demo_notebook

September 10, 2024

1 Example Jupyter notebook to work with the data

2 Read in and plot the Apollo 12 Grade A catalog

[1]:
```
# Import libraries
import numpy as np
import pandas as pd
from obspy import read
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import os
```

Let's take a look at the training data for the lunar dataset. In addition to the data itself, we include a catalog that will tell you which events happen when in the data. The catalog includes the name of the file, the absolute time, the relative time in seconds (relative to the start of the file), the event ID (evid), and the type of moonquake. The types of moonquakes include impacts, deep moonquakes, and shallow moonquakes. You do not have to worry about predicting the type of moonquakes, that's just fun information for you to know!

Note: For your prediction, feel free to include either the absolute time or relative time, just make sure to mark it using the same header in the CSV file so we can easily score it!

[2]:
```
cat_directory = './data/lunar/training/catalogs/'
cat_file = cat_directory + 'apollo12_catalog_GradeA_final.csv'
cat = pd.read_csv(cat_file)
cat
```

[2]:

	filename	time_abs(%Y-%m-%dT%H:%M:%S.%f) \
0	xa.s12.00.mhz.1970-01-19HR00_evid00002	1970-01-19T20:25:00.000000
1	xa.s12.00.mhz.1970-03-25HR00_evid00003	1970-03-25T03:32:00.000000
2	xa.s12.00.mhz.1970-03-26HR00_evid00004	1970-03-26T20:17:00.000000
3	xa.s12.00.mhz.1970-04-25HR00_evid00006	1970-04-25T01:14:00.000000
4	xa.s12.00.mhz.1970-04-26HR00_evid00007	1970-04-26T14:29:00.000000
..	…	…
71	xa.s12.00.mhz.1974-10-14HR00_evid00156	1974-10-14T17:43:00.000000
72	xa.s12.00.mhz.1975-04-12HR00_evid00191	1975-04-12T18:15:00.000000
73	xa.s12.00.mhz.1975-05-04HR00_evid00192	1975-05-04T10:05:00.000000
74	xa.s12.00.mhz.1975-06-24HR00_evid00196	1975-06-24T16:03:00.000000

```
75  xa.s12.00.mhz.1975-06-26HR00_evid00198     1975-06-26T03:24:00.000000

    time_rel(sec)       evid    mq_type
0         73500.0  evid00002  impact_mq
1         12720.0  evid00003  impact_mq
2         73020.0  evid00004  impact_mq
3          4440.0  evid00006  impact_mq
4         52140.0  evid00007    deep_mq
..            …        …          …
71        63780.0  evid00156  impact_mq
72        65700.0  evid00191  impact_mq
73        36300.0  evid00192  impact_mq
74        57780.0  evid00196  impact_mq
75        12240.0  evid00198  impact_mq

[76 rows x 5 columns]
```

2.1 Select a detection

Let's pick the first seismic event in the catalog and let's take a look at the absolute time data. The way we show it here is by using pandas `.iloc` and datetime `.strptime`. We are going to keep the format shown in the absolute time header, which is `'%Y-%m-%dT%H:%M:%S.%f'`

[3]:
```
row = cat.iloc[6]
arrival_time = datetime.strptime(row['time_abs(%Y-%m-%dT%H:%M:%S.
  ↪%f)'],'%Y-%m-%dT%H:%M:%S.%f')
arrival_time
```

[3]:
```
datetime.datetime(1970, 6, 26, 20, 1)
```

[4]:
```
# If we want the value of relative time, we don't need to use datetime
arrival_time_rel = row['time_rel(sec)']
arrival_time_rel
```

[4]:
```
72060.0
```

[5]:
```
# Let's also get the name of the file
test_filename = row.filename
test_filename
```

[5]:
```
'xa.s12.00.mhz.1970-06-26HR00_evid00009'
```

2.2 Read the CSV file corresponding to that detection

We will now find the csv data file corresponding to that time and plot it!

[6]:
```
data_directory = './data/lunar/training/data/S12_GradeA/'
csv_file = f'{data_directory}{test_filename}.csv'
```

```
data_cat = pd.read_csv(csv_file)
data_cat
```

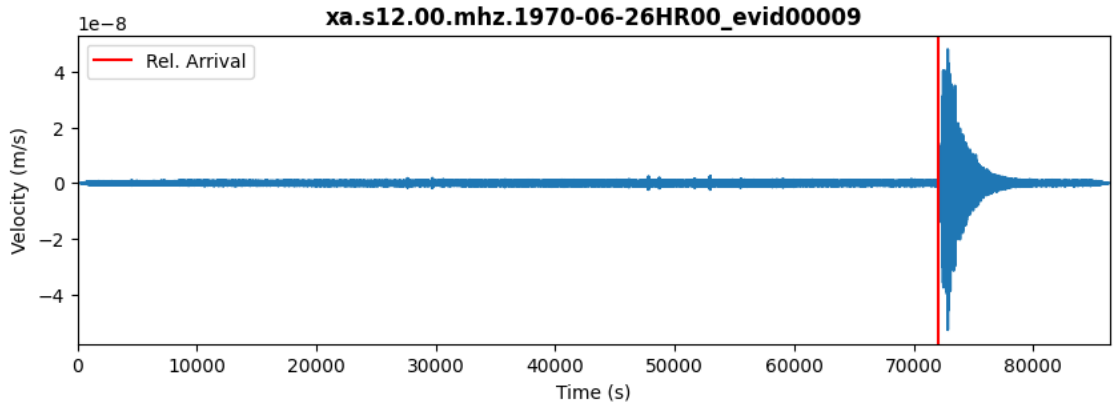
[6]:

	time_abs(%Y-%m-%dT%H:%M:%S.%f)	time_rel(sec)	velocity(m/s)
0	1970-06-26T00:00:00.116000	0.000000	-6.727977e-16
1	1970-06-26T00:00:00.266943	0.150943	-8.646711e-16
2	1970-06-26T00:00:00.417887	0.301887	-9.298738e-16
3	1970-06-26T00:00:00.568830	0.452830	-8.589095e-16
4	1970-06-26T00:00:00.719774	0.603774	-7.139047e-16
…	…	…	…
572418	1970-06-27T00:00:02.832981	86402.716981	5.039820e-17
572419	1970-06-27T00:00:02.983925	86402.867925	-9.191068e-18
572420	1970-06-27T00:00:03.134868	86403.018868	-2.796955e-17
572421	1970-06-27T00:00:03.285811	86403.169811	-9.037156e-17
572422	1970-06-27T00:00:03.436755	86403.320755	-2.439395e-16

[572423 rows x 3 columns]

[7]:
```
# Read in time steps and velocities
csv_times = np.array(data_cat['time_rel(sec)'].tolist())
csv_data = np.array(data_cat['velocity(m/s)'].tolist())

# Plot the trace!
fig,ax = plt.subplots(1,1,figsize=(10,3))
ax.plot(csv_times,csv_data)

# Make the plot pretty
ax.set_xlim([min(csv_times),max(csv_times)])
ax.set_ylabel('Velocity (m/s)')
ax.set_xlabel('Time (s)')
ax.set_title(f'{test_filename}', fontweight='bold')

# Plot where the arrival time is
arrival_line = ax.axvline(x=arrival_time_rel, c='red', label='Rel. Arrival')
ax.legend(handles=[arrival_line])
```

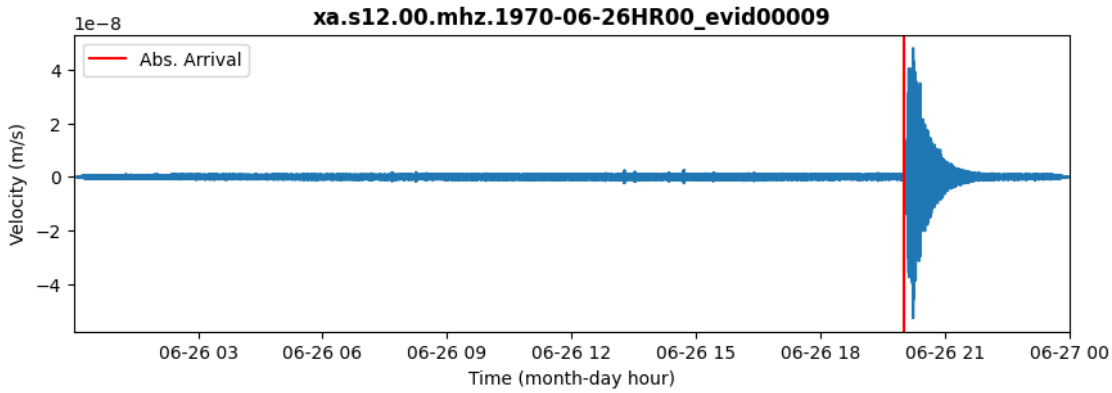
[7]:
```
<matplotlib.legend.Legend at 0x27b41dbca30>
```



What if you wanted to plot in absolute time instead? The operations are very similar, just with a little extra datetime. It takes a bit longer, so we recommend working in relative time to start with!

[8]:
```
# Read in time steps and velocities
csv_times_dt = []
for absval_str in data_cat['time_abs(%Y-%m-%dT%H:%M:%S.%f)'].values:
    csv_times_dt.append(datetime.strptime(absval_str,'%Y-%m-%dT%H:%M:%S.%f'))

csv_data = np.array(data_cat['velocity(m/s)'].tolist())

# Plot the trace!
fig,ax = plt.subplots(1,1,figsize=(10,3))
ax.plot(csv_times_dt,csv_data)

# Make the plot pretty
ax.set_xlim((np.min(csv_times_dt),np.max(csv_times_dt)))
ax.set_ylabel('Velocity (m/s)')
ax.set_xlabel('Time (month-day hour)')
ax.set_title(f'{test_filename}', fontweight='bold')

# Plot where the arrival time is
arrival_line = ax.axvline(x=arrival_time, c='red', label='Abs. Arrival')
ax.legend(handles=[arrival_line])
```

[8]:
```
<matplotlib.legend.Legend at 0x27b47983ee0>
```



2.2.1 Alternatively: read the miniseed file corresponding to that detection

Same procedure as above, just using the miniseed file.

[9]:
```
data_directory = './data/lunar/training/data/S12_GradeA/'
mseed_file = f'{data_directory}{test_filename}.mseed'
st = read(mseed_file)
st
```

[9]:
```
1 Trace(s) in Stream:
XA.S12.00.MHZ | 1970-06-26T00:00:00.116000Z - 1970-06-27T00:00:03.436755Z | 6.6
Hz, 572423 samples
```

[10]:
```
# The stream file also contains some useful header information
st[0].stats
```

[10]:
```
         network: XA
         station: S12
        location: 00
         channel: MHZ
       starttime: 1970-06-26T00:00:00.116000Z
         endtime: 1970-06-27T00:00:03.436755Z
   sampling_rate: 6.625
           delta: 0.1509433962264151
            npts: 572423
           calib: 1.0
         _format: MSEED
           mseed: AttribDict({'dataquality': 'D', 'number_of_records': 1136,
'encoding': 'FLOAT64', 'byteorder': '>', 'record_length': 4096, 'filesize':
4653056})
```

[11]:
```
# This is how you get the data and the time, which is in seconds
tr = st.traces[0].copy()
```

```
tr_times = tr.times()
tr_data = tr.data

# Start time of trace (another way to get the relative arrival time using␣
 ↪datetime)
starttime = tr.stats.starttime.datetime
arrival = (arrival_time - starttime).total_seconds()
arrival
```

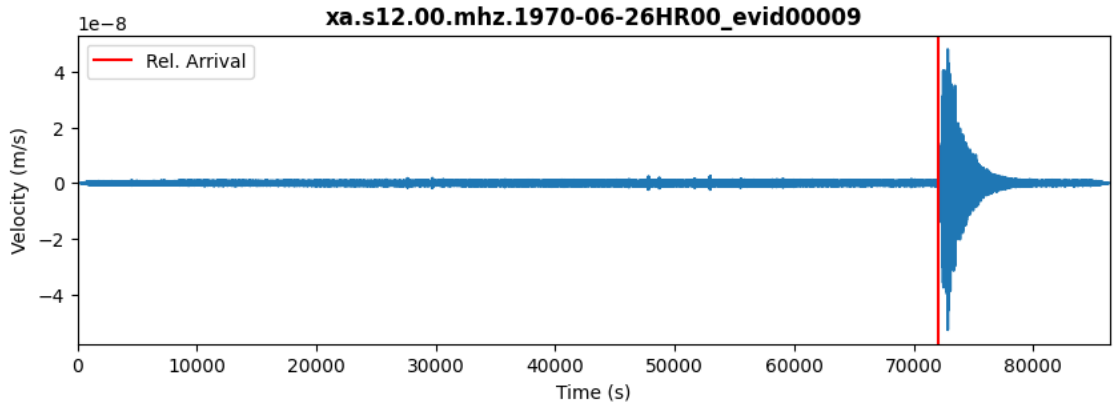
[11]: 72059.884

2.2.2 Plot the trace and mark the arrival!

Use a similar method to plot the miniseed data and seismic arrival.

[12]:
```
# Initialize figure
fig,ax = plt.subplots(1,1,figsize=(10,3))

# Plot trace
ax.plot(tr_times,tr_data)

# Mark detection
ax.axvline(x = arrival, color='red',label='Rel. Arrival')
ax.legend(loc='upper left')

# Make the plot pretty
ax.set_xlim([min(tr_times),max(tr_times)])
ax.set_ylabel('Velocity (m/s)')
ax.set_xlabel('Time (s)')
ax.set_title(f'{test_filename}', fontweight='bold')
```

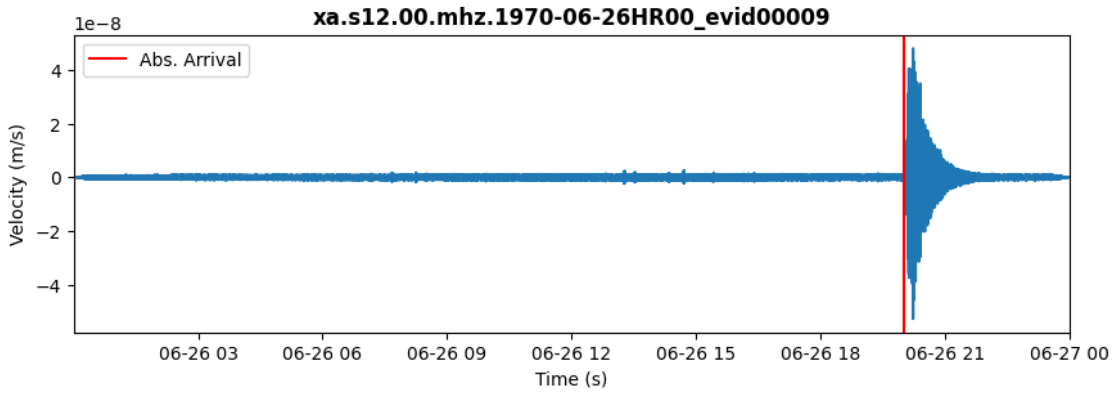
[12]: Text(0.5, 1.0, 'xa.s12.00.mhz.1970-06-26HR00_evid00009')



There are multiple ways that we can do the absolute time using datetime, here is a simple way using the `.timedelta` method

[13]:
```
# Create a vector for the absolute time
tr_times_dt = []
for tr_val in tr_times:
    tr_times_dt.append(starttime + timedelta(seconds=tr_val))

# Plot the absolute result
fig,ax = plt.subplots(1,1,figsize=(10,3))

# Plot trace
ax.plot(tr_times_dt,tr_data)

# Mark detection
arrival_line = ax.axvline(x=arrival_time, c='red', label='Abs. Arrival')
ax.legend(handles=[arrival_line])

# Make the plot pretty
ax.set_xlim([min(tr_times_dt),max(tr_times_dt)])
ax.set_ylabel('Velocity (m/s)')
ax.set_xlabel('Time (s)')
ax.set_title(f'{test_filename}', fontweight='bold')
```

[13]:
```
Text(0.5, 1.0, 'xa.s12.00.mhz.1970-06-26HR00_evid00009')
```



It's completely up to you whether to work with the CSV file or the miniseed files. We recommend working with the miniseed file as it's a bit faster to run.

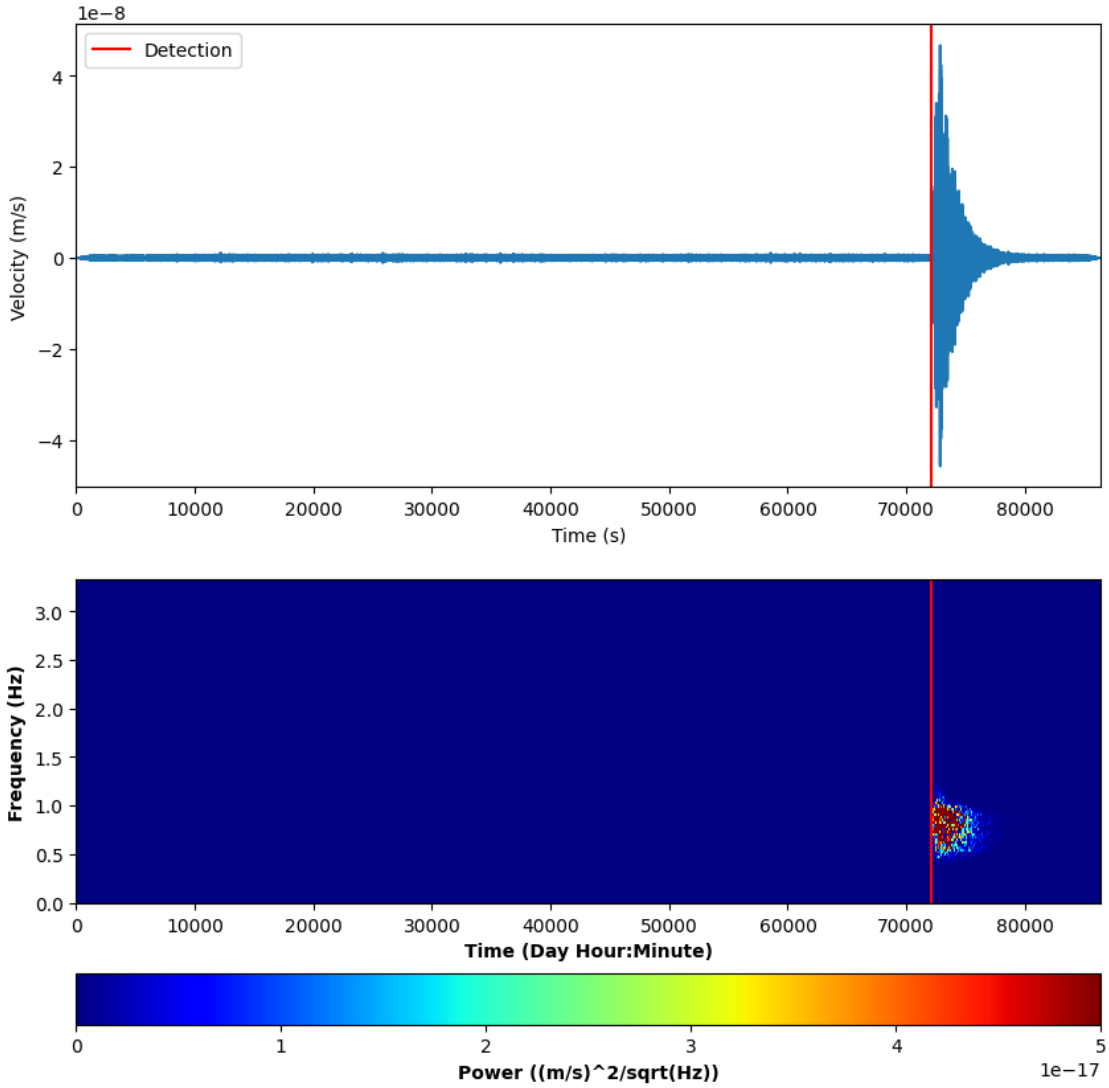
2.3 Let's filter the trace

Sometimes, it's useful to filter the trace to bring out particular frequencies. This will change the shape of the data and make it easier to see certain parts of the signal. In this example, we will filter the data using a bandpass filter between 0.01 Hz to 0.5 Hz.

[14]:
```
# Set the minimum frequency
minfreq = 0.5
maxfreq = 1.0

# Going to create a separate trace for the filter data
st_filt = st.copy()
st_filt.filter('bandpass',freqmin=minfreq,freqmax=maxfreq)
tr_filt = st_filt.traces[0].copy()
tr_times_filt = tr_filt.times()
tr_data_filt = tr_filt.data
```

[15]:
```
# To better see the patterns, we will create a spectrogram using the scipy␣
 ↪function
# It requires the sampling rate, which we can get from the miniseed header as␣
 ↪shown a few cells above
from scipy import signal
from matplotlib import cm
f, t, sxx = signal.spectrogram(tr_data_filt, tr_filt.stats.sampling_rate)
```

[16]:
```
# Plot the time series and spectrogram
fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(2, 1, 1)
# Plot trace
ax.plot(tr_times_filt,tr_data_filt)

# Mark detection
ax.axvline(x = arrival, color='red',label='Detection')
ax.legend(loc='upper left')

# Make the plot pretty
ax.set_xlim([min(tr_times_filt),max(tr_times_filt)])
ax.set_ylabel('Velocity (m/s)')
ax.set_xlabel('Time (s)')

ax2 = plt.subplot(2, 1, 2)
vals = ax2.pcolormesh(t, f, sxx, cmap=cm.jet, vmax=5e-17)
ax2.set_xlim([min(tr_times_filt),max(tr_times_filt)])
ax2.set_xlabel(f'Time (Day Hour:Minute)', fontweight='bold')
ax2.set_ylabel('Frequency (Hz)', fontweight='bold')
ax2.axvline(x=arrival, c='red')
cbar = plt.colorbar(vals, orientation='horizontal')
cbar.set_label('Power ((m/s)^2/sqrt(Hz))', fontweight='bold')
```



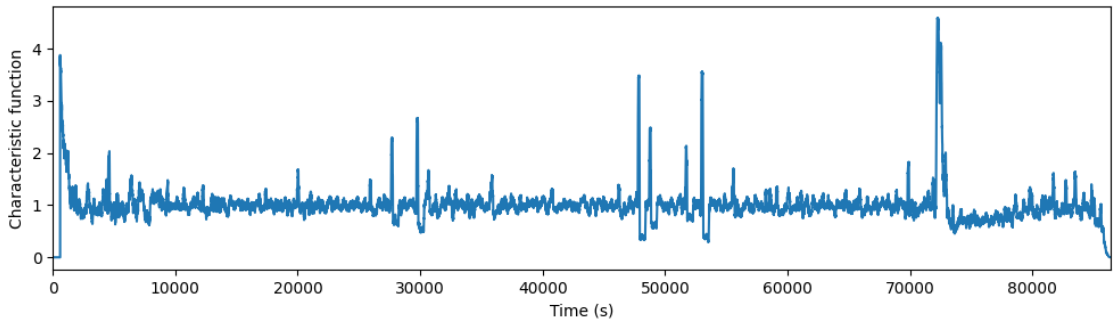
3 Sample short-term average / long-term average (STA/LTA) detection algorithm

A STA/LTA algorithm moves two time windows of two lengths (one short, one long) across the seismic data. The algorithm calculates the average amplitude in both windows, and calculates the ratio between them. If the data contains an earthquake, then the short-term window containing the earthquake will be much larger than the long-term window – resulting in a detection.

[17]:
```
from obspy.signal.invsim import cosine_taper
from obspy.signal.filter import highpass
from obspy.signal.trigger import classic_sta_lta, plot_trigger, trigger_onset

# Sampling frequency of our trace
```

```
df = tr.stats.sampling_rate

# How long should the short-term and long-term window be, in seconds?
sta_len = 120
lta_len = 600

# Run Obspy's STA/LTA to obtain a characteristic function
# This function basically calculates the ratio of amplitude between the␣
 ↪short-term
# and long-term windows, moving consecutively in time across the data
cft = classic_sta_lta(tr_data, int(sta_len * df), int(lta_len * df))

# Plot characteristic function
fig,ax = plt.subplots(1,1,figsize=(12,3))
ax.plot(tr_times,cft)
ax.set_xlim([min(tr_times),max(tr_times)])
ax.set_xlabel('Time (s)')
ax.set_ylabel('Characteristic function')
```

[17]: `Text(0, 0.5, 'Characteristic function')`



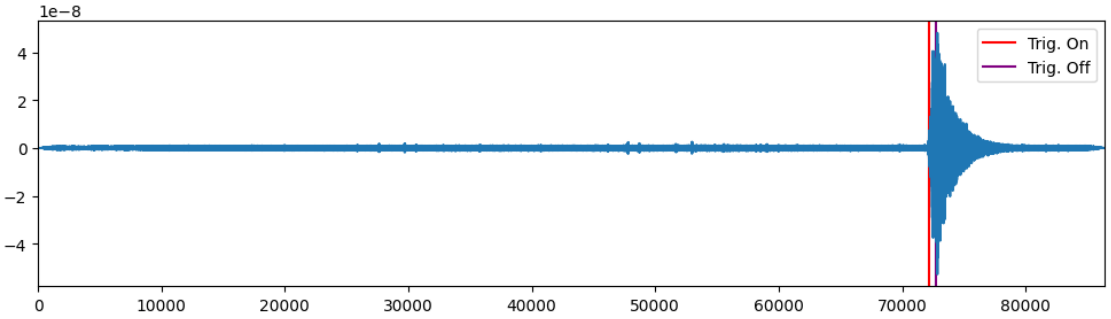
Next, we define the values of the characteristic function (i.e. amplitude ratio between short-term and long-term windows) where we flag a seismic detection. These values are called triggers. There are two types of triggers – "on" and "off", defined as follows:

1. "on" : If the characteristic function is above this value, then a seismic event begins.
2. "off" : If the characteristic function falls below this value (after an "on" trigger), than a seismic event ends.

[18]:
```
# Play around with the on and off triggers, based on values in the␣
 ↪characteristic function
thr_on = 4
thr_off = 1.5
on_off = np.array(trigger_onset(cft, thr_on, thr_off))
# The first column contains the indices where the trigger is turned "on".
```

```
# The second column contains the indices where the trigger is turned "off".

# Plot on and off triggers
fig,ax = plt.subplots(1,1,figsize=(12,3))
for i in np.arange(0,len(on_off)):
    triggers = on_off[i]
    ax.axvline(x = tr_times[triggers[0]], color='red', label='Trig. On')
    ax.axvline(x = tr_times[triggers[1]], color='purple', label='Trig. Off')

# Plot seismogram
ax.plot(tr_times,tr_data)
ax.set_xlim([min(tr_times),max(tr_times)])
ax.legend()
```

[18]: <matplotlib.legend.Legend at 0x27b01c16a60>



Note: You do not have to worry about marking the end of the seismic trace (as you can see, even for us it's not very accurate!). For this challenge, all we care about is the start of the seismic waveform.

3.1 Sample detection export into a catalog!

There are many ways to do this, but we'll show a way to do it using pandas.

[19]:
```
# File name and start time of trace
fname = row.filename
starttime = tr.stats.starttime.datetime

# Iterate through detection times and compile them
detection_times = []
fnames = []
for i in np.arange(0,len(on_off)):
    triggers = on_off[i]
    on_time = starttime + timedelta(seconds = tr_times[triggers[0]])
    on_time_str = datetime.strftime(on_time,'%Y-%m-%dT%H:%M:%S.%f')
```

```
    detection_times.append(on_time_str)
    fnames.append(fname)

# Compile dataframe of detections
detect_df = pd.DataFrame(data = {'filename':fnames, 'time_abs(%Y-%m-%dT%H:%M:%S.
 ↪%f)':detection_times, 'time_rel(sec)':tr_times[triggers[0]]})
detect_df.head()
```

[19]:

	filename	time_abs(%Y-%m-%dT%H:%M:%S.%f)	\
0	xa.s12.00.mhz.1970-06-26HR00_evid00009	1970-06-26T20:03:21.323547	

	time_rel(sec)
0	72201.207547

This can then be exported to a csv using:

```
detect_df.to_csv('output/path/catalog.csv', index=False)
```

4 Download additional data from Earth-based stations

You may find that you need to download additional data from Earth stations to supplement your models and algorithms. We recommend that you download any events from IRIS (Incorporated Research Institutations for Seismology).

https://www.iris.edu/hq/

Note: The organization has been recently renamed to SAGE (Seismological Facility for the Advancement of Geoscience), but all the previous links should still work.

They maintain and curate data from seismic stations all around the world. There are many different ways to get data from them, but I recommend using the utility *PyWeed*:

https://ds.iris.edu/ds/nodes/dmc/software/downloads/pyweed/

We can use the utility to select seismic stations and the earthquake data (or **events**) recorded at those stations.

For this test case, let's download all of the earthquakes magnitude 3 and above that are within 1 degree distance (approximately 110 km) from a site called PFO (Pinon Flat Observatory) in California. **Location** is a number designating the instrument at a particular site (sites may have multiple instruments), and **channel** is an IRIS code that specifies instrument information.

In short, the first latter refers to the samplerate of the instrument (how many data points it records per second), the second to the type of instrument (certain types of seismometers are better at recording nearby earthquakes while others are more suited for distant earthquakes), and the last to the directional component being recored (most seismometers will record motion across two horizontal directions and the vertical). We will pick the channel HHZ, which refers to a (H) high-samplerate (100 samples per second) (H) strong-motion accelerometer (best resolution for nearby strong earthquakes) recording in the (Z) vertical direction. Once you've selected all the earthquakes, you can download the traces.

An earthquake is composed of the following types of waves (in order): pressure (P-wave), shear (S-wave), and surface (Rayleigh and Love). For our challenge, we are only interested in identifying the start of the earthquake. The IRIS dataset contains P-wave arrivals (onset of the P-wave at the seismometer) for each earthquake. In order to get noise prior to the earthquake arrival, we pick our data traces to span 101 seconds before to 60 seconds past the P-wave arrival:

As you can see from the output list, some of the earthquakes don't record any earthquake data (3.4 Ml 2005-08-31) and others have an incorrect P-wave arrival time (4.0 Ml 2005-08-31). Make sure to go through the earthquakes and remove those types of events from the waveform preview prior to download. For output file type, choose miniseed to match the planetary data (SAC is probably fine too, but the file sizes tend to be a bit bigger).

4.1 Thank you very much for being a part of this challenge! Good luck!!!