from \_\_future\_\_ import unicode\_literals

import matplotlib.pyplot as plt

from scipy.fftpack import fft

from scipy.io import wavfile

from scipy import signal

from obspy import read

from obspy import UTCDateTime

from scipy.signal import get\_window

from scipy.signal import spectrogram

from matplotlib.colors import Normalize

import sys

import os

import wave

import numpy as np

import pylab

import datetime

dt = UTCDateTime("2019-01-01T01:00:00.008300Z")

st = read('ABC.Z.2019001.A.sac',

debug\_headers=True, starttime=dt, endtime=dt+1200)

######################

tr = st[0]

# Filtering with a highpass on a copy of the original Trace

tr\_filt = tr.copy()

tr\_filt.filter('highpass', freq=5.0, corners=2, zerophase=True)

# Now let's plot the raw and filtered data...

t = np.arange(0, tr.stats.npts / tr.stats.sampling\_rate, tr.stats.delta)

plt.subplot(211)

plt.plot(t, tr.data, 'k', color='red')

plt.ylabel('Raw Data')

plt.subplot(212)

plt.plot(t, tr\_filt.data, 'k')

plt.ylabel('Highpass Data')

plt.xlabel('Time [s]')

plt.suptitle(tr.stats.starttime)

plt.show()

dt = UTCDateTime("2019-01-01T01:00:00.008300Z")

st = read('ABC.Z.2019001.A.sac',

debug\_headers=True, starttime=dt, endtime=dt+1200)

for tr in st:

my\_sta = tr.stats.station

my\_chan = tr.stats.channel

chan\_samps = tr.stats.sampling\_rate

spec\_scale = 'Log'

my\_winlen = 5.0

my\_overlap = 0.9

idx1 = 0

idx2 = 60 \* 10

dbscale = True

log = False

clip = [0.0, 1.0]

axes = False

cmap = 'jet'

zorder = None

tr\_time = tr.stats.starttime

tr\_1 = tr.slice(tr\_time+idx1,tr\_time+idx2)

tr1\_filt = tr\_1.filter("highpass",freq=5)

print(tr1\_filt)

npts = tr1\_filt.stats.npts

end = npts / chan\_samps

fig, ax = plt.subplots(1,1, sharey=False, sharex = False)

fig.set\_size\_inches(12,8)

freq, time, specgram = spectrogram(tr1\_filt.data, fs=chan\_samps,

nperseg=int(my\_winlen\*chan\_samps),

noverlap=int(my\_winlen\*chan\_samps\*my\_overlap),

window=get\_window('hann',int(my\_winlen\*chan\_samps)))

if dbscale:

specgram = 10 \* np.log10(specgram[1:,:])

else:

specgram = np.sqrt(specgram[1:,:])

freq = freq[1:]

#vmin, vmax, = clip

#if vmin < 0 or vmax > 1 or vmin >= vmax:

# msg = "invalid parameters for clip option"

#calc half bin width

halfbin\_time = (time[1] - time[0]) / 2.0

halfbin\_freq = (freq[1] - freq[0]) / 2.0

kwargs = {k: v for k, v in (('cmap', cmap),('zorder',zorder))

if v is not None}

if log:

freq = np.concatinate((freq,[freq[-1] + 2 \* halfbin\_freq]))

time = np.concatinate((time,[time[-1] + 2 \* halfbin\_time]))

time = halfbin\_time

freq = halfbin\_freq

ax.set\_yscale('log')

ax.pcolormesh(time, freq, specgram,norm=norm, \*\*kwargs)

else:

specgram = np.flipud(specgram)

extent = (time[0] - halfbin\_time,time[-1] + halfbin\_time, freq[0] - halfbin\_freq, freq[-1] + halfbin\_freq)

ax.imshow(specgram, interpolation='nearest',extent=extent, \*\*kwargs)

ax.axis('tight')

ax.set\_xlim(0,end)

ax.grid(False)

plt.show()