Seismic Processing
Prac 2 - Processing a synthetic dataset.

ERTH3021

October 14, 2014

This is the second of two pracs on seismic data processing. In this prac we will be processing the dataset generated in prac 1.

If you do not have the dataset from prac 1 is it available for download at **http://goo.gl/YuZgE9**

As with prac 1 code templates have been provided. The prac can be downloaded from
https://github.com/stuliveshere/Seismic-Processing-Prac2

1 Introduction

Seismic data processing involves taking raw seismic data and generating an image which can be related to the geology targeted by the seismic survey. Seismic processing usually involves a number of steps, including

- velocity analysis
- geometric corrections (normal moveout, statics, migration)
- amplitude corrections (amplitude recovery, AGC)
- noise attenuation
- CMP stacking
- deconvolution

Although not necessarily in this order.

For this prac we will use the following processing sequence:

1. load and QC (quality control) data
2. sort into CDP gathers
3. amplitude recovery
4. move out with constant velocity
5. field stack (aka brute stack)
6. first velocity analysis
7. pre-stack noise attenuation
8. first velocity stack
9. post-stack noise attenuation

This is somewhat like a processing sequence used in a commercial processing house. Note the iterative nature of the sequence - this is a common trait of seismic processing.

The final image quality is influenced by the accuracy of the seismic velocities and the processing parameters used. Inaccurate velocities can lead to imperfect moveout which can affect stack quality. Poor selection of processing parameters can harm the data - for example a harsh noise attenuation routine might attenuate the signal as well as the noise. Finding the optimum processing parameters is the art of processing, and it usually involves significant amounts of testing.

In this prac you will be picking your own velocities and selecting your own processing parameters. You will be expected to test a range of parameters and discuss why you chose those parameters in your report.

Exercise 1: Load data and perform QC

In this exercise we will create an initialization routine which will load our dataset into numpy, and set up some initial parameters. We will then have a look at some raw shot gathers to ensure the data is loaded correctly (QC).

Exercise 2: Sort into CMPs

CMP gathers collect all traces which pass through the same CMP into a single gather (collection of traces). CMP gathers are generally used for velocity analysis and stacking. The number of traces in a CMP gather is referred to as the fold.

Sort the dataset into CMP gathers and perform QC. Compare the CMP gathers from this exercise to the shot gathers from exercise 1 and discuss why they look different.

Extract one CDP gather as a file for later testing.

Exercise 3: Amplitude Recovery

We have already seen in the raw gathers that deeper signals are weaker and much more difficult to see. In the last prac we compensated for this using an AGC. An AGC normalizes the amplitude in a sliding window over every trace, based upon the amplitudes in the window. Sometimes in seismic processing we use an AGC prior to stack, but sometimes we want to preserve the relative amplitude of samples between traces. We call this amplitude recovery.

In theory we could correct directly for spherical divergence, but it is much more challenging to correct for the remaining sources of attenuation. Thus a generic, tunable formula is generally used, where testing is used to fine tune the result until it "looks right". One such formula is

$$R = e^{\gamma t}$$

,
where

- R is the amplitude correction
- γ is the tunable parameter
- t time

Write a python function which implements this formula, and apply to your test gather. Test various values of γ such that the amplitude of the bottom reflector in the gather is closer in amplitude to the top reflectors. Show your gather before and after amplitude recovery.

Hint: e^x is called by the numpy function np.exp().

Exercise 4: Normal moveout correction

Notice in the raw CMP gathers the reflectors have a hyperbolic shape. In order to stack a CMP gather the reflectors need to be flat. The reason for the hyperbolic shape is normal moveout, and flattening reflectors is generally referred to as normal moveout correction. For a flat, horizontal reflector, the travel-time equation is

$$t_x{}^2 = t_0{}^2 + \frac{x^2}{v^2}$$

where

- t_x is the travel time at offset x
- t_0 is the travel-time at offset 0

- x is the offset
- v is the velocity

Write a python function which implements the normal moveout correction. That is, re-arrange the above equation so that we can calculate the zero offset travel-time t_0 for some offset and velocity.

The function which applies your move out correction to the dataset has been supplied. Using your test CMP gather, try to find a single velocity (aka constant velocity) which flattens the reflectors the most. Show that gather before and after moveout correction. Discuss the issues associated with normal moveout with a constant velocity.

Note the code relating to "stretch". The amount of moveout correction is not linear with time. Slow, shallow data tends to suffer from significant stretching. Experiment with stretch values on your test CMP gather. Show a gather with and without stretch. Discuss the potential impact of stretched data on a stack.

Exercise 5: Field Stack

Stacking a gather involves taking all of the traces in a single CMP gather and summing them together into once trace. The amplitude of the trace is then normalized by the fold of the gather. Thus stacking can also be considered as finding the average trace at a given CMP location. When all CMP gathers are stacked, an image can be built with one trace at each CMP location. This image is generally referred to as a stack or a stacked section.

Write a function which will stack any gather given to it. Note this can be done in one line.

Hint: mean $= \bar{x} = \frac{x_1+x_2+...+x_n}{n}$

An even bigger hint: think about the axis you want to use numpy.mean() along

Once you have stacking working, you are ready to do a field stack. Initialise the entire seismic volume, apply TAR, NMO, and then stack. Ask the tutors to check your code before you run it - running the NMO will take a few minutes. Try it with and without pre-stack AGC. What are some of the problems with the stack? What is the large feature above the coals?

Exercise 6: First Velocity Analysis

In exercise 4 we moved out the test CDP with a constant velocity. A more realistic scenario would be to pick a number of velocities, one for each layer. One method of determining the optimal stacking velocities is known as semblance analysis. Essentially, a given CDP is moved out and stacked with a range of velocities. The best stacking velocity will result in the highest amplitude stack for a given reflector. By visualizing all the velocities at once, you can see exactly where the maximum stacking energy occurs.

A simplistic semblance routine has been given to you in exercise 6. Run exercise 6 on your test CDP. Pick velocities and times for the 3 layers, and make a note of these for later. What is the large high energy event which can also be seen in the semblance plot?

Exercise 7: Noise attenuation - Top Mute

As seen in the stack in exercise 5, and in the semblance plot in exercise 6, some of the refractor energy is leaking through the stack. There are a number of different ways we could attenuate or even entirely remove the refractor. The refractor often has lower frequency than our reflectors, so a frequency filter might knock back the high amplitude low frequency components. But in this case the refractor is very linear and can be targeted by a mute window.

We could calculate the dip of the refractor, come up with a geometric formula, calculate the time to refractor at a given geophone, and then apply a mute, but an easier method is to use what is called a linear move-out (LMO). An LMO is similar in concept to an NMO, but instead of flattening a hyperbolic reflector it flattens a linear event. Because the event is linear, the travel-time equation is our old friend the velocity equation

$$v = \frac{x}{t}$$

where

- v = velocity

- x = (absolute) offset
- t = time

Derive an equation which will apply a linear moveout *correction* to a dataset.

Hint: we want to remove the dip. Remove kind of sounds like we want to subtract something

The function which applies your linear moveout correction to a dataset has been supplied. Experiment with a couple of velocities to make sure it's working ok. Calculate the exact velocity which will flatten the refractor. *Hint:*$\Delta v = \Delta x/\Delta t$

Once you have your refractor nice and flat, we will need to mute it out. Find the time of the now flat refractor, and overwrite the refractor with zeros.

You will now need to reverse the LMO so the CDP gather looks normal again. Show your CDP gather before LMO, after apply LMO, after muting, and after removing the LMO. Describe how you calculated the LMO velocity.

Exercise 8 : First Velocity stack

You now have all the components to generate a minimally processed stack with picked velocities. This stack is sometimes referred to as the "first velocity stack". You will need to

1. initialise the full volume
2. apply the amplitude recovery
3. apply LMO to mute refractor
4. apply NMO to flatten refractors
5. apply AGC so we can see it better
6. stack volume
7. view

Make sure you have used all your parameters correctly from previous exercises - you dont want to have to re-run your code. The tutors will check your code before you start. You will also want to output the stack as a file, so that you dont have to re-run your code for the next exercise.

Compare your first velocity stack to the brute stack we generated earlier. What are the main differences? Discuss why we applied the LMO before the NMO.

Exercise 9: Post-stack noise attenuation

Consider what happens when you run a smoothing window along a trace. Events which are shorter than the smoothing window will tend to be smoothed out - events which are longer than the smoothing window will tend to be left along. Now consider what happens when you run a smoothing window along a time slice, between traces. Events shorter than our smoothing window, for example random noise, will tend to be attenuated, and events longer than our window will tend to be left alone. This sounds like a good thing, so lets try it.

The process we are going to use is sometimes referred to as trace mixing. This code has been supplied to you in exercise 9. Experiment with mixing window sizes to find one that works well. What are some of the downsides of trace mixing?

Extra marks: perform the trace mixing pre-stack. What are the pros and cons of a pre-stack trace mix?

```
#import su files from prac1
#view the gathers to make sure
#they imported properly

from toolbox import io
import toolbox
import numpy as np
import os
import matplotlib.pyplot as pylab
import pprint


#-----------------------------------------------------------------------
#               useful functions
#-----------------------------------------------------------------------

def initialise(file):
        #intialise empty parameter dictionary
        #kwargs stands for keyword arguments
        kwargs = {}
        #load file
        dataset = toolbox.read(file)

        #allocate stuff
        dataset['cdp'] = (dataset['gx'] + dataset['sx'])/2.
        kwargs['ns'] = 1000
        kwargs['dt'] = 0.001

        #also add the time vector - it's useful later
        kwargs['times'] = np.linspace(0.001, 1.0, 1000)
        return dataset, kwargs


#-----------------------------------------------------------------------
#               main functions
#-----------------------------------------------------------------------

if __name__ == "__main__":
        #intialise workspace and parameter dictionary
        workspace, params = initialise('survey.su')

        #the scan tool scans the file headers for non-zero values
        toolbox.scan(workspace)

        #the scroll tool allows you to skip through an entire volume, but
        #you have to define the sort keys, and step size
        params['primary'] = 'sx'
        params['secondary'] = 'gx'
        params['step'] = 20

        #the scroll tool is like the display tool, but
        #you can use the left and right arrows to
        #scroll through a volume
        toolbox.scroll(workspace, None, **params)

        #the viewing tools have been changed so that
        #you can view more than one thing at once.
        #but you need to put this at the end.
        pylab.show()
```

../exercise1.py

```
#sort into cdp/offsets
#view a cdp gather

from toolbox import io
import toolbox
import numpy as np
import os
import matplotlib.pyplot as pylab
from exercise1 import initialise


#-----------------------------------------------------------------------
#               useful functions
```

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

None


#------------------------------------------------------------------------------
#               main functions
#------------------------------------------------------------------------------

if __name__ == "__main__":
        #intialise workspace and parameter dictionary
        workspace, params = initialise('survey.su')

        # we are going to pull out 1 cdp for testing with.
        #firstly, use the scroll tool to view the cdps, and
        #then pick one near the middle of the volume
        params['primary'] = 'cdp'
        params['secondary'] = 'offset'
        params['step'] = 20
        #toolbox.scroll(workspace, None, **params)

        #we then want to extract that single cdp
        #for testing with later. we can do that
        #the following way
        cdp201 = workspace[workspace['cdp'] == 201]
        #view it
        #toolbox.agc(cdp201, None ,**params)
        #toolbox.display(cdp201, None, **params)

        #we have the right cdp = but the traces are in the wrong
        #order. lets sort by offset

        cdp201 = np.sort(cdp201, order=['cdp', 'offset'])

        #output it for later
        toolbox.cp(cdp201, 'cdp201.su', None)
        params['clip'] = 1e-6
        toolbox.display('cdp201.su', None, **params)


        pylab.show()
```

../exercise2.py

```
#import su files from prac1
#sort into cdp/offsets
#view a cdp gather

from toolbox import io
import toolbox
import numpy as np
import os
import matplotlib.pyplot as pylab
from exercise1 import initialise


#------------------------------------------------------------------------------
#               useful functions
#------------------------------------------------------------------------------
@io
def tar(dataset, **kwargs):
  #pull some values out of the
  #paramter dictionary
  gamma = kwargs['gamma']
  t = kwargs['times']

  #calculate the correction coeffieicnt
  r  = np.exp(gamma * t)

  #applyt the correction to the data
  dataset['trace'] *= r
  return dataset


#------------------------------------------------------------------------------
#               main functions
```

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

if __name__ == "__main__":
  #intialise workspace and parameter dictionary
  workspace, params = initialise('cdp201.su')

  #set the value of gamma you want to test here
  params['gamma'] = 10
  #and apply
  tar(workspace, None, **params)

  #we cant use agc to look at this.  The display
  #function has been modified so you can adjust
  #the clip
  params['clip'] = 1e-4
  toolbox.display(workspace, None, **params)
  pylab.show()
```

../exercise3.py

```
#write a function which will flatten a cdp

from toolbox import io
import toolbox
import numpy as np
from exercise1 import initialise
import pylab
import cProfile

#------------------------------------------------------------------------------
#               useful functions
#------------------------------------------------------------------------------

def _nmo_calc(tx, vels, offset):
  '''calculates the zero offset time'''
  t0 = np.sqrt(tx*tx - (offset*offset)/(vels*vels))
  return t0

@io
def nmo(dataset, **kwargs):
  it = np.nditer(dataset, flags=['f_index'])
  for trace in it:
    index= it.index
    aoffset = np.abs(trace['offset']).astype(np.float)
    ns = kwargs['ns']
    dt = kwargs['dt']
    tx = kwargs['times']

    #calculate time shift for each sample in trace
    t0 = _nmo_calc(tx, kwargs['vels'], aoffset)

    #calculate stretch between each sample
    stretch = 100.0*(np.pad(np.diff(t0),(0,1), 'reflect')-dt)/dt

    #filter stretch
    filter = [(stretch >0.0) & ( stretch < kwargs['smute'])]
    values = np.interp(tx, t0[filter], trace['trace'][filter])
    values[tx < np.amin(t0[filter])] = 0.0
    values[tx > np.amax(t0[filter])] = 0.0

    #write corrected values back to dataset
    dataset[index]['trace'] *= 0
    dataset[index]['trace'] += values
  return dataset

#------------------------------------------------------------------------------
#               main functions
#------------------------------------------------------------------------------

if __name__ == "__main__":
  #intialise workspace and parameter dictionary
  workspace, params = initialise('cdp201.su')

  #set stretch mute and vels
  params['smute'] = 10000.0
  params['vels'] = toolbox.build_vels([0.5], [1500])
```

```
  nmo(workspace, None, **params)


  toolbox.agc(workspace, None, None)
  toolbox.display(workspace, None, None)
  pylab.show()
```

../exercise4.py

```

from toolbox import io
import toolbox
import numpy as np
from exercise3 import tar
from exercise4 import nmo
from exercise1 import initialise
import pylab

#-----------------------------------------------------------------------
#               useful functions
#-----------------------------------------------------------------------

def _stack_gather(gather):
  '''stacks a single gather into a trace.
  uses header of first trace. normalises
  by the number of traces'''
  gather['trace'][0] = np.mean(gather['trace'], axis=-2)
  return gather[0]

@io
def stack(dataset, **kwargs):
  cdps = np.unique(dataset['cdp'])
  sutype = np.result_type(dataset)
  result = np.zeros(cdps.size, dtype=sutype)
  for index, cdp in enumerate(cdps):
    gather = dataset[dataset['cdp'] == cdp]
    trace = _stack_gather(gather)
    result[index] = trace
  return result

if __name__ == '__main__':
  #initialise your test cdp first
  workspace, params = initialise('survey.su')

  #first do the true amplitude recovery
  params['gamma'] = 10
  tar(workspace, None, **params)

  #then apply NMO
  params['smute'] = 100.0
  params['vels'] = toolbox.build_vels([0.5], [1500])
  nmo(workspace, None, **params)

  #we will apply a pre-stack agc
  toolbox.agc(workspace, None, None)

  #stack it
  section = stack(workspace, None, **params)

  #display it
  #params['clip'] = 1e-5
  toolbox.display(section, None, **params)

  pylab.show()
```

../exercise5.py

```
#import su files from prac1
#sort into cdp/offsets
#view a cdp gather

from toolbox import io
import toolbox
import numpy as np
import os
```

```python
import matplotlib.pyplot as pylab
from exercise1 import initialise
from exercise3 import tar
from exercise4 import nmo
from exercise5 import _stack_gather


#-----------------------------------------------------------------------
#               useful functions
#-----------------------------------------------------------------------

def semb(workspace,**kwargs):
  vels = kwargs['velocities']
  nvels = vels.size
  ns = kwargs['ns']
  result = np.zeros((nvels,ns),'f')
  for v in range(nvels):
    panel = workspace.copy()
    kwargs['vels'] = np.ones(1000, 'f') * vels[v]
    nmo(panel, None, **kwargs)
    toolbox.agc(panel, None, None)
    result[v,:] += np.abs(_stack_gather(panel)['trace'])


  pylab.imshow(result.T, aspect='auto', extent=(min(vels), max(vels),1.,0.), cmap='spectral')
  pylab.xlabel('velocity')
  pylab.ylabel('time')
  pylab.colorbar()
  pylab.show()


#-----------------------------------------------------------------------
#               main functions
#-----------------------------------------------------------------------

if __name__ == "__main__":
  workspace, params = initialise('cdp201.su')
  velocities = np.arange(800, 4000, 50)
  params['velocities'] = velocities
  params['smute'] = 100
  semb(workspace, **params)
  v = [1417, 1510,1878]
  t = [0.171, 0.215, 0.381]
  params['vels'] = toolbox.build_vels(t, v)
  nmo(workspace, None, **params)
  toolbox.agc(workspace, None, None)
  toolbox.display(workspace, None, None)


  pylab.show()
```

../exercise6.py

```python
#write a function which will flatten a cdp

from toolbox import io
import toolbox
import numpy as np
from exercise1 import initialise
import pylab

#-----------------------------------------------------------------------
#               useful functions
#-----------------------------------------------------------------------

def _lmo_calc(aoffset, velocity):
        t0 = -1.0*aoffset/velocity
        return t0

@io
def lmo(dataset, **kwargs):
        for index, trace in enumerate(dataset):
                aoffset = np.abs(trace['offset']).astype(np.float)
                ns = trace['ns']
                dt = trace['dt'] * 1e-6
                tx = np.linspace(dt, dt*ns, ns)
```

```
                #calculate time shift
                shift = _lmo_calc(aoffset, kwargs['lmo'])
                #turn into samples
                shift  = (shift*1000).astype(np.int)
                #roll
                result = np.roll(trace['trace'], shift)
                dataset[index]['trace'] *= 0
                dataset[index]['trace'] += result
        return dataset

#-----------------------------------------------------------------------
#               main functions
#-----------------------------------------------------------------------

if __name__ == "__main__":
        workspace, params = initialise('cdp201.su')

        params['lmo'] =2200.0
        toolbox.agc(workspace, None, None)
        lmo(workspace, None, **params)
        workspace['trace'][:,80:110].fill(0)
        params['lmo'] =-2000.0
        lmo(workspace, None, **params)

        toolbox.display(workspace, None, None)
        pylab.show()
```

../exercise7.py

```
#restack the data with new vels
#and the refractor muted out
from toolbox import io
import toolbox
import numpy as np
import matplotlib.pyplot as pylab
from exercise1 import initialise
from exercise3 import tar
from exercise4 import nmo
from exercise5 import stack
from exercise7 import lmo


#-----------------------------------------------------------------------
#               useful functions
#-----------------------------------------------------------------------

None


#-----------------------------------------------------------------------
#               main functions
#-----------------------------------------------------------------------

if __name__ == "__main__":
        #intialise workspace and parameter dictionary
        workspace, params = initialise('survey.su')

        #set our TAR
        params['gamma'] = 10
        tar(workspace, None, **params)


        #apply LMO
        params['lmo'] =2200.0
        lmo(workspace, None, **params)
        workspace['trace'][:,80:110] *= 0
        params['lmo'] = -2200.0
        lmo(workspace, None, **params)

        #apply our NMO
        params['smute'] = 100.0
        v = [1417, 1510,1878]
        t = [0.171, 0.215, 0.381]
        params['vels'] = toolbox.build_vels(t, v)
        nmo(workspace, None, **params)

```

```
        #apply AGC
        toolbox.agc(workspace, None, **params)

        #stack
        stack(workspace, 'stack1.su', **params)

        #view
        toolbox.display('stack1.su', None, **params)

        pylab.show()
```

../exercise8.py

```
#restack the data with new vels
#and the refractor muted out
from toolbox import io
import toolbox
import numpy as np
import matplotlib.pyplot as pylab
from exercise1 import initialise
from exercise3 import tar
from exercise4 import nmo
from exercise5 import stack
from exercise7 import lmo


#-----------------------------------------------------------------------
#               useful functions
#-----------------------------------------------------------------------

@io
def trace_mix(dataset, **kwargs):
        ns = kwargs['ns']
        window = np.ones(kwargs['mix'], 'f')/kwargs['mix']
        for i in range(ns):
                dataset['trace'][:,i] = np.convolve(dataset['trace'][:,i], window, mode='same')
        return dataset



#-----------------------------------------------------------------------
#               main functions
#-----------------------------------------------------------------------

if __name__ == "__main__":
        #intialise workspace and parameter dictionary
        workspace, params = initialise('stack1.su')
        params['mix'] = 10
        trace_mix(workspace, None, **params)
        toolbox.display(workspace, None, **params)
        pylab.show()
```

../exercise9.py