Obspy_Tutorial_Follow_Along_Notebook

June 26, 2020

1 Obspy: a Python Toolbox for seismology

1.1 ROSES Tutorial Outline

1. Reading data from a file
2. Downloading data from online repositories
3. Removing instrument response
4. Writing data to a file
5. Some Obspy stream and trace methods
6. Plotting with Matplotlib

1.1.1 Instructor: Sydney Dybing (sdybing@uoregon.edu)

Dependencies needed: Obspy, Numpy, Matplotlib Since this lecture is only an hour, I intend for it to serve as a basic introduction to using Obspy. I've written it to include information about how to use the tools I find myself using most often in my research.

If you want to learn more or there are functions I didn't cover that you need, there are several fantastic options for learning more about Obspy online. In particular, I will link the official Obspy Tutorial, and I also recommend the Seismo-Live Juypter Notebooks for Seismology, which includes an Obspy section among other topics.

I'll start showing you some of the basic capabilities of Obspy with a sample data file that I already downloaded, and was provided on Slack. This is in miniSEED format, but Obspy can also automatically detect many other data formats, such as SAC.

1.2 Reading data from a file

[1]:
```
from obspy import read
```

With the read function, you basically just only need the path to the file on your computer. There are additional options, which you can check out at the Obspy documentation page for the read function.

The data is read into a stream object.

[2]:
```
!ls
```

```
B082_EHZ.mseed
CWC_HNE.mseed
CWC_HNN.mseed
```

```
CWC_HNZ.mseed
Obspy_Tutorial_Follow_Along_Notebook.ipynb
Obspy_Tutorial_Lab_Notebook.ipynb
```

[3]:
```
st = read('B082_EHZ.mseed')
print(st)
```

```
1 Trace(s) in Stream:
PB.B082..EHZ | 2010-04-04T22:40:42.368400Z - 2010-04-04T22:45:42.358400Z | 100.0
Hz, 30000 samples
```

Stream objects are basically collections of trace objects, which contain the data and associated metadata. You can grab a trace from a stream element the way you would an item from a list.

[4]:
```
tr = st[0]
print(tr)
```

```
PB.B082..EHZ | 2010-04-04T22:40:42.368400Z - 2010-04-04T22:45:42.358400Z | 100.0
Hz, 30000 samples
```

To view the data:

[5]:
```
data = tr.data
print(data)
```

```
[  49   52   50 … 6979 7201 7440]
```

This is a numpy array, so you can do any operations then with this array that you normally would in Python.

To view the metadata:

[6]:
```
print(tr.stats)
```

```
         network: PB
         station: B082
        location:
         channel: EHZ
       starttime: 2010-04-04T22:40:42.368400Z
         endtime: 2010-04-04T22:45:42.358400Z
   sampling_rate: 100.0
           delta: 0.01
            npts: 30000
           calib: 1.0
         _format: MSEED
           mseed: AttribDict({'dataquality': 'M', 'number_of_records': 134,
'encoding': 'STEIM2', 'byteorder': '>', 'record_length': 512, 'filesize':
68608})
```

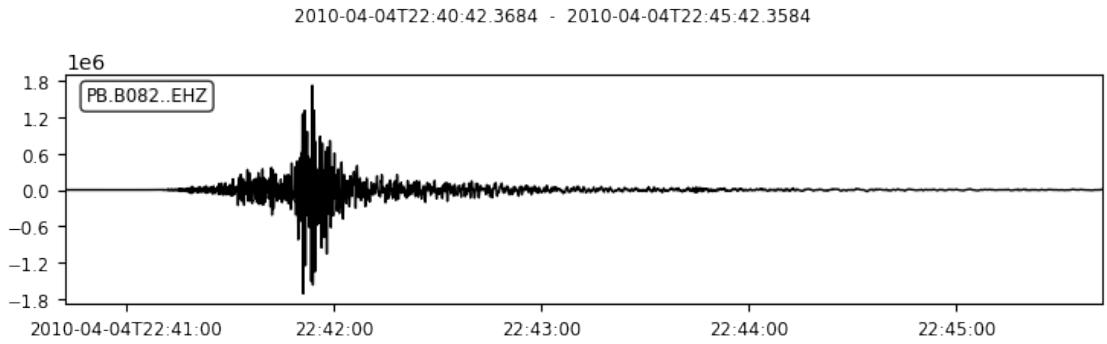
If you want to use any of these particular things, you can grab them individually as follows:

[7]:
```
print(tr.stats.network)
print(tr.stats.npts)
```

```
PB
30000
```

You can also make simple plots of traces.

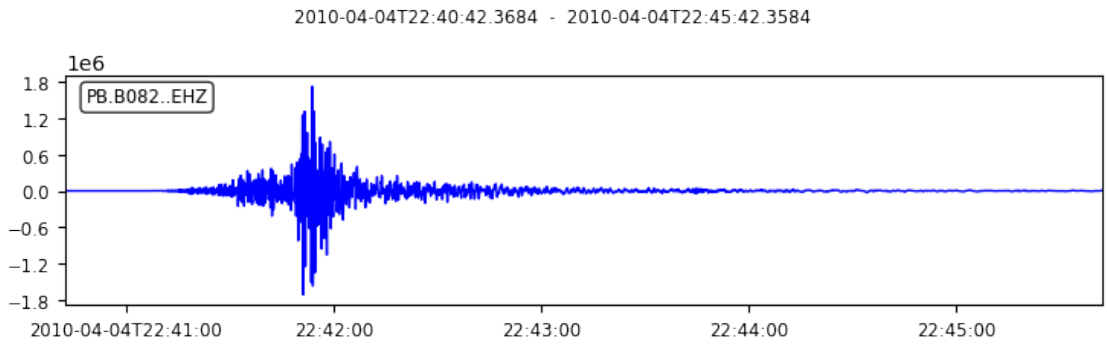
[8]:
```
tr.plot();
```



You can change these simple Obspy plots to look more unique and plot different things. More info can be found in the Obspy tutorial pages online.

Here is an example of changing the color:

[11]:
```
tr.plot(color='b')
```



So far, we've been working with just one trace from a stream, which has a single channel of data from a single station. Obspy streams are useful because they can hold multiple traces. You can read them in several different ways.

First, you can add individual files to a stream one at a time.

[12]:
```
!ls
```

```
B082_EHZ.mseed
CWC_HNE.mseed
CWC_HNN.mseed
CWC_HNZ.mseed
Obspy_Tutorial_Follow_Along_Notebook.ipynb
Obspy_Tutorial_Lab_Notebook.ipynb
```

[14]:
```
st = read('CWC_HNE.mseed')
st += read('CWC_HNN.mseed')
st += read('CWC_HNZ.mseed')
print(st)
```

```
3 Trace(s) in Stream:
CI.CWC..HNE | 2019-07-05T11:06:53.048393Z - 2019-07-05T11:10:53.038393Z | 100.0
Hz, 24000 samples
CI.CWC..HNN | 2019-07-05T11:06:53.048393Z - 2019-07-05T11:10:53.038393Z | 100.0
Hz, 24000 samples
CI.CWC..HNZ | 2019-07-05T11:06:53.048393Z - 2019-07-05T11:10:53.038393Z | 100.0
Hz, 24000 samples
```

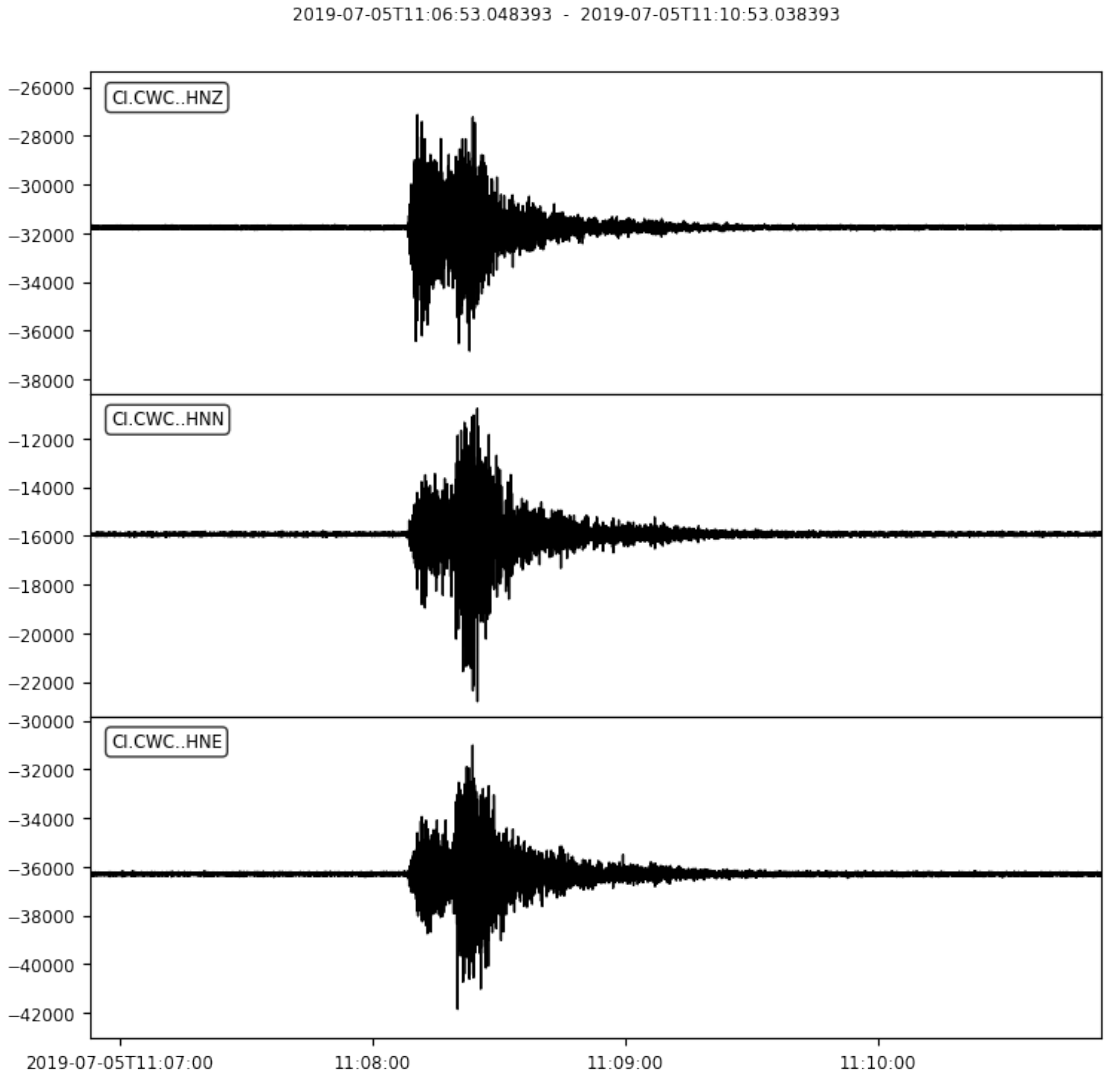
Another option is to read individual files in as traces, and then add them to a stream. The output is ultimately the same.

[15]:
```
tr1 = read('CWC_HNE.mseed')
tr2 = read('CWC_HNN.mseed')
tr3 = read('CWC_HNZ.mseed')
st = tr1 + tr2 + tr3
print(st)
```

```
3 Trace(s) in Stream:
CI.CWC..HNE | 2019-07-05T11:06:53.048393Z - 2019-07-05T11:10:53.038393Z | 100.0
Hz, 24000 samples
CI.CWC..HNN | 2019-07-05T11:06:53.048393Z - 2019-07-05T11:10:53.038393Z | 100.0
Hz, 24000 samples
CI.CWC..HNZ | 2019-07-05T11:06:53.048393Z - 2019-07-05T11:10:53.038393Z | 100.0
Hz, 24000 samples
```

You can make simple plots of stream objects the same way you can with traces.

[16]:
```
st.plot()
```



To convert our y-axis units from digital counts to actual ground motion, we need to remove the instrument response, which I'll discuss more later. You can download responses alongside data. So let's talk about downloading data from online repositories like the IRIS Data Management Center.

1.3 Downloading data from online repositories

IRIS is just one "client" that stores data, and other client options can be found here. FDSN is the International Federation of Digital Seismograph Networks, which helps coordinate global data access.

[17]:
```
from obspy.clients.fdsn import Client
client = Client('IRIS')
```

Times are managed in Obspy using a class called UTCDateTime.

[18]:
```
from obspy import UTCDateTime
```

UTCDateTime objects are easy to use and the format is easy to understand. If you're like me and you ocassionally forget exactly how they're formatted, it's simple to check the Obspy documentation here.

To create a UTCDateTime object, you just need a date and a time, and to string them together in the right order, which is as follows:

"YYYY-MM-DDTHH:MM:SS"

If you're working in juldays, which is just counting the day of the year out of 365, with January 1 being day 001, this is the format:

"YYYY-DDDTHH:MM:SS"

Let's try an example from last year's M7.1 Ridgecrest earthquake.

[19]:
```
time = UTCDateTime('2019-07-06T03:19:53.04')
print(time)
```

```
2019-07-06T03:19:53.040000Z
```

You can add or subtract a number of seconds from a UTCDateTime object, which is useful for downloading a specific segment of data.

[20]:
```
starttime = time - 60
print(starttime)
```

```
2019-07-06T03:18:53.040000Z
```

[21]:
```
endtime = time + 15*60
print(endtime)
```

```
2019-07-06T03:34:53.040000Z
```

Great! We have some times - so where do we want to get the data from?

The widely-adopted SEED standard for data follows this convention for data identification:

Network code: Identifies which network the data belongs to. You can check out all of the available networks at IRIS MetaData Aggregator here. Network codes are unique and assigned by the FDSN.

Station code: The station within a network - these aren't necessarily unique, so you need to use network codes with station codes to identify a station.

Location ID: Sometimes stations can have more than one instrument at them, which is specified by the location ID.

Channel codes: Three character code - Emily will talk more about this next week!

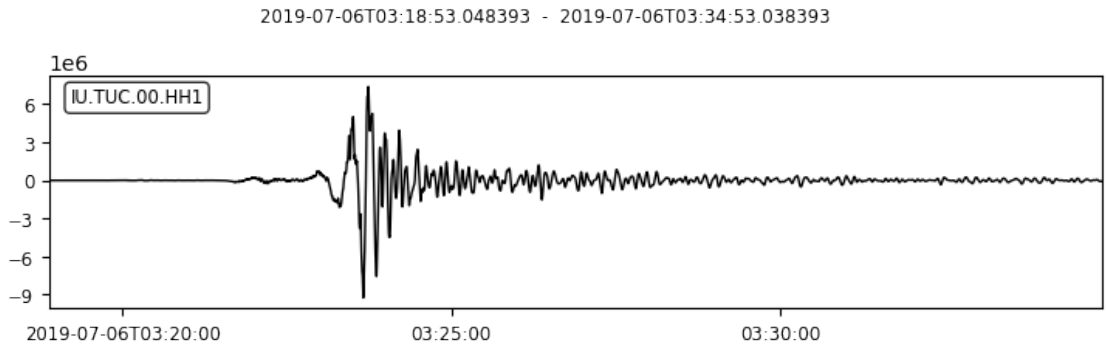
Let's start with a simple example, where we choose one network, one station, one location (one instrument basically that's part of the station), and one channel. Here is the MDA page for the Tucson, AZ station we will use.

[22]:
```
net = 'IU'
sta = 'TUC'
loc = '00'
chan = 'HH1'
```

To actually download the data, we can use an Obspy function called get_waveforms.

[23]:
```
st = client.get_waveforms(net, sta, loc, chan, starttime, endtime)
print(st)
st.plot()
```

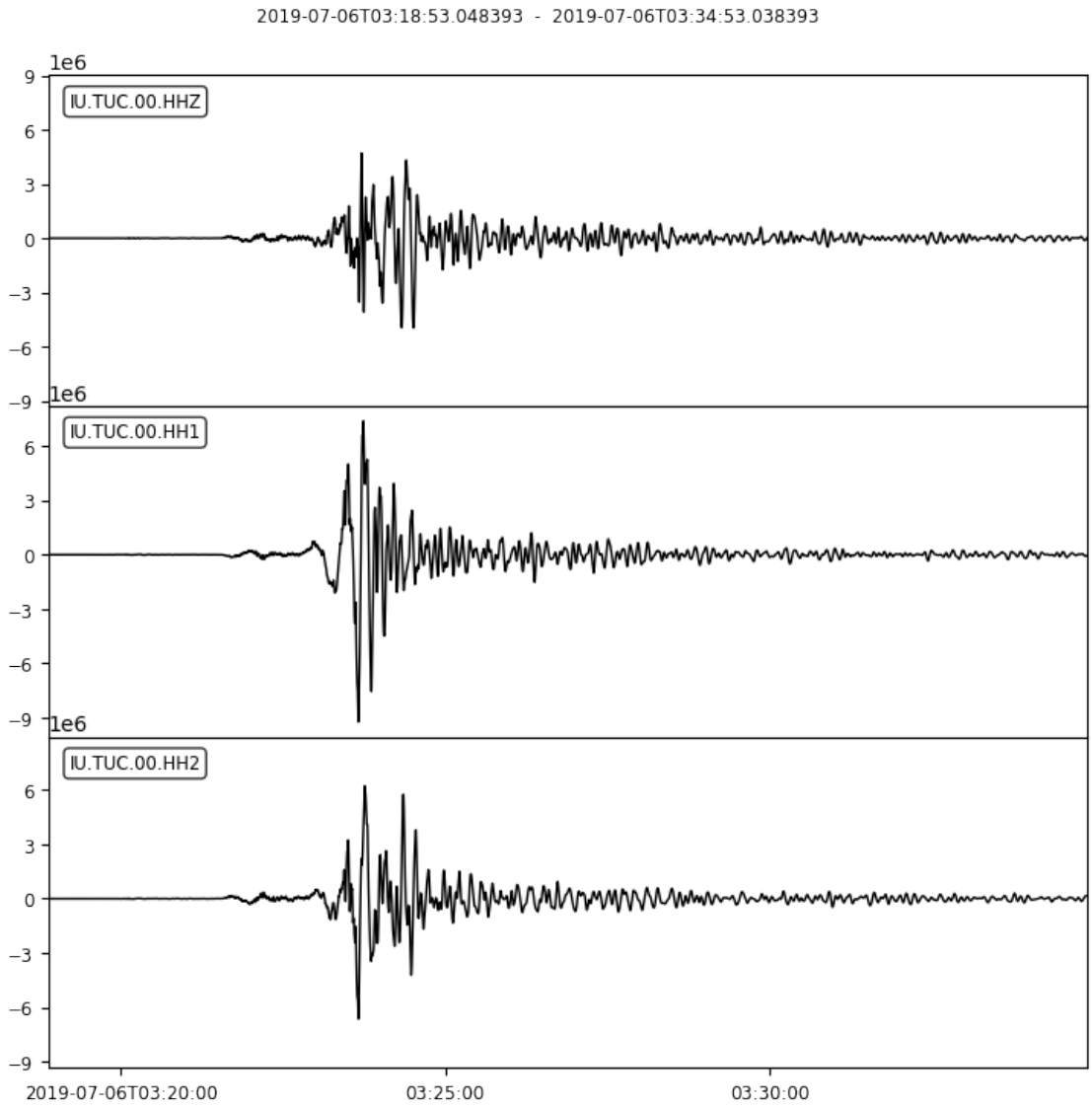
```
1 Trace(s) in Stream:
IU.TUC.00.HH1 | 2019-07-06T03:18:53.048393Z - 2019-07-06T03:34:53.038393Z |
100.0 Hz, 96000 samples
```



If we want to download multiple channels, in Obspy you can use Linux syntax like wildcards to accomplish this.

[24]:
```
chan = 'HH*'
st = client.get_waveforms(net, sta, loc, chan, starttime, endtime)
print(st)
st.plot()
```

```
3 Trace(s) in Stream:
IU.TUC.00.HH1 | 2019-07-06T03:18:53.048393Z - 2019-07-06T03:34:53.038393Z |
100.0 Hz, 96000 samples
IU.TUC.00.HH2 | 2019-07-06T03:18:53.048393Z - 2019-07-06T03:34:53.038393Z |
100.0 Hz, 96000 samples
IU.TUC.00.HHZ | 2019-07-06T03:18:53.048393Z - 2019-07-06T03:34:53.038393Z |
100.0 Hz, 96000 samples
```



1.4 Removing instrument response

This data right now is quantitatively meaningless because we haven't removed the instrument response.

Theoretically, this is relatively complicated (see Havskov and Alguacil, 2015). In practice, Obspy makes correcting for instrument response easier! You just have to make sure you download the response with the data.

We'll keep working with the same station and earthquake. As a refresher, here are the variables we have set. We'll stick with working on one channel for now to make visualization simpler.

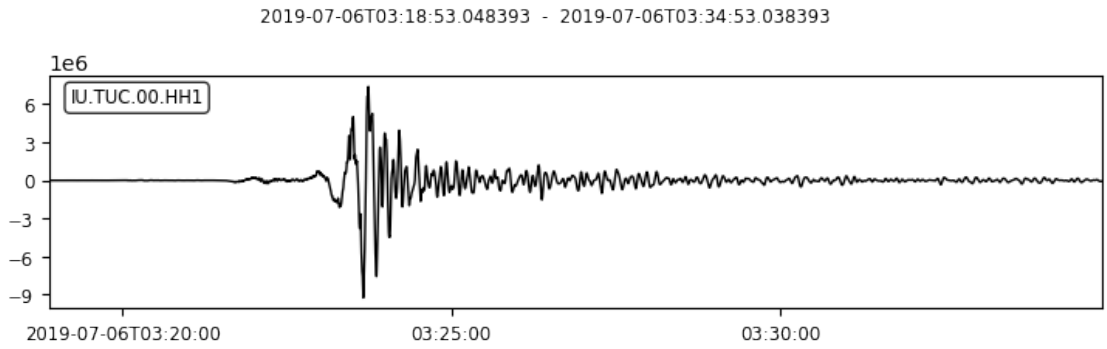
[25]:
```
time = UTCDateTime("2019-07-06T03:19:53.04")
starttime = time - 60
endtime = time + 60*15

net = "IU"
sta = "TUC"
loc = "00"
chan = "HH1"
```

Now, all we have to do is add one option when we use get_waveforms:

[26]:
```
st = client.get_waveforms(net, sta, loc, chan, starttime, endtime,␣
 ↪attach_response = True)
print(st)
st.plot();
```

```
1 Trace(s) in Stream:
IU.TUC.00.HH1 | 2019-07-06T03:18:53.048393Z - 2019-07-06T03:34:53.038393Z |
100.0 Hz, 96000 samples
```

```
/Users/utpalkumar50/miniconda3/envs/roses/lib/python3.7/site-
packages/obspy/io/stationxml/core.py:84: UserWarning: The StationXML file has
version 1.1, ObsPy can deal with version 1.0. Proceed with caution.
  root.attrib["schemaVersion"], SCHEMA_VERSION))
```



It doesn't look any different yet, but the responses have been downloaded. Before removing the response, you might want to copy your data, since the function remove_response acts directly on the data, and will overwrite the original. Copying a stream is simple:

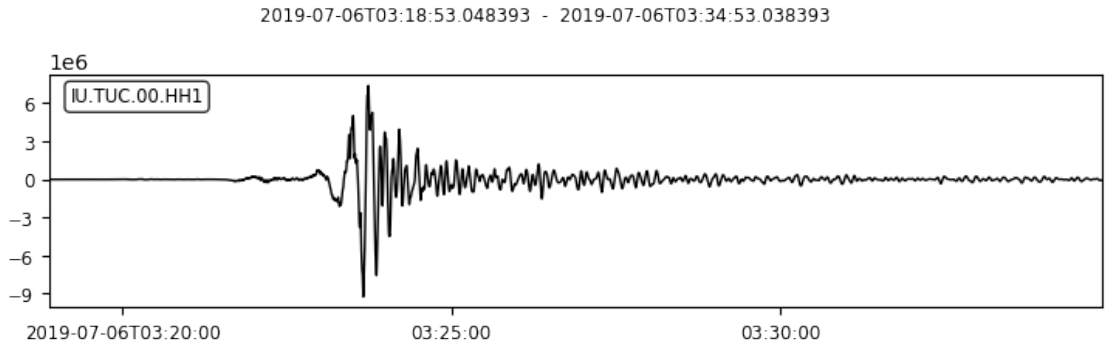
[27]:
```
st_rem = st.copy()
print(st_rem)
print(st)
```

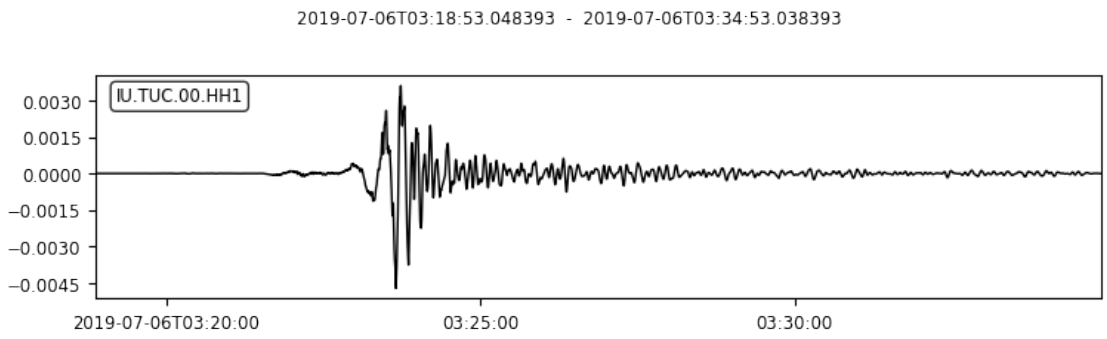
```
1 Trace(s) in Stream:
IU.TUC.00.HH1 | 2019-07-06T03:18:53.048393Z - 2019-07-06T03:34:53.038393Z |
```

```
100.0 Hz, 96000 samples
1 Trace(s) in Stream:
IU.TUC.00.HH1 | 2019-07-06T03:18:53.048393Z - 2019-07-06T03:34:53.038393Z |
100.0 Hz, 96000 samples
```

Now we can remove the response. You can set which units you want the ground motion to be represented in using the output option.

[28]:
```
st_rem.remove_response(output='VEL') #DISP, ACC
st.plot()
st_rem.plot()

# Remember, if you run this cell multiple times, your output will be strange␣
 ↪because you already removed the
# response from st_rem. If you want to do it again and try something else, you␣
 ↪either have to make a new copy
# of the original st again, or go back and re-run the previous cell that copied␣
 ↪st.
```



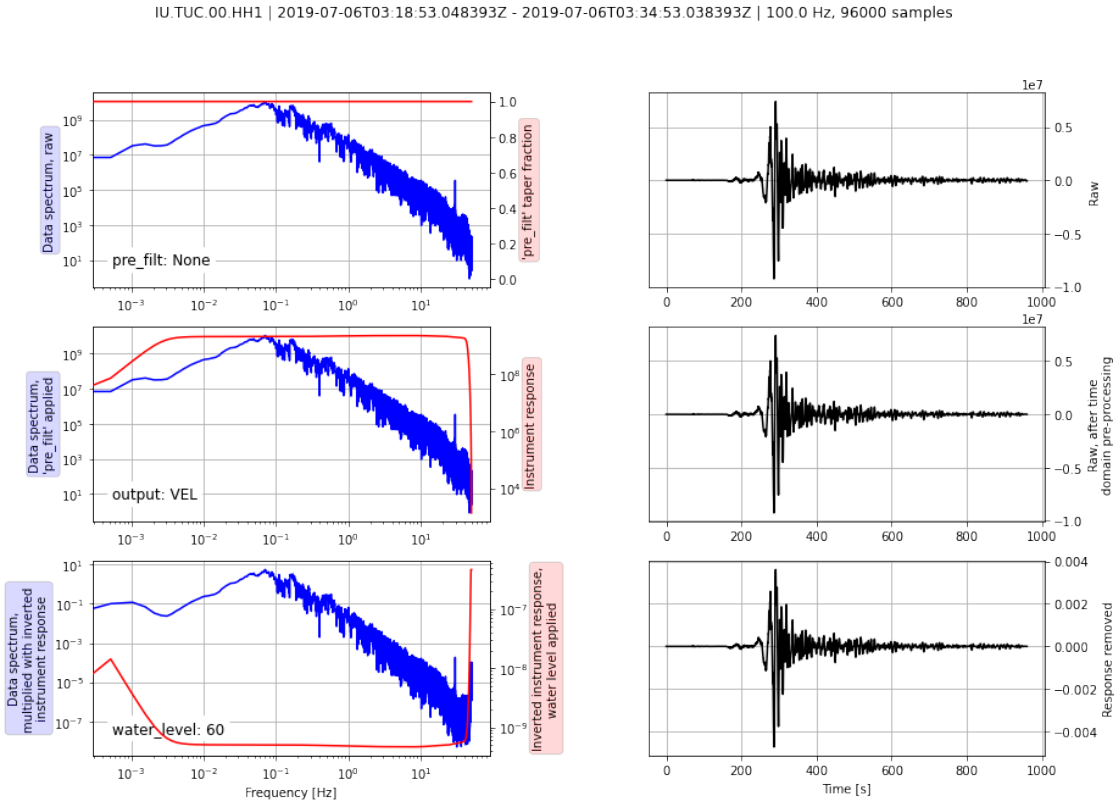


This is an oversimplification of the response removal process.

We can visualize what remove_response is doing by using the plot = True option. This is what it looks like if we don't do any kind of filtering.

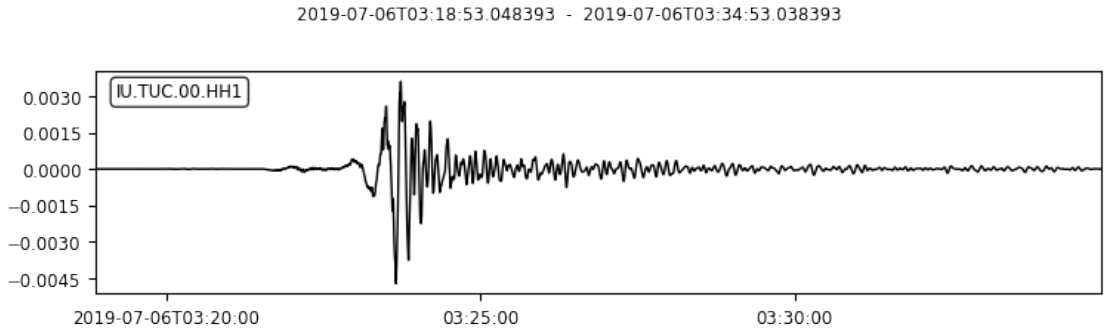
[29]:
```
st_rem = st.copy() # repeating this since the last cell will have removed the response from st_rem already
st_rem.remove_response(output='VEL', plot=True)
# other options: output = 'DISP', 'ACC'
```

[29]:
```
1 Trace(s) in Stream:
IU.TUC.00.HH1 | 2019-07-06T03:18:53.048393Z - 2019-07-06T03:34:53.038393Z |
100.0 Hz, 96000 samples
```



This is what our data looks like now:

[30]:
```
st_rem.plot()
```



1.5 Writing downloaded data to a file

This can be done with st.write() - you just specify the desired file path, name, extension, and data format (SAC, MSEED, etc.). It looks something like this:

stream.write('/path/filename.mseed', format='MSEED')

For this example:

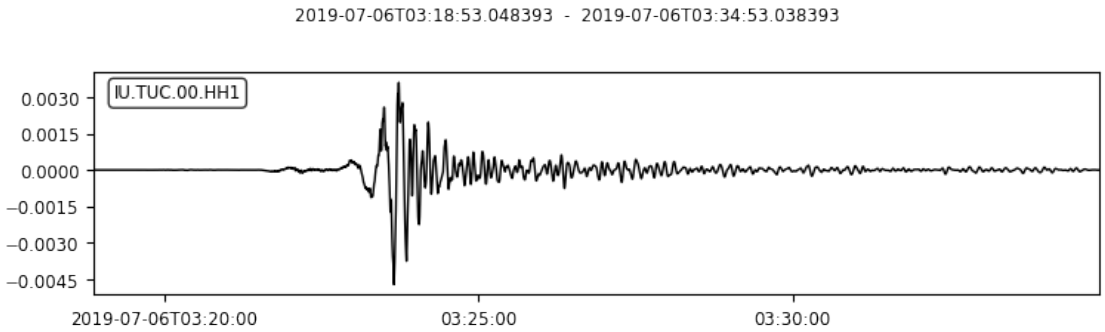
[31]:
```
filename = f'{sta}_{chan}.mseed'
st_rem.write(filename, format='MSEED')
```

```
/Users/utpalkumar50/miniconda3/envs/roses/lib/python3.7/site-
packages/obspy/io/mseed/core.py:772: UserWarning: The encoding specified in
trace.stats.mseed.encoding does not match the dtype of the data.
A suitable encoding will be chosen.
  warnings.warn(msg, UserWarning)
```

Now we can read it in as a stream just as we did with the sample file at the beginning of this tutorial.

[32]:
```
st = read('TUC_HH1.mseed')
print(st)
st.plot()
```

```
1 Trace(s) in Stream:
IU.TUC.00.HH1 | 2019-07-06T03:18:53.048393Z - 2019-07-06T03:34:53.038393Z |
100.0 Hz, 96000 samples
```



1.6 Some Stream and Trace Methods

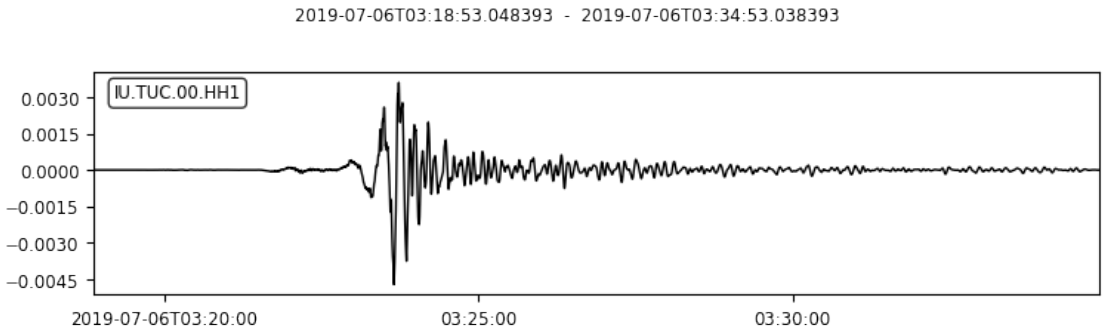
The stream and trace objects in Obspy both have a number of public methods that can be used to modify the data. I'll go over a few of them that I use most often, and if there is anything else you're interested in doing, you can check the Obspy documentation for traces and for streams.

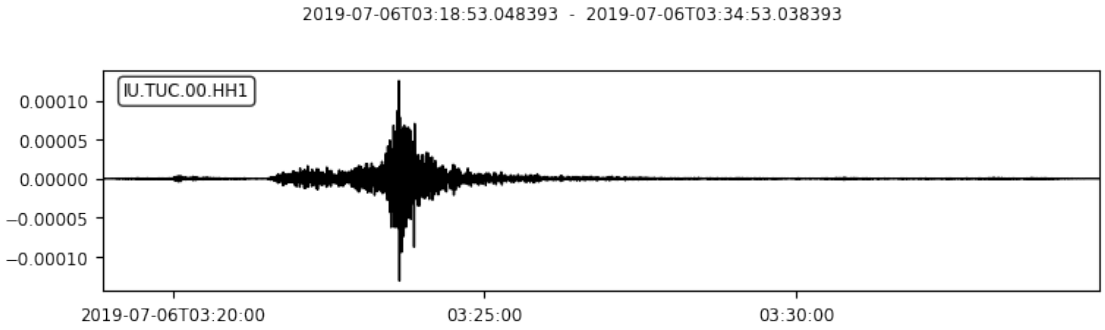
1.6.1 Filtering

One thing we might want to do is look at a specific frequency range of data in our earthquake. We can use the "filter" method for this. There are many different filter methods available, and all of these options can be found in the Obspy documentation here.

Let's try a bandpass example:

[33]:
```
st_filt = st.copy()
st_filt.filter('bandpass',freqmin = 1.0, freqmax = 20.0)
st.plot()
st_filt.plot()
```





1.6.2 Trimming data

You can shorten a stream and remove unwanted data with the "trim" method. For this method, you need UTCDateTime objects for the time you want your new trimmed trace to start and end.

Let's refresh our memories about the times in this data:

[34]:
```
print(st[0].stats)
```

```
network: IU
station: TUC
```

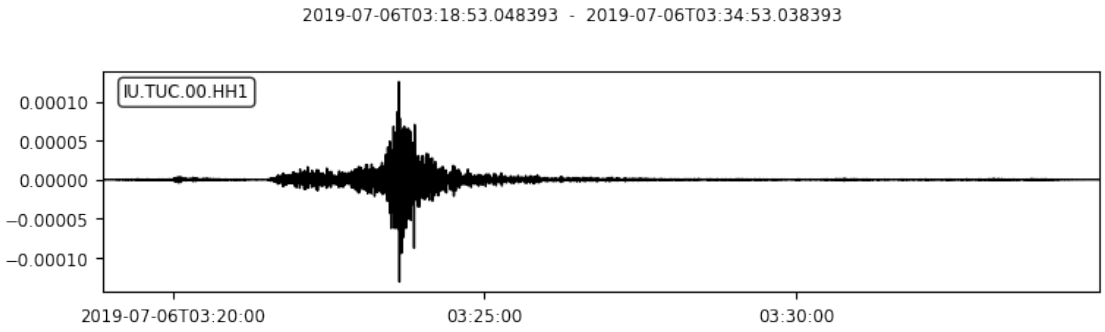
```
        location: 00
         channel: HH1
       starttime: 2019-07-06T03:18:53.048393Z
         endtime: 2019-07-06T03:34:53.038393Z
   sampling_rate: 100.0
           delta: 0.01
            npts: 96000
           calib: 1.0
         _format: MSEED
           mseed: AttribDict({'dataquality': 'M', 'number_of_records': 1715,
'encoding': 'FLOAT64', 'byteorder': '>', 'record_length': 512, 'filesize':
878080})
```

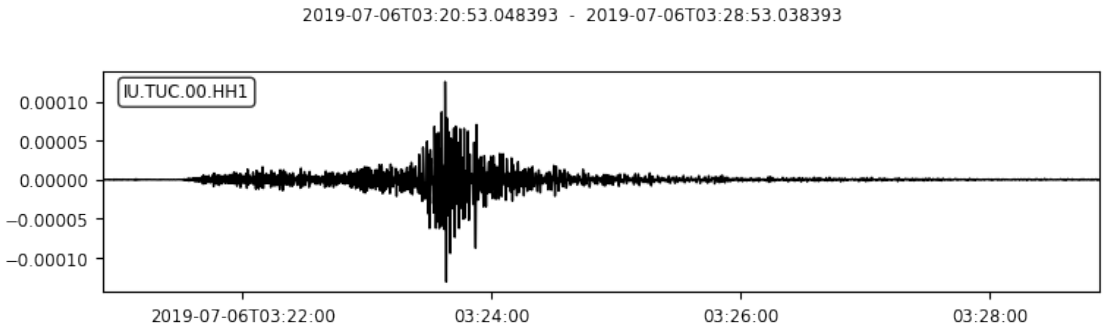
So let's say we want to chop 6 minutes off the end of this trace, and 2 minutes off the front.

[35]:
```
starttime = st[0].stats.starttime + 2*60
endtime = st[0].stats.endtime - 6*60
```

Don't forget to copy the stream again if you want to keep the original.

[37]:
```
st_trim = st_filt.copy()
st_trim.trim(starttime=starttime, endtime=endtime)
st_filt.plot()
st_trim.plot()
```





While trimming data, you also have an option to fill gaps with a value, and for this you just set fill_value=somenumber in the method after defining the start and end time.

1.6.3 Changing sampling rates

There are three methods in Obspy for changing the sampling rates of data in a stream or trace:

1. Decimate: downsamples data by an integer factor
2. Interpolate: increase sampling rate by interpolating (many method options)
3. Resample: resamples data using a Fourier method

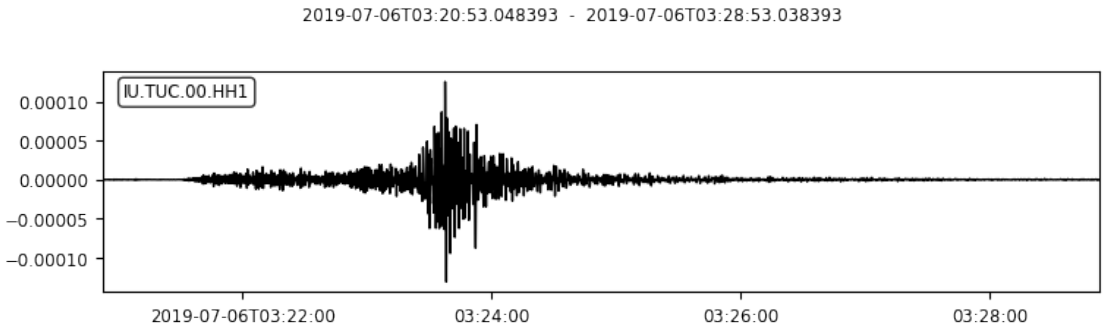
I'll demonstrate using decimate. Let's refresh our memories about this data's sampling rate:

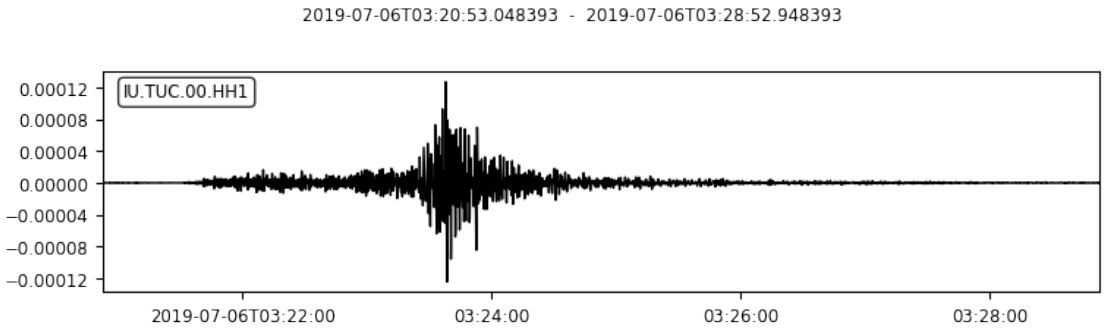
[38]:
```
st_trim[0].stats.sampling_rate
```

[38]:
```
100.0
```

Let's decimate by a factor of 10.

[39]:
```
st_dec = st_trim.copy()
st_dec.decimate(10)
st_trim.plot()
st_dec.plot()
```





If we check the stats again, you can see the sampling rate is different!

[40]:
```
st_dec[0].stats.sampling_rate
```

[40]:
```
10.0
```

Other methods you might be interested in using: differentiate, integrate, stack, merge, sort, and many others. To see how to use these, check the Obspy documentation for streams and traces, which I linked at the beginning of this section, as there are lists and links to how to implement each of them!

1.7 Matplotlib

Let's examine how we can make some basic plots of seismic data in Matplotlib.

We need to separate from the stream the times (independent variable) and the data (dependent variable).

[41]:
```
data = st[0].data

print(data)
```

```
[-9.40377808e-07 -9.38652106e-07 -9.40177916e-07 … -9.52581960e-07
 -1.10261625e-06 -9.28648806e-07]
```

Just as before, this pulls the data out into a Numpy array that can then be used for plotting.

Getting the times is slightly different in that it's a method ("times") instead of a property.

[42]:
```
times = st[0].times()

print(times)
```

```
[0.0000e+00 1.0000e-02 2.0000e-02 … 9.5997e+02 9.5998e+02 9.5999e+02]
```

Because these both come from the same data trace, the arrays should be the same length, and we should be able to stick them in a Matplotlib plot easily.

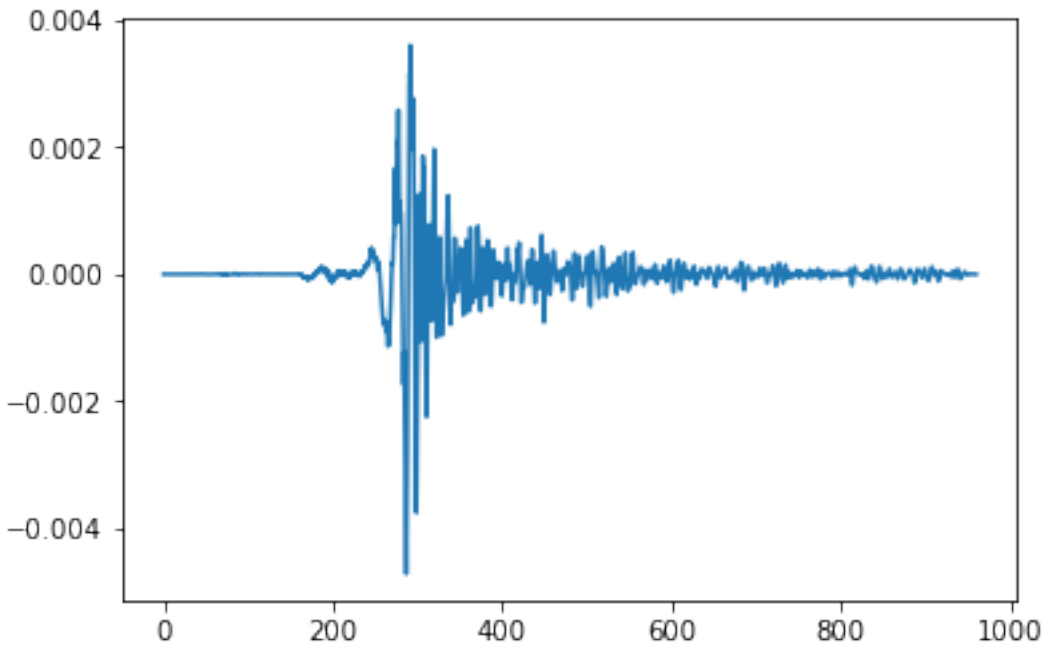
[43]:
```
print(len(data))
print(len(times))
```

```
96000
96000
```

Great! Let's try it out:

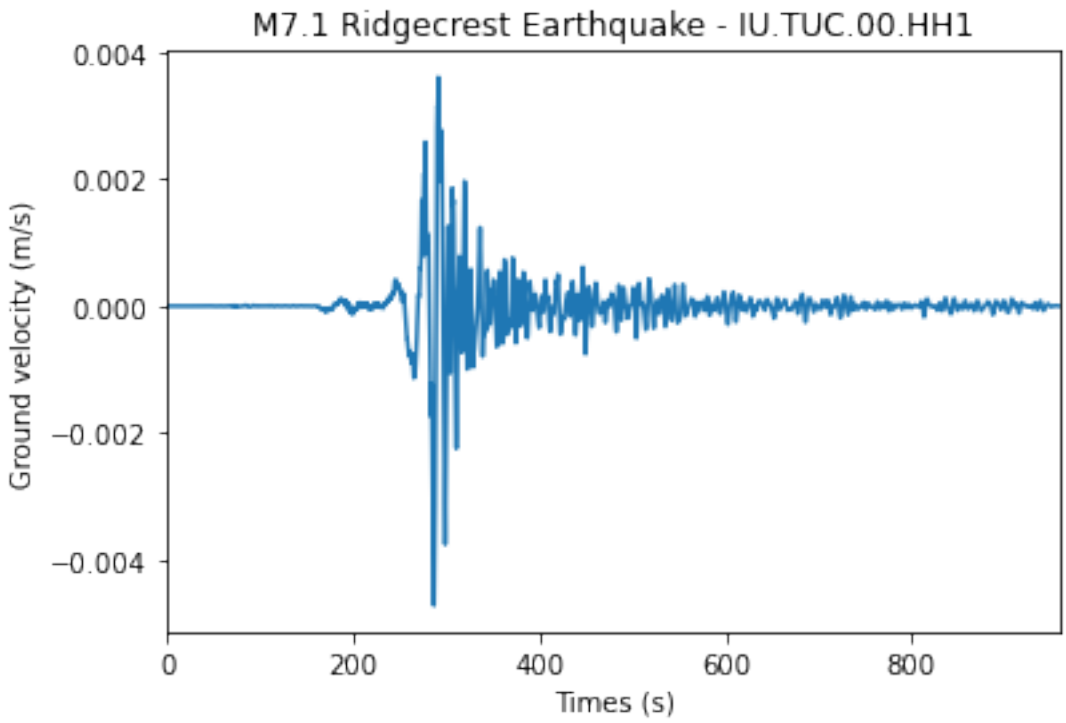
[44]:
```
import matplotlib.pyplot as plt
```

[47]:
```
plt.plot(times, data);
```



Here is what I might do to clean up this basic plot a bit:

[49]:
```
net = st[0].stats.network
sta = st[0].stats.station
loc = st[0].stats.location
chan = st[0].stats.channel
plt.plot(times, data)
plt.xlim(0,960)
plt.xlabel('Times (s)')
plt.ylabel('Ground velocity (m/s)')
plt.title(f'M7.1 Ridgecrest Earthquake - {net}.{sta}.{loc}.{chan}');
```



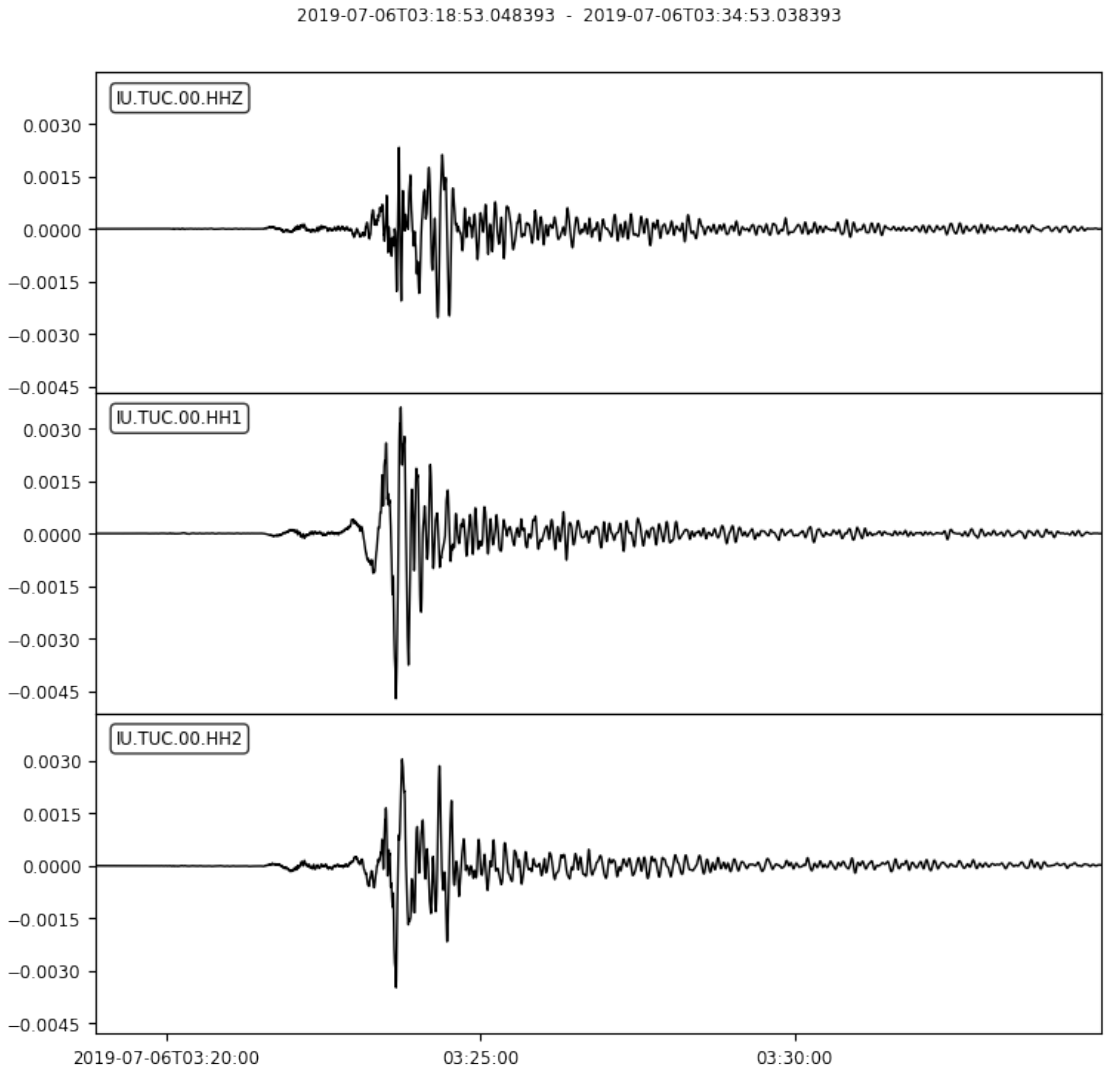
If you want to plot the data from three channels from the same instrument at the same time, you can use subplots.

To avoid confusion, let's just go back and re-download the data from this earthquake, and include all three channels. This code is copied from earlier in the notebook - the only difference is that I'll remove the response now from all three channels.

[50]:
```
time = UTCDateTime("2019-07-06T03:19:53.04")
starttime = time - 60
endtime = time + 60*15

net = "IU"
sta = "TUC"
loc = "00"
chan = "HH*"

st = client.get_waveforms(net, sta, loc, chan, starttime, endtime,␣
 ↪attach_response = True)
print(st)

st.remove_response(output = 'VEL')

st.plot();
```

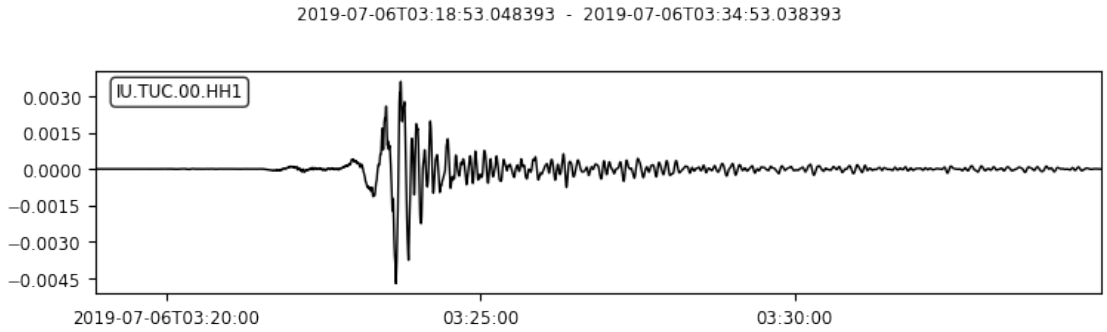
```
/Users/utpalkumar50/miniconda3/envs/roses/lib/python3.7/site-
```

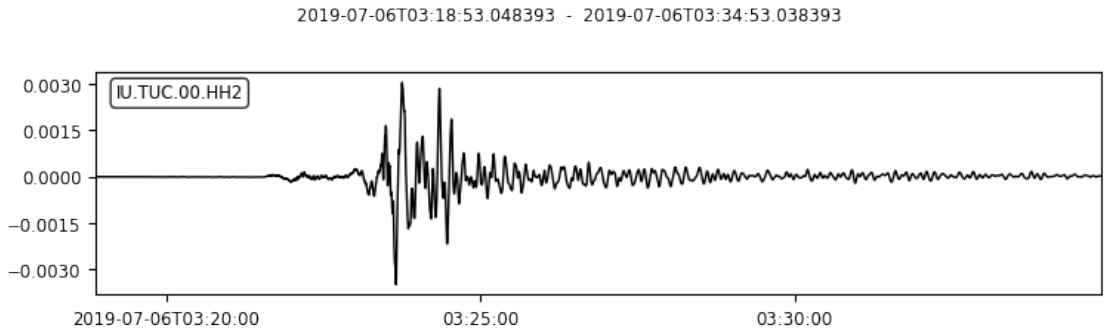
```
packages/obspy/io/stationxml/core.py:84: UserWarning: The StationXML file has
version 1.1, ObsPy can deal with version 1.0. Proceed with caution.
  root.attrib["schemaVersion"], SCHEMA_VERSION))

3 Trace(s) in Stream:
IU.TUC.00.HH1 | 2019-07-06T03:18:53.048393Z - 2019-07-06T03:34:53.038393Z |
100.0 Hz, 96000 samples
IU.TUC.00.HH2 | 2019-07-06T03:18:53.048393Z - 2019-07-06T03:34:53.038393Z |
100.0 Hz, 96000 samples
IU.TUC.00.HHZ | 2019-07-06T03:18:53.048393Z - 2019-07-06T03:34:53.038393Z |
100.0 Hz, 96000 samples
```

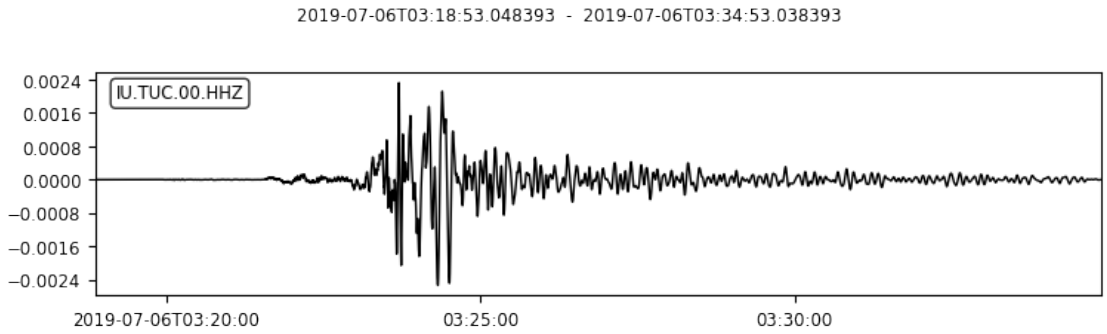


Alright! Let's plot each of these three channels on their own subplots. First, we have to separate each trace's data and times out.

[51]:
```
for i in range(3):
    st[i].plot();
```







We can then define them as such.

[56]:
```
HH1_data = st[0].data
HH1_times = st[0].times()

HH2_data = st[1].data
```

```
HH2_times = st[1].times()

HHZ_data = st[2].data
HHZ_times = st[2].times()
```

Now we can start plotting. We'll start by making a big "main" figure, and then we add subplots and the data to each of them.

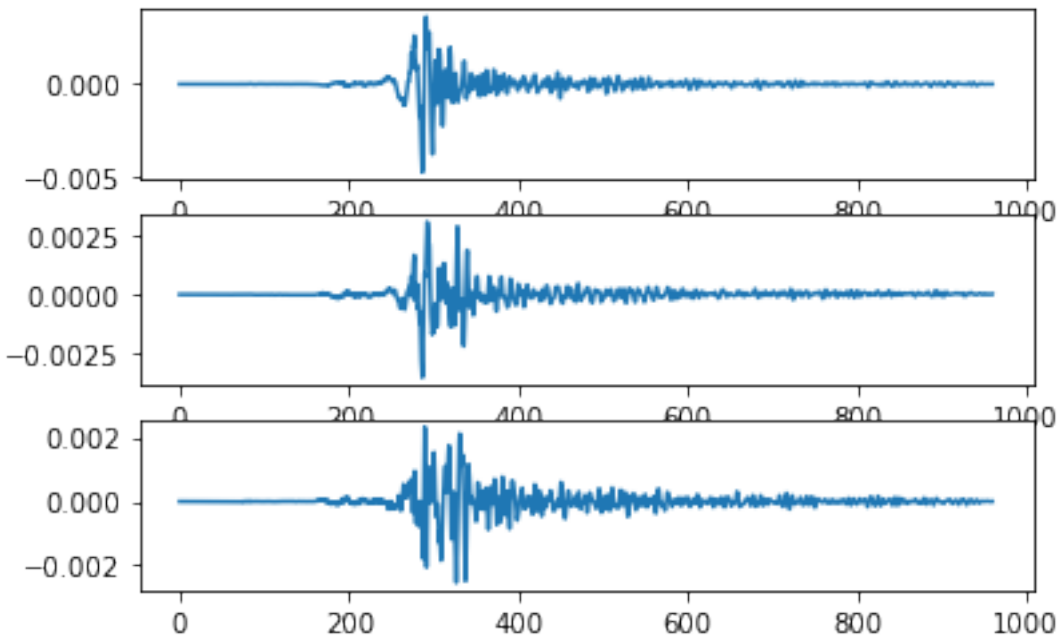
[57]:
```
fig= plt.figure(figsize=(8,10))
```

```
<Figure size 576x720 with 0 Axes>
```

To define subplots in Matplotlib, for each iteration you list the number of rows and number of columns you want, as well as which one you're on.

[58]:
```
plt.subplot(311)
plt.plot(HH1_times, HH1_data)

plt.subplot(312)
plt.plot(HH2_times, HH2_data)

plt.subplot(313)
plt.plot(HHZ_times, HHZ_data)
```

[58]:
```
[<matplotlib.lines.Line2D at 0x7fe5c48c2ad0>]
```

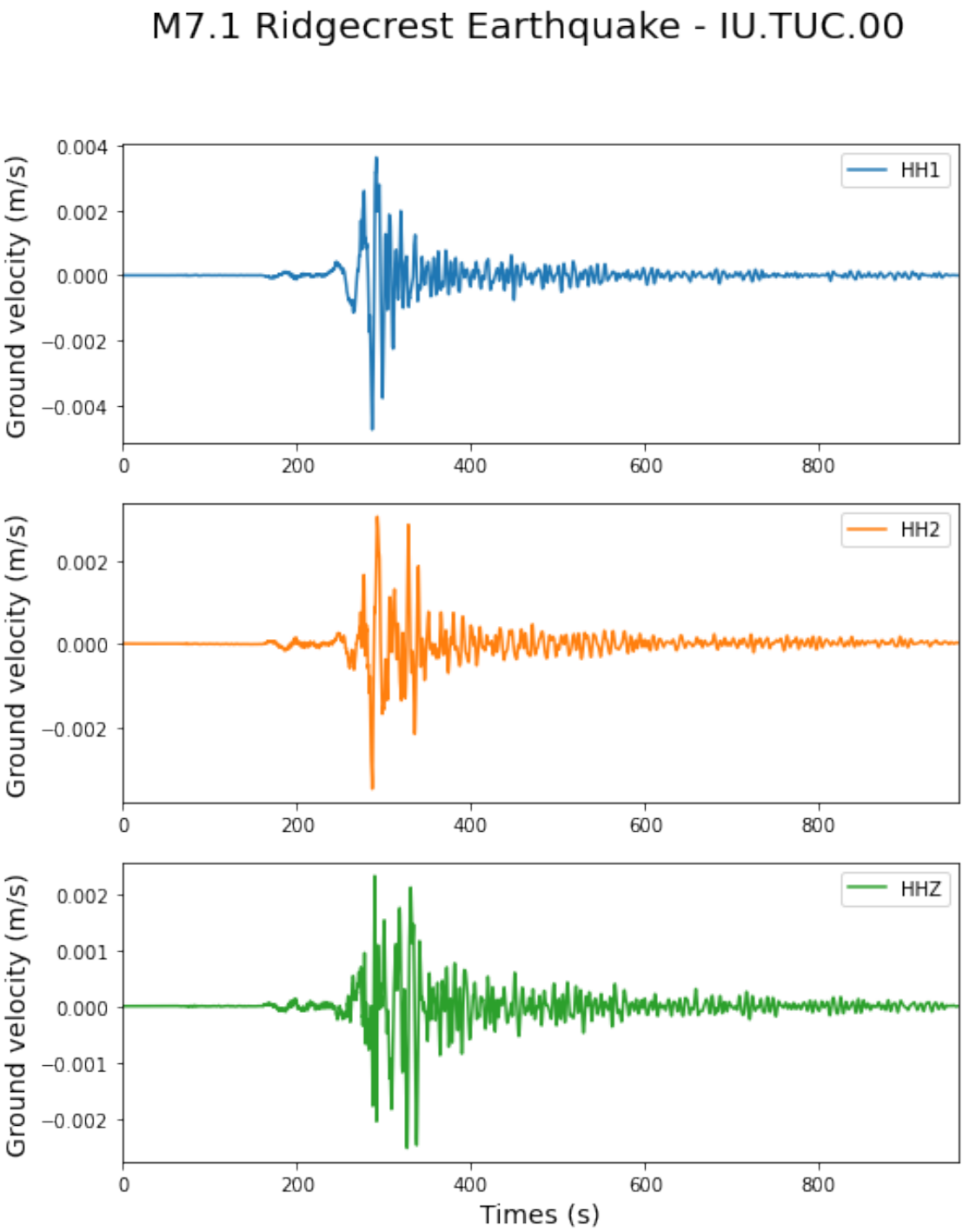


Let's try making the figure bigger. We can define a size in inches, with the width first and the height second.

[ ]:
```
fig = plt.figure()

plt.subplot(311)
plt.plot(HH1_times, HH1_data)

plt.subplot(312)
plt.plot(HH2_times, HH2_data)

plt.subplot(313)
plt.plot(HHZ_times, HHZ_data);
```

Now let's clean it up with some labels, like we did last time, and add some other features like legends and colors.

[59]:
```
fig = plt.figure(figsize = (8,10))
plt.suptitle(f"M7.1 Ridgecrest Earthquake - {net}.{sta}.{loc}",fontsize=20)
plt.subplot(311)
plt.plot(HH1_times, HH1_data, label='HH1', color='C0')
plt.xlim(0,960)
plt.ylabel('Ground velocity (m/s)',fontsize=14)
plt.legend()

plt.subplot(312)
plt.plot(HH2_times, HH2_data, label='HH2', color='C1')
plt.xlim(0,960)
plt.ylabel('Ground velocity (m/s)',fontsize=14)
plt.legend()

plt.subplot(313)
plt.plot(HHZ_times, HHZ_data, label='HHZ', color='C2')
plt.xlim(0,960)
plt.ylabel('Ground velocity (m/s)',fontsize=14)
plt.legend()
plt.xlabel('Times (s)', fontsize=14);
```



You could make this plot even better with more advanced Matplotlib capabilities that you can look up and mess with.

Here is one final useful Matplotlib capability, with code provided by Liam Toney (who will be teaching the PyGMT tutorial): plotting so that you have your UTC times on the x-axis, rather

than just seconds from zero, using matplotlib.dates.

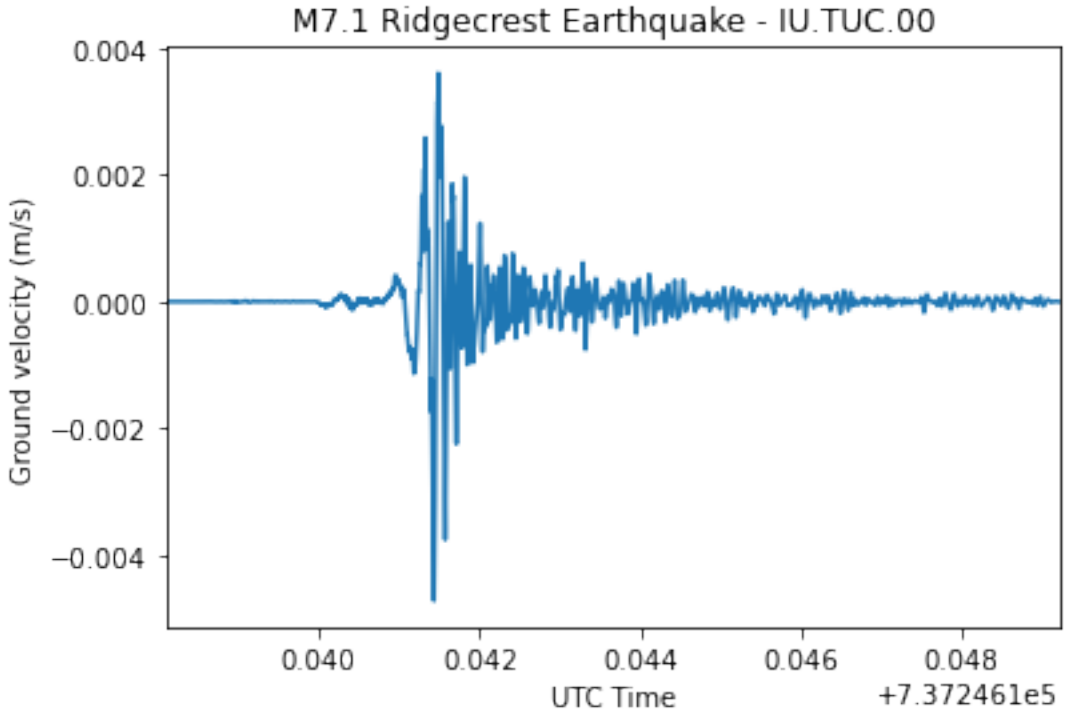
[60]:
```
import matplotlib.dates as mdates
```

We'll just demonstrate this on one channel - HH1. When we calculate the times this time, instead of leaving the parentheses blank, we specify the "type" of times as "matplotlib" so that we can plot them in UTC time.

[61]:
```
chan = 'HH1'
HH1_times_mpl = st[0].times(type = 'matplotlib')
```

Much of the plotting here is the same - it just incorporates another Matplotlib capability where you build your axes more manually. You can check out the documentation for this here. The basic plotting setup is as follows:

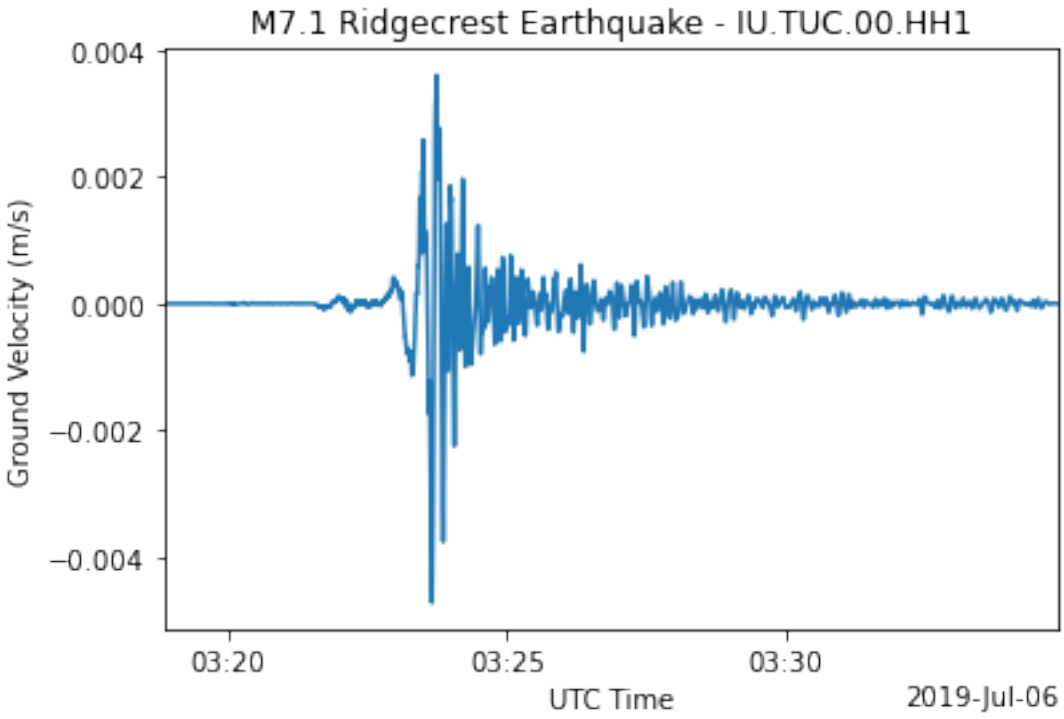
[63]:
```
fig, ax = plt.subplots()
ax.plot(HH1_times_mpl,HH1_data)
ax.set_xlim(HH1_times_mpl[0],HH1_times_mpl[-1])
ax.set_xlabel('UTC Time')
ax.set_ylabel('Ground velocity (m/s)')
ax.set_title(f"M7.1 Ridgecrest Earthquake - {net}.{sta}.{loc}");
```



Now we need to make the dates on the x-axis actually useful. We can do this with a locator. You can read more about this at the matplotlib.dates documentation linked above, but what this code does is allow Matplotlib to pick the best locations for the ticks on the x-axis and the best format

for them.

[64]:
```
fig, ax = plt.subplots()

ax.plot(HH1_times_mpl, HH1_data)
ax.set_xlim(HH1_times_mpl[0], HH1_times_mpl[-1])
ax.set_xlabel("UTC Time")
ax.set_ylabel("Ground Velocity (m/s)")
ax.set_title("M7.1 Ridgecrest Earthquake - " + net + "." + sta + "." + loc + ".
 ↪" + chan);

locator = ax.xaxis.set_major_locator(mdates.AutoDateLocator())
ax.xaxis.set_major_formatter(mdates.ConciseDateFormatter(locator))
```



There we go!

I hope this notebook serves as a good introduction and starting point to working with seismic data in Obspy, and a resource for your future coding needs!