In this assignment reverse time migration (RTM) is performed on data with and without signal with a direct wave arrival. The earth model for velocity and density shown in Figures 1 and 2 was utilized to construct data shown in Figures 3 and 4 with and without a direct arrival, respectively.

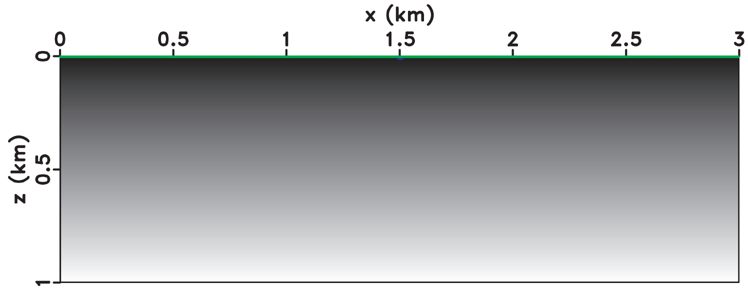


Figure 1: Velocity.

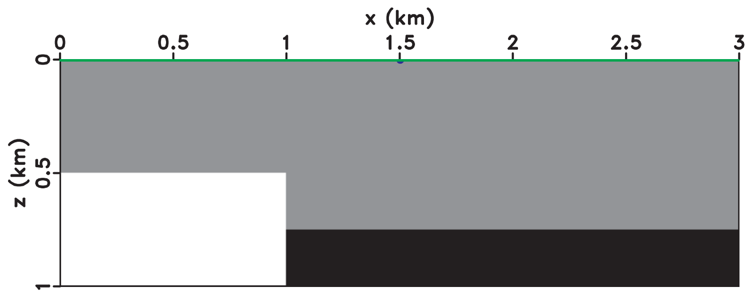


Figure 2: Density.

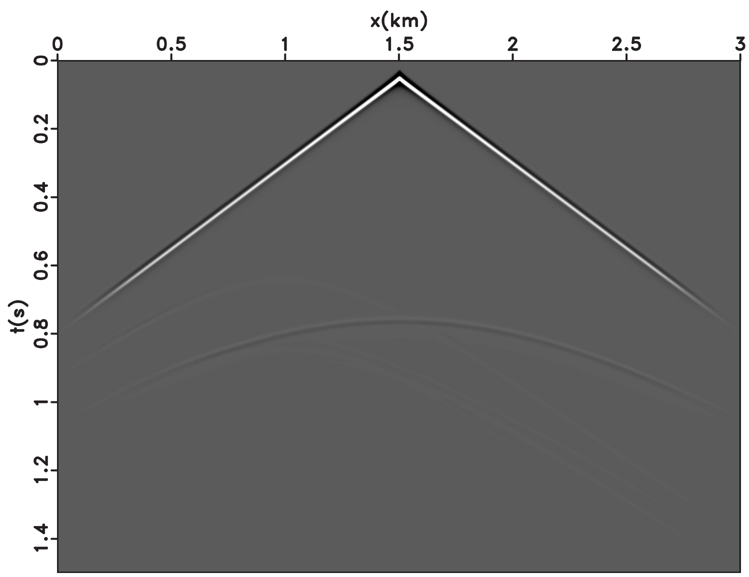


Figure 3: Data w/ direct arrival.

APPENDIX

The plots below were computed using the program `sfcicold2d` which implements cross-correlation between receiver and source wavefields for the imaging condition. The plots from `sfcicold2d` don't exactly match the plots above using a "homebrewed" CIC implementation. The images in the previous section were deduced using "Interferometric cross-correlation" according to the documentation of `sftcor`. Honestly, I don't know exactly how the interferometric cross-correlation works, but it appears to suppress the backscattering effects that contaminate the CIC image using `sfcicold2d` (especially in the case of variable density without a direct wave in the model).

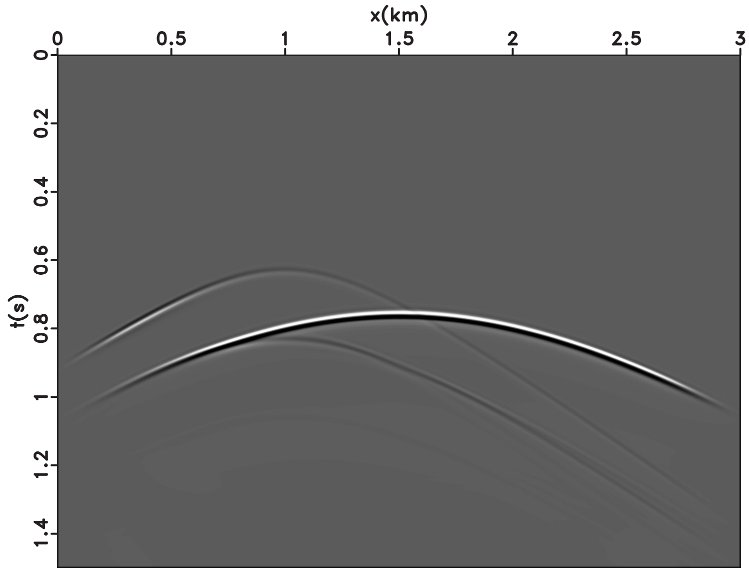


Figure 4: Data w/o direct arrival.

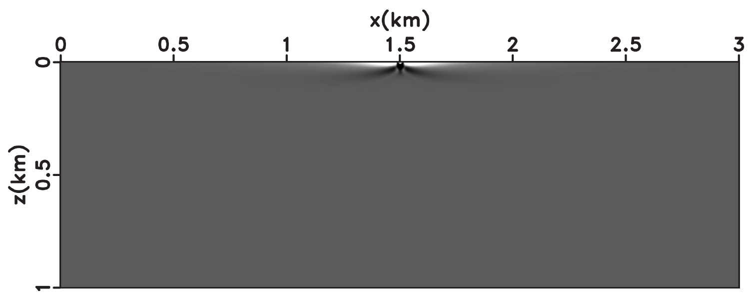


Figure 5: CIC using a constant density model (the incorrect density model) for imaging with direct arrival in data.

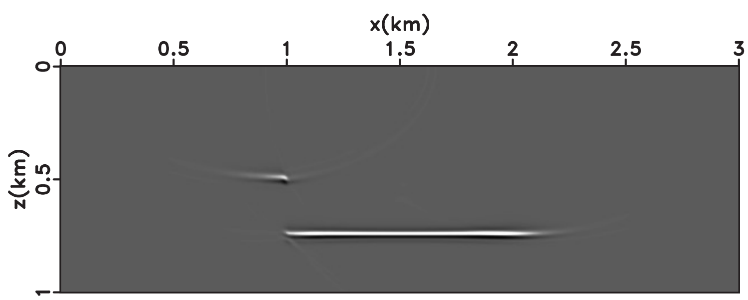


Figure 6: CIC using a constant density model (the incorrect density model) for imaging without direct arrival in data.

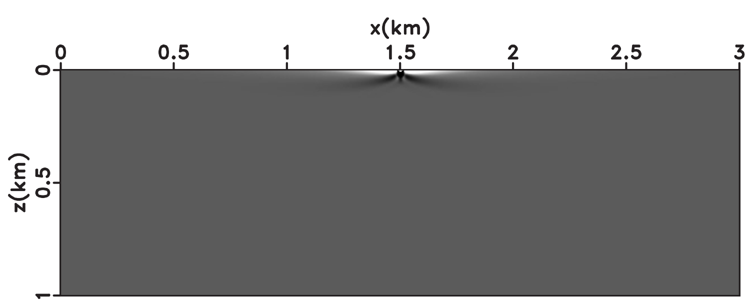


Figure 7: CIC using a variable density model (the correct density model) for imaging with direct arrival in data.

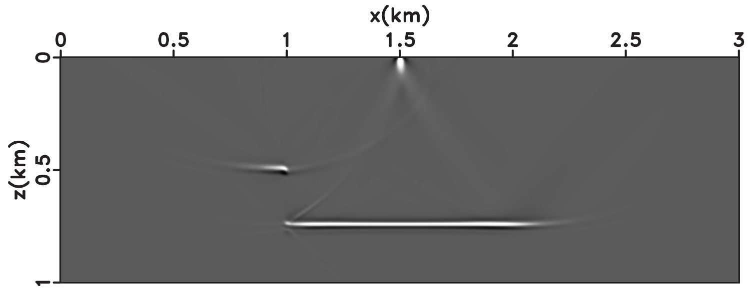
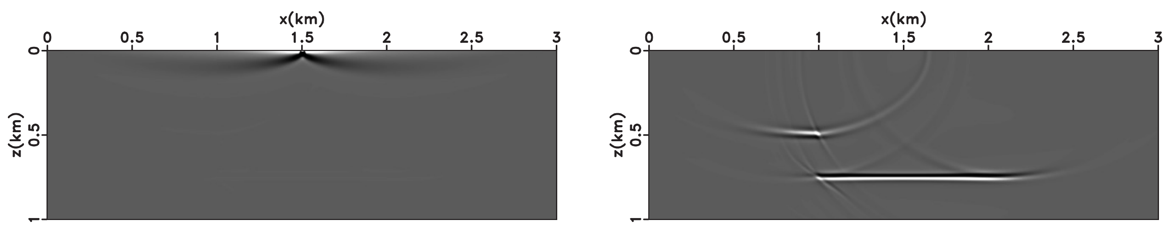
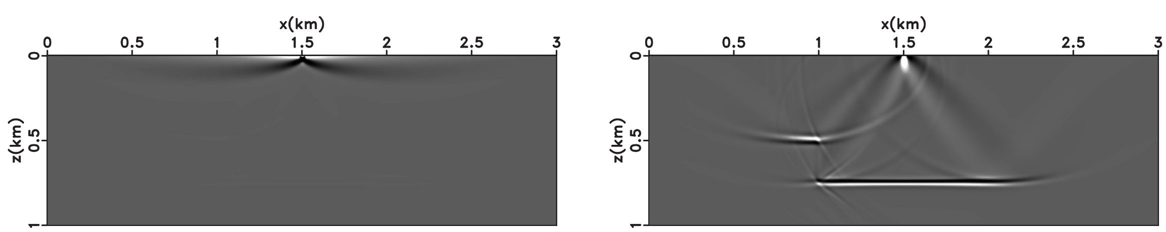


Figure 8: CIC using a variable density model (the correct density model) for imaging without direct arrival in data.



(a) (b)

Figure 9: Images using `sfcicold2d` to implement CIC with a constant density model (the incorrect density model) with and without direct arrival in data.



(a) (b)

Figure 10: Images using `sfcicold2d` to implement CIC with a variable density model (the correct density model) with and without direct arrival in data.

EXERCISE

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.

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

WRAP-UP

After you are satisfied that your document looks ok, print it from the PDF viewer and bring it to class.

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))

# ————————————————————————————————————————————————————————————
##
 # Here add rules for your assignment.
 #
 # use Flow()
 # use Result()
 #
 # find Madagascar programs using the command "sfdoc —k ."
 # find the documentation of Madagascar programs by typing the program name
 ##

# ————————————————————————————————————————————————————————————
## Construct source wavefields
# This step was performed when data was constructed
# ww is wavefield with vertically varying density
```

```
# wo is wavefield with constant density=1

# ----------------------------------------------------------------
## Construct receiver wavefields (four total)
# wb0_reverse - constant density with direct wave
# wb1_reverse - constant density without direct wave
# wa0_reverse - varying density with direct wave
# wa1_reverse - varying density without direct wave
# (takes about 30 seconds a pop at 1500 time steps...)

# flip the data along the time axis both with and without direct wave
# *** ensure that opt=i flag is used to prevent swapping of offset and step
Flow('dr0_reverse','dr0','reverse which=2 opt=i'); # with direct wave
Flow('dr1_reverse','dr1','reverse which=2 opt=i'); # without
Result('dr0_reverse','transp |' + fdm.dgrey('pclip=99.9',par))
Result('dr1_reverse','transp |' + fdm.dgrey('pclip=99.9',par))

# inject data into model
# *** Make sure the source locations are changed to the locations where
#     the data were recorded.
#     (i.e. the receiver locations used to generate data)

# varying density (use ra)
fdm.awefd('dda0','wa0','dr0_reverse','vo','ra','rr','rr','jsnap=1 fsrf=n',par)
fdm.awefd('dda1','wa1','dr1_reverse','vo','ra','rr','rr','jsnap=1 fsrf=n',par)
Result('wa0','window j3=%(jsnap)d |'%par + fdm.wgrey('pclip=99.9',par))
Result('wa1','window j3=%(jsnap)d |'%par + fdm.wgrey('pclip=99.9',par))

# constant density (use rb)
fdm.awefd('ddb0','wb0','dr0_reverse','vo','rb','rr','rr','jsnap=1 fsrf=n',par)
fdm.awefd('ddb1','wb1','dr1_reverse','vo','rb','rr','rr','jsnap=1 fsrf=n',par)
Result('wb0','window j3=%(jsnap)d |'%par + fdm.wgrey('pclip=99.9',par))
Result('wb1','window j3=%(jsnap)d |'%par + fdm.wgrey('pclip=99.9',par))

# transpose wavefields along time axis again (remove later for speed)
Flow('wa0_reverse','wa0','reverse which=4');
Flow('wa1_reverse','wa1','reverse which=4');
Flow('wb0_reverse','wb0','reverse which=4');
Flow('wb1_reverse','wb1','reverse which=4');
Result('wa0_reverse','window j3=%(jsnap)d |'%par + fdm.wgrey('pclip=99.9',par))
Result('wa1_reverse','window j3=%(jsnap)d |'%par + fdm.wgrey('pclip=99.9',par))
Result('wb0_reverse','window j3=%(jsnap)d |'%par + fdm.wgrey('pclip=99.9',par))
Result('wb1_reverse','window j3=%(jsnap)d |'%par + fdm.wgrey('pclip=99.9',par))

# ----------------------------------------------------------------
## Imaging condition 1: Homebrew correlation (CIC)
# note that "ww" has vertically varying density contrast
# note that "wo" has a constant density contrast
# modifying ompchunk doesn't seem to speed things up . . . why?...

#### varying density
###### with direct arrival
Flow('Ra01',['ww','wa0_reverse'],'sftcor ur=${SOURCES[1]} ompchunk=1')
###### without direct arrival
Flow('Ra11',['ww','wa1_reverse'],'sftcor ur=${SOURCES[1]} ompchunk=1')

##### constant density
####### with direct arrival
Flow('Rb01',['wo','wb0_reverse'],'sftcor ur=${SOURCES[1]} ompchunk=1')
####### without direct arrival
Flow('Rb11',['wo','wb1_reverse'],'sftcor ur=${SOURCES[1]} ompchunk=1')


# ----------------------------------------------------------------
## Imaging condition 2: CIC using sfcicold2d
#### *** sfcicop2d was seg-faulting

##### varying density
####### with direct arrival
Flow('Ra02',['ww','wa0'],'sfcicold2d ur=${SOURCES[1]}')
####### without direct arrival
Flow('Ra12',['ww','wa1'],'sfcicold2d ur=${SOURCES[1]}')

##### constant density
####### with direct arrival
Flow('Rb02',['wo','wb0'],'sfcicold2d ur=${SOURCES[1]}')
#Flow('Rb02',['wo','wb0_reverse'],'sfcicold2d ur=${SOURCES[1]} isreversed=y')
####### without direct arrival
Flow('Rb12',['wo','wb1'],'sfcicold2d ur=${SOURCES[1]}')

# ----------------------------------------------------------------

# Placeholder results

#varying density
#Flow('Ra01','Rtemp','math x=${SOURCES[0]} output=x')
#Flow('Ra02','Rtemp','math x=${SOURCES[0]} output=x')
#Flow('Ra11','Rtemp','math x=${SOURCES[0]} output=x')
#Flow('Ra12','Rtemp','math x=${SOURCES[0]} output=x')

#constant density
#Flow('Rb01','Rtemp','math x=${SOURCES[0]} output=x')
#Flow('Rb02','Rtemp','math x=${SOURCES[0]} output=x')
#Flow('Rb11','Rtemp','math x=${SOURCES[0]} output=x')
#Flow('Rb12','Rtemp','math x=${SOURCES[0]} output=x')

#plot
Result('Ra01',fdm.wgrey('pclip=99.9',par))
Result('Ra02',fdm.wgrey('pclip=99.9',par))
Result('Ra11',fdm.wgrey('pclip=99.9',par))
Result('Ra12',fdm.wgrey('pclip=99.9',par))
Result('Rb01',fdm.wgrey('pclip=99.9',par))
Result('Rb02',fdm.wgrey('pclip=99.9',par))
Result('Rb11',fdm.wgrey('pclip=99.9',par))
Result('Rb12',fdm.wgrey('pclip=99.9',par))


End()
```