BASIC MADAGASCAR PROCESSING FLOWS

JEFFREY SHRAGGE

1. SConstruct BASICS

You are beginning with a basic SConstruct file:

```
from rsf.proj import *

% Workflow goes in here

End()
```

Your task will be to create a processing flow by entering various Madagascar *Flow*, *Plot* and *Result* commands between the start and end lines in your *SConstruct* file. Note that text in **bold** (e.g. **scons test.rsf**) are commands that should be executed on the command line in a terminal window.

1.1. Creating basic RSF file.

The first thing we are going to do is to generate a basic RSF file using a **Flow** command. Recall that this has the following structure:

Flow('target','source','command').

where the *sources* and used as input to the *command* to generate the *target*. Open up the *SConstruct* file with your favourite text editor and enter the following command after the first line:

```
Flow('spike',None,
    'spike n1=200 n2=200 o1=0 o2=0 d1=0.005 d2=0.005 label1=Depth unit1=km label2=Time unit2=s')
```

Here we use*None* to say that no input files were needed to create the target. Save your edits it in your text editor (but leave it open for further editing). In a terminal window, use *scons* to build the target *spike.rsf* by entering **scons spike.rsf**. If you entered it correctly, you should have seen something like:

```
/Users/jeffreyshragge/Software/RSF1.5/RSFSRC/bin/sfspike n1=200 n2=200 o1=0 o2=0 d1=0.5
d2=0.5 label1=Z unit1=km label2=X unit2=km > spike.rsf
```

You may have noticed that *scons* called *sfspike* not *spike*. The extra *sf* has been prepended automatically. You could have hard coded the directory path into the command; however, you would have to write out the full program name (i.e. *sfspike*) and this would make your work hard to reproduce on another computer!

Date: August 10, 2013.

```
Flow('spike2',None,
    '/Users/jeffreyshragge/Software/RSF1.3/RSFSRC/bin/sfspike n1=200 n2=200 o1=0 o2=0 d1=0.005 d2=
```

You could also call non-Madagascar programs (e.g. scripts). You can check to see whether the file dimensions are correct by entering **sfin spike.rsf**. You can also type see some of the attributes by entering **sfattr < spike.rsf**.

Do the following:

(1) Write a Flow command that writes something to the terminal (e.g. using *echo*)

1.2. **First piping.** You notice that you have entered the incorrect d1 and d2 numbers above. Let's correct this goof by using the *sfput* command. Note: you can get information about any particular RSF program by simply entering its name - e.g. type **sfput**. Insert the following *Flow* command into your *SConstruct* file:

```
Flow('newspike','spike','put d1=0.005 d2=0.005')
```

Enter **scons newspike.rsf** and then check that the new d1 and d2 variables have been updated by **sfin newspike.rsf**. Note that we could have also written this command as a combination of two separate commands that are linked by a "pipe". Put the following command into your *SConstruct* file.

```
Flow('newspike2','spike','put d1=0.005 | put d2=0.005')
```

In this case the Unix pipe takes the standard output (*stdout*) of the first command and uses it as standard input (*stdin*) to the second command. A great advantage of using pipes is that data processing commands can be strung together in a single *Flow* command! Compare the two implementations by entering **sfin newspike.rsf newspike2.rsf**

1.3. **Using Strings.** You notice that you have put the wrong units into your spikes! Let's fix this up by changing the label using the following command:

```
xlabel='Depth'
Flow('newspike3','newspike2','put label1='+xlabel)
```

In this case, the *+xlabel* is not arithmetic addition; rather, it is a (Python) concatenation of two strings. Running **scons newspike3.rsf** should generate a message like

```
  < newspike2.rsf /Users/jeffreyshragge/Software/RSF1.3/RSFSRC/bin/sfput label1=Depth
> newspike3.rsf
```

1.4. **Making a constant velocity model.** We are now going to take *newspike3.rsf* and make a 2D velocity model. Let's start with a constant $v_0 = 2$ km/s background. To do this we are going to use *sfmath* which is a very useful command to know. Insert the following into your *SConstruct* file:

```
Flow('V0','newspike3','math output="2" ')
```

Run **scons V0.rsf** and look at the attributes using **sfattr < V0.rsf**. You'll notice that the min/max values are all equal to 2. This command took the input file, set every element equal to 2, and then wrote out a file with the same dimensions as the input file. The *sfspike* command is very useful for making dummy files of certain size.

1.5. **V(z) Velocity.** We are now going to make the velocity model a little bit more interesting by adding a $v = v(z)$ velocity gradient. We can do this by revisiting the command we just did. Enter the following into your *SConstruct*:

```
Flow('VZ','V0','math output="input+0.5*x1" ')
```

This *sfmath* call will take the input file and add 0.5 times the x1 axis (i.e. z) that ranges from 0 to 0.995. Execute this command with **scons VZ.rsf**. You will often want to use variables in your SConstruct file (e.g., the velocity gradient above), and then substitute its value into a Madagascar command. Here's an example you can try:

```
vz=0.5
Flow('VZ2','V0','math output="input+%g*x1" '%vz)
```

Here, we define a (float) variable *vz* and then use it in a *Flow* command, substituting it in place of *%g*. Note that *%g* indicates that it is a float; you might also want to use *%d* for an integer and *%s* for a string.

1.6. **Plotting.** We haven't yet generated any graphical output. Let's take a look at the $v = v(z)$ velocity model we just generated. Enter the following:

```
Plot('VZ','VZ','grey color=j mean=y scalebar=y pclip=100 title="V(z)" ')
```

and then run **scons VZ.vpl; sfpen VZ.vpl**. We have called a new Madagascar command: *Plot*. This will create a .vpl (vector plot) file that can be viewed with the *sfpen* or *xtpen* programs. We have also passed a number of different variables to the *sfgrey* program. To see a listing of those you can type **sfgrey**. There are a number of other standard graphics plots in **sfdoc stdplot**.

1.7. **Independent Tasks.** Do the following:

(1) Write SConstruct commands to include both v_z and v_x gradients, and the plot the result using *Plot* into a file called *VXZ.vpl*.
(2) Change the colour maps of your plots to be shades of red, white and blue.

1.8. **Combining Figures.** You can combine more than one plot into a single graphics file. Here are three common usages:

```
Result('VEL','VZ VXZ','Movie')
Result('VEL2','VZ VXZ','SideBySideAniso')
Result('VEL2','VZ VXZ','OverUnderAniso')
```

You can look at them by entering **scons VEL.view** , **scons VEL2.view** and **scons VEL3.view**. Here I have used the Madagascar *Result* command, which has built the .vpl files and put them into the *./Fig/* directory. Note that you could also view them together with *sfpen Fig/VEL∗.vpl*. Use the *s* and *f* keys to make the movie frames go by slower or faster, respectively.

1.9. **Gaussian Perturbation.** Do the following:

(1) Use *sfmath* to generate a RSF file called *Gauss.rsf* that is a Gaussian velocity perturbation in the centre of the model. HINT: A Gaussian can be formed by:

$$g(x,z) = A_0 \mathrm{exp}\left(-\frac{(x-x_0)^2}{\sigma_x^2} - \frac{(z-z_0)^2}{\sigma_z^2}\right) \tag{1}$$

where A_0 is an amplitude, x_0 and z_0 are fixed coordinates, and σ_x and σ_z are standard deviations.

1.10. **Multiple inputs.** We'd like to make a composite velocity model using the *VZ.rsf* and *Gauss.rsf* files. This can easily be accomplished with the *sfadd* command.

```
Flow('GaussVz','Vz Gauss','add ${SOURCES[1]}')
```

You'll notice that there are two input SOURCES to generate a single TARGETS. The different files can be called using (Python) indexing of the SOURCES. Thus, ${SOURCES[0]} refers to file *Vz.rsf* and ${SOURCES[1]} refers to file *Gauss.rsf*. Another way you could have done this is with the following *sfmath* command, e.g.,

```
Flow('GaussVz2',['Vz','Gauss'],'math gauss=${SOURCES[1]} output="input+gauss" ')
```

I have used square brackets in the SOURCES area, which is another way to write it in Python. Often there are multiple ways in Madagascar to generate the same result!

You might also run into a situation where you need multiple inputs and multiple outputs. These can be done by passing extra tags (below *otherin* and *otherout*) and assigning them ${SOURCES[1]} and ${TARGETS[1]}

```
Flow('out1 out2,'in1 in2','command otherin=${SOURCES[1]} otherout=${TARGETS[1]}')
```

1.11. **Independent Tasks.** Do the following:

(1) Generate a figure of the file *GaussVz.rsf* with appropriate tile and correct aspect ratio (HINT: screenratio) using a *Result* command.
(2) Use the *sfwindow* and *sfcat* commands to taking the left half of the **GaussVz.rsf** velocity model and swap it to the right side of the velocity model.

1.12. **Looking ahead.** One thing about the above exercises is that we have hard-coded a lot of variable values into out SConstruct file. One way to deal with this is to use *parameter dictionaries* that can be defined in one place and then substituted into the various *Flow*, *Plot* and *Result* commands. For example, we could have defined a dictionary called *par* at the start of the *SConstruct* and then written the first *Flow* command as:

```
par = {
'n1':200,'n2':200,
'ox':0.,'oz':0.,
'dx':0.5,'dz':0.5,
'l1':'Depth',
'l2':'Time',
'u1':'km',
'u2':'s',
}

Flow('spike',None,
    '''
    spike n1=%(nz)d n2=%(nx)d o1=%(oz)g o2=%(ox)g d1=%(dz)g d2=%(dx)g
    label1=%(l1)s label2=%(l2)s unit1=%(u1)s unit2=%(u2)s
    '''%par)
```

Here, all of the parameter values are now passed at the end of the *Flow* command with the inclusion of *%par*. This is a great way to make sure you keep a consistent set of values throughout your *SConstruct* file! Note also the triple quotes are just another way you can write the various commands. You can write free-form within this region.

Do the following:

(1) Check your reproducibility of your work. Enter **scons -c** to clean, then rebuild all your work with **scons** and look at figures with **scons view**
(2) Backup your *SConstruct* under a different name, and then rewrite your *SConstruct* file placing all of the variables you used into a parameter dictionary.

```
from rsf.proj import *
# . . Generate a 2D matrix
Flow('spike',None,
        'spike n1=200 n2=200 o1=0 o2=0 d1=0.5 d2=0.5 label1=Depth unit1=km label2=Time unit2=s')

# . . Correct the axis sampling
Flow('newspike','spike','put d1=0.005 d2=0.005')

# . . Correct the axis sampling (again)
Flow('newspike2','spike','put d1=0.005 | put d2=0.005')

# . . Example of using string concatenation
xlabel='Depth'
Flow('newspike3','newspike2','put label1='+xlabel)

# . . Generate velocity model
Flow('V0','newspike3','math output="2" ')

# . . Create V of Z
Flow('VZ','V0','math output="input+0.5*x1" ')

# . . Again substituting a float number
vz=0.5
Flow('VZ2','V0','math output="input+%g*x1" '%vz)

# . . Plot result
Plot('VZ','VZ','grey color=j mean=y scalebar=y pclip=100 title="V(z)" ')

# . . Horizontal gradients too!
Flow('VXZ','V0','math output="input+0.5*x1+0.5*x2" ')

# . . Plot of both gradients
Plot('VXZ','VXZ','grey color=j mean=y scalebar=y pclip=100 title="V(x,z)" ')

# . . Example of different plots
Result('VEL','VZ VXZ','Movie')
Result('VEL2','VZ VXZ','SideBySideAniso')
Result('VEL3','VZ VXZ','OverUnderAniso')

# . . Make Gaussian perturbation
Flow('Gauss','VZ','math output="exp(-49*(x1-0.5)^2-25*(x2-0.5)^2)" ')

# . . Add two files together and plot
Flow('GaussVz','Vz Gauss','add ${SOURCES[1]}')
Result('GaussVz','grey color=j mean=y pclip=100 scalebar=y')

# . . A different way to do it
Flow('GaussVz2','Vz Gauss','math gauss=${SOURCES[1]} output="input+gauss" ')
Result('GaussVz2','grey color=j mean=y pclip=100 scalebar=y')
End()
```