BASIC MADAGASCAR PROCESSING FLOWS - ACOUSTIC/ELASTIC MODELING

JEFFREY SHRAGGE & AARON GIRARD

Your aim is to develop an SConstruct file that can be used for acoustic and elastic finite-difference time-domain wavefield propagation.

1. IMPORTING THE BP MODEL

You are beginning with a basic SConstruct file. (Recall what first and last line do you need to use?) Start a new basic SConstruct file and add the following script:

```
par = {
   'dx':0.005,'dz':0.005,
   'minx':35,'maxx':45,'minz':0,'maxz':6,
   'f':15,'nt':6000,'dt':0.0005,
   'sx':40,'sz':0.01,'rz':0.00625,
   'jsnap':100,'jdata':10
}

par['kt'] = 1.2/(par['dt']*par['f'])
par['nz'] = par['maxz']/par['dz']
par['nx'] = (par['maxx']-par['minx'])/par['dx']
velsegy = 'vel_z6.25m_x12.5m_exact.segy'
densegy = 'density_z6.25m_x12.5m.segy'
```

If you reached the end of the previous exercise you should recognise the par statement. This parameter dictionary is a useful way of collecting all your parameters into a single structure for quick editing and referencing. Parameter dictionaries are structured as:

```
Dictionary_name = {
   'parameter1_name':parameter1_value,
   'parameter2_name':parameter2_value, ...
}
```

The par['kt'] (e.g.) is the syntax for changing, adding or editing the value of parameters from the dictionary. In this case we are using it to define the value of parameters derived from other parameters. We also define the values of two string variables *velsegy* and *densegy*; both methods of defining variables can be used interchangeably.

```
Fetch(velsegy+'.gz',dir='2004_BP_Vel_Benchmark',
   server='ftp://software.seg.org',top='pub/datasets/2D')
Fetch(densegy+'.gz',dir='2004_BP_Vel_Benchmark',
   server='ftp://software.seg.org',top='pub/datasets/2D')
```

These *Fetch* commands are similar to *Flow*, *Plot* and *Result*; however, rather than processing input from your local directory they 'fetch' files from a remote directory via a public ftp server and download them to the target files in your local directory. For now please

Date: December 13, 2023.

comment the commands out by adding hashes (#) in front of them to tell python to ignore them and copy the files from your USB to your local directory. This is quicker and will avoid overloading the internet in this room. As the names suggest, these are zipped *segy* files containing the velocity and density models we will use in our simulations.

Now copy the following Flow commands into your SConstruct file:

```
#Flow(velsegy,velsegy+'.gz','gzip -d')
#Flow(densegy,densegy+'.gz','gzip -d')
Flow('BPvelocity',velsegy,'segyread |
   put d1=0.00625 label1="Z" unit1="km" d2=0.0125 label2="X" unit2="km"')
Flow('BPdensity',densegy,'segyread |
   put d1=0.00625 label1="Z" unit1="km" d2=0.0125 label2="X" unit2="km" ' %par)
```

The first two flow commands use the Unix utility *gzip* to unzip the velocity and density files. We have commented these out for the same reason we comment the *Fetch* commands above. The second pair of commands uses the Madagascar command *sfsegyread* to read the data from the segy files into rsf format. Unfortunately, *sfsegyread* cannot interpret the header files correctly, so we need to use *sfput* to ensure the correct values are in the rsf header files. If you type *scons* on the command line Madagascar will convert the model files to rsf format. You can use *sfin* to check that the BP velocity and density header files are correct. Try using *sfattr* to look at the properties of the models.

1.1. **Exercise.**

(1) Use a *Plot* command to create a color image of the BP velocity model at a 2:1 ratio.

2. USING PARAMETER DICTIONARIES

The full BP velocity model is too large to use during the workshop, so we will use only a small subsection for our simulations. To do this, enter the Flow command below:

```
Flow('velocity','BPvelocity','''
   window max1=%(maxz)d max2=%(maxx)g min2=%(minx)g |
   add scale=0.001 |
   transp |
   sinc n1=%(nx)d d1=%(dx)g o1=%(minx)g |
   transp ''' %par)
```

This command uses *sfwindow* to extract a subset of the velocity model as defined by the parameter dictionary. It also uses *sfadd* to convert it from m/s to km/s for the modeling code. This is the first command to use the parameter dictionary, which is called by the %par at the end of the command. This tells python to replace all segments with the format %(parameter)c. The character c denotes the format used to display the parameter, with the most useful characters being d, g and s which denote integer, float and string formats.

The *sftransp* and *sfsinc* commands are an ad-hoc approach to convert the velocity model to finer sampling on the x axis. sfsinc is a one-dimensional interpolation code on the first axis, the two sftransp commands are required to make the x-axis the first axis. Sufficiently fine spatial sampling is required in wave-propagation modeling to prevent grid dispersion. If you wish to perform this operation regularly, there is a Madagascar code, *sfresample*, which can be installed (in the development version).

2.1. **Exercises.**

(1) Create a plot of the subsampled velocity model to check that it meets your expectations.
(2) Create a file using *Flow* named *density.rsf* using the corresponding section of the density model.
(3) Create a file using *Flow* named *wav.rsf* using *sfspike* that has a spike at 'kt' then uses *sfricker1* to convolve that spike with a Ricker wavelet of frequency 'f'. Check the output using *sfgraph*.

You will now need to add this *Flow* command to convert the wavelet to the correct format

---

```
Flow('acousticwavelet','wav','transp plane=12 | pad n1=2')
```

Add the following *Flow* commands to define the source and receiver locations:

```
Flow('sx',None,'math n1=1 output=%(sx)g' %par)
Flow('sz',None,'math n1=1 output=%(sz)g' %par)
Flow('source','sx sz','cat axis=1 ${SOURCES[1]}')
Flow('rx',None,'math n1=%(nx)d d1=%(dx)g o1=%(minx)g output=x1' %par)
Flow('rz',None,'math n1=%(nx)d d1=%(dx)g o1=%(minx)g output=%(rz)g' %par)
Flow('receivers','rx rz','cat axis=2 ${SOURCES[1]} | transp')
```

Run *scons source.rsf* and *scons receivers.rsf* and use *sfin* to have a look at their structure to see if they are what you expected. You can also use *sfdisfil* to print the data in the terminal window. They are both two dimensional files that contain the x and z coordinates of the source and receivers, in this case we are only using one source, so the source file only contains 2 x 1 values.

3. ACOUSTIC MODELING

Enter the following *Flow* command:

```
Flow('arecfield awavefield','acousticwavelet density receivers source velocity','''
   awefd2d
   den=${SOURCES[1]}
   rec=${SOURCES[2]}
   sou=${SOURCES[3]}
   vel=${SOURCES[4]}
   wfl=${TARGETS[1]}
   verb=y dabc=y snap=y jsnap=%(jsnap)d jdata=%(jdata)d free=y
   ''' %par)
```

This command uses the acoustic modeling code *sfawefd2d* and all the parameter files we have created to simulate a single shot and output wave-field snapshots to *awavefield.rsf* and the receiver data to *arecfield.rsf*. You should be familiar with the ${SOURCES[]} and ${TARGETS[]} syntax from the previous session. The triple quotes (''') enable the commands to be broken up over multiple lines. Several parameters are defined; you can examine their definitions in the code's self-documentation. The two most important parameters

at this point are *jsnap* and *jdata* which define the sampling rate of the output wavefield snapshots and the receiver data, respectively, as multiples of 'dt'. Run the acoustic modeling by using *scons awavefield.rsf*.

3.1. **Exercises.**

(1) Use *Result* to create a plot of the shot gather from *arecfield.rsf*.
(2) Use *Result* to create a movie of the wave propagating in the model from *awavefield.rsf*.

4. ELASTIC MODELING

Add this command to correctly format the wavelet for elastic modeling:

```
Flow('elasticwavelet','wav','put n2=1 n3=1 | transp plane=13 | pad n2=2')
```

For elastic modeling we need to create the elasticity tensor rather than just using a velocity model:

4.1. **Exercises.**

(1) Create c11.rsf from the velocity and density models. HINT: $C_{11} = \rho V_p^2$.
(2) Use Flow to copy *c11.rsf* to *c33.rsf*. HINT: look up sfcp.
(3) Create a V_s model assuming a Poisson solid, i.e. $V_s = V_p/\sqrt{3}$.
(4) Create *c55.rsf*. HINT: $C_{55} = \rho V_s^2$.
(5) Create *c13.rsf*. HINT: $C_{13} = C_{11} - 2C_{55}$
(6) Create the elasticity model by concatenating (sfcat) it c11, c33, c55 and *c13* along their third axis.
(7) Run the elastic modeling code *sfewefd2d* using ccc = the elastic model instead of the velocity model, remember to use *elasticwavelet.rsf*. HINT: also add the parameter *ssou=y* and switch free to *free=n*.
(8) Separate the x and z components of both the wavefield snapshots and the receiver data.
(9) Use *Result* to view at least one component each of the wave-field snapshots and the shot gathers.
(10) Examine the differences between the acoustic and elastic modeling.

5. EXTRA INVESTIGATIONS

Feel free to do the following in any order:

(1) The shot gathers you generated with the initial parameters are a bit incomplete. Adjust the parameters to model a deeper source.
(2) The parameters weÕve provided produce stable, non-dispersive modeling, experiment with them to see if you can produce these effects. HINT: Reduce the number of time steps you model to reduce the amountof time each run takes.
(3) Try extracting different subsets of the BP model to model waves propagating in different areas.
(4) Some operating systems have a function called /usr/bin/time that enables you to see the running time of individual commands. Experiment with this command to see the effects different parameters have on running time.
(5) Try adjusting the elasticity tensor to see if you can simulate anisotropy.

```
from rsf.proj import *
import math

# Define parameters
par = {
'dx':0.005,'dz':0.005,
'minx':35,'maxx':45,'minz':0,'maxz':6,
'f':15,'nt':6000,'dt':0.0005,
'sx':40,'sz':0.01,'rz':0.00625,
'jsnap':100,'jdata':10
}
par['kt'] = 1.2/(par['dt']*par['f'])
par['nz'] = par['maxz']/par['dz']
par['nx'] = (par['maxx']-par['minx'])/par['dx']
velsegy = 'vel_z6.25m_x12.5m_exact.segy'
densegy = 'density_z6.25m_x12.5m.segy'

# Grab the data from the internet
#Fetch(velsegy+'.gz',dir='2004_BP_Vel_Benchmark',
#server='ftp://software.seg.org',top='pub/datasets/2D')
#Fetch(densegy+'.gz',dir='2004_BP_Vel_Benchmark',
#server='ftp://software.seg.org',top='pub/datasets/2D')

# Prepare the velocity and density files
#Flow('BPvelocity.segy',velsegy+'.gz','gzip -d')
#Flow('BPdensity.segy',densegy+'.gz','gzip -d')
Flow('BPvelocity',velsegy,'segyread | put d1=0.00625 label1="Z" unit1="km" d2=0.0125 label2="X" un
Flow('BPdensity',densegy,'segyread | put d1=0.00625 label1="Z" unit1="km" d2=0.0125 label2="X" uni

# Create a plot of the velocity model
Result('velocity_plot','BPvelocity','grey color=j mean=y scalebar=y title="BP Velocity Model"')

# Subset the velocity model
Flow('velocity','BPvelocity','window max1=%(maxz)d max2=%(maxx)g min2=%(minx)g | add scale=0.001

# Subset the density model
Flow('density','BPdensity','window max1=%(maxz)d max2=%(maxx)g min2=%(minx)g | add scale=0.001 |t

# Plot a subset of the velocity model
Result('velocity_plot_subset','velocity','grey color=j mean=y scalebar=y title="BP Velocity Model
Result('density_plot_subset','density','grey color=j mean=y scalebar=y title="BP Density Model su

# Create the ricker wavelet
#Flow('wav',None,'spike n1=%(nt)d d1=%(dt)g o1=0 k1=%(kt)d | ricker1 frequency=%(f)d ' %par )
Flow('wav',None,'spike n1=%(nt)d d1=%(dt)g o1=0 k1=%(kt)d | ricker1 frequency=%(f)d' % par)
Result('plot_wav','wav','window min1=0.07 max1=0.09 | graph')
Flow('acousticwavelet','wav','transp plane=12 | pad n1=2')

# Put the source and receiver information together
Flow('sx',None,'math n1=1 output=%(sx)g' %par)
Flow('sz',None,'math n1=1 output=%(sz)g' %par)
Flow('source','sx sz','cat axis=1 ${SOURCES[1]}')
```

```
Flow('rx',None,'math n1=%(nx)d d1=%(dx)g o1=%(minx)g output=x1' %par)
Flow('rz',None,'math n1=%(nx)d d1=%(dx)g o1=%(minx)g output=%(rz)g' %par)
Flow('receivers','rx rz','cat axis=2 ${SOURCES[1]} | transp')

# Acoustic wavefield modelling
Flow('arecfield awavefield','acousticwavelet density receivers source velocity',''' 
awefd2d
den=${SOURCES[1]}
rec=${SOURCES[2]}
sou=${SOURCES[3]}
vel=${SOURCES[4]}
wfl=${TARGETS[1]}
verb=y dabc=y snap=y jsnap=%(jsnap)d jdata=%(jdata)d free=y
''' %par)

# Plot the acoustic wavefield results
Result("acoustic_shot_gather","arecfield",'transp plane=12 | grey mean=y gainpanel=a title="Acous
Result("acoustic_wavefield_movie","awavefield",'grey gainpanel=a title="Acoustic wavefield"')

# Elastic modelling
Flow('elasticwavelet','wav','put n2=1 n3=1 | transp plane=13 | pad n2=2')

# Create C11
Flow("C11","velocity density",'math dens=${SOURCES[1]} vp=${SOURCES[0]} output=dens*vp*vp')
Flow("C33","C11",'cp')
Flow("vp","velocity",'cp')

# Create a Vs model assuming a Poisson solid
par['sqrt3']=math.sqrt(3.0)
Flow("vs","velocity",'math vel=${SOURCES[0]} output=vel/%(sqrt3)f' % par)
Flow("C55","density vs",'math dens=${SOURCES[0]} vs=${SOURCES[1]} output=dens*vs*vs')
Flow("C13","C11 C55",'math c11=${SOURCES[0]} c55=${SOURCES[1]} output=c11-2*c55')
Flow("ccc","C11 C33 C55 C13",'cat axis=3 ${SOURCES[1:4]}')

# Do the elastic modelling
Flow('erecfield ewavefield','elasticwavelet density receivers source ccc',
'''
ewefd2d
den=${SOURCES[1]}
rec=${SOURCES[2]}
sou=${SOURCES[3]}
ccc=${SOURCES[4]}
wfl=${TARGETS[1]}
dabc=y snap=y verb=y jsnap=%(jsnap)d jdata=%(jdata)d
ssou=y nb=20 nbell=11
''' %par)

# Separate components
Flow('ewfldZ','ewavefield','window n3=1 f3=0 min2=%(minx)g max2=%(maxx)g max1=%(maxz)g min1=%(min
Flow('ewfldX','ewavefield','window n3=1 f3=1 min2=%(minx)g max1=%(maxz)g min1=%(minz)g' % par)

# Plot components of the Elastic wavefield
Result("ewfld_z_movie","ewfldZ",'grey gainpanel=a title="Elastic wavefield, Z component"')
Result("ewfld_x_movie","ewfldX",'grey gainpanel=a title="Elastic wavefield, X component"')
```

```
End()
```