eps130_hw1_gutenberg_richter_v0.3_DISTRIBUTION

February 5, 2019

1 Earthquake Occurrence Statistics

The statistics of earthquake occurrence is revealed from catalogs of seismicity, which include event time, location and magnitude. We will talk about how earthquakes are located and how magnitudes are estimated separately, but for now it is sufficient to know that this information can be easily acquired. With such catalogs it is possible to compare the seismic activity of different regions, make informed assessments about the frequency of earthquake occurrence, and learn about the fault rupture process. Maps of the earthquakes in catalogs over time reveal the structure of faulting in a region, and provide a framework with which to study the seismotectonics of a region.

There are two primary earthquake statistics used by seismologists. They are the Gutenberg-Richter relationship (Gutenberg and Richter, 1949), and the Omori Law (Omori, 1894).

Gutenberg and Richter found that when the logarithm of the number of earthquakes is plotted vs. magnitude that the distribution (data) may be described by the line (model), log(N)=A+Bm, where N is the number of earthquakes, m is the magnitude and A (y-intercept) and B (slope) are refered to as the Gutenberg-Richter statistics or coefficients. They found that on a global scale, and subsequently more generally, the B-value or the slope of the Gutenberg-Richter line is approximately equal to -1. Thus for each increase in earthquake magnitude there are approximately 10 times fewer earthquakes. If, for example, there are 100 M3 events in a region per year, then the Gutenberg-Richter relationship generally finds that there would be approximately 10 M4 events and 1 M5 event in each year. For magnitudes larger than M5 there would be fewer than one event per year. Gutenberg-Richter is a very important earthquake statistic because it is used to determine the rates of earthquake occurrence, which is a key step in characterizing earthquake hazards (we will see this in future homework exercises).

The Omori Law is used to characterize the rate at which aftershocks occur following a large mainshock event. This statistic is used for comparing the aftershock productivity of different earthquakes and regions, make forecasts of the likelihood of large damaging aftershocks and to distinguish between earthquake faulting and possibly geothermal or volcanic-related seismicity by examining whether the distribution describes a "mainshock/aftershock" pattern or is more "swarm-like".

In this homework you will use python code in this notebook to investigate the Gutenberg-Richter and Omori statistics for the San Francisco Bay Area, as well as develop numerical analysis skills using python.

Note: This is not a python class, but the primary programming tool that will be used is python. However, if you know MatLab or have other programing background and would prefer to use it, you are free to use those tools instead. It will be helpful to read sections 9.6 and 9.8 of Lay and Wallace (1995) prior to working on this laboratory for background on the Gutenberg-Richter relation and the Omori Law.

```
In [100]: !pip install cartopy -q
          import math
          import datetime
          import numpy as np
          from scipy import stats
          import matplotlib
          import matplotlib.pyplot as plt
          import cartopy.crs as ccrs
          import cartopy.feature as cfeature
          import pandas as pd
```

```
In [101]: def haversine_np(lon1, lat1, lon2, lat2):
              """
              Calculate the great circle distance between two geographic points
              on the earth (specified in decimal degrees)

              All args must be of equal length.

              The first pair can be singular and the second an array

              """
              lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])

              dlon = lon2 - lon1
              dlat = lat2 - lat1

              a = np.sin(dlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2.0)**2

              c = 2 * np.arcsin(np.sqrt(a))
              km = 6371.0 * c
              return km
```

```
In [102]: def countDays(c,y,m,d):
              '''
              Function to count days in the array
              '''
              days=np.zeros(c)
              for i in range(0,c,1):
                  d0 = datetime.date(y[0], m[0], d[0])
                  d1 = datetime.date(y[i], m[i], d[i])
                  delta = d1 - d0
                  days[i]=delta.days
              return days
```

```
In [103]: def readAnssCatalog(p):
              '''
              Function to slice an ANSS catalog loaded as a pandas dataframe and return arrays
              '''
```

```
    d=np.array(p)         # load the dataframe into numpy as an array
    year=d[:,0].astype(int)  # define variables from the array
    month=d[:,1].astype(int)
    day=d[:,2].astype(int)
    hour=d[:,3].astype(int)
    minute=d[:,4].astype(int)
    sec=d[:,5].astype(int)
    lat=d[:,6]
    lon=d[:,7]
    mag=d[:,8]
    days = countDays(len(year),year,month,day)
    return year,month,day,hour,minute,sec,lat,lon,mag,days
```

1.0.1 The Catalog

We have downloaded the Advanced National Seismic System (ANSS) catalog from 1900 to 2018 for you to use (also available here: http://www.quake.geo.berkeley.edu/anss/catalog-search.html), and saved it as a text-file named "anss_catalog_1900to2018all.txt". This catalog has all events in the aforementioned time range located within 100 km of UC Berkeley. Columns of this catalog include information about the catalogued earthquakes, including the date and time of each event, its location in latitude, longitude and depth, and the event magnitude.

The following python code reads this catalog file and places the information in arrays for analysis.

```
In [104]: #Read data and create data arrays

          # This catalog is a M0+ search centered at Berkeley radius=100.
          # big enough to include Loma Prieta but exclude Geysers
          data= pd.read_csv('anss_catalog_1900to2018all.txt', sep=' ', delimiter=None, header=N
                          names = ['Year','Month','Day','Hour','Min','Sec','Lat','Lon','Mag'])

          year,month,day,hour,minute,sec,lat,lon,mag,days = readAnssCatalog(data)
```

```
In [112]: # Exercise 1: Explore the raw catalog (10 pts)
          readAnssCatalog(data)
          print (year)
          print (month)
          print (hour)
          print (mag)
```

```
[1911 1932 1932 ... 2017 2017 2017]
[ 7  6  6 ... 12 12 12]
[22  9  0 ... 10 10 16]
[6.6  4.5  3.   ... 0.7  1.53 1.03]
```

1.0.2 Print the number of events, the number of days from the first event, the minimum magnitude, and the maximum magnitude

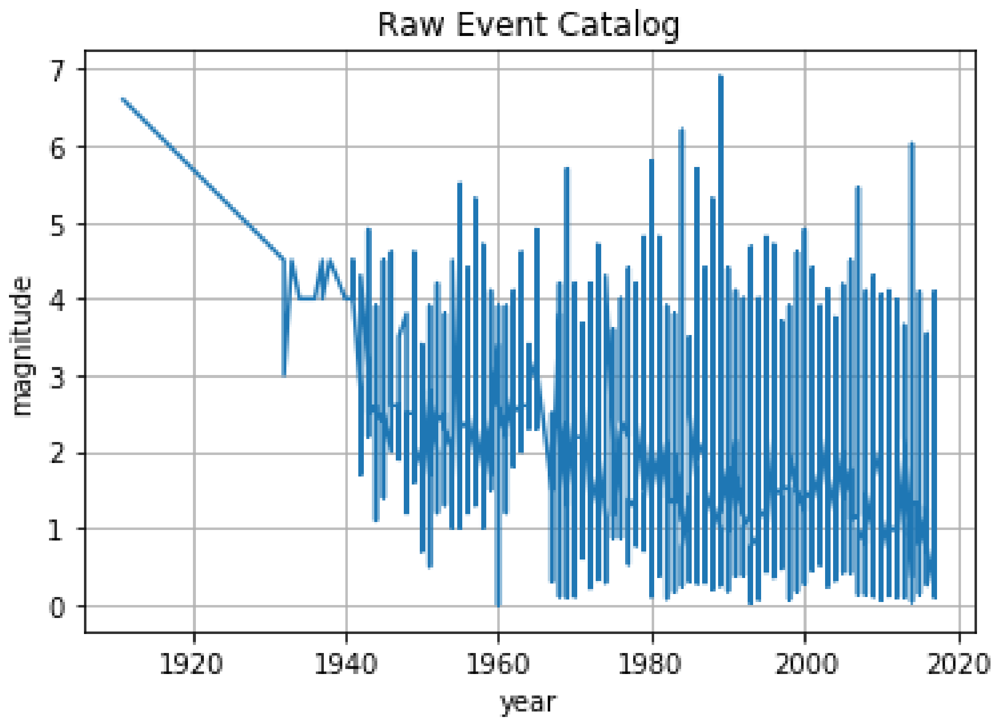
```
In [106]: nevt=len(year)
          print (nevt)
          firsteventday = (day[0])
          print (firsteventday)
          daysfromfirstevent = (day - day[0])
          minimum_magnitude=min(mag)
          print (minimum_magnitude)
          maximum_magnitude = max(mag)
          print (maximum_magnitude)
```

```
69804
1
0.0
6.9
```

1.0.3 Plot the catalog time series

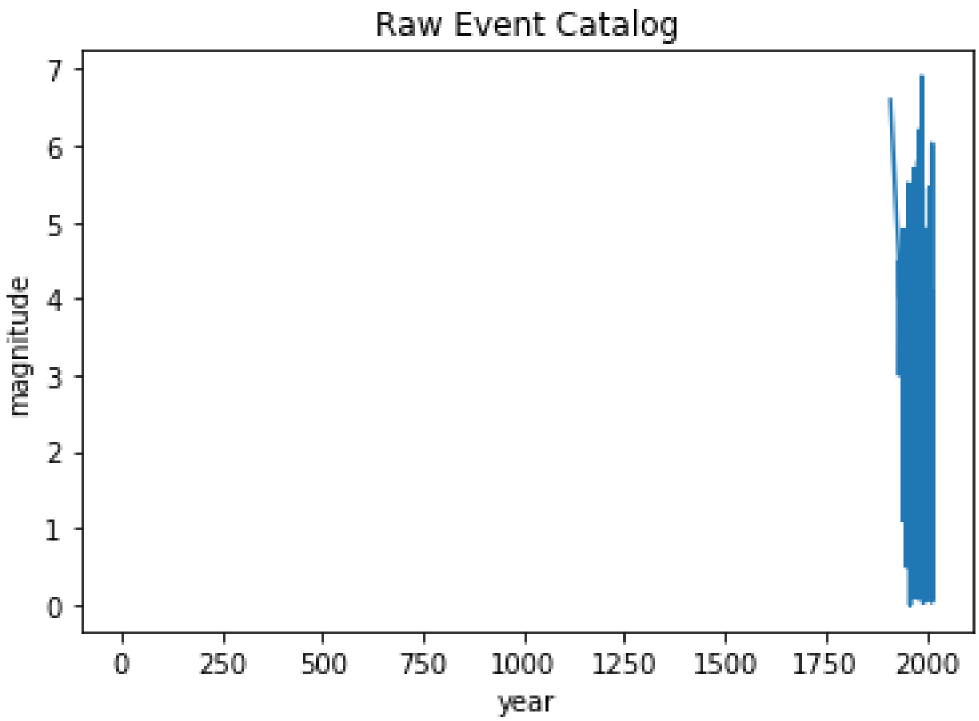
Make an x-y plot showing the magnitude of the earthquake on the y-axis and the time of the event on the x-axis. For this it is useful to have already determined the days since the beginning of the catalog. The plot will show that the catalog is not uniform due to the fact that over time as more seismic recording stations were installed more earthquakes could be detected and properly located.

```
In [107]: fig, ax = plt.subplots()
          ax.plot(year,mag) # ******
          ax.set(xlabel='year', ylabel='magnitude',
                 title='Raw Event Catalog')
          ax.grid()
          fig.savefig("hw1_ex1_fig1.png")


          plt.show()
```



In [108]:
```
### Plot the catalog in map view
#Familiarize yourself with the code example below as there will be additional exerci

#- Describe the seismicity and any patterns that you see.
#In this raw event catalog of a 100 year period, One ppattern I notice is that there
```

In [109]:
```
#- Try adding fault traces.
fig, ax = plt.subplots()
fault_traces = max(mag)
ax.plot(year,mag,fault_traces)
ax.set(xlabel='year', ylabel='magnitude',
       title='Raw Event Catalog')
plt.show()
```



In [58]:
```
#- How well does the seismicity show the region's major faults?

#I can't tell if the regions major fault is the concentrated region on the first grap
```

In [110]:
```
#Make a Map

#Set Corners of Map
lat0=36.75
lat1=39.0
lon0=-123.75
lon1=-121.0
tickstep=0.5 #for axes
latticks=np.arange(lat0,lat1+tickstep,tickstep)
lonticks=np.arange(lon0,lon1+tickstep,tickstep)
ydim=10      #height of plot
xdim=ydim*(haversine_np(lon0,lat0,lon1,lat0)/haversine_np(lon0,lat0,lon0,lat1)) #sca

###
plt.figure(figsize=(ydim,xdim))
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_extent([lon0, lon1, lat0, lat1], crs=ccrs.PlateCarree())
ax.set_aspect('auto')
ax.coastlines(resolution='10m',linewidth=1) #downloaded 10m, 50m
ax.set_xticks(lonticks)
```

```
ax.set_yticks(latticks, crs=ccrs.PlateCarree())
ax.set(xlabel='longitude', ylabel='Latitude',
       title='Raw Catalog')
ax.add_feature(cfeature.BORDERS, linestyle=':')
ax.add_feature(cfeature.LAKES,alpha=0.5)
ax.add_feature(cfeature.RIVERS)
ax.add_feature(cfeature.STATES.with_scale('10m'))

# Plot events as open circles with size and color proportional to event magnitude
indx=np.argsort(mag)   #determine sort index #Sort Descending to plot largest events
x=lon[indx]            #apply sort index
y=lat[indx]
z=np.exp(mag[indx])    #exponent to scale size
c = plt.cm.plasma(z/max(z))
plt.scatter(x,y, s=c, facecolors='none', edgecolors=c, marker='o', linewidth=2, alpha


# Add Berkeley, CA as a red square with size proportional to event magnitude
plt.plot(-122.2727,37.8716,'rs',markersize=10) # *****

#Save the plot by calling plt.savefig() BEFORE plt.show()
plt.savefig('hw1_ex2_seismap_raw.pdf')
plt.savefig('hw1_ex2_seismap_raw.png')

plt.show()
```

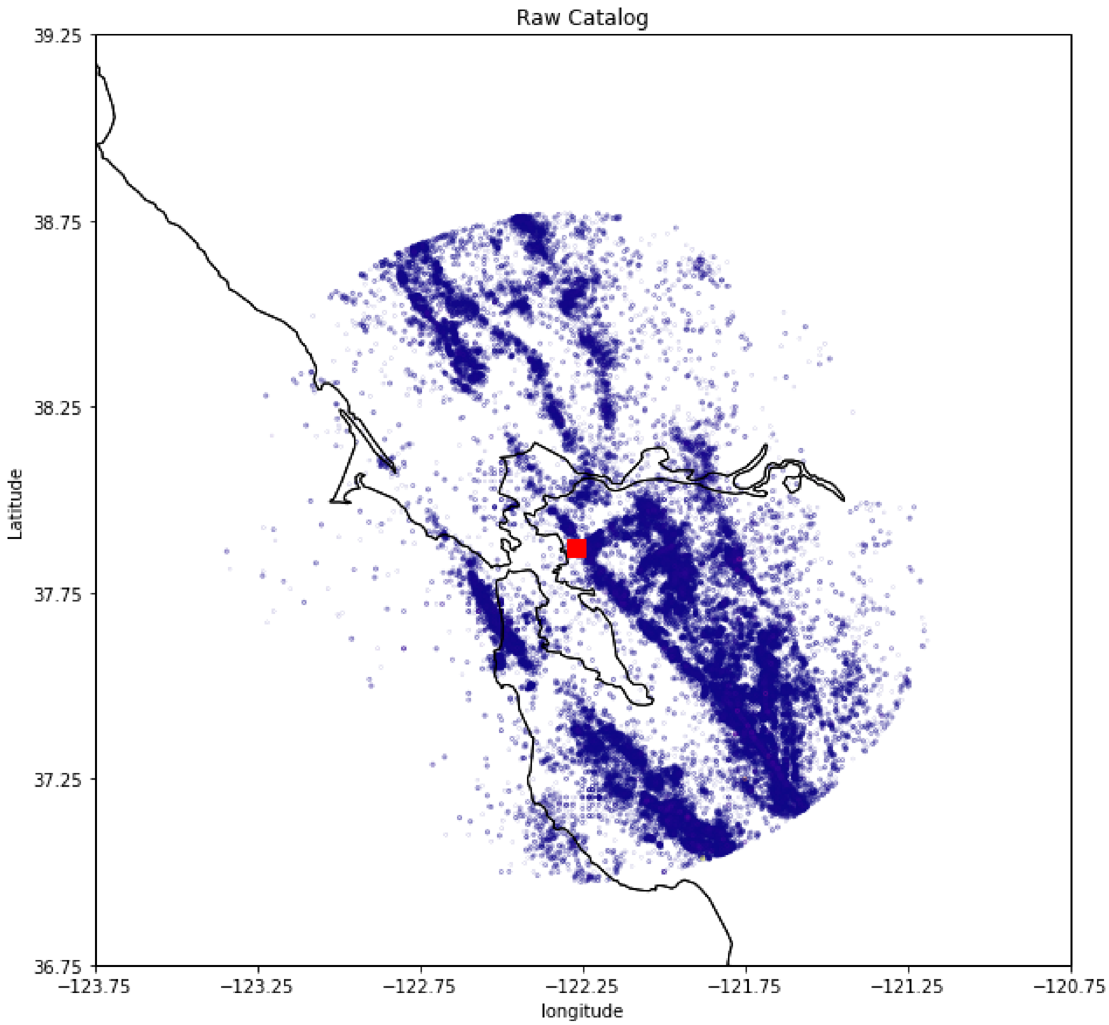
```
---------------------------------------------------------------------------

KeyboardInterrupt                         Traceback (most recent call last)

<ipython-input-110-56f86f307f5c> in <module>
     40
     41 #Save the plot by calling plt.savefig() BEFORE plt.show()
---> 42 plt.savefig('hw1_ex2_seismap_raw.pdf')
     43 plt.savefig('hw1_ex2_seismap_raw.png')
     44


/srv/app/venv/lib/python3.6/site-packages/matplotlib/pyplot.py in savefig(*args, **kwa
    687 def savefig(*args, **kwargs):
    688     fig = gcf()
--> 689     res = fig.savefig(*args, **kwargs)
    690     fig.canvas.draw_idle()   # need this if 'transparent=True' to reset colors
    691     return res


/srv/app/venv/lib/python3.6/site-packages/matplotlib/figure.py in savefig(self, fname,
```

```
   2092             self.set_frameon(frameon)
   2093
-> 2094         self.canvas.print_figure(fname, **kwargs)
   2095
   2096         if frameon:


    /srv/app/venv/lib/python3.6/site-packages/matplotlib/backend_bases.py in print_figure(s
   2073                     orientation=orientation,
   2074                     bbox_inches_restore=_bbox_inches_restore,
-> 2075                     **kwargs)
   2076             finally:
   2077                 if bbox_inches and restore_bbox:


    /srv/app/venv/lib/python3.6/site-packages/matplotlib/backends/backend_pdf.py in print_p
   2563                 RendererPdf(file, dpi, height, width),
   2564                 bbox_inches_restore=bbox_inches_restore)
-> 2565             self.figure.draw(renderer)
   2566             renderer.finalize()
   2567             if not isinstance(filename, PdfPages):


    /srv/app/venv/lib/python3.6/site-packages/matplotlib/artist.py in draw_wrapper(artist,
     48                 renderer.start_filter()
     49
---> 50             return draw(artist, renderer, *args, **kwargs)
     51         finally:
     52             if artist.get_agg_filter() is not None:


    /srv/app/venv/lib/python3.6/site-packages/matplotlib/figure.py in draw(self, renderer)
   1647
   1648             mimage._draw_list_compositing_images(
-> 1649                 renderer, self, artists, self.suppressComposite)
   1650
   1651             renderer.close_group('figure')


    /srv/app/venv/lib/python3.6/site-packages/matplotlib/image.py in _draw_list_compositing
    136     if not_composite or not has_images:
    137         for a in artists:
--> 138             a.draw(renderer)
    139     else:
    140         # Composite any adjacent images together


    /srv/app/venv/lib/python3.6/site-packages/matplotlib/artist.py in draw_wrapper(artist,
```

```
     48                 renderer.start_filter()
     49
---> 50             return draw(artist, renderer, *args, **kwargs)
     51         finally:
     52             if artist.get_agg_filter() is not None:


/srv/app/venv/lib/python3.6/site-packages/cartopy/mpl/geoaxes.py in draw(self, rendere
    386
    387         return matplotlib.axes.Axes.draw(self, renderer=renderer,
--> 388                                          inframe=inframe)
    389
    390     def __str__(self):


/srv/app/venv/lib/python3.6/site-packages/matplotlib/artist.py in draw_wrapper(artist,
     48                 renderer.start_filter()
     49
---> 50             return draw(artist, renderer, *args, **kwargs)
     51         finally:
     52             if artist.get_agg_filter() is not None:


/srv/app/venv/lib/python3.6/site-packages/matplotlib/axes/_base.py in draw(self, rende
   2626             renderer.stop_rasterizing()
   2627
-> 2628         mimage._draw_list_compositing_images(renderer, self, artists)
   2629
   2630         renderer.close_group('axes')


/srv/app/venv/lib/python3.6/site-packages/matplotlib/image.py in _draw_list_compositin
    136     if not_composite or not has_images:
    137         for a in artists:
--> 138             a.draw(renderer)
    139     else:
    140         # Composite any adjacent images together


/srv/app/venv/lib/python3.6/site-packages/matplotlib/artist.py in draw_wrapper(artist,
     48                 renderer.start_filter()
     49
---> 50             return draw(artist, renderer, *args, **kwargs)
     51         finally:
     52             if artist.get_agg_filter() is not None:


/srv/app/venv/lib/python3.6/site-packages/matplotlib/collections.py in draw(self, rende
```

```
    881     def draw(self, renderer):
    882         self.set_sizes(self._sizes, self.figure.dpi)
--> 883         Collection.draw(self, renderer)
    884
    885


    /srv/app/venv/lib/python3.6/site-packages/matplotlib/artist.py in draw_wrapper(artist,
     48                 renderer.start_filter()
     49
---> 50             return draw(artist, renderer, *args, **kwargs)
     51         finally:
     52             if artist.get_agg_filter() is not None:


    /srv/app/venv/lib/python3.6/site-packages/matplotlib/collections.py in draw(self, rende
    330                 self._linewidths, self._linestyles,
    331                 self._antialiaseds, self._urls,
--> 332                 self._offset_position)
    333
    334         gc.restore()


    /srv/app/venv/lib/python3.6/site-packages/matplotlib/backends/backend_pdf.py in draw_pa
   1737                 offsets, offsetTrans, facecolors, edgecolors,
   1738                 linewidths, linestyles, antialiaseds, urls,
-> 1739                 offset_position)
   1740
   1741         padding = np.max(linewidths)


    /srv/app/venv/lib/python3.6/site-packages/matplotlib/backend_bases.py in draw_path_coll
    239             transform = transforms.Affine2D(
    240                             transform.get_matrix()).translate(xo, yo)
--> 241             self.draw_path(gc0, path, transform, rgbFace)
    242
    243     def draw_quad_mesh(self, gc, master_transform, meshWidth, meshHeight,


    /srv/app/venv/lib/python3.6/site-packages/matplotlib/backends/backend_pdf.py in draw_pa
   1689             path, transform,
   1690             rgbFace is None and gc.get_hatch_path() is None,
-> 1691             gc.get_sketch_params())
   1692         self.file.output(self.gc.paint())
   1693


    /srv/app/venv/lib/python3.6/site-packages/matplotlib/backends/backend_pdf.py in writePa
```

```
   1495         cmds = self.pathOperations(path, transform, clip, simplify=simplify,
   1496                                    sketch=sketch)
-> 1497         self.output(*cmds)
   1498
   1499     def reserveObject(self, name=''):


/srv/app/venv/lib/python3.6/site-packages/matplotlib/backends/backend_pdf.py in output
    616
    617     def output(self, *data):
--> 618         self.write(fill([pdfRepr(x) for x in data]))
    619         self.write(b'\n')
    620


/srv/app/venv/lib/python3.6/site-packages/matplotlib/backends/backend_pdf.py in fill(s
    103     result = []
    104     for i, s in enumerate(strings):
--> 105         length = len(s)
    106         if currpos + length < linelen:
    107             currpos += length + 1


KeyboardInterrupt:
```



2 Exercise 2: Compute the Gutenberg-Richter statitistics (30 pts)

Follow the steps below to compute the Gutenberg Richter statistics for the raw catalog.

2.0.1 Determine and plot the Gutenberg-Richter Distribution

First, define a range of magnitudes to bin the data. You can use a range of magnitude, m from 0.0 to 6.9 in increments of 0.1 magnitude unit.

```
In [111]: m=np.arange(0, 6.9, 0.1) # ***** define a range of mag bins with width 0.1 magnitude
          print(m)
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7
 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.  3.1 3.2 3.3 3.4 3.5
 3.6 3.7 3.8 3.9 4.  4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.  5.1 5.2 5.3
```

```
 5.4 5.5 5.6 5.7 5.8 5.9 6.  6.1 6.2 6.3 6.4 6.5 6.6 6.7 6.8]
```

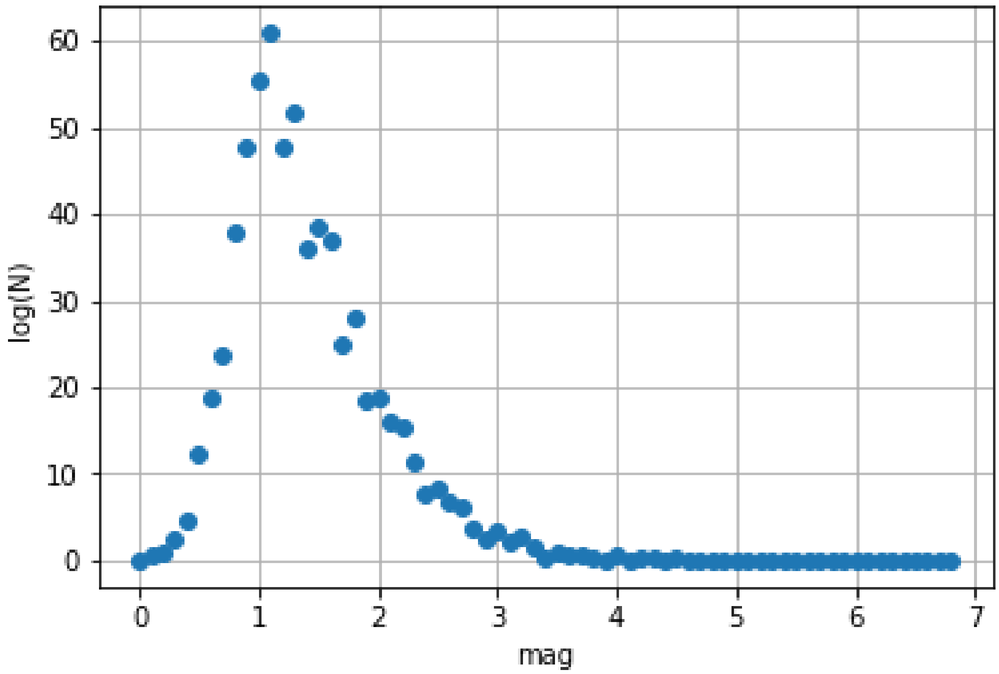
Next, count the number of events above a given magnitude. That is count the number of events above and equal to magnitude 0.0, then above and equal to 0.1, and so forth all the way to the maximum magnitude. You can do this by placing the code for vectorized counting of array elements passing a logical test (numpy.count_nonzero()) inside a for loop over the incremental magnitudes, m. We are interested in the annual rate of the events so you will need to divide by the total number of years the catalog spans. N is the log of of the number of events per year, so take the log base 10 (numpy.log10) of the annual number of earthquakes for each magnitude bin. Note you can place all of the operations in one line of code inside the for loop.

```
In [123]: # Preallocate the vector N with a zeros vector of size len(m)

          N = np.zeros(m.shape) # *****
          # Find N
          for i in range(m.size):
              N[i] = np.count_nonzero(np.vectorize(lambda e: 1 if (e >= m[i] and (True if i ==
          print(N)

          # note that the final variable N should actually be log(N)
```

```
[1.13207547e-01 4.71698113e-01 1.00000000e+00 2.55660377e+00
 4.71698113e+00 1.23490566e+01 1.86698113e+01 2.38396226e+01
 3.78113208e+01 4.78113208e+01 5.55094340e+01 6.08773585e+01
 4.76981132e+01 5.17169811e+01 3.60000000e+01 3.85754717e+01
 3.68867925e+01 2.50000000e+01 2.78490566e+01 1.83867925e+01
 1.86415094e+01 1.59433962e+01 1.53867925e+01 1.14056604e+01
 7.62264151e+00 8.32075472e+00 6.83018868e+00 6.18867925e+00
 3.73584906e+00 2.37735849e+00 3.29245283e+00 2.19811321e+00
 2.77358491e+00 1.37735849e+00 3.11320755e-01 9.43396226e-01
 6.79245283e-01 6.69811321e-01 3.01886792e-01 3.77358491e-02
 6.22641509e-01 4.71698113e-02 2.16981132e-01 1.50943396e-01
 1.13207547e-01 1.79245283e-01 9.43396226e-03 1.13207547e-01
 0.00000000e+00 2.83018868e-02 1.88679245e-02 0.00000000e+00
 2.83018868e-02 0.00000000e+00 3.77358491e-02 1.88679245e-02
 0.00000000e+00 2.83018868e-02 0.00000000e+00 0.00000000e+00
 9.43396226e-03 0.00000000e+00 9.43396226e-03 0.00000000e+00
 0.00000000e+00 9.43396226e-03 0.00000000e+00 0.00000000e+00
 9.43396226e-03]
```

Make a plot of the distribution.

```
In [124]: plt.figure()
          plt.plot(m, N,'o') # ******
          plt.xlabel('mag')
          plt.ylabel('log(N)')
          plt.grid()
```



2.0.2 Fit the data to find the Gutenberg Richter statistics.

Now, fit the data with the Gutenberg Richter relationship $log_{10}(N(m))$=A+Bm. In other words, "invert" the data to find the applied model parameters. We will walk through the steps of this inversion.

1) First, create the model parameter matrix, G, which has one column of 1's and a second column of magnitude bins, m.

$$G = \begin{pmatrix} 1 & m_0 \\ 1 & m_1 \\ 1 & m_2 \\ . & . \\ . & . \\ . & . \\ 1 & m_n \end{pmatrix}$$

2) Next, create the data matrix, d, which in this case is a single column and contains the $log_{10}(N(m))$ values. Note that we already defined this vector above.

$$d = \begin{pmatrix} log_{10}(N(m_0)) \\ log_{10}(N(m_1)) \\ log_{10}(N(m_2)) \\ . \\ . \\ . \\ log_{10}(N(m_n)) \end{pmatrix}$$

```
In [ ]: tmp=np.ones(len(N))        # column of 1's
        G=np.column_stack((tmp,m)) # matrix A
```

3) Next, compute the $G^T G$ matrix (G-transpose times G) using numpy functions

```
In [ ]: GTG=np.dot(np.transpose(G),G)
```

4) Next, compute the $G^T D$ (A-transpose times D)

```
In [ ]: D = N # Recall that we have already defined the D-matrix above as "N"
        GTD=np.dot( , )    # *****
```

5) Finally, solve the inverse problem. Invert the equation $(G^T G)x = G^T D$ using the numpy linear algebra solver (numpy.linalg.solv()). The result, x, will be a vector of the Gutenberg-Richter coefficients, in which the A-value is x[0] and the B-value is x[1]. The values you should get are A=3.418 and B=-0.809.

```
In [ ]: # solve the linear inverse problem
        soln=np.linalg.solve( , ) # ******
        print(f'A-value={soln[0]:.3f}, B-value={soln[1]:.3f}')
```

The linear Gutenberg-Richter model is fully defined by the A and B coefficients, however in order to plot a line through the distribution we need to take the dot product of G with our two-parameter solution vector.

```
In [ ]: x=m # the independent variable of the best-fit line is the same as for the data (magni
        y=np.dot(G,soln) # synthetic data
```

The resulting vector, "y", is called the synthetic data because it is a synthetic estimate of the real data. The difference between synthetic data and real data can be quantified through uncertainty analysis

2.0.3 Uncertainty analysis of Gutenberg-Richter model

Next, compute the uncertainties of the model (best-fit line defined by the A and B coefficients).

The following steps outline how to compute 95% confidence intervals for the model using the numpy and scipy packages in Python.

1) df=(length_of_data) - (number_of_model_parameters) #degree of freedom

2) e=data-(model predictions) #prediction error

3) variance=np.sum(e*e)/df

4) se_y=np.sqrt(var) #standard error of the estimate

5) sdev=np.sqrt(var) #standard deviation

6) t=stats.t.ppf(1-0.05/2,degfree) #two-sided students t-distribution

7) lower95=np.exp(np.log(modeled_pga)-t*se_y)

8) upper95=np.exp(np.log(modeled_pga)+t*se_y)

9) se_b=sdev/np.sqrt(np.sum((x-np.mean(x))**2)) # standard error of slope

10) se_a=sdev*np.sqrt(1/len(x) + np.mean(x)**2/np.sum((x-np.mean(x))**2)) # standard error of intercept (9 and 10 will be important for incorporating Gutenberg Richter uncertainty in PSHA (a future homework)

```
In [ ]: #Compute the uncertainty in Gutenberg-Richter Parameters

        length_of_data = len(N)
        number_of_model_parameters = 2
        df=(length_of_data) - (number_of_model_parameters) #degree of freedom

        e=N-y #prediction error

        var=np.sum(e*e)/df

        se_y=np.sqrt(var)              #standard error of the estimate
        sdev=np.sqrt(var)              #standard deviation

        #Calculate 95% confidence bounds
        t=stats.t.ppf(1-0.05/2,df)     #two-sided students t-distribution
        tmp=np.sqrt(1/len(x)+((x-np.mean(x))**2)/np.sum((x-np.mean(x))**2))
        tmp=tmp/max(tmp)
        lower95=y-t*se_y*tmp
        upper95=y+t*se_y*tmp
        se_b=sdev/np.sqrt(np.sum((x-np.mean(x))**2))                      #standard error slop
        se_a=sdev*np.sqrt(1/len(x) + np.mean(x)**2/np.sum((x-np.mean(x))**2)) #standard error
        a95=se_a*t
        b95=se_b*t
```

2.0.4 Plot the fit to the data

To visualize the fit to the distribution, we would like to make a plot showing the magnitude distribution as plotted above, log10(N) vs m, with the best fit line that we just found the coefficients for. Visalize the uncertainty by plotting the upper and lower 95 percentiles of the data.

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

        #plot the distribution like above
```

```
ax.plot(m,N,'bo')

#plot the best-fit line with uncertainties
ax.plot(x,y,'r-',linewidth=2)

# plot the 95% confidence intervals
ax.plot( , ,'k-',linewidth=2) # *****
ax.plot( , ,'k-',linewidth=2) # *****

ax.set(xlabel='magnitude', ylabel='Number of Earthquakes (log10)',
       title='Initial Gutenberg-Richter Distribution')
ax.grid()
plt.savefig("hw1_ex3_figure2_withErrorBounds.png")
plt.show()
```

2.0.5 Questions

1. What do the Gutenberg Ricther statistics represent for the earthquake distribution?
2. What happens to the shape of the distribution (log(N)) if you reduce the magnitude bin size by a factor of 10?
3. Why does the model parameter matrix (G) have a column of 1's?
4. What determines the size of $G^T G$ and $G^T d$?
5. How well does the Gutenberg-Richter model fit the data? Quantify your answer in terms of uncertainty.
6. Where does the fit begin to breakdown and why?
7. Based on your Gutenberg-Richter coefficients what is the annual rates of a M4 earthquakes? For a M7 earthquake?
8. On average how many years are there between M7 earthquakes based on this catalog.
9. How many M7 earthquakes are in the catalog?
10. What is your assessment of the quality or suitability of the forecast of average M7 occurrence?

3 Exercise 3: Declustering (30 pts)

In the above analysis mainshocks (primary events) and aftershocks are mixed together. The results for the Gutenberg-Richter statistics were generally pretty good, however a correct implementation of Gutenberg-Richter considers only the primary events. Therefore, we seek a catalog with aftershocks removed in order to improve our assessment of the Gutenberg-Richter statistics. The process to remove aftershocks is called declustering.

In this exercise, you will evaluate a published declustering method as you use it to decluster the catalog analyzed above. Then you will re-compute the Gutenberg-Richter coefficients for the declustered catalog in order to examine the affect on the G-R statistics.

3.0.1 Declustering Algorithm

The analysis that was just performed was for the raw catalog, which means that it includes all events. However Gutenberg-Richter is really interested in the occurrence of primary "main shock" events, and therefore it is necessary to decluster the catalog to obtain an unbiased estimate of the G-N coefficients. Declustering here means remove the aftershocks from the catalog. This is done using an algorithm that relates the "expected" time and distance range of aftershocks from a given mainshock. Large mainshocks will result in aftershock populations that, statistically speaking, have a greater likelihood to occur over longer time periods and greater mainshock-aftershock distances compared with smaller mainshock-aftershock series.

The code block below defines a declustering algorithm. This algorithm uses distance and time metrics that are magnitude dependent, called 'Dtest' and 'Ttest'. If a given event falls within the maximal values defined by Dtest and Ttest for its magnitude it is deemed an aftershock and removed from the catalog. After all events are processed, the remaining catalog is then comprised of only primary events. This declustered catalog can be used to estimate more accurate Gutenberg-Richter statistics. Furthermore, we can study the aftershock events that the algorithm removed for a given earthquake in the context of the Omori Law statistics (Exercise 4).

Because aftershock identification is an empirical procedure, there are many different ways to define the Dtest and Ttest relationships. Stiphout et al., (2012, on page 10) summarizes three different definitions of the Dtest/Ttest relationships originally proposed by Uhrhammer (1986), Knopoff and Gardner (1972), and Gruenthal.

Compare the event reduction rate (final number divided by the initial number of events) for the three different proposed distance and time windows. You can do this by adding a logical (if statement) tree to enable switching between different definitions of Dtest and Ttest in declustering_algorithm below. The first definition (Eqn 1 from Stiphout et al., 2012, p.10) has already been completed.

```
In [ ]: def declustering_algorithm(cat,definition=1):
            '''
            Decluster a catalog

            note: This function may take a few minutes to complete

            calls readAnssCatalog()

            Inputs:

            cat must be an anss formatted pandas datafram
            definition is the algorithm (1 - 3) from Stiphout, 2012, which determines Dtest an
                Definition = 1 : Gardner and Knopoff, 1974 [default]
                Definition = 2 : Gruenthal
                Definition = 3 : Uhrhammer, 1986

            '''
            import numpy as np

            # do not edit
            cnt=0
```

```
save=np.zeros((1,10000000),dtype=int)

# grab catalog arrays
year,month,day,hour,minute,sec,lat,lon,mag,days = readAnssCatalog(cat)
ne=len(year)

# main for-loop over events
for i in range(0,ne,1):

    if definition == 1:

        # Definition #1 : Knopoff and Gardner, 1972
        Dtest=np.power(10,0.1238*mag[i]+0.983)
        if mag[i] >= 6.5:
            Ttest=np.power(10,0.032*mag[i]+2.7389)
        else:
            Ttest=np.power(10,0.5409*mag[i]-0.547)

    elif definition == 2:

        # Definition #2 : Gruenthal # *****
        Dtest=np.exp(1.77+(0.037+1.02*mag[i])**2)   # distance bounds
        if mag[i] >= 6.5:
            Ttest=abs(np.exp(-3.95+(0.62+17.32*mag[i])**2))  # aftershock time bou
        else:
            Ttest=np.power(10,0.024*mag[i]+2.8)  # aftershock time bounds for M <

#         elif definition == 3:
#
        # Definition #3 # *****



    a=days[i+1:ne]-days[i]
    m=mag[i+1:ne]
    b=haversine_np(lon[i],lat[i],lon[i+1:ne],lat[i+1:ne])

    icnt=np.count_nonzero(a <= Ttest)
    if icnt > 0:
        itime=np.array(np.nonzero(a <= Ttest)) + (i+1)
        for j in range(0,icnt,1):
            if b[j] <= Dtest and m[j] < mag[i]:
                save[0][cnt]=itime[0][j]
                cnt += 1 # save contains index of aftershocks in cat

#Note this is an array of indexes that will be used to delete events flagged
                    #as aftershocks
save=np.delete(np.unique(save),0)
```

```
    # Filter or slice out the declustered and aftershock dataframe catalogs from the
    # original dataframe catalog "data" using "save" above.
    cat_aftershocks = cat.iloc[np.unique(save)] # *****
    cat_declustered = cat.iloc[~cat.index.isin(save)]

    cat_aftershocks.reset_index(drop=True, inplace=True)
    cat_declustered.reset_index(drop=True, inplace=True)

    return cat_declustered, cat_aftershocks
```

In [ ]:
```
# Run the declustering algorithm
data_declustered, data_aftershocks = declustering_algorithm(data,definition=1)

# This condition should print out "True" if the catalogs were separated correctly
len(data) == len(data_aftershocks) + len(data_declustered)
```

3.0.2 Plot a map showing the declustered catalog

In [ ]:
```
# load the declustered dataframe into numpy as an array with different d"" variable na
dyear,dmonth,dday,dhour,dmn,dsec,dlat,dlon,dmag,ddays = readAnssCatalog(data_decluster

#Make map
lat0=36.75 #Set Corners of Map
lat1=39.0
lon0=-123.75
lon1=-121.0
tickstep=0.5 #for axes
latticks=np.arange(lat0,lat1+tickstep,tickstep)
lonticks=np.arange(lon0,lon1+tickstep,tickstep)
ydim=10      #height of plot
xdim=ydim*(haversine_np(lon0,lat0,lon1,lat0)/haversine_np(lon0,lat0,lon0,lat1)) #scale
plt.figure(figsize=(ydim,xdim))
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_extent([lon0, lon1, lat0, lat1], crs=ccrs.PlateCarree())
ax.set_aspect('auto')
ax.coastlines(resolution='10m',linewidth=1) #downloaded 10m, 50m
ax.set_xticks(lonticks)
ax.set_yticks(latticks, crs=ccrs.PlateCarree())
ax.set(xlabel='longitude', ylabel='Latitude',
       title='Declustered Catalog')
ax.add_feature(cfeature.BORDERS, linestyle=':')
ax.add_feature(cfeature.LAKES,alpha=0.5)
ax.add_feature(cfeature.RIVERS)
ax.add_feature(cfeature.STATES.with_scale('10m'))

# Plot events as open circles with size and color proportional to event magnitude
indx=np.argsort(dmag)   #determine sort index #Sort Descending to plot largest events
```

```
x=dlon[indx]            #apply sort index
y=dlat[indx]
z=np.exp(dmag[indx])    #exponent to scale size
c = plt.cm.plasma(z/max(z))
plt.scatter(x, y, s=(z/2), facecolors='none', edgecolors=c, marker='o', linewidth=2, al

# Add Berkeley, CA as a red square with size proportional to event magnitude
plt.plot(-122.2727,37.8716,'rs',markersize=10) # *****

#Save the plot by calling plt.savefig() BEFORE plt.show()
plt.savefig('hw1_ex4_seismap_declust.pdf')
plt.savefig('hw1_ex4_seismap_declust.png')

plt.show()
```

3.0.3 Re-compute the Gutenberg-Richter statistics as above for the declustered catalog

```
In [ ]: #Determine and plot the Gutenberg-Richter Distribution for De-clustered data
        #You may want to adjust the magnitude range of the analysis to focus on where the cata
        m=np.arange(1.5,6.9,0.1)
        N=np.zeros(len(m))

        for i in range(0,len(m),1):
            N[i]=np.log10(np.count_nonzero(dmag >= m[i])/numyr)

        #Invert for A and B values
        G=np.column_stack(( ,m)) # *****
        GTG=np.dot(np.transpose(G),G)
        GTD=np.dot(np.transpose(G),N)
        soln=np.linalg.solve( , ) # *****
        x=m
        y=np.dot( ,soln) # *****

        #Compute the uncertainty in Gutenberg-Richter Parameters
        df=len(N) - 2               #degree of freedom
        e=N-y                       #prediction error
        var=np.sum(e**2)/df
        se_y=np.sqrt(var)           #standard error of the estimate
        sdev=np.sqrt(var)           #standard deviation

        #Calculate 95% confidence bounds
        t=stats.t.ppf(1-0.05/2,df)   #two-sided students t-distribution
        tmp=np.sqrt(1/len(x)+((x-np.mean(x))**2)/np.sum((x-np.mean(x))**2))
        tmp=tmp/max(tmp)
        lower95=y-t*se_y*tmp
        upper95=y+t*se_y*tmp
        se_b=sdev/np.sqrt(np.sum((x-np.mean(x))**2))                    #standard error slop
        se_a=sdev*np.sqrt(1/len(x) + np.mean(x)**2/np.sum((x-np.mean(x))**2)) #standard error
```

```
a95=se_a*t
b95=se_b*t

#Now Plot
fig, ax = plt.subplots()
ax.plot(m, N,'b.',x,y,'k-',x,lower95,'r-',x,upper95,'r-')
ax.set(xlabel='magnitude', ylabel='Number of Earthquakes (log10)',
       title='Declustered Gutenberg-Richter Distribution')
ax.grid()

fig.savefig("hw1_ex4_figure4.png")
plt.show()

print(f'A_value= {soln[0]:.3f} B_value={soln[1]:.3f}')
print(f'95%intercept= {a95:.3f} 95%slope={b95:.3f}')
```

3.0.4 Questions

1. How many events were removed from the catalog by each declustering algorithm?
2. Compare the spatial distribution of earthquakes between the raw and declustered catalogs.
3. For the two other methods of declusting, how many events were removed from the catalog?
4. Compare the Gutenberg-Richter A and B coefficients for the three versions of the declustered catalog.
5. What is the annual rate of occurrence of M4 earthquakes for each of the declustered catalogs?
6. What is the average M7 return period (inverse of annual occurrence of M7 events) for each of the declustered catalogs?
7. Compare your estimated values with what has been presented in the USGS Earthquake Hazard Assessments of the return period for Hayward fault earthquakes.

4 Exercise 4: Omori Law for Loma Prieta M6.9 Event (30 pts)

Here we will use the declustering algorithm to identify aftershocks of the October 18 1989 at 04:15am (October 17 at 5:15pm PDT) the M6.9 Loma Prieta earthquake occurred in the Santa Cruz mountains approximately 80 km southwest of the Berkeley Campus. This wiki has some background information for the earthquake: https://en.wikipedia.org/wiki/1989_Loma_Prieta_earthquake

4.0.1 Load the Earthquake Catalog

Load the .csv data file of all the earthquakes 1900 - 2018 in the ANSS (Advanced National Seismic System) catalog from 100 km around Berkeley.

```
In [26]: # This catalog is a M0+ search centered at Berkeley radius=100km.
         # A big enough radius to include Loma Prieta but exclude Geysers.
```

```
data=pd.read_csv('anss_catalog_1900to2018all.txt', sep=' ', delimiter=None, header=Nor
                 names = ['Year','Month','Day','Hour','Min','Sec','Lat','Lon','Mag'])
year,month,day,hour,minute,sec,lat,lon,mag,days = readAnssCatalog(data)
```

4.0.2 Select earthquakes related to the Loma Prieta Earthquake

Use Boolean indexing to select events from the full catalog from between October 18, 1989 (date of mainshock) and December 18, 1989 (3-months following).

```
In [65]: EQ_1989 = data[(data.Year>=1) & (data.Year<1)]  # *****        #get one year of data

         fall_eq = EQ_1989[(EQ_1989.Month>9) & (EQ_1989.Month<=12)]    #collect months of Oct,
         LP_eq = fall_eq[(~((fall_eq.Month==10) & (fall_eq.Day<18)))]  #negate events before d
         LP_eq = LP_eq[(~((LP_eq.Month==12) & (LP_eq.Day>18)))]         #negate events after da
         LP_eq.reset_index(drop=True, inplace=True)
```

Create data arrays for 3-month period beginning with Loma Prieta Earthquake

```
In [66]: # year,month,day,hour,minute,sec,lat,lon,mag = readAnssCatalog(EQ_1989)#override for
         year,month,day,hour,minute,sec,lat,lon,mag,days = readAnssCatalog(LP_eq)
         nevt = len(mag)
```

4.0.3 Plot the Loma Preita time series

```
In [67]: # plot magnitude vs. day
         fig, ax = plt.subplots(figsize=(7,7))
         ax.plot(  , ,'o',alpha=0.2,markersize=5) # *****
         ax.set(xlabel='Days', ylabel='Magnitude',
                title='Raw Event Catalog')
         ax.grid()
         ax.set_ylim([0,7])
         fig.savefig("hw1_ex4_ts_raw.png")
         plt.show()

         print(f'Number={nevt:d} MinMag={min(mag):.2f} MaxMag={max(mag):.2f}')


  File "<ipython-input-67-e81dfd2ff7f1>", line 3
    ax.plot(  , ,'o',alpha=0.2,markersize=5) # *****
              ^
SyntaxError: invalid syntax
```

4.0.4 Plot the Loma Preita Earthquake Catalog in map view

```
In [ ]: #Set Corners of Map
        lat0=36.75
        lat1=39.0
```

```
lon0=-123.75
lon1=-121.0
tickstep=0.5 #for axes
latticks=np.arange(lat0,lat1+tickstep,tickstep)
lonticks=np.arange(lon0,lon1+tickstep,tickstep)

plt.figure(1,(10,10))
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_extent([lon0, lon1, lat0, lat1], crs=ccrs.PlateCarree())
ax.coastlines(resolution='10m',linewidth=1)
ax.set_aspect('auto')
ax.set_xticks(lonticks)
ax.set_yticks(latticks, crs=ccrs.PlateCarree())
ax.set(xlabel='Longitude', ylabel='Latitude',
       title='Earthquake Catalog')

#Sort Descending to plot largest events on top
indx=np.argsort(mag)   #determine sort index
x=lon[indx]            #apply sort index
y=lat[indx]
z=np.exp(dmag[indx])    #exponent to scale size
c = plt.cm.viridis_r(z/max(z)) # colormap scales with magnitude
plt.scatter(x, y, s=(z), facecolors=c, alpha=0.5, edgecolors='k', marker='o', linewidth
plt.plot(-122.2727,37.8716,'rs',markersize=8)  # plot red square on Berkeley

plt.savefig("hw1_ex4_map_raw.png")

plt.show()
```

4.0.5 Decluster the Raw Catalog for the Loma Prieta time period

We use the same decluster algorithm previously to identify aftershocks and remove them from the 30-day Loma Preita catalog.

```
In [ ]: data_dec, data_after = declustering_algorithm(LP_eq,definition=2)

        # This condition should print out "True" if the catalogs were separated correctly
        len(LP_eq) == len(data_after) + len(data_dec)
```

Create two sets of arrays, one for the declustered catalog and one for the aftershock catalog. Use `np.delete()` to delete the aftershock events for the declustered catalog, and use `after` to select the aftershock events for the aftershock calalog.

```
In [ ]: dyear,dmonth,dday,dhour,dminute,dsec,dlat,dlon,dmag,ddays = readAnssCatalog(data_dec)
        ayear,amonth,aday,ahour,aminute,asec,alat,alon,amag,adays = readAnssCatalog(data_after
        dnevt =len(ddays)
        anevt=len(adays)
```

In [ ]:
```
#Plot Aftershock Catalog in time series
fig, ax = plt.subplots(figsize=(7,7))
ax.plot( ,  ,'o',alpha=0.2,markersize=5)
ax.set(xlabel='days', ylabel='magnitude',
       title='Aftershock Event Catalog')
ax.grid()
ax.set_ylim([0,7])
fig.savefig("hw1_ex4_ts_aftershockOnly.png")
plt.show()

print(f'Number={anevt:d} MinMag={min(amag):.2f} MaxMag={max(amag):.2f}')
```

In [ ]:
```
#Make a Map of the declustered events

#Set Corners of Map
lat0=36.75
lat1=39.0
lon0=-123.75
lon1=-121.0
tickstep=0.5 #for axes
latticks=np.arange(lat0,lat1+tickstep,tickstep)
lonticks=np.arange(lon0,lon1+tickstep,tickstep)

plt.figure(1,(10,10))
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_extent([lon0, lon1, lat0, lat1], crs=ccrs.PlateCarree())
ax.set_aspect('auto')
ax.coastlines(resolution='10m',linewidth=1) #downloaded 10m, 50m
ax.set_xticks(lonticks)
ax.set_yticks(latticks, crs=ccrs.PlateCarree())
ax.set(xlabel='Longitude', ylabel='Latitude',
       title='Aftershock Catalog')

#Sort Descending to plot largest events on top
indx=np.argsort(dmag)   #determine sort index
x=dlon[indx]            #apply sort index
y=dlat[indx]
z=np.exp(dmag[indx])    #exponent to scale size
c = plt.cm.viridis_r(z/max(z))
plt.scatter(x, y, s=(z/2), facecolors=c, alpha=0.4, edgecolors='k', marker='o', linewic
plt.plot(-122.2727,37.8716,'rs',markersize=8)

plt.savefig("hw1_ex4_map_mainshockOnly.png")


plt.show()
```

In [ ]:
```
#Make a Map of Aftershock events
```