c
   ```

   to remove (clean) previously generated files.
3. For the prestack migration exercise, we will use the same input as in the previous section but resort it into shot gathers (Figure 9). To display the data on your screen, run

   ```
   scons shots.view
   ```

4. The first step in the Kirchhoff method is computing traveltime tables. To display the computed table on your screen (as a movie over source locations), run

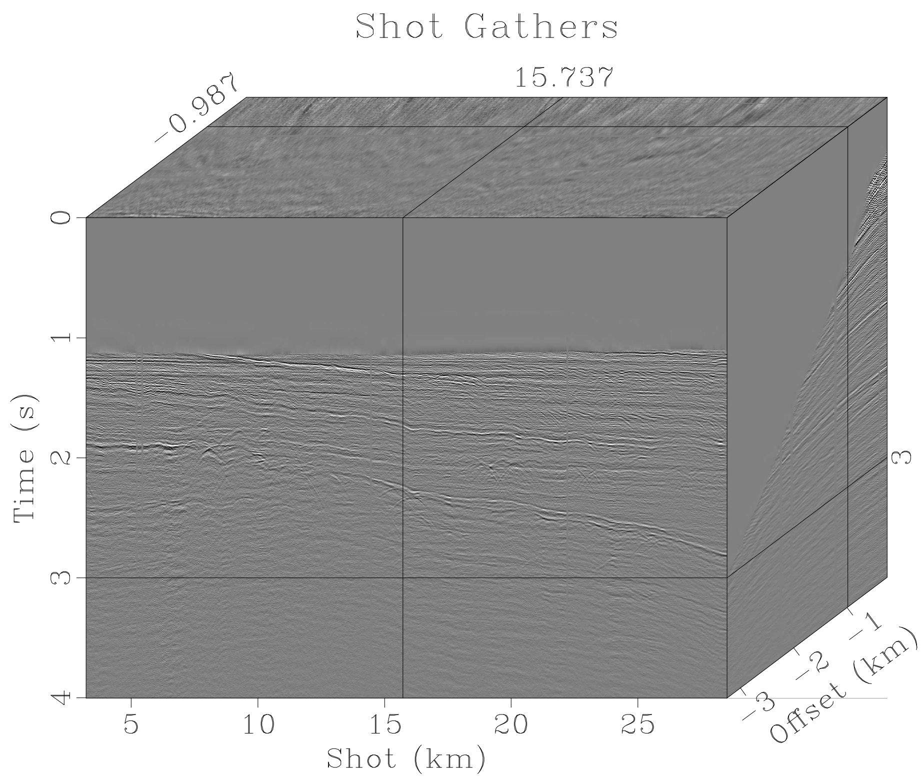


Figure 9: Preprocessed shot gathers from the Teapot Dome dataset.

```
scons times.vpl
```

In addition to traveltimes themselves, we compute the traveltime derivatives with respect to the source and receiver locations (Li and Fomel, 2013).

5. To compute and display the PSDM image (Figure 10), run

   ```
   scons psdm.view
   ```

   Caution: this computation is expensive and may take some time.

6. Your task: replace input data with the result of your processing (parabolic Radon demultiple) from the Computational Assignment 5 and compare the results.

7. (**EXTRA CREDIT**) For an extra credit, replace `cig=n` with `cig=y` in the `SConstruct` file to generate CIGs (Common Image Point Gathers) instead of a stacked migration image. Examine the flatness of the gathers as an indication of the velocity model correctness. Can you improve the image by updating the velocity model or processing the gathers?

kirchhoff/SConstruct

```
from rsf.proj import *

```

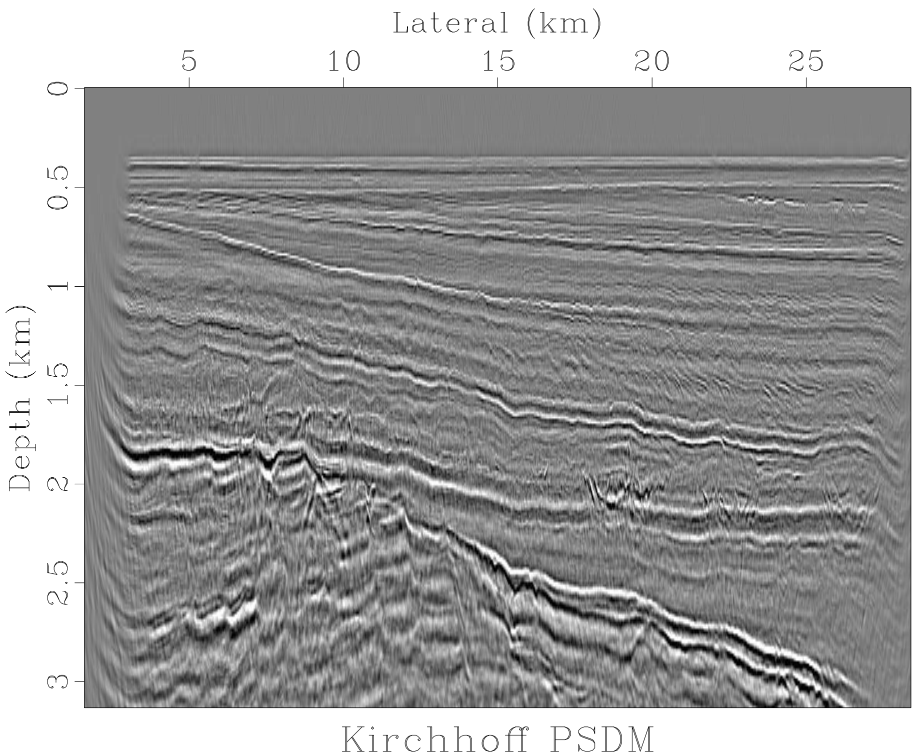


Figure 10: Kirchoff prestack depth migration of the Teapot Dome dataset (using every 10th shot).

```
# Pre-processed CMP gathers from the rtm directory
Fetch(['paracdp.rsf','tparacdp.rsf'],
      'rtm',top='..',server='local')

# Convert to shot gathers, time-power gain and high-pass filter
Flow('shot',  'tparacdp','headermath output="(sx-3237)/25" ')
Flow('offset','tparacdp','headermath output="(offset+3237)/25" ')
Flow('head','offset shot','cat axis=1 ${SOURCES[1]}')

Flow('shots','paracdp head',
     '''
     intbin head=${SOURCES[1]} xkey=0 ykey=1 |
     window max1=4 | pow pow1=1 | bandpass flo=5 |
     put
     label2=Offset unit2=km o2=-3.237 d2=0.025
     label3=Shot   unit3=km o3=3.237  d3=0.025
     ''')

Result('shots',
       '''
       byte gainpanel=500 |
       transp plane=23 memsize=5000 |
       grey3 frame1=750 frame2=500 frame3=90
       point1=0.8 point2=0.8 flat=n
       title="Shot Gathers"
       ''')

# Source and receiver coordinates
Flow('ys','shots','window n1=1 n2=1 | math output=x1')
Flow('xs','ys','math output=0')
Flow('zs','ys','math output=0.01')
Flow('scoord','zs ys xs','cat axis=2 ${SOURCES[1:3]} | transp')

Flow('yr',None,'math n1=1131 o1=0 d1=0.025 output=x1')
Flow('xr','yr','math output=0')
Flow('zr','yr','math output=0.006')
Flow('rcoord','zr yr xr','cat axis=2 ${SOURCES[1:3]} | transp')

# Velocity model from the rtm directory
Fetch('vofz.rsf','rtm',top='..',server='local')
Flow('velocity','vofz','remap1 n1=251 o1=0 d1=0.0125 | put o3=0')

# First-arrival traveltime tables
Flow('times tdls tdss','velocity scoord',
     '''
     eikods shotfile=${SOURCES[1]}
     tdl1=${TARGETS[1]} tds1=${TARGETS[2]} b1=2 b2=2 |
     put o4=3.237 d4=0.025 | window
```

```
    ''')
Flow('timer tdlr tdsr','velocity rcoord',
    '''
    eikods shotfile=${SOURCES[1]}
    tdl1=${TARGETS[1]} tds1=${TARGETS[2]} b1=2 b2=2 |
    put o4=0 d4=0.025 | window
    ''')

Plot('times',
    '''
    window j3=10 | grey color=j title="Traveltime Table"
    scalebar=y barlabel=Time barunit=s allpos=y
    minval=0 maxval=10
    ''',view=1)

# Kirchhoff PSDM

Flow('psdm','shots times tdss timer tdsr',
    '''
    kirmigsr cig=n type=l
    stable=${SOURCES[1]} sderiv=${SOURCES[2]}
    rtable=${SOURCES[3]} rderiv=${SOURCES[4]}
    ''',split=[3,'omp',[0]],reduce='add')

Result('psdm','grey title="Kirchhoff PSDM" ')

End()
```

REFERENCES

Dix, C. H., 1955, Seismic velocities from surface measurements: Geophysics, **20**, 68–86.

Fomel, S., and L. Vaillant, 2001, Evaluating the Stolt-stretch parameter: Journal of Seismic Exploration, **9**, 319–335. (`http://ahay.org/RSF/book/sep/stoltst/paper_html/`).

Fomel, S., L. Ying, and X. Song, 2013, Seismic wave extrapolation using lowrank symbol approximation: Geophysical Prospecting, **61**, 526–536. (`http://ahay.org/RSF/book/tccs/lowrank/paper_html/`).

Gazdag, J., 1978, Wave equation migration with the phase-shift method: Geophysics, **43**, 1342–1351.

Li, S., and S. Fomel, 2013, Kirchhoff migration using eikonal-based computation of traveltime source derivatives: Geophysics, **78**, S211–S219. (`http://ahay.org/RSF/book/tccs/eikods/paper_html/`).

Stolt, R. H., 1985, Migration by Fourier transform: Geophysics, **50**, 2219–2244. (Discussion and reply in GEO-60-5-1583).