EqLock: Statistics of Earthquakes and Detector Lock Loss

Ashley Patron

August 29, 2022

Abstract

The seismometers of LIGO's detectors are highly sensitive to earthquakes. Events of high ground motion not only contribute a significant amount of noise to LIGO's data, but they can cause lock loss in the detector, meaning that its cavities are no longer at resonant frequency. Therefore, gathering statistics between high ground motion and the lock status can ultimately improve understanding of how earthquakes affect the stability of the detector. For this project, called EqLock, we looked in the earthquake band (0.03-0.1 Hz) for significant peaks in ground motion at ETMX, ETMY, and ITMY stations during every month of LIGO's third observing run (O3). Specifically focusing on the vertical ground motion at the peaks, earthquakes are categorized as "survived" or "lost lock" based on if the lock value drops within 20 minutes before and after the peak in velocity. Next, the code gathers relevant information about the ground motion, times of peak ground velocity, times of lock loss, and more. The final result of EqLock is a plot of the probability of survival based on the magnitude of each earthquake's peak ground velocity. As expected, the probability that the detector survives an earthquake event without losing lock is higher for smaller ground velocities and lower for higher ground velocities. At Livingston, 177 out of the 362 earthquake events resulted in lock loss, and 194 out of 434 earthquakes resulted in lock loss at Hanford. According to the results, no earthquakes above $5 * 10^3$ nm/s were survived during O3 and Hanford and Livingston.

Introduction

Ground motion contributes a lot of noise to LIGO's data, and the high velocities produced by earthquakes can cause the detector to lose lock. This project takes interest in the relationship between the lock status and the magnitude of seismic motion, specifically the probability of LIGO surviving an earthquake event.

I aimed to reproduce similar results shown in Figure 10 of the paper "Improving the Robustness of the Advanced LIGO Detectors to Earthquakes" [1], which shows that in general, the probability of surviving an earthquake lowers as its magnitude increases. My project originated when Arnaud Pele sent the idea out to LIGO members to translate the code used for the paper from Matlab to Python. While the code I wrote is not a direct translation, it does find the probability of surviving an earthquake of a certain vertical ground velocity without losing lock. The original Matlab codes I reference, one for Livingston and one for Hanford, are provided in the Appendix.

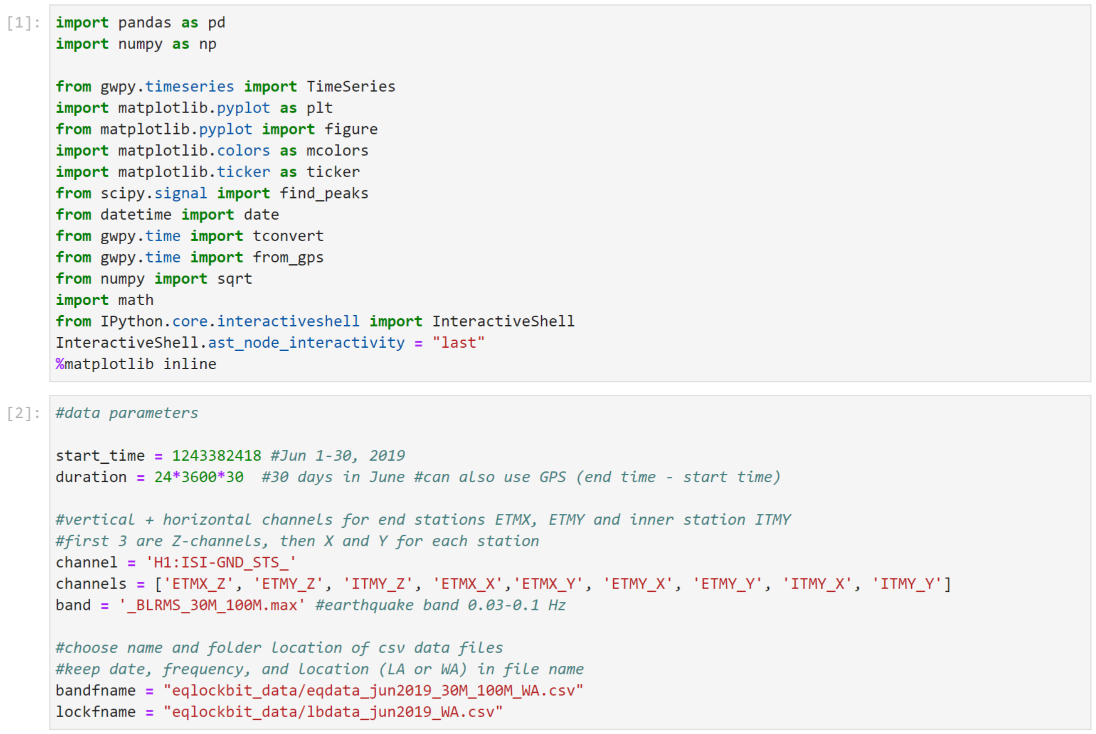
Lastly, the Python code I've provided, which I call EqLock (shortened from "Earthquake Lock"), can be generalized to analysis of other frequency bands by altering certain parameters. This allows LIGO members to use EqLock for more than what is shown in this particular report about earthquakes.

The main goals of this project are to identify earthquakes in the earthquake band (0.03-0.1 Hz), get statistics involving seismic motion and lock status of the detector, and compare the results of Livingston and Hanford for O3, the third observing run. Another goal of this project is to provide a tool for other LIGO members to use for similar analyses. Ultimately, I hope this project can help lower noise caused by earthquakes, improve the seismic isolation systems, and reduce the effect of earthquakes on the detector's stability.

Methods and Example Code

I completed this notebook in Python on JupyterLab in the CalTech servers for Livingston and Hanford. The EqLock notebook is divided into five sections. The following information about parameters and outputs for each section is also provided in markdown cells in EqLock, but it is less comprehensive than in this documentation.

Figure 1 shows the introductory cells of EqLock, which imports necessary packages and defines seven parameters used throughout the code; these parameters are the (1) GPS start time of the segment of interest, (2) duration of the time segment, (3) channel name, (4) channel stations, (5) channel frequency band, (6) name of the file of seismic data, and (7) name of the file for lock data. Note that parameters 3, 4, and 5 are combined together to form the complete channel name.



```
[1]: import pandas as pd
     import numpy as np

     from gwpy.timeseries import TimeSeries
     import matplotlib.pyplot as plt
     from matplotlib.pyplot import figure
     import matplotlib.colors as mcolors
     import matplotlib.ticker as ticker
     from scipy.signal import find_peaks
     from datetime import date
     from gwpy.time import tconvert
     from gwpy.time import from_gps
     from numpy import sqrt
     import math
     from IPython.core.interactiveshell import InteractiveShell
     InteractiveShell.ast_node_interactivity = "last"
     %matplotlib inline

[2]: #data parameters

     start_time = 1243382418 #Jun 1-30, 2019
     duration = 24*3600*30  #30 days in June #can also use GPS (end time - start time)

     #vertical + horizontal channels for end stations ETMX, ETMY and inner station ITMY
     #first 3 are Z-channels, then X and Y for each station
     channel = 'H1:ISI-GND_STS_'
     channels = ['ETMX_Z', 'ETMY_Z', 'ITMY_Z', 'ETMX_X','ETMX_Y', 'ETMY_X', 'ETMY_Y', 'ITMY_X', 'ITMY_Y']
     band = '_BLRMS_30M_100M.max' #earthquake band 0.03-0.1 Hz

     #choose name and folder location of csv data files
     #keep date, frequency, and location (LA or WA) in file name
     bandfname = "eqlockbit_data/eqdata_jun2019_30M_100M_WA.csv"
     lockfname = "eqlockbit_data/lbdata_jun2019_WA.csv"
```

Figure 1: Import statements and declaration of original parameters.

If importing large amounts of data, I recommend breaking the time up and repeating Sections 1, 2, and 3. If separated by month, for example, Section 1 saves two CSV files (raw seismic motion and lock data) for each month. Section 2 then loads the data from these CSV files, and Section 3 relies on the earthquakes found in Section 2 to obtain and save a dataframe of statistics. Section 4 is optional but can combine these files if Sections 1-3 are repeated. Lastly, Section 5 finds the probability of survival. This part of the paper elaborates on the methods alongside an example code.

Section 1: Get earthquake and lockbit data

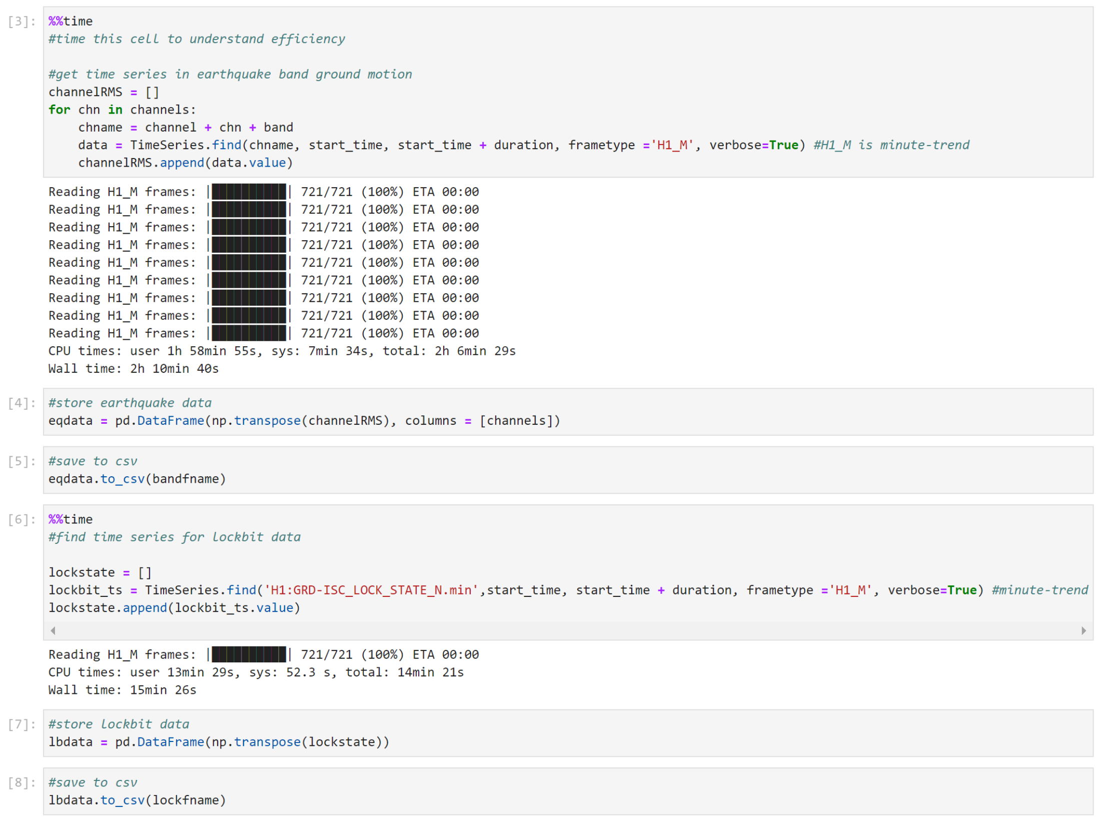
First, import the time series of the seismic motion and lock data as minute trends, then save the resulting dataframes to CSV files.

Note on efficiency: this is most time-consuming part of code! For one month of data, it takes between 1.5-4.5 hours (typically around 3 hours) to get the seismic TimeSeries and 10-30 minutes to get lockbit TimeSeries data, as shown in cells [3] and [6] of Figure 2, respectively.

Section 2: Combine data and find earthquakes

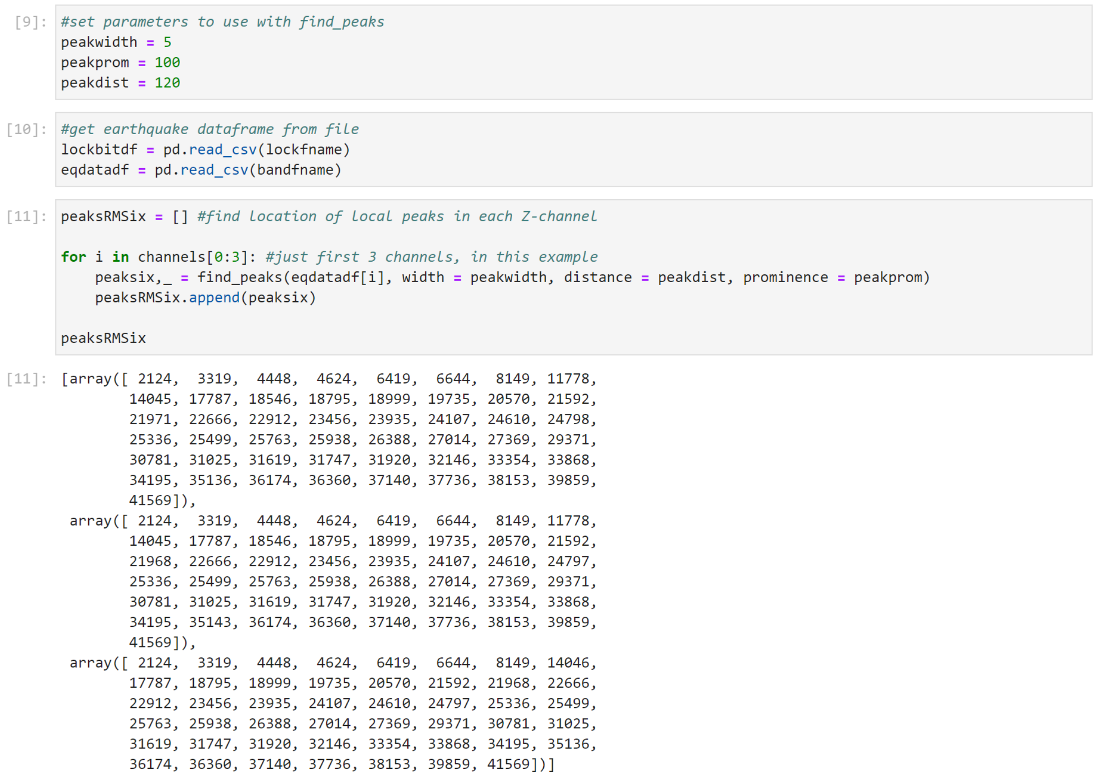
Three key parameters set here—width, prominence, and distance—define what is considered a peak in the data, thus what characterizes an earthquake. These parameters are used as arguments in the *find_peaks* command.

I chose width of 5 minutes, prominence of 100, and distance of 120 minutes. The process for choosing the peak parameters is completely up to the scientist, and my method was to start from the given parameters and change them based on my intermediate results. I began with the parameters chosen in the original Matlab code (width 5 minutes, prom. 200 or 130, and distance at least 60 minutes) then altered them until the code best identified seismic motion peaks. These three parameters are defined in cell [9] of Figure 3.



```
[3]: %%time
     #time this cell to understand efficiency

     #get time series in earthquake band ground motion
     channelRMS = []
     for chn in channels:
         chname = channel + chn + band
         data = TimeSeries.find(chname, start_time, start_time + duration, frametype ='H1_M', verbose=True) #H1_M is minute-trend
         channelRMS.append(data.value)

     Reading H1_M frames: |██████████| 721/721 (100%) ETA 00:00
     Reading H1_M frames: |██████████| 721/721 (100%) ETA 00:00
     Reading H1_M frames: |██████████| 721/721 (100%) ETA 00:00
     Reading H1_M frames: |██████████| 721/721 (100%) ETA 00:00
     Reading H1_M frames: |██████████| 721/721 (100%) ETA 00:00
     Reading H1_M frames: |██████████| 721/721 (100%) ETA 00:00
     Reading H1_M frames: |██████████| 721/721 (100%) ETA 00:00
     Reading H1_M frames: |██████████| 721/721 (100%) ETA 00:00
     Reading H1_M frames: |██████████| 721/721 (100%) ETA 00:00
     CPU times: user 1h 58min 55s, sys: 7min 34s, total: 2h 6min 29s
     Wall time: 2h 10min 40s

[4]: #store earthquake data
     eqdata = pd.DataFrame(np.transpose(channelRMS), columns = [channels])

[5]: #save to csv
     eqdata.to_csv(bandfname)

[6]: %%time
     #find time series for lockbit data

     lockstate = []
     lockbit_ts = TimeSeries.find('H1:GRD-ISC_LOCK_STATE_N.min',start_time, start_time + duration, frametype ='H1_M', verbose=True) #minute-trend
     lockstate.append(lockbit_ts.value)

     Reading H1_M frames: |██████████| 721/721 (100%) ETA 00:00
     CPU times: user 13min 29s, sys: 52.3 s, total: 14min 21s
     Wall time: 15min 26s

[7]: #store lockbit data
     lbdata = pd.DataFrame(np.transpose(lockstate))

[8]: #save to csv
     lbdata.to_csv(lockfname)
```

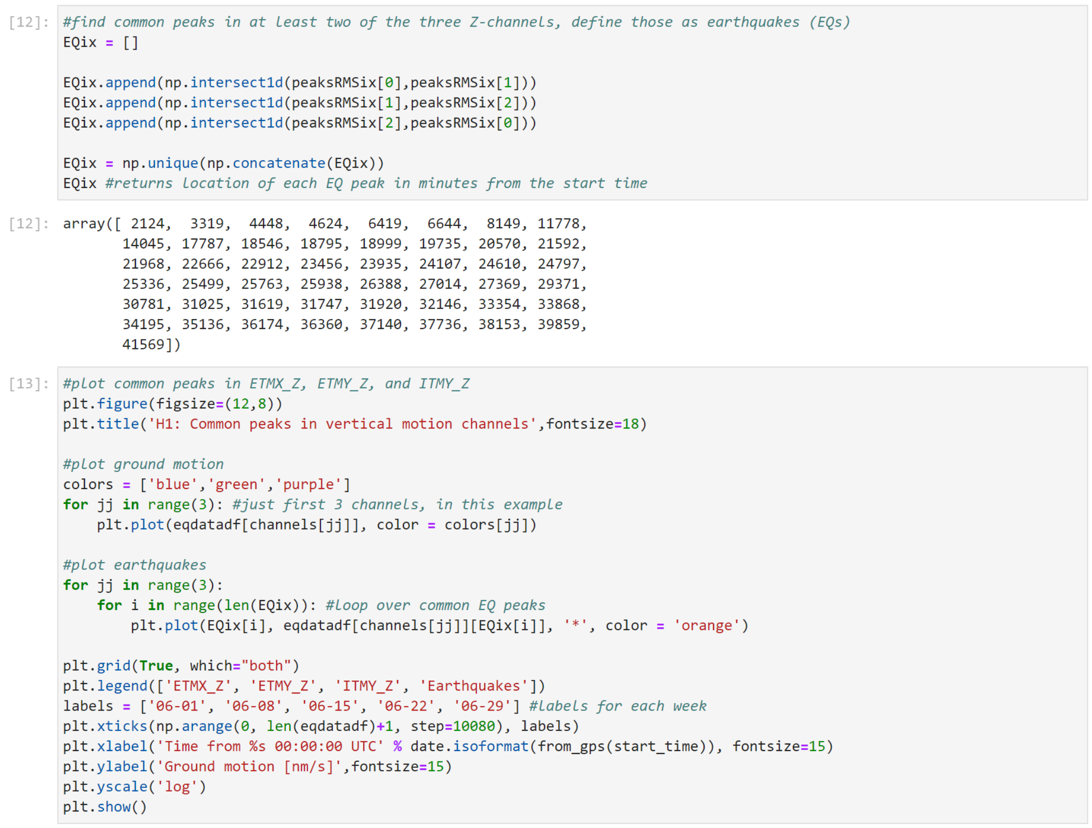
Figure 2: Download seismic motion data and lock data from LIGO detectors.



```
[9]: #set parameters to use with find_peaks
     peakwidth = 5
     peakprom = 100
     peakdist = 120

[10]: #get earthquake dataframe from file
      lockbitdf = pd.read_csv(lockfname)
      eqdatadf = pd.read_csv(bandfname)

[11]: peaksRMSix = [] #find location of local peaks in each Z-channel

      for i in channels[0:3]: #just first 3 channels, in this example
          peaksix,_ = find_peaks(eqdatadf[i], width = peakwidth, distance = peakdist, prominence = peakprom)
          peaksRMSix.append(peaksix)

      peaksRMSix

[11]: [array([ 2124,  3319,  4448,  4624,  6419,  6644,  8149, 11778,
              14045, 17787, 18546, 18795, 18999, 19735, 20570, 21592,
              21971, 22666, 22912, 23456, 23935, 24107, 24610, 24798,
              25336, 25499, 25763, 25938, 26388, 27014, 27369, 29371,
              30781, 31025, 31619, 31747, 31920, 32146, 33354, 33868,
              34195, 35136, 36174, 36360, 37140, 37736, 38153, 39859,
              41569]),
       array([ 2124,  3319,  4448,  4624,  6419,  6644,  8149, 11778,
              14045, 17787, 18546, 18795, 18999, 19735, 20570, 21592,
              21968, 22666, 22912, 23456, 23935, 24107, 24610, 24797,
              25336, 25499, 25763, 25938, 26388, 27014, 27369, 29371,
              30781, 31025, 31619, 31747, 31920, 32146, 33354, 33868,
              34195, 35143, 36174, 36360, 37140, 37736, 38153, 39859,
              41569]),
       array([ 2124,  3319,  4448,  4624,  6419,  6644,  8149, 14046,
              17787, 18795, 18999, 19735, 20570, 21592, 21968, 22666,
              22912, 23456, 23935, 24107, 24610, 24797, 25336, 25499,
              25763, 25938, 26388, 27014, 27369, 29371, 30781, 31025,
              31619, 31747, 31920, 32146, 33354, 33868, 34195, 35136,
              36174, 36360, 37140, 37736, 38153, 39859, 41569])]
```

Figure 3: Define peaks parameters.

After loading the dataframes of raw LIGO data, loop through the vertical channels and use the parameters to find the peaks in each set of data. Figure 3 shows the output of three lists, one for each vertical channel, that contain the location in minutes of the peak from the start time. Note that the vertical channels ETMX_Z, ETMY_Z, and ITMY_Z are purposefully listed as the first 3 in the "channels" list (cell [2], Figure 1) so I can easily loop through them.



```
[12]: #find common peaks in at least two of the three Z-channels, define those as earthquakes (EQs)
      EQix = []

      EQix.append(np.intersect1d(peaksRMSix[0],peaksRMSix[1]))
      EQix.append(np.intersect1d(peaksRMSix[1],peaksRMSix[2]))
      EQix.append(np.intersect1d(peaksRMSix[2],peaksRMSix[0]))

      EQix = np.unique(np.concatenate(EQix))
      EQix #returns location of each EQ peak in minutes from the start time

[12]: array([ 2124,  3319,  4448,  4624,  6419,  6644,  8149, 11778,
             14045, 17787, 18546, 18795, 18999, 19735, 20570, 21592,
             21968, 22666, 22912, 23456, 23935, 24107, 24610, 24797,
             25336, 25499, 25763, 25938, 26388, 27014, 27369, 29371,
             30781, 31025, 31619, 31747, 31920, 32146, 33354, 33868,
             34195, 35136, 36174, 36360, 37140, 37736, 38153, 39859,
             41569])

[13]: #plot common peaks in ETMX_Z, ETMY_Z, and ITMY_Z
      plt.figure(figsize=(12,8))
      plt.title('H1: Common peaks in vertical motion channels',fontsize=18)

      #plot ground motion
      colors = ['blue','green','purple']
      for jj in range(3): #just first 3 channels, in this example
          plt.plot(eqdatadf[channels[jj]], color = colors[jj])

      #plot earthquakes
      for jj in range(3):
          for i in range(len(EQix)): #loop over common EQ peaks
              plt.plot(EQix[i], eqdatadf[channels[jj]][EQix[i]], '*', color = 'orange')

      plt.grid(True, which="both")
      plt.legend(['ETMX_Z', 'ETMY_Z', 'ITMY_Z', 'Earthquakes'])
      labels = ['06-01', '06-08', '06-15', '06-22', '06-29'] #labels for each week
      plt.xticks(np.arange(0, len(eqdatadf)+1, step=10080), labels)
      plt.xlabel('Time from %s 00:00:00 UTC' % date.isoformat(from_gps(start_time)), fontsize=15)
      plt.ylabel('Ground motion [nm/s]',fontsize=15)
      plt.yscale('log')
      plt.show()
```

Figure 4: Define common peaks as earthquakes and plot with seismic data.

Earthquakes should affect the whole detector, therefore seeing a peak in just one station could signal something else. We can find earthquake events by only keeping peaks that are common to all three stations. Since the data TimeSeries are minute-trends, an earthquake that happens near one of the minute's boundaries could be recorded as a different time from the peak in other stations, even if it's only off by one minute. So, instead of finding peaks exactly common in all three channels, cell [12] in Figure 4 filters the list to just peaks that are found in 2 or more vertical channels. The result is a list (defined as EQix) of the location of each earthquake peak measured in minutes from the start time. Cell [13] in Figure 4 plots the identified earthquakes on top of the raw seismic data of each vertical channel, which is shown in Figure 5.

Section 3: Get statistics

Now, step through a chosen interval to find if and when the detector loses lock. The delta parameter defines this time interval around a peak in seismic motion data, and it's value represents the number of minutes both before and after the peak. The total length of the time interval is therefore 2 x delta centered around the peak. In this example, delta is 20 minutes. Also defined in Cell [14] of Figure 6 is maxlock, which refers to the low noise mode value that remains when the detector is locked; this value is different for each detector, 2000 at Livingston and 600 at Hanford. Lastly, in Cell [14], define the name and location of the CSV file that will contain all of the statistics found in Section 3 of EqLock.

Now, create two lists of earthquakes: one for survived events and one for events that occurred around the same time as lock loss. Start delta minutes from the peak, step through until delta minutes after the peak to find if the lock value drops, and categorize event as "lost lock" if it does. Next (in Cell [16]), loop through these lost lock events to find the first time that the detector lost lock.

Figures 7, 8, and 9 show three cells in which the earthquake statistics are calculated, stored in a dataframe, and saved in a CSV file, respectively. The statistics found include: the date, yes/no if

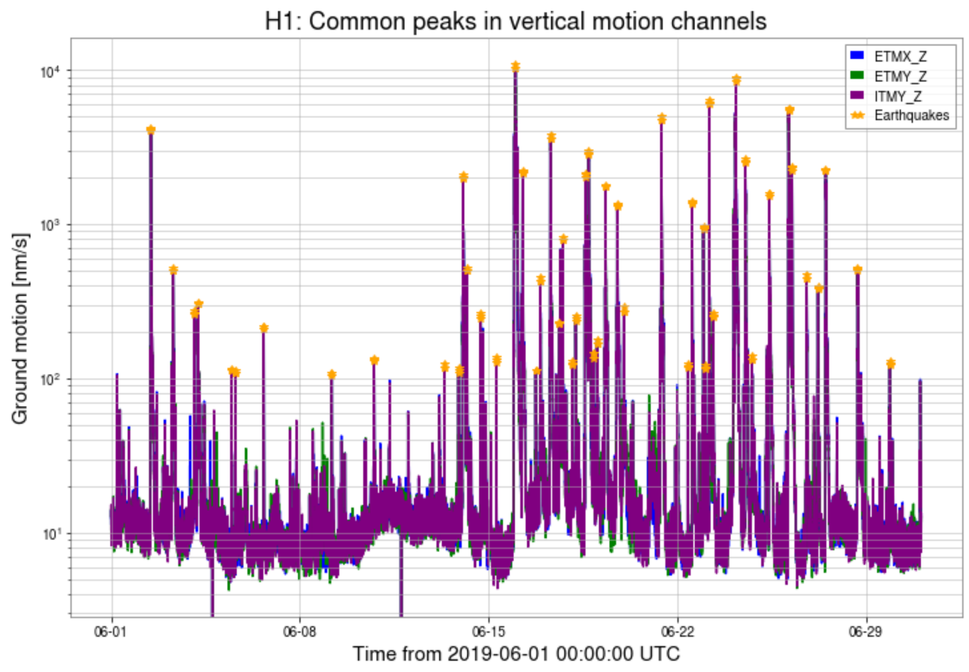


Figure 5: Earthquakes and vertical ground motion in June 2019 at Hanford, Washington.

```
[14]: #set parameters
      delta = 20  #minutes before and after EQ
      maxlock = 600 #2000 for LA, 600 for WA

      eqfname = "eqlockbit_data/eqtotaldf_jun2019_30M_100M_WA.csv" #will store dataframe of earthquake statistics
```

```
[15]: #find if detector survived EQ or lost lock status
      #create lists for each outcome
      survived = []
      lostlock = []

      #find if the lockbit value drops in the 20-min interval before and after EQ
      for jj in EQix:
          if np.min(lockbitdf['0'][jj-delta:jj+delta])>=maxlock: #if no drop, detector survived
              survived.append(jj)
          else: #if value drops below maxlock around time of EQ, we say detector lost lock
              lostlock.append(jj)

      lostlock
```

```
[15]: [2124, 4624, 14045, 18795, 18999, 21592, 22666, 22912, 23456, 24610, 25336, 25499, 25763, 26388, 27014, 27369, 29371,
      31025, 31920, 33354, 35136, 36174, 36360, 37736]
```

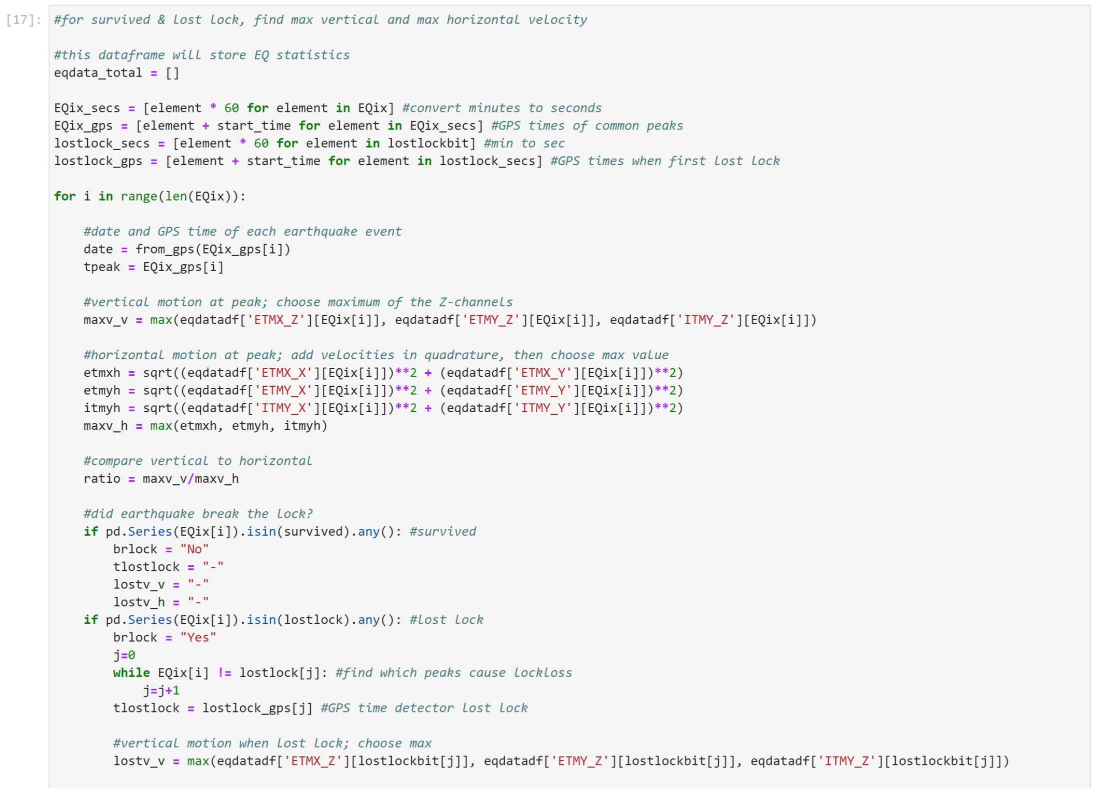
```
[16]: #find times of lockloss within +/- delta of each event
      lostlockbit = []

      for jj in lostlock:
          delta = 20
          while lockbitdf['0'][jj-delta]>=maxlock: #step through 20-min interval
              delta = delta - 1
          lostlockbit.append(jj-delta)  #first number we lost lock: associate w EQ

      lostlockbit
```

```
[16]: [2120, 4610, 14025, 18799, 18979, 21572, 22646, 22911, 23453, 24593, 25316, 25479, 25743, 26368, 26994, 27349, 29365,
      31008, 31915, 33334, 35116, 36154, 36340, 37716]
```

Figure 6: Find which earthquakes result in lock loss and what time the lock loss occurs.

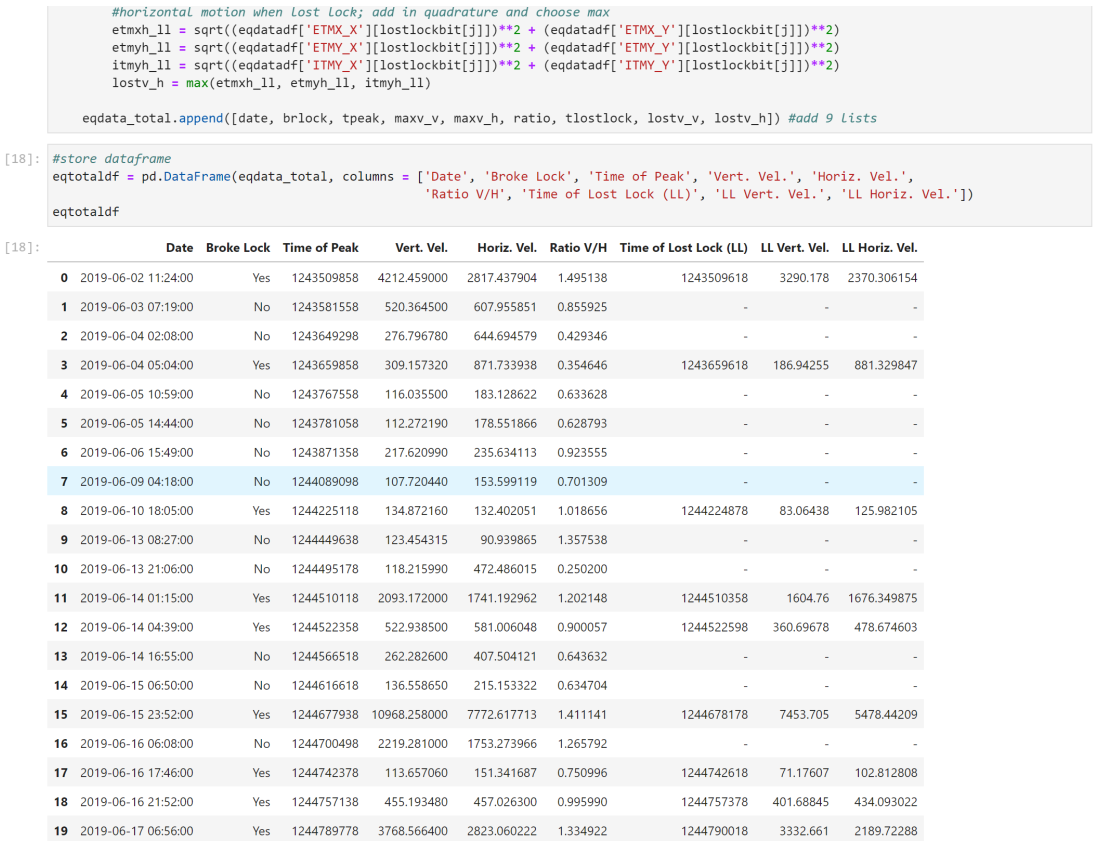
earthquake broke lock, time of peak, vertical ground velocity, horizontal ground velocity, ratio of vertical/horizontal velocities, and also for the earthquakes that did not survive, the time of lost lock, and the vertical and horizontal ground velocities at that time.

The beginning of Cell [17] converts the data (in minutes) to seconds and create lists of the earthquake and lock loss GPS times. Then, it loops through the peaks and calculates the desired statistics by calling data from the earthquake and lock dataframes. To calculate the horizontal velocities at the time of peak, the velocity in the x- and y- directions are added in quadrature to find the resultant horizontal velocity.



[17]:
```python
#for survived & lost lock, find max vertical and max horizontal velocity

#this dataframe will store EQ statistics
eqdata_total = []

EQix_secs = [element * 60 for element in EQix] #convert minutes to seconds
EQix_gps = [element + start_time for element in EQix_secs] #GPS times of common peaks
lostlock_secs = [element * 60 for element in lostlockbit] #min to sec
lostlock_gps = [element + start_time for element in lostlock_secs] #GPS times when first lost lock

for i in range(len(EQix)):

    #date and GPS time of each earthquake event
    date = from_gps(EQix_gps[i])
    tpeak = EQix_gps[i]

    #vertical motion at peak; choose maximum of the Z-channels
    maxv_v = max(eqdatadf['ETMX_Z'][EQix[i]], eqdatadf['ETMY_Z'][EQix[i]], eqdatadf['ITMY_Z'][EQix[i]])

    #horizontal motion at peak; add velocities in quadrature, then choose max value
    etmxh = sqrt((eqdatadf['ETMX_X'][EQix[i]])**2 + (eqdatadf['ETMX_Y'][EQix[i]])**2)
    etmyh = sqrt((eqdatadf['ETMY_X'][EQix[i]])**2 + (eqdatadf['ETMY_Y'][EQix[i]])**2)
    itmyh = sqrt((eqdatadf['ITMY_X'][EQix[i]])**2 + (eqdatadf['ITMY_Y'][EQix[i]])**2)
    maxv_h = max(etmxh, etmyh, itmyh)

    #compare vertical to horizontal
    ratio = maxv_v/maxv_h

    #did earthquake break the lock?
    if pd.Series(EQix[i]).isin(survived).any(): #survived
        brlock = "No"
        tlostlock = "-"
        lostv_v = "-"
        lostv_h = "-"
    if pd.Series(EQix[i]).isin(lostlock).any(): #lost lock
        brlock = "Yes"
        j=0
        while EQix[i] != lostlock[j]: #find which peaks cause lockloss
            j=j+1
        tlostlock = lostlock_gps[j] #GPS time detector lost lock

        #vertical motion when lost lock; choose max
        lostv_v = max(eqdatadf['ETMX_Z'][lostlockbit[j]], eqdatadf['ETMY_Z'][lostlockbit[j]], eqdatadf['ITMY_Z'][lostlockbit[j]])
```

Figure 7: Convert to GPS times and calculate desired statistics.



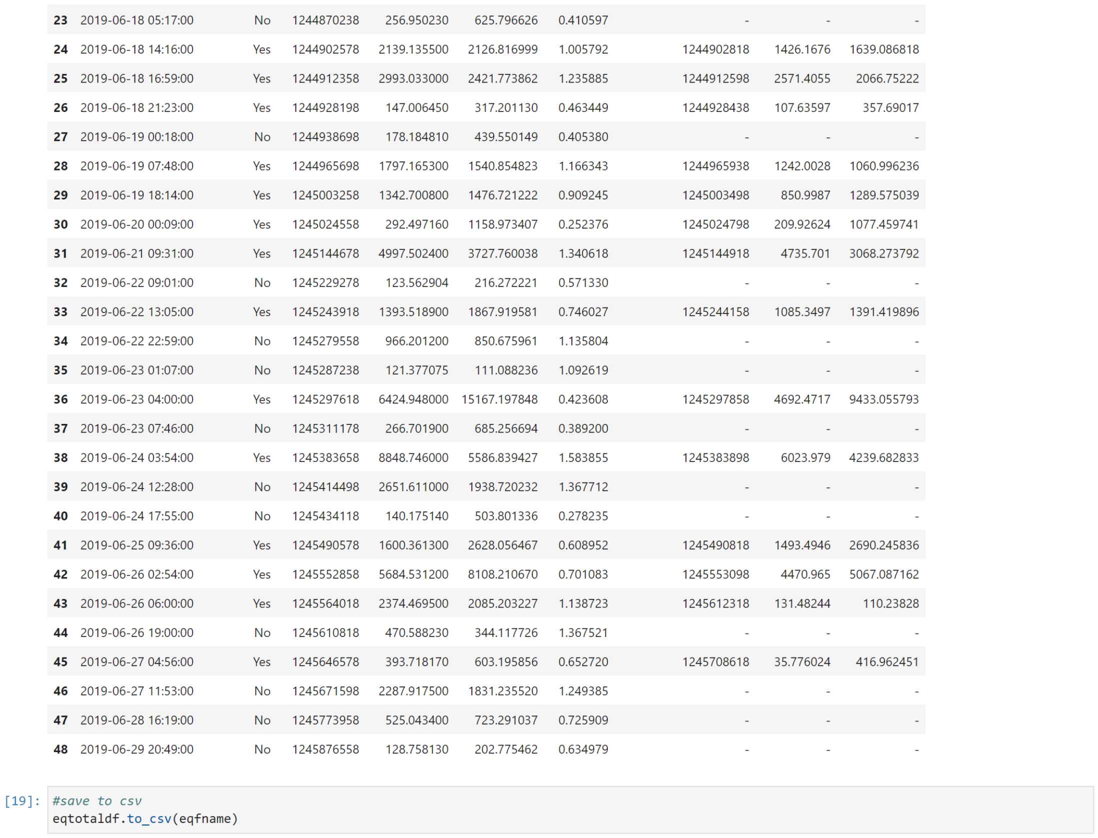
```python
        #horizontal motion when lost lock; add in quadrature and choose max
        etmxh_ll = sqrt((eqdatadf['ETMX_X'][lostlockbit[j]])**2 + (eqdatadf['ETMX_Y'][lostlockbit[j]])**2)
        etmyh_ll = sqrt((eqdatadf['ETMY_X'][lostlockbit[j]])**2 + (eqdatadf['ETMY_Y'][lostlockbit[j]])**2)
        itmyh_ll = sqrt((eqdatadf['ITMY_X'][lostlockbit[j]])**2 + (eqdatadf['ITMY_Y'][lostlockbit[j]])**2)
        lostv_h = max(etmxh_ll, etmyh_ll, itmyh_ll)

    eqdata_total.append([date, brlock, tpeak, maxv_v, maxv_h, ratio, tlostlock, lostv_v, lostv_h]) #add 9 lists
```

[18]:
```python
#store dataframe
eqtotaldf = pd.DataFrame(eqdata_total, columns = ['Date', 'Broke Lock', 'Time of Peak', 'Vert. Vel.', 'Horiz. Vel.',
                                                  'Ratio V/H', 'Time of Lost Lock (LL)', 'LL Vert. Vel.', 'LL Horiz. Vel.'])
eqtotaldf
```

[18]:

	Date	Broke Lock	Time of Peak	Vert. Vel.	Horiz. Vel.	Ratio V/H	Time of Lost Lock (LL)	LL Vert. Vel.	LL Horiz. Vel.
0	2019-06-02 11:24:00	Yes	1243509858	4212.459000	2817.437904	1.495138	1243509618	3290.178	2370.306154
1	2019-06-03 07:19:00	No	1243581558	520.364500	607.955851	0.855925	-	-	-
2	2019-06-04 02:08:00	No	1243649298	276.796780	644.694579	0.429346	-	-	-
3	2019-06-04 05:04:00	Yes	1243659858	309.157320	871.733938	0.354646	1243659618	186.94255	881.329847
4	2019-06-05 10:59:00	No	1243767558	116.035500	183.128622	0.633628	-	-	-
5	2019-06-05 14:44:00	No	1243781058	112.272190	178.551866	0.628793	-	-	-
6	2019-06-06 15:49:00	No	1243871358	217.620990	235.634113	0.923555	-	-	-
7	2019-06-09 04:18:00	No	1244089098	107.720440	153.599119	0.701309	-	-	-
8	2019-06-10 18:05:00	Yes	1244225118	134.872160	132.402051	1.018656	1244224878	83.06438	125.982105
9	2019-06-13 08:27:00	No	1244449638	123.454315	90.939865	1.357538	-	-	-
10	2019-06-13 21:06:00	No	1244495178	118.215990	472.486015	0.250200	-	-	-
11	2019-06-14 01:15:00	Yes	1244510118	2093.172000	1741.192962	1.202148	1244510358	1604.76	1676.349875
12	2019-06-14 04:39:00	Yes	1244522358	522.938500	581.006048	0.900057	1244522598	360.69678	478.674603
13	2019-06-14 16:55:00	No	1244566518	262.282600	407.504121	0.643632	-	-	-
14	2019-06-15 06:50:00	No	1244616618	136.558650	215.153322	0.634704	-	-	-
15	2019-06-15 23:52:00	Yes	1244677938	10968.258000	7772.617713	1.411141	1244678178	7453.705	5478.44209
16	2019-06-16 06:08:00	No	1244700498	2219.281000	1753.273966	1.265792	-	-	-
17	2019-06-16 17:46:00	Yes	1244742378	113.657060	151.341687	0.750996	1244742618	71.17607	102.812808
18	2019-06-16 21:52:00	Yes	1244757138	455.193480	457.026300	0.995990	1244757378	401.68845	434.093022
19	2019-06-17 06:56:00	Yes	1244789778	3768.566400	2823.060222	1.334922	1244790018	3332.661	2189.72288

Figure 8: Store earthquake statistics in a dataframe.



23	2019-06-18 05:17:00	No	1244870238	256.950230	625.796626	0.410597	-	-	-
24	2019-06-18 14:16:00	Yes	1244902578	2139.135500	2126.816999	1.005792	1244902818	1426.1676	1639.086818
25	2019-06-18 16:59:00	Yes	1244912358	2993.033000	2421.773862	1.235885	1244912598	2571.4055	2066.75222
26	2019-06-18 21:23:00	Yes	1244928198	147.006450	317.201130	0.463449	1244928438	107.63597	357.69017
27	2019-06-19 00:18:00	No	1244938698	178.184810	439.550149	0.405380	-	-	-
28	2019-06-19 07:48:00	Yes	1244965698	1797.165300	1540.854823	1.166343	1244965938	1242.0028	1060.996236
29	2019-06-19 18:14:00	Yes	1245003258	1342.700800	1476.721222	0.909245	1245003498	850.9987	1289.575039
30	2019-06-20 00:09:00	Yes	1245024558	292.497160	1158.973407	0.252376	1245024798	209.92624	1077.459741
31	2019-06-21 09:31:00	Yes	1245144678	4997.502400	3727.760038	1.340618	1245144918	4735.701	3068.273792
32	2019-06-22 09:01:00	No	1245229278	123.562904	216.272221	0.571330	-	-	-
33	2019-06-22 13:05:00	Yes	1245243918	1393.518900	1867.919581	0.746027	1245244158	1085.3497	1391.419896
34	2019-06-22 22:59:00	No	1245279558	966.201200	850.675961	1.135804	-	-	-
35	2019-06-23 01:07:00	No	1245287238	121.377075	111.088236	1.092619	-	-	-
36	2019-06-23 04:00:00	Yes	1245297618	6424.948000	15167.197848	0.423608	1245297858	4692.4717	9433.055793
37	2019-06-23 07:46:00	No	1245311178	266.701900	685.256694	0.389200	-	-	-
38	2019-06-24 03:54:00	Yes	1245383658	8848.746000	5586.839427	1.583855	1245383898	6023.979	4239.682833
39	2019-06-24 12:28:00	No	1245414498	2651.611000	1938.720232	1.367712	-	-	-
40	2019-06-24 17:55:00	No	1245434118	140.175140	503.801336	0.278235	-	-	-
41	2019-06-25 09:36:00	Yes	1245490578	1600.361300	2628.056467	0.608952	1245490818	1493.4946	2690.245836
42	2019-06-26 02:54:00	Yes	1245552858	5684.531200	8108.210670	0.701083	1245553098	4470.965	5067.087162
43	2019-06-26 06:00:00	Yes	1245564018	2374.469500	2085.203227	1.138723	1245612318	131.48244	110.23828
44	2019-06-26 19:00:00	No	1245610818	470.588230	344.117726	1.367521	-	-	-
45	2019-06-27 04:56:00	Yes	1245646578	393.718170	603.195856	0.652720	1245708618	35.776024	416.962451
46	2019-06-27 11:53:00	No	1245671598	2287.917500	1831.235520	1.249385	-	-	-
47	2019-06-28 16:19:00	No	1245773958	525.043400	723.291037	0.725909	-	-	-
48	2019-06-29 20:49:00	No	1245876558	128.758130	202.775462	0.634979	-	-	-

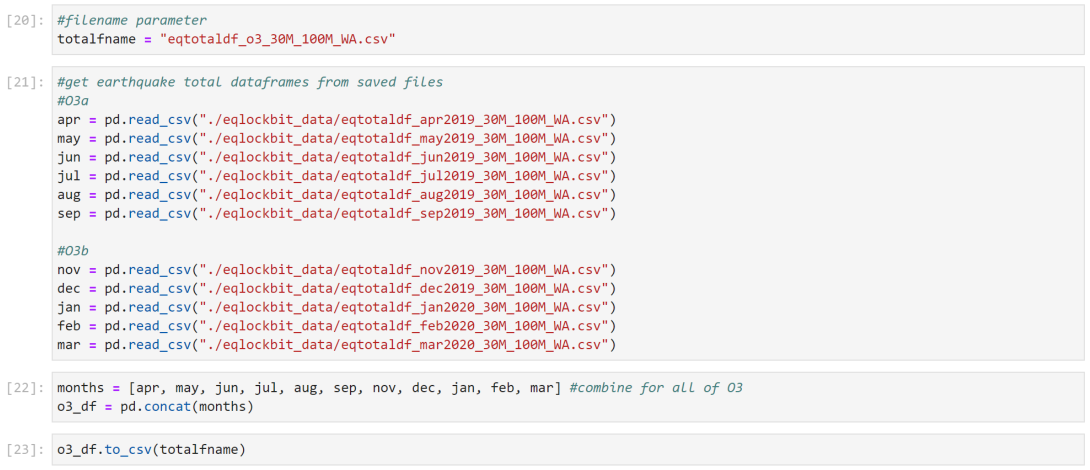
```
[19]: #save to csv
      eqtotaldf.to_csv(eqfname)
```

Figure 9: Save dataframe of earthquake statistics to a CSV file.

Then, for both the final vertical and horizontal velocities, the maximum value between the three stations at the time of each peak is saved in the dataframe. This process is also applied to the velocities at the times of lock loss and stored in the dataframe and saved to a CSV file (Cells [18] and [19]).

Section 4: Combine data files (optional)

This section is made to concatenate CSV files and only needs to be used if more than one CSV is saved. In the example code, I saved CSV files of earthquake statistics (from Section 3) for each month in O3. As shown in Figure 10, the concatenated file name is defined in Cell [20], the files for each month are loaded in Cell [21], the files are concatenated in Cell [22], and Cell [23] saves the CSV of the complete earthquake data statistics.

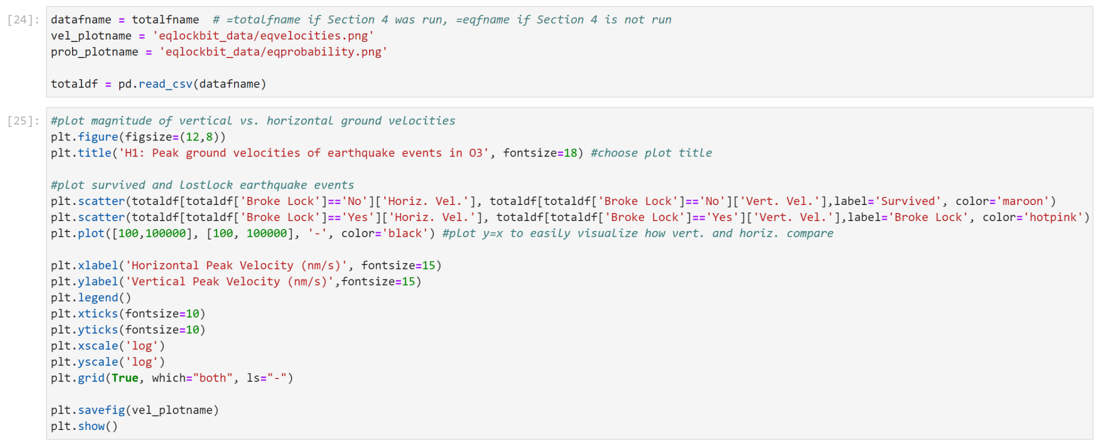


```
[20]: #filename parameter
      totalfname = "eqtotaldf_o3_30M_100M_WA.csv"

[21]: #get earthquake total dataframes from saved files
      #O3a
      apr = pd.read_csv("./eqlockbit_data/eqtotaldf_apr2019_30M_100M_WA.csv")
      may = pd.read_csv("./eqlockbit_data/eqtotaldf_may2019_30M_100M_WA.csv")
      jun = pd.read_csv("./eqlockbit_data/eqtotaldf_jun2019_30M_100M_WA.csv")
      jul = pd.read_csv("./eqlockbit_data/eqtotaldf_jul2019_30M_100M_WA.csv")
      aug = pd.read_csv("./eqlockbit_data/eqtotaldf_aug2019_30M_100M_WA.csv")
      sep = pd.read_csv("./eqlockbit_data/eqtotaldf_sep2019_30M_100M_WA.csv")

      #O3b
      nov = pd.read_csv("./eqlockbit_data/eqtotaldf_nov2019_30M_100M_WA.csv")
      dec = pd.read_csv("./eqlockbit_data/eqtotaldf_dec2019_30M_100M_WA.csv")
      jan = pd.read_csv("./eqlockbit_data/eqtotaldf_jan2020_30M_100M_WA.csv")
      feb = pd.read_csv("./eqlockbit_data/eqtotaldf_feb2020_30M_100M_WA.csv")
      mar = pd.read_csv("./eqlockbit_data/eqtotaldf_mar2020_30M_100M_WA.csv")

[22]: months = [apr, may, jun, jul, aug, sep, nov, dec, jan, feb, mar] #combine for all of O3
      o3_df = pd.concat(months)

[23]: o3_df.to_csv(totalfname)
```

Figure 10: Concatenate CSV files of stored statistics, if necessary.

Section 5: Produce probability plot

Plots of the ground velocities and the probability of survival will be reproduced in this final section. In Cell [24] of Figure 11, the file names are defined for the statistics dataframe and the two final plots. Note that the statistics dataframe is renamed here but originally named in Cell [14], and if Section 4 was run, it is named in Cell [20].



```
[24]: datafname = totalfname  # =totalfname if Section 4 was run, =eqfname if Section 4 is not run
      vel_plotname = 'eqlockbit_data/eqvelocities.png'
      prob_plotname = 'eqlockbit_data/eqprobability.png'

      totaldf = pd.read_csv(datafname)
```

```
[25]: #plot magnitude of vertical vs. horizontal ground velocities
      plt.figure(figsize=(12,8))
      plt.title('H1: Peak ground velocities of earthquake events in O3', fontsize=18) #choose plot title

      #plot survived and lostlock earthquake events
      plt.scatter(totaldf[totaldf['Broke Lock']=='No']['Horiz. Vel.'], totaldf[totaldf['Broke Lock']=='No']['Vert. Vel.'],label='Survived', color='maroon')
      plt.scatter(totaldf[totaldf['Broke Lock']=='Yes']['Horiz. Vel.'], totaldf[totaldf['Broke Lock']=='Yes']['Vert. Vel.'],label='Broke Lock', color='hotpink')
      plt.plot([100,100000], [100, 100000], '-', color='black') #plot y=x to easily visualize how vert. and horiz. compare

      plt.xlabel('Horizontal Peak Velocity (nm/s)', fontsize=15)
      plt.ylabel('Vertical Peak Velocity (nm/s)',fontsize=15)
      plt.legend()
      plt.xticks(fontsize=10)
      plt.yticks(fontsize=10)
      plt.xscale('log')
      plt.yscale('log')
      plt.grid(True, which="both", ls="-")

      plt.savefig(vel_plotname)
      plt.show()
```

Figure 11: Define the file names to store final results, then plot the vertical vs. horizontal velocities at each peak. Designate different colors for the survived and not-survived earthquakes.

Next, Cell [25] produces a scatter plot of the vertical vs. horizontal velocities of each earthquake, where different colors distinguish the survived earthquakes and those that result in lock loss. The line y = x is plotted for reference. The output plot of this cell is given as Figure 12.

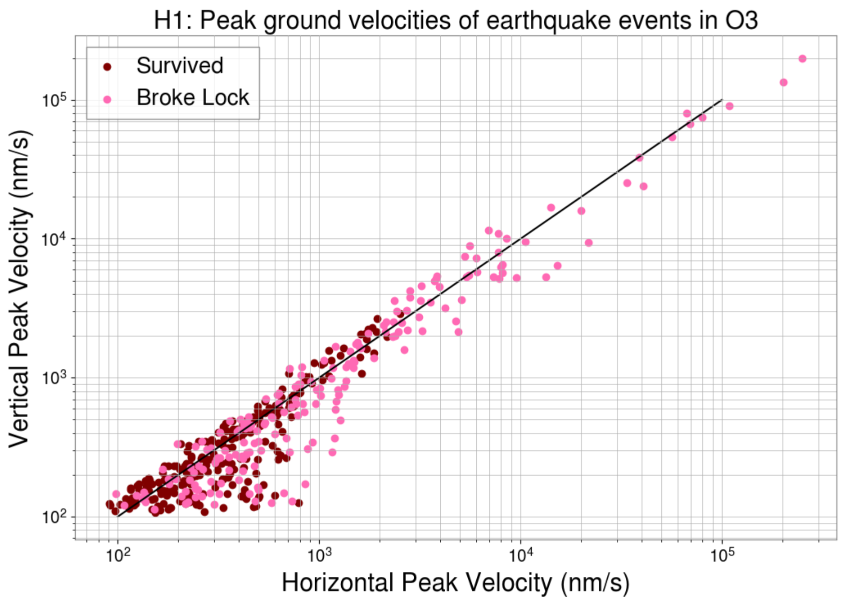
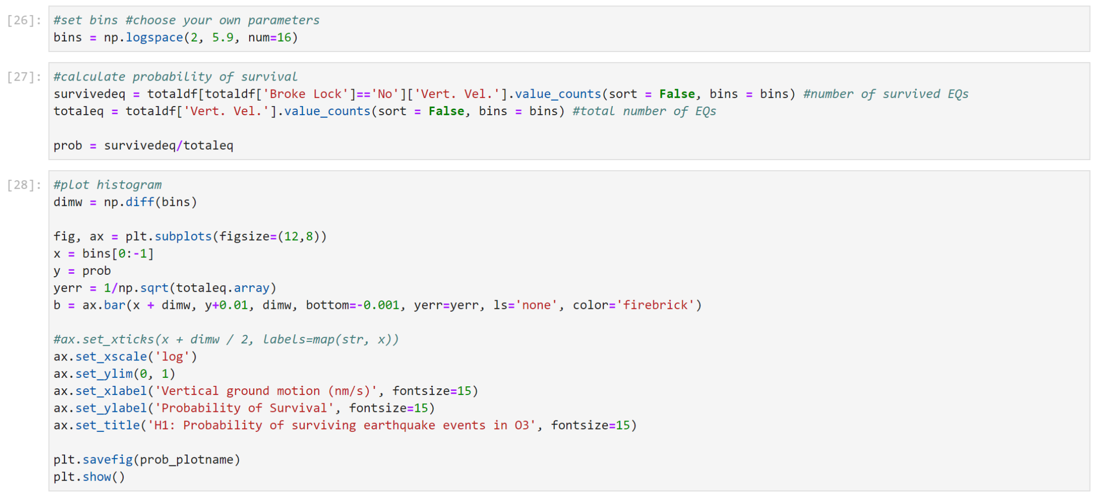


Figure 12: Magnitudes of the vertical ground motion (measured in nm/s) at the peak vs. the horizontal ground motion at the peak for earthquakes found during O3 at Hanford. Each earthquake event we either "survived" or the earthquake "broke lock" according to the parameters chosen for peaks in the data within a certain time interval.

Lastly, we can calculate the probability of survival—the probability that the detector will maintain lock during an earthquake event of a certain magnitude. Divide the number of survived earthquakes by the number of total earthquakes to get this probability. First, choose the beginning, end, and number of the bins that will group the earthquake velocities. This parameter is also completely up to the scientist and likely requires some experimentation; the bins I chose (see Figure 13, Cell [26]) resulted in plots that I thought represents the data well. Then, Cell [27] finds the number of survived earthquakes within each bin and divides it by the number of total earthquakes within that bin.



```
[26]: #set bins #choose your own parameters
      bins = np.logspace(2, 5.9, num=16)

[27]: #calculate probability of survival
      survivedeq = totaldf[totaldf['Broke Lock']=='No']['Vert. Vel.'].value_counts(sort = False, bins = bins) #number of survived EQs
      totaleq = totaldf['Vert. Vel.'].value_counts(sort = False, bins = bins) #total number of EQs

      prob = survivedeq/totaleq

[28]: #plot histogram
      dimw = np.diff(bins)

      fig, ax = plt.subplots(figsize=(12,8))
      x = bins[0:-1]
      y = prob
      yerr = 1/np.sqrt(totaleq.array)
      b = ax.bar(x + dimw, y+0.01, dimw, bottom=-0.001, yerr=yerr, ls='none', color='firebrick')

      #ax.set_xticks(x + dimw / 2, labels=map(str, x))
      ax.set_xscale('log')
      ax.set_ylim(0, 1)
      ax.set_xlabel('Vertical ground motion (nm/s)', fontsize=15)
      ax.set_ylabel('Probability of Survival', fontsize=15)
      ax.set_title('H1: Probability of surviving earthquake events in O3', fontsize=15)

      plt.savefig(prob_plotname)
      plt.show()
```

Figure 13: Set bins, find probability of survival in each bin, and plot survival in a histogram.

A histogram of the probability of survival vs. magnitudes of ground velocity is then plotted in Cell [28] using the defined bins. In the example code, the y-error is defined as 1/(square root of total earthquakes) for each bin. The probability of survival is the final result of EqLock, and the output plot for Hanford is shown in Figure 15 in the Results section of this documentation.

Results and Discussion

This code gathers statistics regarding seismic motion and the lock status of the detectors, then uses these calculations to find the probability of surviving an earthquake without lock loss. This happens by isolating peaks in the seismic data and comparing the times of these peaks with times of lock loss.

The most important parameters chosen in this earthquake analysis are the peaks parameters and delta. The peaks parameters I chose are width = 5 min, prominence = 100, and distance = 120 minutes, and these are used as arguments in the *find_peaks* command. The delta interval I chose is 20 minutes, so I evaluate only the 20 minutes before and 20 minutes after each peak to find lock loss.

With the chosen parameters discussed, EqLock resulted in a total of 362 earthquakes found in Livingston and 434 earthquakes in Hanford during LIGO's third observing run (April 1, 2019 to March 27, 2020). The Livingston detector survived 185 events and lost lock for 177 events. The Hanford detector survived 240 events and lost lock for 194 events. All of these earthquakes are represented in the scatter plots in Figure 14, which compares the magnitudes of ground velocities and distinguishes the survived and not-survived earthquakes.

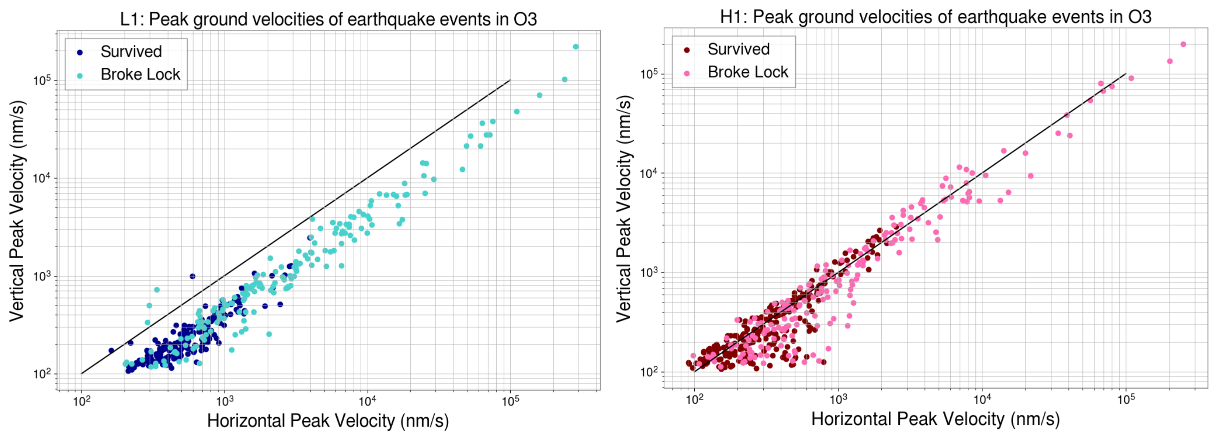


Figure 14: Plot of vertical and horizontal velocities for each earthquake event.

There were more total earthquakes found at the Hanford detector. This could be due to the fact that in general, the Hanford location experiences more earthquakes than Livingston. Many of these earthquakes are relatively small, therefore they are less likely to break lock and the probability of survival is higher.

An interesting trend shown in Figure 14 is that the horizontal peak velocity is generally larger than the vertical velocity at Livingston's detector, whereas at Hanford, these quantities are more or less the same. One hypothesis for why the horizontal motion is consistently larger than vertical motion at Livingston is that the type of seismic wave that reaches Livingston is different than Hanford, but the exact reason for this trend is unknown.

Past a certain magnitude of ground motion, LIGO is very unlikely to survive the earthquake without losing lock. To better understand this likelihood, the probability of survival at each detector is shown in Figure 15.

According to my chosen parameters, there are certain velocities that always result in lock loss at both locations. At Livingston, the maximum ground velocities survived are 2,475.5 nm/s in the vertical direction and 3,953.1 nm/s in the horizontal direction. At Hanford, the maximum ground velocities survived are 2,875.9 nm/s in the vertical direction and 2,509.2 nm/s in the horizontal direction.

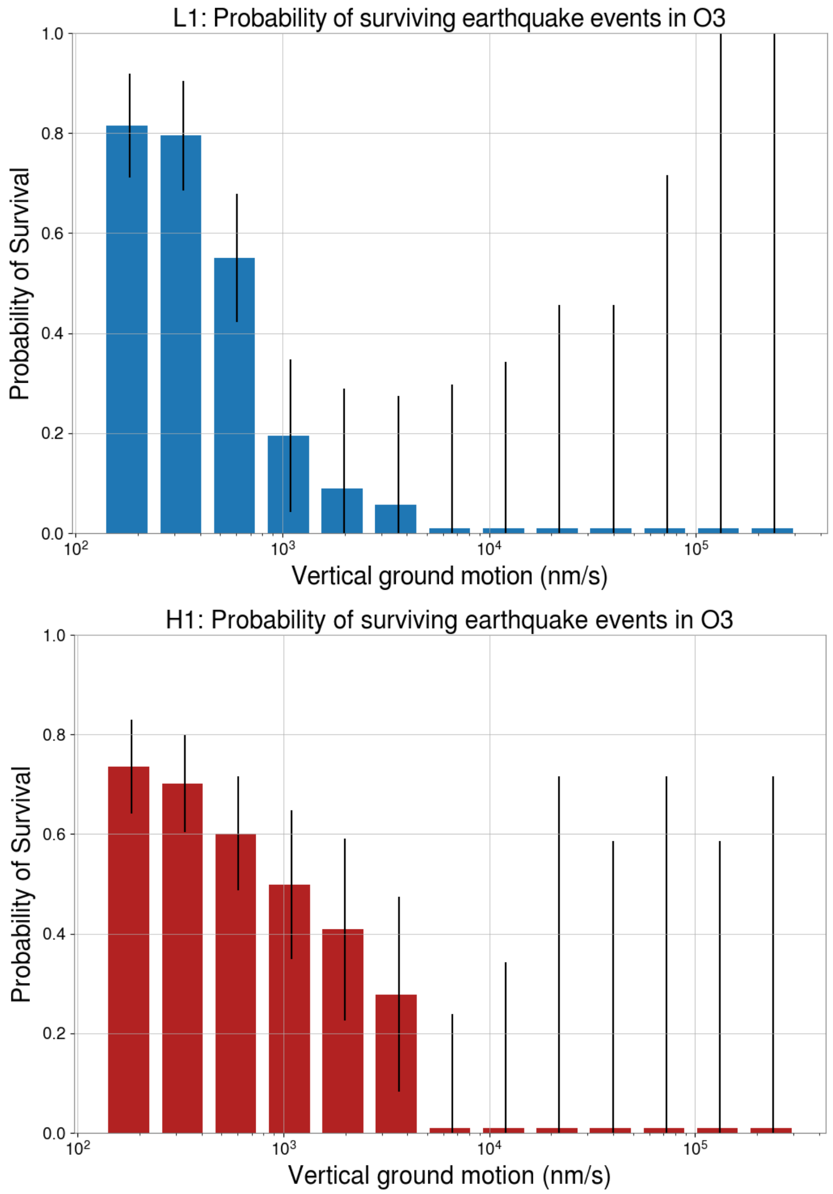


Figure 15: The probability of LIGO's detector surviving an earthquake event without losing lock for different magnitudes of seismic motion.

Note that there are less earthquakes per bin in the higher ground motion range, so the uncertainty of the probabilities is also higher (shown with the black bars in the plots of Figure 15). The errors here are calculated simply as $1/\sqrt{total}$ for the total earthquakes in each bin. We can decrease this error as we experience more earthquakes above a certain magnitude and get more data points in the higher bins.

Overall, in this paper, I have described a code that can find the probability of LIGO's detector surviving an earthquake of a magnitude within a certain range of ground velocities. The EqLock project was created for the purpose of analyzing earthquakes, but the code itself can be generalized for more uses. One of the project goals is to create a code that is accessible to LIGO members who want to perform similar analyses. Therefore, by changing the certain parameters outlined in this documentation, the code can be used for finding peaks and calculating probability for any frequency band. In the end, this project hopes to help LIGO better understand the effect of earthquakes on lock loss, to eventually improve seismic isolation controls, and to provide a tool for LIGO members to use in their research.

References

[1] E Schwartz et al. "Improving the robustness of the advanced LIGO detectors to earthquakes". In: *Classical and Quantum Gravity* 37.23 (Nov. 2020), p. 235007. ISSN: 1361-6382. DOI: 10.1088/1361-6382/abbc8c. URL: http://dx.doi.org/10.1088/1361-6382/abbc8c.