In this homework, you will use a finite-differences modeling code, similar to the one you wrote in the preceding homework, to implement basic reverse time migration. I do not expect you to be concerned with the efficiency of your implementation at this time. This implementation of reverse-time migration does not require that you write any new C code. You will use pre-existing Madagascar programs, but you will modify the `SConstruct` file to combine those programs.

This is an individual assignment and absolutely no collaboration on code is allowed.

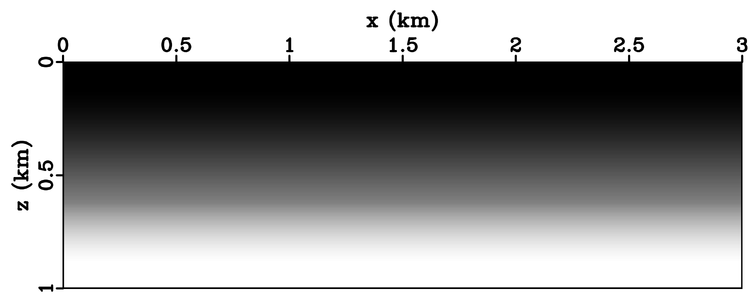


Figure 1: Velocity.

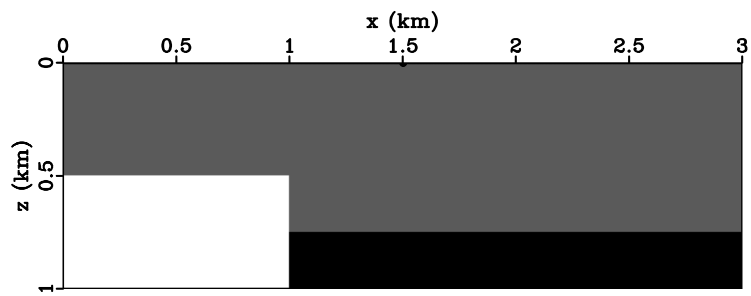


Figure 2: Density.

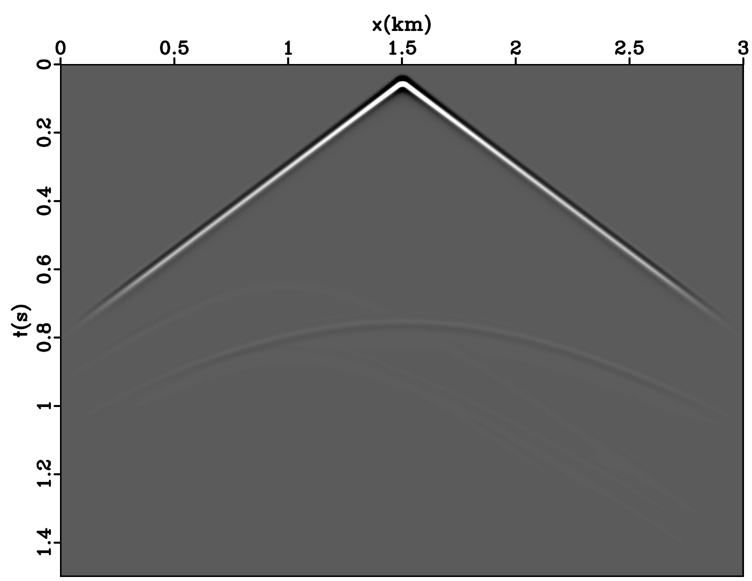
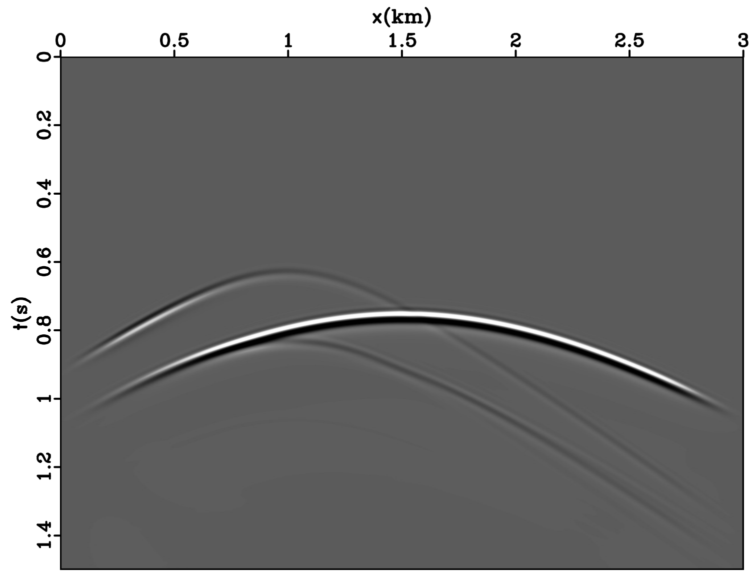


Figure 3: Data w/ direct arrival.

EXERCISE

Using the finite-differences modeling function `awefd`, construct an image of the subsurface. This function takes the following parameters:

Figure 4: Data w/o direct arrival.



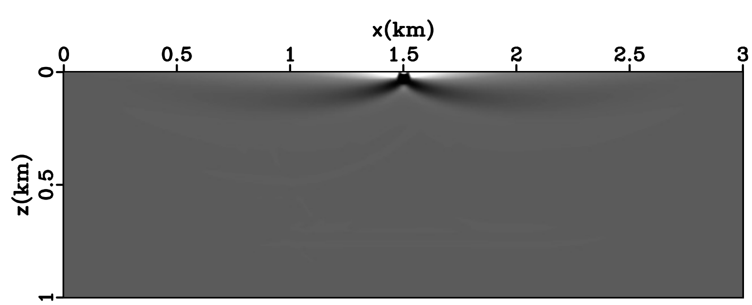

```
awefd(odat,owfl,idat,velo,dens,sou,rec,custom,par)
```

- `odat`: output data $d(x,t)$
- `owfl`: output wavefield $u(z,x,t)$
- `idat`: input data (wavelet)
- `velo`: velocity model $v(z,x)$
- `dens`: density model $\rho(z,x)$
- `sou`: source coordinates
- `rec`: receiver coordinates
- `custom`: custom parameters
- `par`: parameter dictionary

Design an imaging procedure following the generic scheme developed in class. Your task is to identify Madagascar programs necessary to implement reverse-time migration in two different ways and generate the appropriate `Flows` in the `SConstruct`. Explain in detail how your imaging procedures work.

1. Use your imaging procedure to generate images based on recorded data in Figures 3 and 4. For this exercise, use the constant density `rb.rsf` for imaging. Include those two images in this document. Are the images different from each-other? How? Why?

Figure 5: data in Figure 3, constant density , and first I.C..



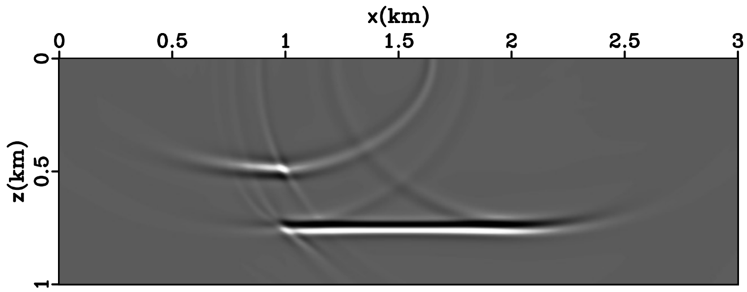


Figure 6: data in Figure 4, constant density , and first I.C..

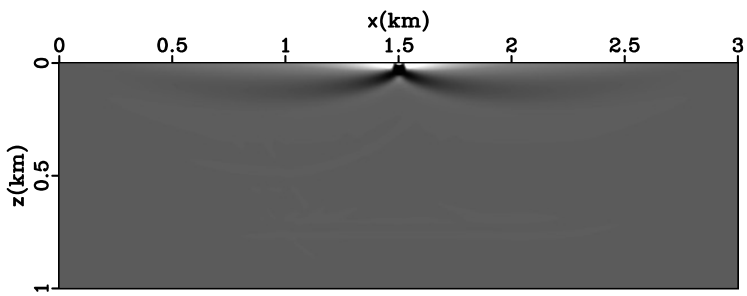


Figure 7: data in Figure 3, constant density , and second I.C..

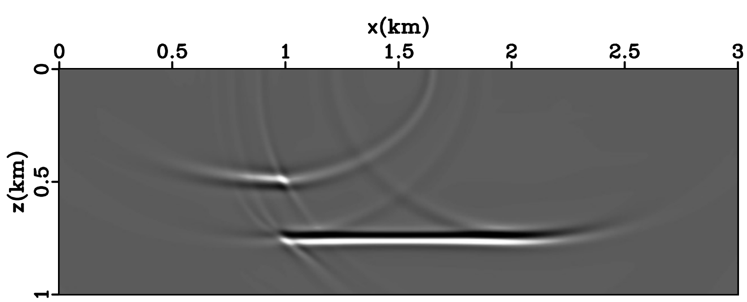


Figure 8: data in Figure 4, constant density , and second I.C..

2. Use your imaging procedure to generate images based on recorded data in Figures 3 and 4. For this exercise, use the variable density `ra.rsf` for imaging. Include those two images in this document. Are the images different from each-other? How? Why? How do your images compare with the ones from the preceding exercise?

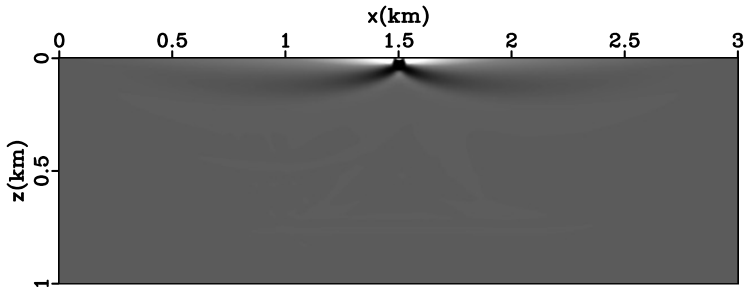


Figure 9: data in Figure 3, variable density , and first I.C..

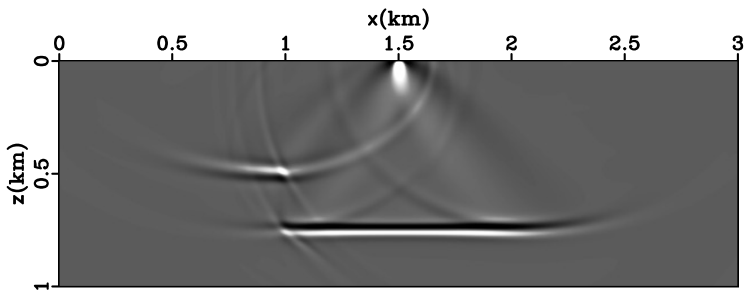


Figure 10: data in Figure 4, variable density , and first I.C..

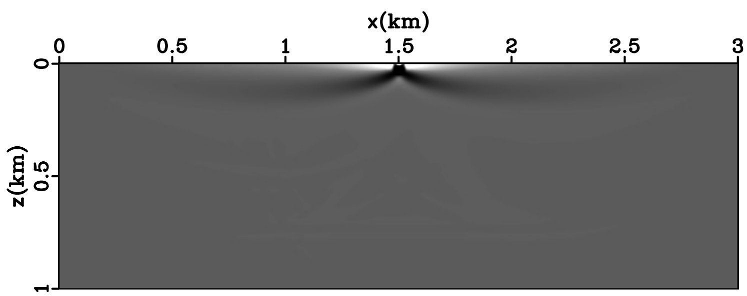


Figure 11: data in Figure 3, variable density , and second I.C..

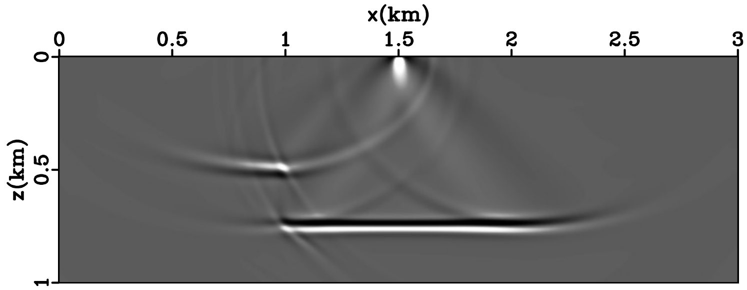


Figure 12: data in Figure 4, variable density , and second I.C..

COMMENTS

Codes

The block of codes with gray background in the attached `SConstruct` are designed to perform reverse-time migration.

Results

1) using constant density
1a) Using two different implementations for the conventional image condition, I obtain the same migrated images as shown in Figures 5 and 7, Figures 6 and 8.
1b) Using the recorded data shown in Figure 3:
In this data, the direct arrival is much stronger than the reflected arrivals, therefore, the interfaces in the migrated images (Figures 5 and 7) are much weaker than the low-frequency feature (at $x = 1.5$ km and $z = 0.0$ km) due to the strong direct arrivals.
1c) Using the recorded data shown in Figure 4:
In this data, we only have the reflected arrivals while the direct arrival is removed, therefore, the interfaces are strong and obvious in the migrated images (Figures 6 and 8) and we do not see the feature at at $x = 1.5$ km and $z = 0.0$ km. In Figures 6 and 8, we observe smile-shape artifacts at the ends of the two horizontal reflectors because of the truncation of the data. We also see some artifacts at the corners at $x = 1$ km because these corners are refractors.
2) using variable density
2a) Using two different implementations for the conventional image condition, I obtain the same migrated images as shown in Figures 9 and 11, Figures 10 and 12.
2b) Using the recorded data shown in Figure 3:
I obtain almost the same migrated images (Figures 9 and 11) as the ones (Figures 5 and 7) with constant density.
2c) Using the recorded data shown in Figure 4:
I obtain the same horizontal reflectors and nearby artifacts in the migrated images (Figures 10 and 12) as using the ones (Figures 6 and 8) with constant density. However, I can also see the low-frequency features extended from the top

to the horizontal reflectors because the consistence of reflected events between the source wavefield and data wavefield always exists.

SCONSTRUCT

```
 # GPGN 658 - reverse-time migration
from rsf.proj import *
import fdm
# ------------------------------------------------------------
par = dict(
    nt=1500, ot=0, dt=0.001, lt='t', ut='s',
    nx=601,  ox=0, dx=0.005, lx='x', ux='km',
    nz=201,  oz=0, dz=0.005, lz='z', uz='km',
    kt=50,nb=100,jsnap=50,jdata=1,frq=35
    )
fdm.param(par)

par['xk']=50
par['xl']=par['nx']-50

par['xsou']=par['ox']+par['nx']/2*par['dx']
par['zsou']=par['oz']

# ------------------------------------------------------------
# wavelet
fdm.wavelet('wav_',par['frq'],par)
Flow(  'wav', 'wav_','transp')
Result('wav','window n2=500 |' + fdm.waveplot('',par))

# ------------------------------------------------------------
# sources coordinates
fdm.point('ss',par['xsou'],par['zsou'],par)
Plot('ss',fdm.ssplot('',par))

# receivers coordinates
fdm.horizontal('rr',0,par)
Plot('rr',fdm.rrplot('',par))

# ------------------------------------------------------------
# velocity
Flow('vo',None,
     '''
     math output="2.0+0.25*x1"
     n1=%(nz)d o1=%(oz)g d1=%(dz)g
     n2=%(nx)d o2=%(ox)g d2=%(dx)g
     ''' % par)

Plot(  'vo',fdm.cgrey('allpos=y bias=2.0 pclip=100',par))
Result('vo',['vo','ss','rr'],'Overlay')

# ------------------------------------------------------------
# density
Flow('ra',None,
     '''
     spike nsp=2 mag=+0.5,-0.5
     n1=%(nz)d o1=%(oz)g d1=%(dz)g k1=101,151 l1=%(nz)d,%(nz)d
     n2=%(nx)d o2=%(ox)g d2=%(dx)g k2=1,201   l2=200,%(nx)d |
     add add=2
     ''' % par)
Plot(  'ra',fdm.cgrey('allpos=y bias=1.5 pclip=100',par))
Result('ra',['ra','ss','rr'],'Overlay')

Flow('rb','ra','math output=1')

# ------------------------------------------------------------
```

```
# edge taper
Flow('taper',None,
     '''
     spike nsp=1 mag=1
     n1=%(nx)d d1=%(dx)g o1=%(ox)g k1=%(xk)d l1=%(xl)d
     n2=%(nt)d d2=%(dt)g o2=%(ot)g |
     smooth rect1=50
     ''' % par)
Result('taper','transp |'+fdm.dgrey('pclip=99',par))

# ------------------------------------------------------------
# finite-differences modeling
fdm.awefd('dd','ww','wav','vo','ra','ss','rr','jsnap=1 fsrf=n',par)
fdm.awefd('do','wo','wav','vo','rb','ss','rr','jsnap=1 fsrf=n',par)

Result('ww','window j3=%(jsnap)d |'%par + fdm.wgrey('pclip=99.9',par))
Result('wo','window j3=%(jsnap)d |'%par + fdm.wgrey('pclip=99.9',par))

# data w/  direct arrivals
Flow(  'dr0','dd taper',
       'add mode=p ${SOURCES[1]}')

# data w/o direct arrivals
Flow(  'dr1','dd do taper',
       'math r=${SOURCES[0]} d=${SOURCES[1]} t=${SOURCES[2]} output="(r-d)*t"')

for j in range(2):
    dtag="%d"%j
    Result('dr'+dtag,'transp |' + fdm.dgrey('pclip=99.9',par))
# ------------------------------------------------------------
# Reverse-time migration
imags = ['imag0','imag1']
odats = ['odat0','odat1']
tdats = ['tdat0','tdat1']
twfls = ['twfl0','twfl1']
rwfls = ['rwfl0','rwfl1']
velo,sou,rec,custom='vo','rr','rr','jsnap=1 fsrf=n'
ics,dens,swfls,idats = ['ic0','ic1'],['ra','rb'],['ww','wo'],['dr0','dr1']
# two implementations of conventional (cross-correlation zero-lag) IC
method1 = 'xcor2d uu=${SOURCES[1]} axis=3 verb=y nbuf=100'
method2 = 'sfadd mode=m ${SOURCES[1]} | sfstack axis=3 ${SOURCES[1]}'
for den in dens:
  for i in range(2):
    Flow(odats[i]+den,idats[i],'reverse which=2 opt=i verb=y')
    fdm.awefd(tdats[i]+den,twfls[i]+den,odats[i]+den,velo,den,sou,rec,custom,par)
    Flow(rwfls[i]+den,twfls[i]+den,'reverse which=4 opt=i verb=y')
    output = imags[i]+den
    inputs = [rwfls[i]+den,swfls[i]]
    for ic in ics:
      if ic=='ic0':
        Flow(output+ic,inputs,method1)
      if ic=='ic1':
        Flow(output+ic,inputs,method2)
      Result(output+ic,'window j3=%(jsnap)d |'%par + fdm.wgrey('pclip=99.9',par))
# ------------------------------------------------------------
End()
```