Obspy_Tutorial_Lab_Notebook

June 26, 2020

1 Obspy: a Python Toolbox for seismology

1.1 Interactive Lab

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

Dependencies needed: Obspy, Numpy, Matplotlib This is the lab notebook associated with my Obspy introduction lecture. As a reminder, here are a few more Obspy resources if there are additional functions you want to use, but I didn't cover: the official Obspy Tutorial, and the Seismo-Live Juypter Notebooks for Seismology.

1.2 Download some earthquake data

Let's practice using the IRIS client to download data from an earthquake.

First: which tools do you need to import to be able to download data? If you need a reminder, check the "Downloading data from online repositories" section of the lecture notebook. If you missed it in the lecture, the full notebook has been posted to the #unit1-obspy channel on Slack. Don't forget to also set your client to IRIS.

[3]:
```
from obspy import read
from obspy import UTCDateTime
from obspy.clients.fdsn import Client
client = Client('IRIS')
```

Now you'll want to choose the time for the data you want to download, and make it into a UTCDateTime object. You can use the same time from the lecture if you want, but it would be more interesting if you chose your own earthquake!

Go to the USGS Earthquake Browser. You can just use one of the earthquakes that shows up on the default map if you want, or you can go to the right side of the page and click the gear, which allows you to change some of your earthquake search settings. If there's a particular earthquake from years ago that you want to use, you can click "Search Earthquake Catalog" for more advanced setting changes.

Once you pick your earthquake, click its icon on the map, and then on the earthquake's header, which will show up in the bottom left corner of your screen. This will open the earthquake information page, where one of the boxes will be the "Origin" information. If you click this, you will be able to get the exact UTC date and time at which the earthquake occurred!

If you are having trouble finding any of these things to click, I included pictures in this [Imgur album]. Hopefully that helps you out!

If you got that far, your next job is to turn that UTC date and time into an actual UTCDateTIme object.

[5]:
```
time = UTCDateTime('2020-01-09T08:38:08')
```

Now set your start time and end time for the segment of data you want to download.

[6]:
```
starttime = time - 60
endtime = time + 15*60
```

Great! You now have your two bookend times for the data you want to download. Where do we want to get it from? Remember, you need to know a network, station, location (if there is one - sometimes locations are just blank. If this happens, you can just set loc to be a wildcard), and channel or channels.

You can check out all of the available networks at IRIS MetaData Aggregator here. If you click on a network, you'll get a list of stations - and then if you click on a station, you get a list of locations and channels.

If you're understandably overwhelmed by the sheer number of networks available, you can just start with IU, which is the Global Seismic Network. There are stations all over the world, so you can just pick whichever is closest to the earthquake you chose, since the IU network page includes a link to a map of all of the stations!

Again, if you're having trouble figuring out what you're looking at on the IRIS MDA, check out this [Imgur album] with some screenshots!

[7]:
```
net = 'IU'
sta = 'PET'
loc = '00'
chan = 'BH*'
```

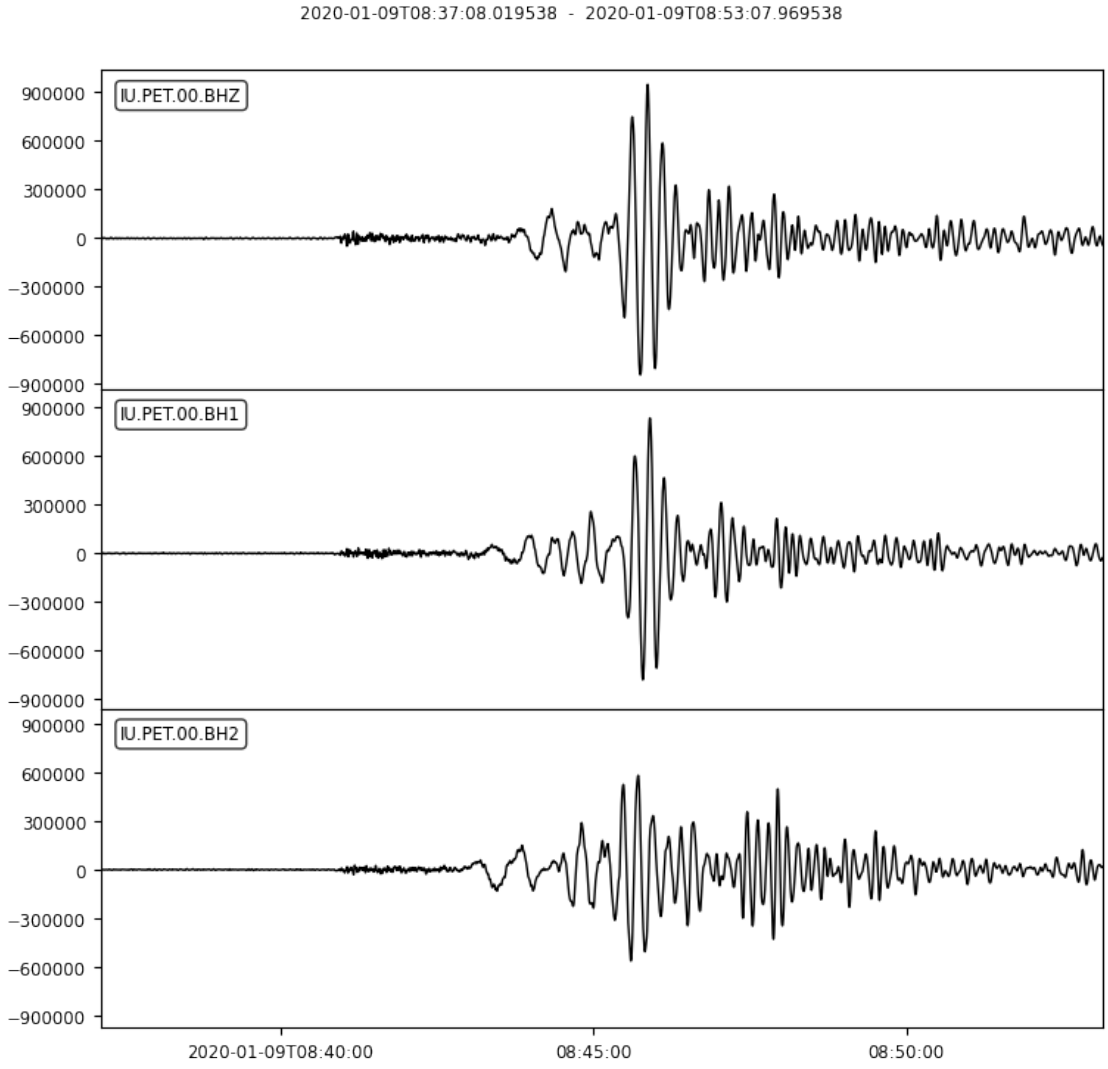
Now use get_waveforms to download the data (make sure you attach the instrument response!), check out the metadata, and make a simple plot with the stream method we used many times in the lecture.

If you get an error "FDSNNoDataException: No data available for request. Detailed response of server:", you either typed something wrong, the earthquake was so recent that data isn't available yet, or the instrument wasn't working. You can try to pick a different location or station, or a different earthquake. Experiment!

[10]:
```
st = client.get_waveforms(net, sta, loc, chan,starttime, endtime,␣
 ↪attach_response=True)
print(st)
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.PET.00.BH1 | 2020-01-09T08:37:08.019538Z - 2020-01-09T08:53:07.969538Z | 20.0
Hz, 19200 samples
IU.PET.00.BH2 | 2020-01-09T08:37:08.019538Z - 2020-01-09T08:53:07.969538Z | 20.0
Hz, 19200 samples
IU.PET.00.BHZ | 2020-01-09T08:37:08.019538Z - 2020-01-09T08:53:07.969538Z | 20.0
Hz, 19200 samples
```



1.3 Remove the instrument response

Use the stream method remove_response to convert your earthquake data from digital counts to velocity, acceleration, or displacement (your choice!). Visit this section of the lecture notebook if you don't remember how, and remember to copy your original data stream first if you want to keep it and be able to plot them side by side to compare them!

[11]:
```
st_rem = st.copy()
st_rem.remove_response(output='VEL')
```

[11]:
```
3 Trace(s) in Stream:
IU.PET.00.BH1 | 2020-01-09T08:37:08.019538Z - 2020-01-09T08:53:07.969538Z | 20.0
Hz, 19200 samples
IU.PET.00.BH2 | 2020-01-09T08:37:08.019538Z - 2020-01-09T08:53:07.969538Z | 20.0
Hz, 19200 samples
IU.PET.00.BHZ | 2020-01-09T08:37:08.019538Z - 2020-01-09T08:53:07.969538Z | 20.0
Hz, 19200 samples
```

1.4 Write your downloaded data to a file

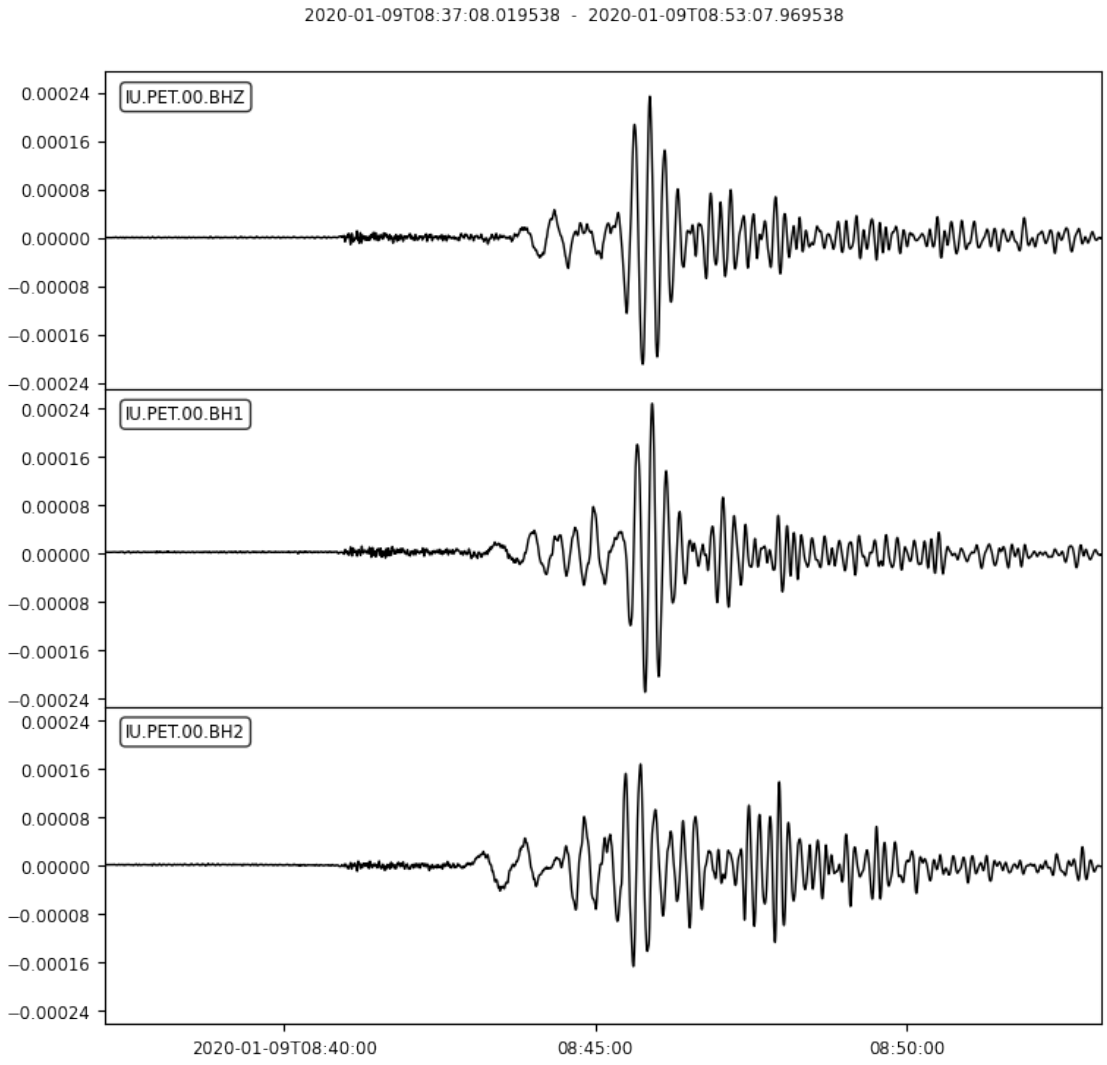
Save your data to a file! You can choose to use miniSEED format like we did in lecture, or pick something else. You can choose to save each channel if you downloaded more than one to its own file if you want.

[14]:
```
# for tr in st:
#     print(tr.stats.channel)
st_rem.write(f'{net}.{sta}.mseed')
```

Now read it back in as a stream to make sure it worked, check the metadata, and make a plot. Don't forget, you need to import read from Obspy first because we haven't used it yet!

[15]:
```
st = read(f'{net}.{sta}.mseed')
print(st)
st.plot()
```

```
3 Trace(s) in Stream:
IU.PET.00.BH1 | 2020-01-09T08:37:08.019538Z - 2020-01-09T08:53:07.969538Z | 20.0
Hz, 19200 samples
IU.PET.00.BH2 | 2020-01-09T08:37:08.019538Z - 2020-01-09T08:53:07.969538Z | 20.0
Hz, 19200 samples
IU.PET.00.BHZ | 2020-01-09T08:37:08.019538Z - 2020-01-09T08:53:07.969538Z | 20.0
Hz, 19200 samples
```



1.5 Try out some Stream and Trace Methods

We worked with a few different public methods for stream and trace objects in the lecture. Try some of them out, changing options if you can! In lecture we filtered our data (try a different kind!), trimmed it, and changed the sampling rate (try a different method!), so you can go back to your notebook or the full one provided in Slack for a reminder of how to do these things if needed.

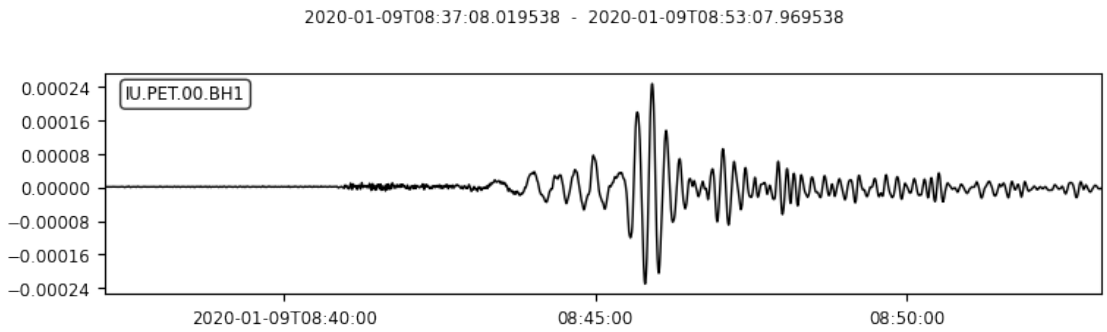
In addition to practicing implementing these methods, pick another one from the lists in the documentation either for traces or streams and see if you can get it working.

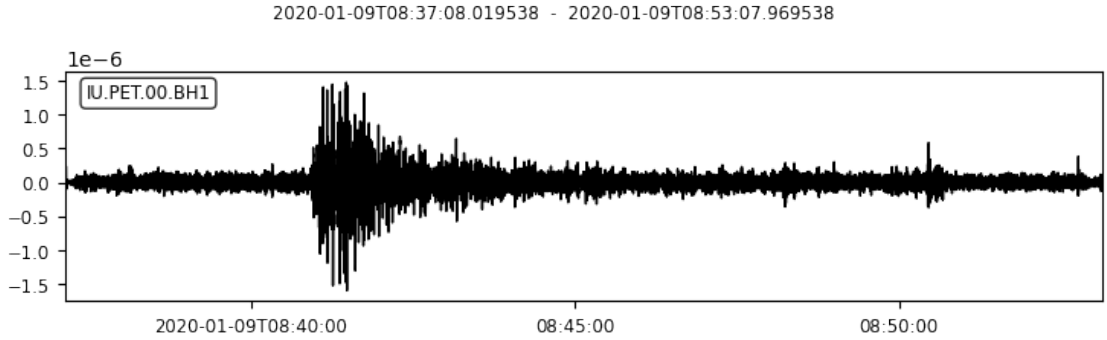
Don't forget to copy your data stream before you implement any of these methods!

1.5.1 Filtering

[17]:
```
st_filt = st[0].copy()
st_filt.filter('bandpass',freqmin = 1.0, freqmax = 20.0)
st[0].plot()
st_filt.plot()
```

```
/Users/utpalkumar50/miniconda3/envs/roses/lib/python3.7/site-
packages/obspy/signal/filter.py:67: UserWarning: Selected high corner frequency
(20.0) of bandpass is at or above Nyquist (10.0). Applying a high-pass instead.
  warnings.warn(msg)
```

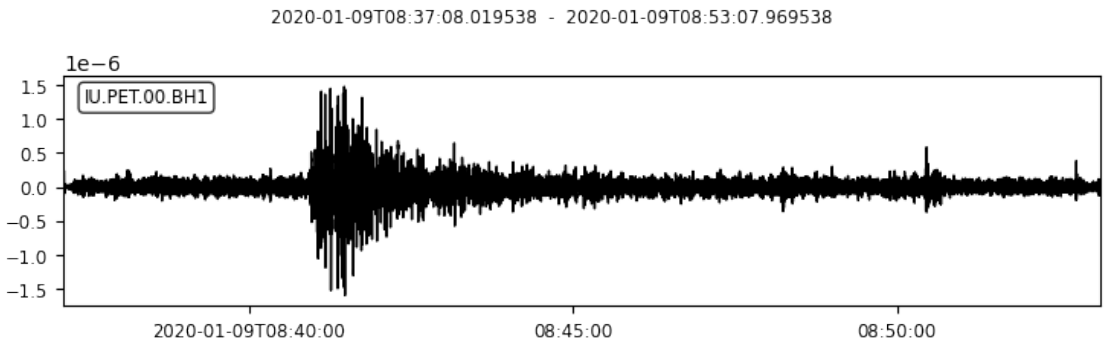


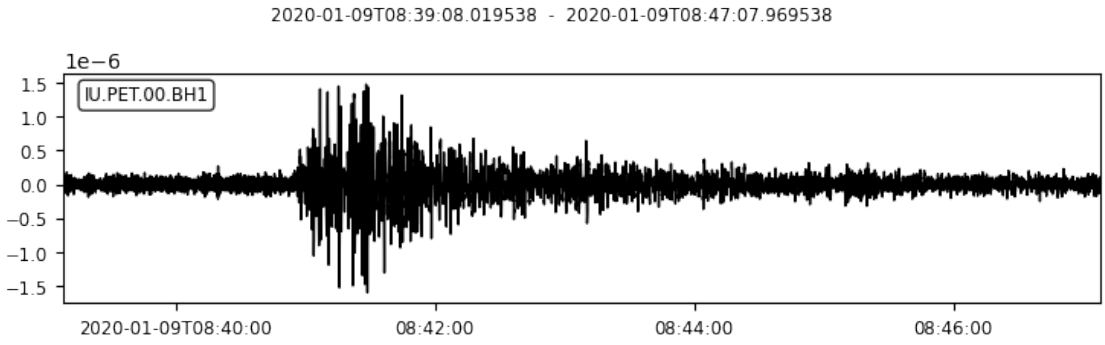


1.5.2 Trimming data

[18]:
```
starttime = st[0].stats.starttime + 2*60
endtime = st[0].stats.endtime - 6*60

st_trim = st_filt.copy()
st_trim.trim(starttime=starttime, endtime=endtime)
st_filt.plot()
st_trim.plot()
```

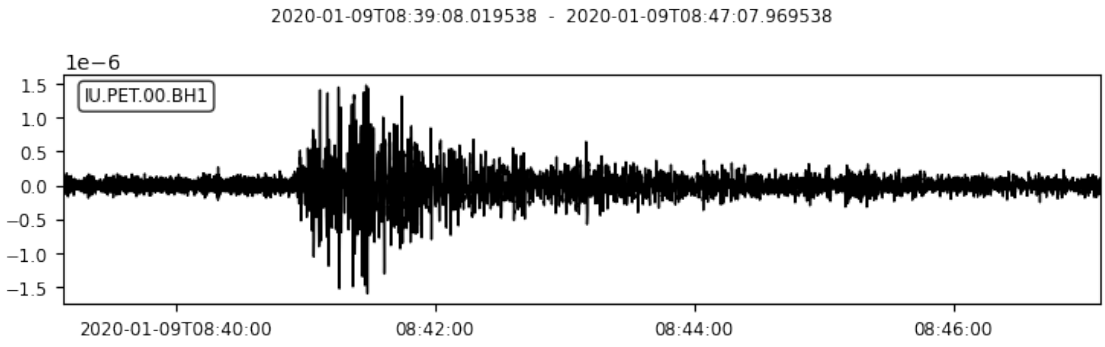


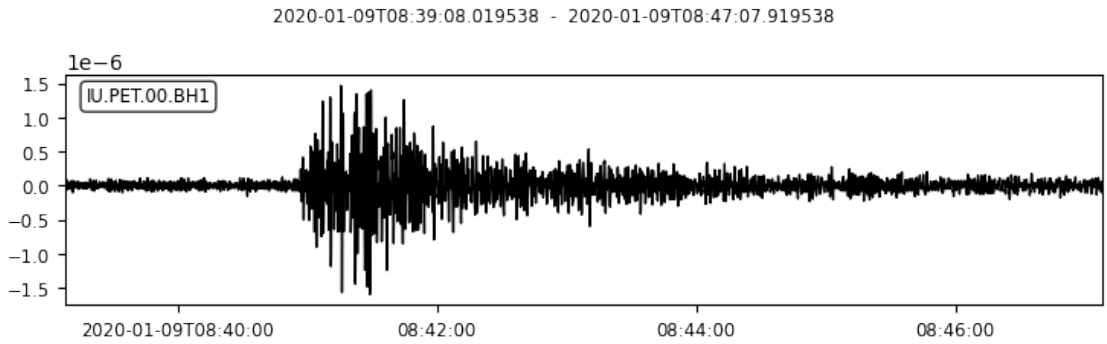


1.5.3 Changing sampling rates

[21]:
```
st_trim.stats.sampling_rate

st_dec = st_trim.copy()
st_dec.decimate(2)
st_trim.plot()
st_dec.plot()
```





1.6 Practice plotting in Matplotlib

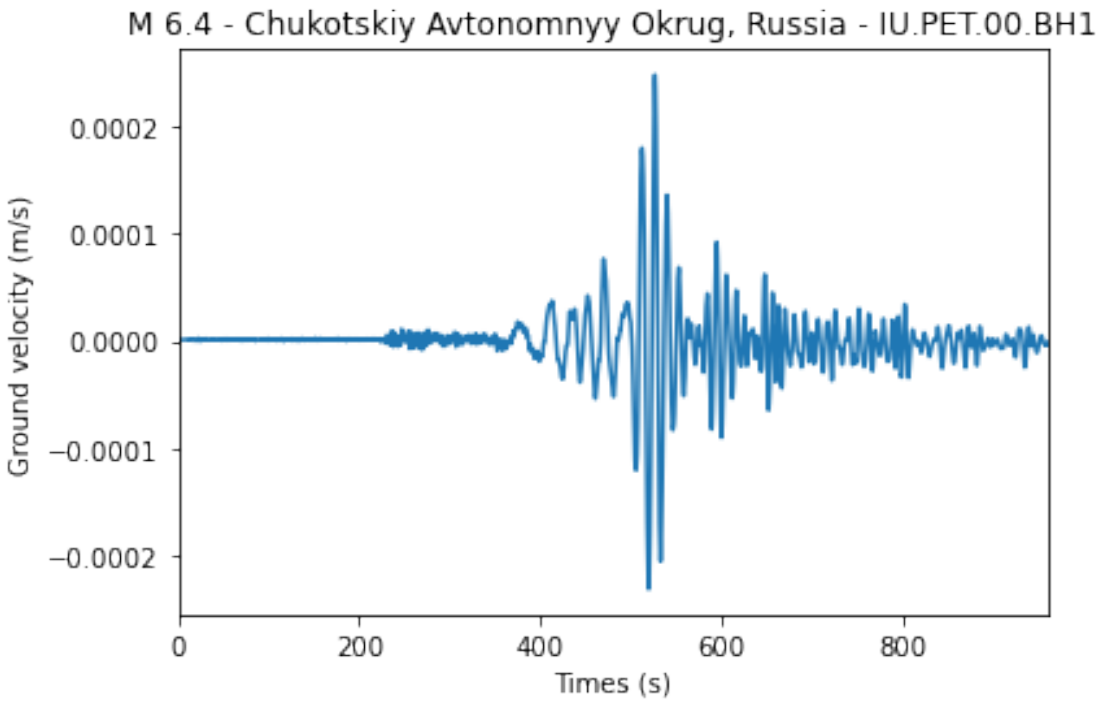
The last thing I'd like you to practice implementing in this lab is to make some nicer plots of your seismic data using Matplotlib. As I mentioned in lecture, you can spend hours messing with plots to make them better, so this can be a very freeform section! Don't forget you'll need to import matplotlib.pyplot, as well as matplotlib.mdates if you want to put UTC times on your x-axis.

Like we did in lecture, start by separating out the times and data into their Numpy arrays so you can plot them. Then use the lecture notebook (or full notebook on Slack) to plot the data and add whatever features you want. You can check out the Matplotlib website if you want to make it more complicated than we did in lecture!

[23]:
```
import matplotlib.pyplot as plt
data = st[0].data
times = st[0].times()

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'M 6.4 - Chukotskiy Avtonomnyy Okrug, Russia - {net}.{sta}.{loc}.
 ↪{chan}');
```

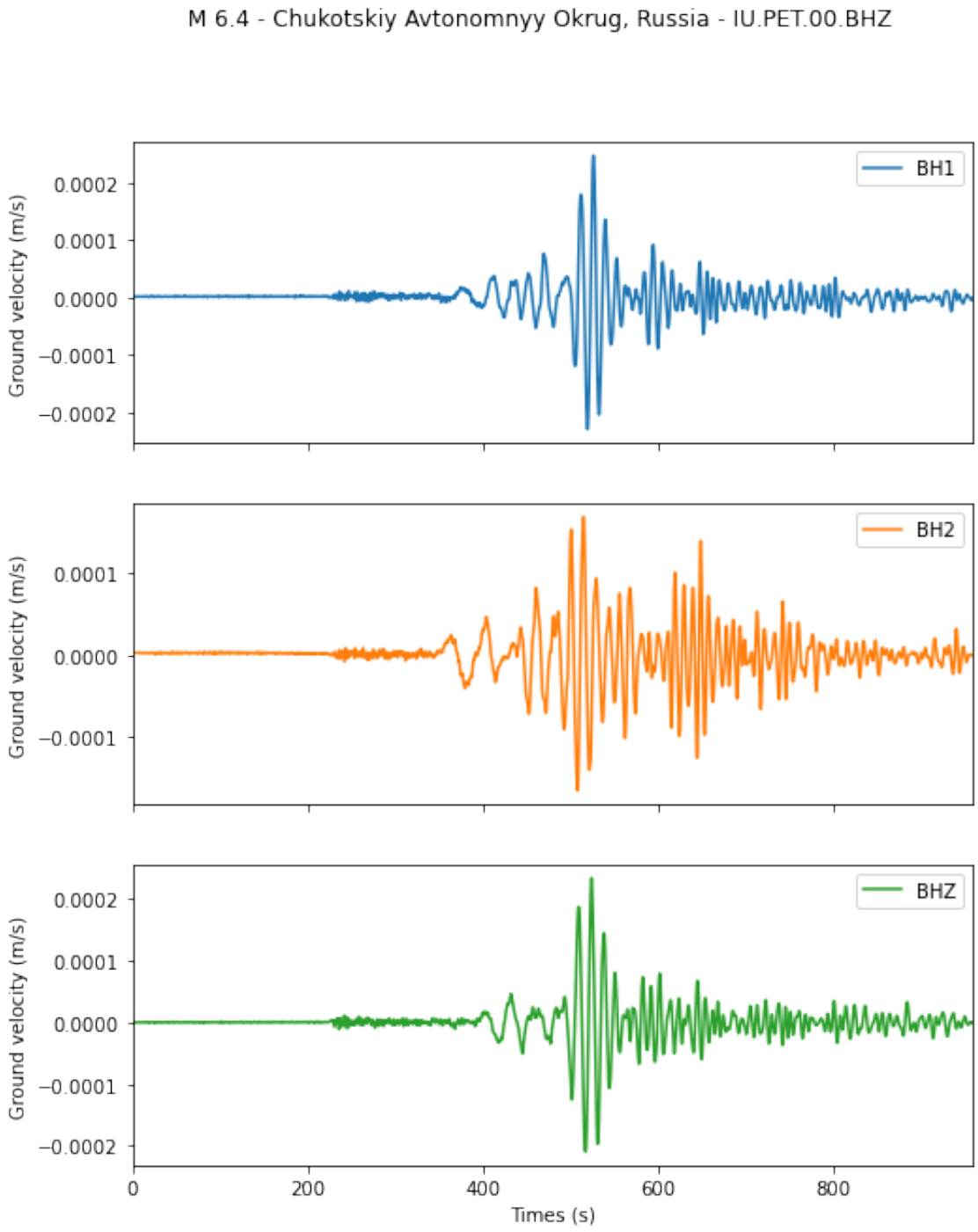


[33]:
```
fig, ax = plt.subplots(3,1,figsize=(8,10),sharex=True)
for i,tr in enumerate(st):
    data = tr.data
    times = tr.times()

    net = tr.stats.network
    sta = tr.stats.station
    loc = tr.stats.location
    chan = tr.stats.channel

    ax[i].plot(times, data,label=chan, color=f'C{i}')
    ax[i].set_xlim(0,960)
    ax[i].set_ylabel('Ground velocity (m/s)')
    ax[i].legend()

plt.xlabel('Times (s)')
plt.suptitle(f'M 6.4 - Chukotskiy Avtonomnyy Okrug, Russia - {net}.{sta}.{loc}.
 ↪{chan}');
```



Thank you all for attending this session today, and I hope it was useful for you!