Generating Synthetic Seismogram in Python

Steps to generate seismogram from well logs

1- Data Prepration

In [64]:
```
import pandas as pd
import lasio as las
import numpy as np
import matplotlib.pyplot as plt
```

Importing the datafrom las file

In [65]:
```
well = las.read('S-KNGM-GM-3_T_133820_0.las')
df = well.df()
```

Unit Conversion

To calculate Acoustic Impedance, we need to have Sonic log (DT) in μs/m unit as well as density log (RHOB) in kg/m3. Looking at header: DT has us/ft unit and RHOB g/cm3.

In [66]:
```
df['DT.US/M'] = df['DT'] / 0.3048   #unit convert to μs/m
df['RHOB.KG/M3'] = df['RHOB'] * 1000  #unit convert to kg/m3
```

Acoustic Impedance

Acoustic Impedance(AI, sometimes showed by Z) is defined as the product of sonic wave velocity and medium density. In reflection seismology, AI variation between geological layers is responsible for the energy reflection toward the surface where we collect seismic data.

In [67]:
```
# Sonic velocity calculate
df["Vsonic"] = 1e6/df["DT.US/M"]                    #(unit: m/s)
```

In [68]:
```
# AI calculate
df['AI'] = df['Vsonic'] * df['RHOB.KG/M3']        #(unit: kg/m2.s)
```

Reflection Coefficient

Reflection Coefficient (Rc) is defined as impedance differences of two adjacent layers divided by their sum. Mathematically, this is derivative with respect to depth.

In [69]:
```
Imp = df['AI'].values
Rc=[]
for i in range(len(Imp)-1):
    Rc.append((Imp[i+1]-Imp[i])/(Imp[i]+Imp[i+1]))

# to adjust vector size copy the last element to the tail
Rc.append(Rc[-1])
```

In [70]:
```
# Let's add Rc into dataframe as new column
df['Rc'] = pd.Series(Rc, index=df.index)
```

Figure 2

Time-Depth relationship

Seismic data is measured in the time domain while well data mainly have depth domain. To convert time to depth and vice versa, we need to establish a relationship. There are several ways to do this but here, as we use the sonic log, which is reciprocal of sound wave velocity. Mathematically if we integrate DT over depth interval, we will compute the time-to-depth relationship. Before that, we should make a remedy for the upper part log data which has a data gap from the surface. As this is on-shore field data, we do not have a water column. The replacement velocity for the section above log is almost 2600 m/s. Simply, dividing gapped interval thickness (log start depth subtracted from kb) by replacement velocity will give us the time.

In [71]:
```
log_start_time = 0      # assuming log start time zero
```

T-D Establishment

Let's implement integration. As depth sampling interval is 0.1524 meter, we need to calculate the time that takes for that thickness to travel with measured sonic data. Then calculate cumulative travel time to downward. Calculating from log starting depth shows zero for first value while we know zero time belongs to ground surface. So, the travel time of this gap should be added to integrated sonic time.

In [72]:
```
#first replace NaN values with zero
dt_iterval = np.nan_to_num(df["DT.US/M"]) * 0.1524 / 1e6
t_cum =  np.cumsum(dt_iterval) * 2
df["TWT"] = t_cum + log_start_time
```

Resampling to the time domain and Rc calculation

Although we established a time-depth relationship(TDR), the components that we need for synthetic seismogram are in depth domain. Reflectivity series should be transferred into time domain for convolution. After defining the time vector, we can use the interpolation function from the numpy library.

In [73]:
```
dt = 0.001   #sampleing interval
t_max = 3.0   # max time to create time vector
t = np.arange(0, t_max, dt)
AI_tdom = np.interp(x=t, xp = df.TWT, fp = df.AI)    #resampling
```

After resampling, if we check AI vector size, it will be definitley diffrent from original one

In [74]:
```
# again Rc calulation but in resampled time domain
Rc_tdom = []
for i in range(len(AI_tdom)-1):
    Rc_tdom.append((AI_tdom[i+1]-AI_tdom[i])/(AI_tdom[i]+AI_tdom[i+1]))
# to adjust vector size copy the last element to the tail
Rc_tdom.append(Rc_tdom[-1])
```

Wavelet & Convolution

In reflection seismology, it is assumed that the seismic data we record on the surface is the reflection of energy (due to rock property variation in depth) which is convolved (masked) with wavelet. From various wavelets, Ricker is one of the most common type as it is zero phase and favorable for seismic interpretation tasks.

Ricker wavelet

In [75]:
```
# define function of ricker wavelet
def ricker(f, length, dt):
    t0 = np.arange(-length/2, (length-dt)/2, dt)
    y = (1.0 - 2.0*(np.pi**2)*(f**2)*(t0**2)) * np.exp(-(np.pi**2)*(f**2)*(t0**2))
    return t0, y
```

Convolution

In this step, we will convolve the reflectivity series with a wavelet to create a synthetic seismogram. It is commonly known as flip and slide a function over the other one. If you are interested to know the mathematic behind this operation you may look at this great video (https://www.youtube.com/watch?v=eqAFP15AMPc).

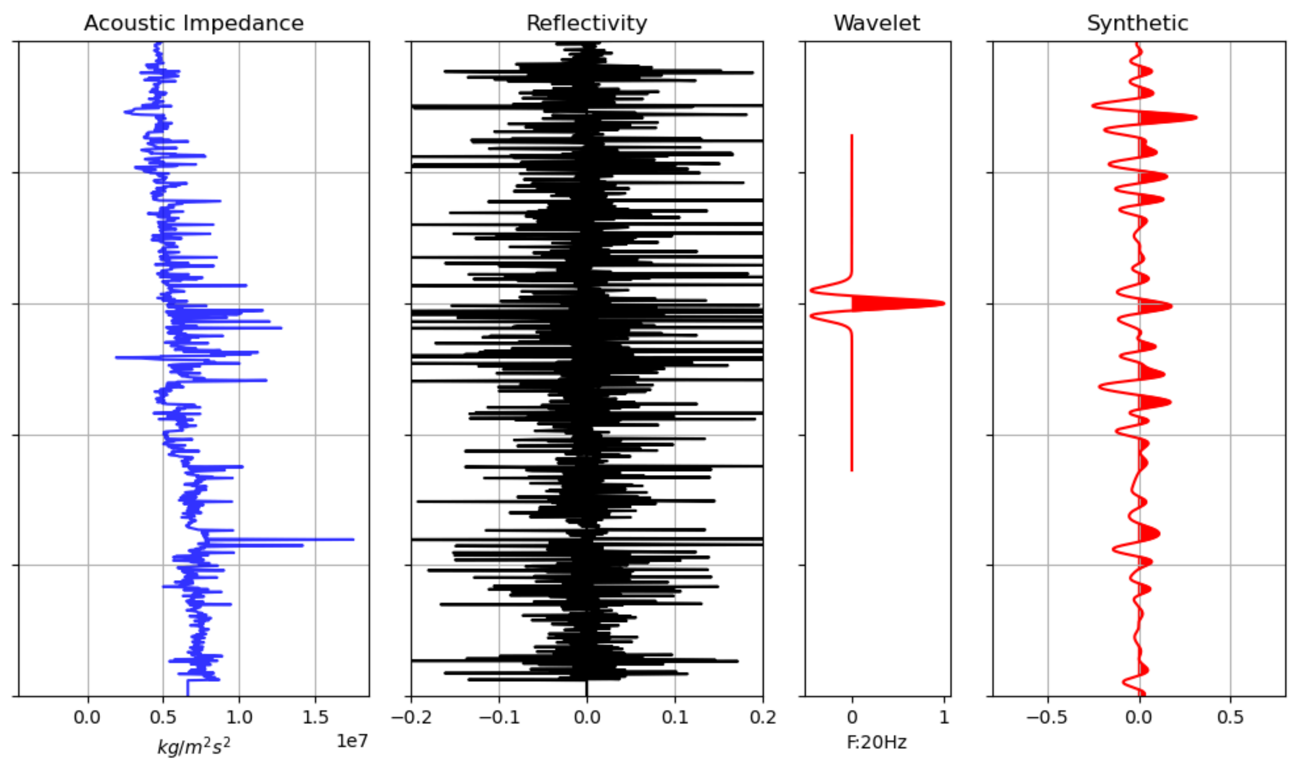
In [76]:
```
f=20            #wavelet frequency
length=0.512    #Wavelet vector length
dt=dt           # Sampling prefer to use smiliar to resampled AI
t0, w = ricker (f, length, dt) # ricker wavelet
synthetic = np.convolve(w, Rc_tdom, mode='same')
```

Figure 3

In [77]:
```
fig, axs = plt.subplots(1, 4, sharey=False, figsize = (10,6), gridspec_kw={'width_rat
fig.subplots_adjust(wspace=0.1)

# ------------------------------------------------------------------------------------
axs[0].plot( AI_tdom, t,'b', alpha=0.8)
axs[0].set_title('Acoustic Impedance', style = 'normal')
axs[0].set_xlabel( r'$kg/m^2s^2$', fontsize = '10')
axs[0].set_ylim(1 , 2.0)
axs[0].set_yticklabels('')
axs[0].invert_yaxis()
axs[0].grid()
# ------------------------------------------------------------------------------------
axs[1].plot( Rc_tdom, t,'k')
axs[1].plot([0, 0], [t.min(), t.max()], '-', c='k')
axs[1].set_title('Reflectivity', style = 'normal')
axs[1].set_ylim(1 , 2.0)
axs[1].set_xlim(-0.2 , 0.2)
axs[1].set_yticklabels('')
axs[1].invert_yaxis()
axs[1].grid()
# ------------------------------------------------------------------------------------
axs[2].plot( w, t0+1.4,'r', alpha=0.99)
axs[2].fill_betweenx(t0+1.4 , w,  0,  w > 0.0,  color='r')
axs[2].set_title('Wavelet', style = 'normal')
axs[2].set_xlabel('F:'+ str(f)+'Hz' , fontsize = '10')
axs[2].set_yticklabels('')
axs[2].set_ylim(1 , 2.0)
axs[2].invert_yaxis()
# ------------------------------------------------------------------------------------
axs[3].plot( synthetic, t ,'r')
axs[3].fill_betweenx(t, synthetic,  0,  synthetic > 0,  color='r')
axs[3].set_title('Synthetic', style = 'normal')
axs[3].set_ylim(1 , 2.0)
axs[3].set_xlim(-0.8 , 0.8)
axs[3].invert_yaxis()
axs[3].set_yticklabels('')
axs[3].grid()

plt.tight_layout()
```



In [ ]:

```

```