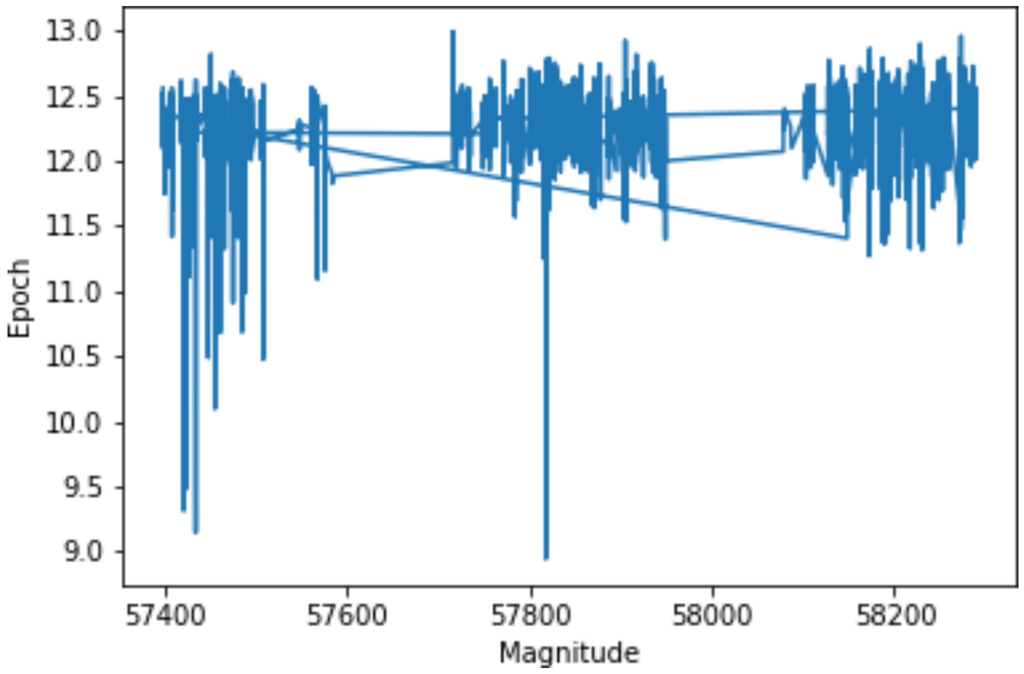
Avery Horvath

**1/21/19:**
When the data was first plotted, it looked very messy. So, when

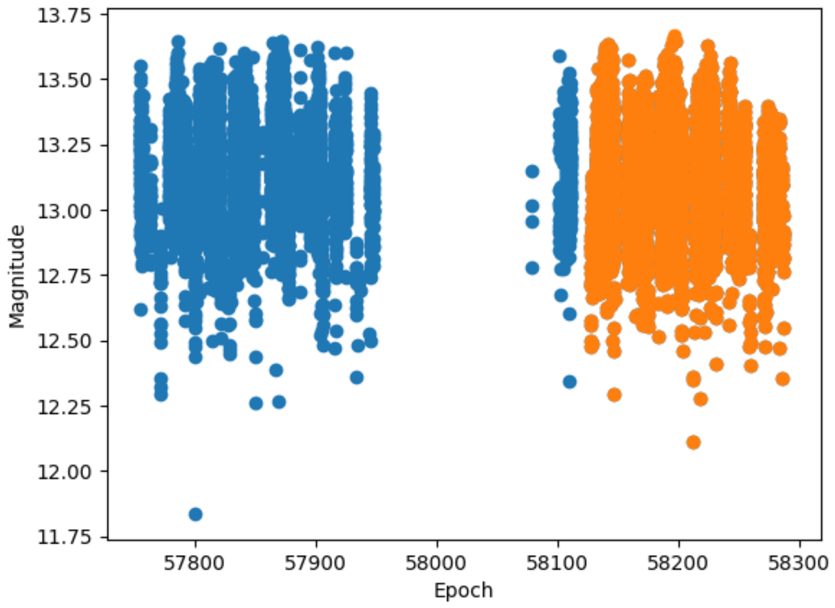
```
epoch = epoch[flag == 0]
mag = mag[flag == 0]
```

was added, the following plot was displayed.



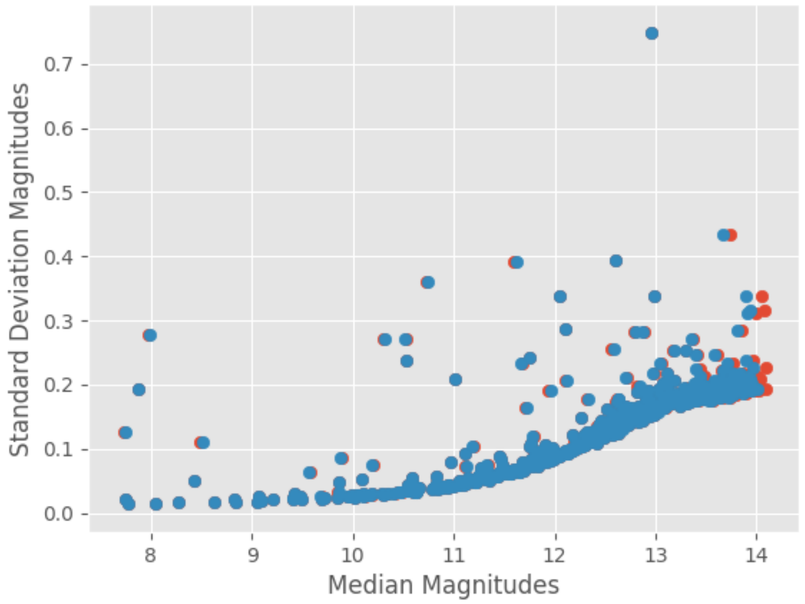
**1/28/19:**

This week magnitude vs. epoch for 2017 and 2018 were plotted. Data from 2016 was removed because this data was not good. Then, we conducted sigma clipping with a sigma of 3. This removes magnitudes that are greater than 3 standard deviations away from the mean of the magnitudes. This



generated the plot below:

Next, all data files were looped through and the median magnitude and magnitude standard deviation was found for each file. The median magnitude and standard deviation for 2017 and 2018 for each file was appended to lists. These lists were then plotted and the result is shown below:



**2/3/19:**

Periods between 0.2 and 10 were put into a supersmoother for each of the stars using:

```
period_arange, delta_mags = supersmoother(epoch,mag,0.2,10.0,0.01)
```

Where the super smoother function is:

```
def supersmoother(epoch_arr, mag_arr, period_min, period_max,stepsize):
    period_arange=np.arange(period_min,period_max,stepsize)
    delta_mags = []
    for period in period_arange:
        phased_time=np.fmod(epoch_arr,period)/period
        phase_sorted_ind = np.argsort(phased_time)
        mag_sorted = mag_arr[phase_sorted_ind]
        delta_mags.append(np.mean(np.fabs(mag_sorted[1:] - mag_sorted[0:-1])))

    return period_arange, delta_mags
```

We can tell whether or not we have the right period based on if the graph looks periodic or volatile. If the graph is volatile, the delta magnitude between one point and the next will be high. The five periods with the smallest delta magnitude were put into an array for each star/file using:

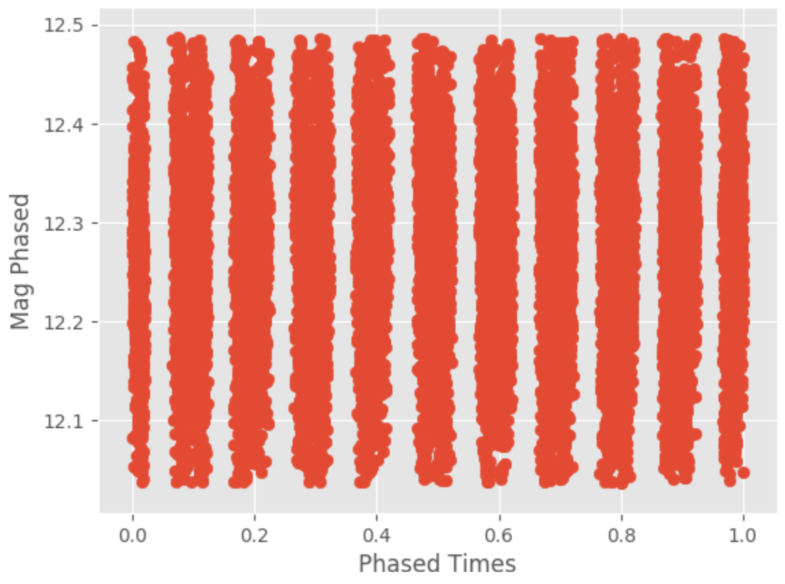
```
mags_sort_ind = np.argsort(delta_mags)
test_periods = period_arange[mags_sort_ind][0:4]
```

These 5 periods represented the best contenders for the period of each star. Therefore, the phased epoch was plotted against the phased magnitude at these 5 periods for each star and saved. This was done with:

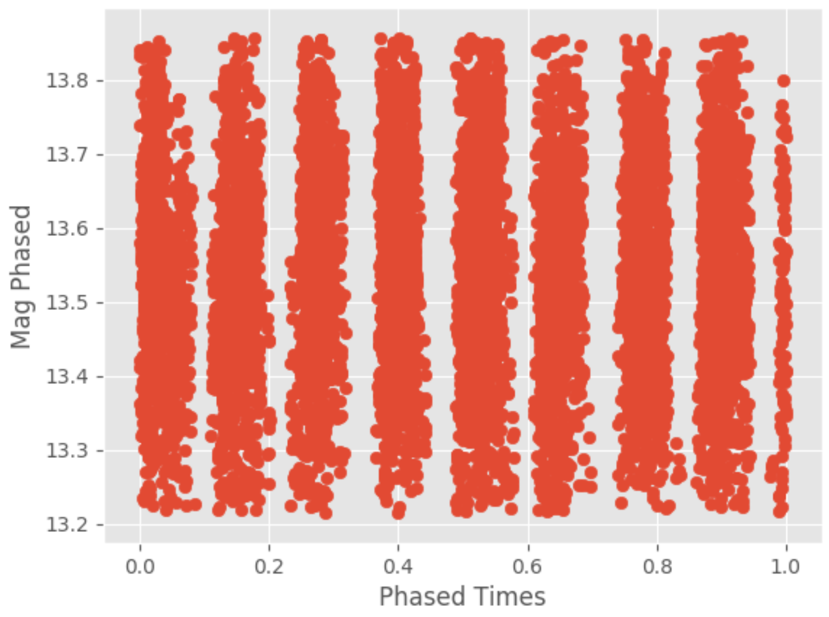
```
for test_period in test_periods:
            phased_epoch_unsorted=np.fmod(epoch,test_period)/test_period
            sort_ind              = np.argsort(phased_epoch_unsorted)
            phased_epoch_sorted   = phased_epoch_unsorted[sort_ind]
            mag_phased            = mag[sort_ind]

            img = filepath.split("/")[2]
            plt.figure()
            plt.scatter(phased_epoch_sorted,mag_phased)
            plt.title("%s \n Period = %s" % (img,test_period))
            plt.xlabel("Phased Times")
            plt.ylabel("Mag Phased")
            if not os.path.exists("./images/%s" % (img)):
                os.makedirs("./images/%s" % (img))
            plt.savefig("./images/%s/phased_period_%s.png" % (img,test_period))
```

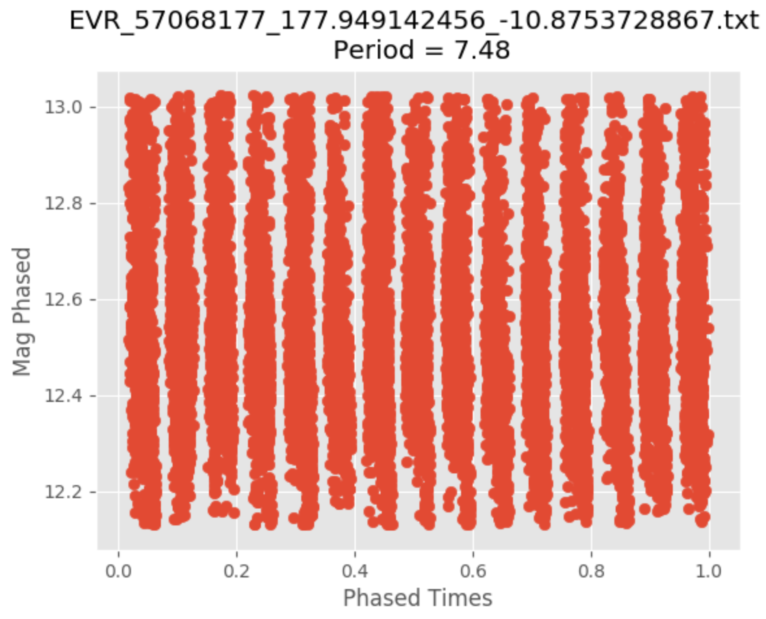
There were many stars with periodic behavior – only a few are shown below:

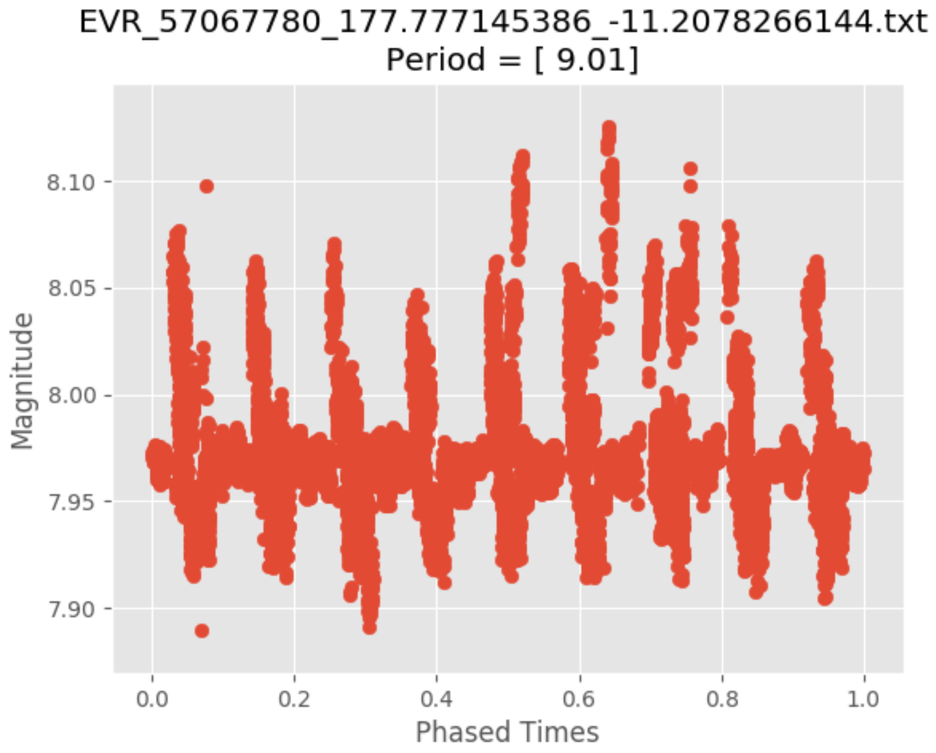
EVR_54848050_176.625320435_-9.99535369873.txt
Period = 9.98

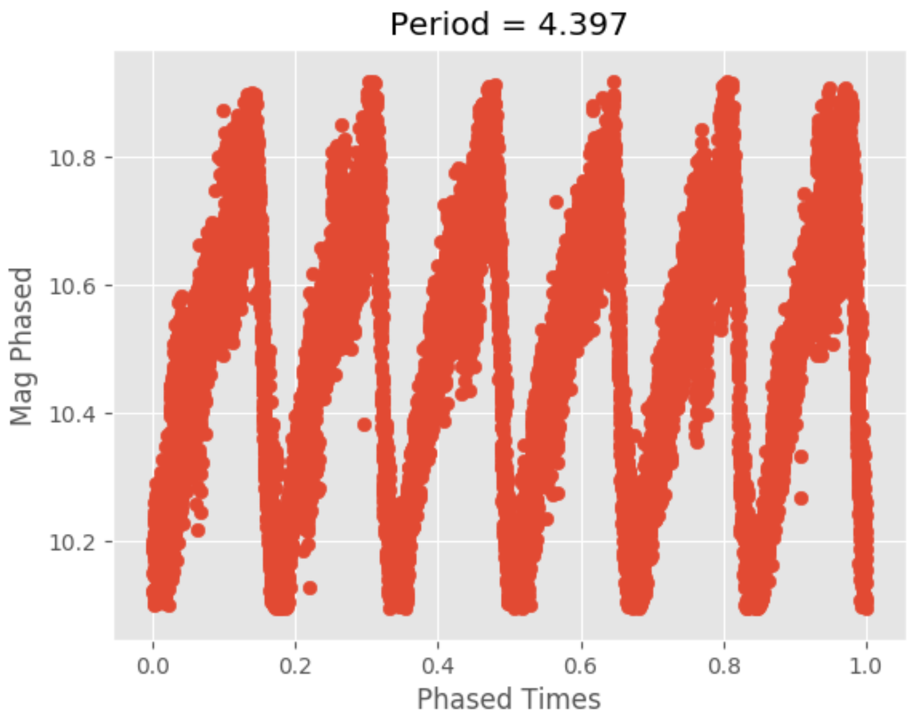


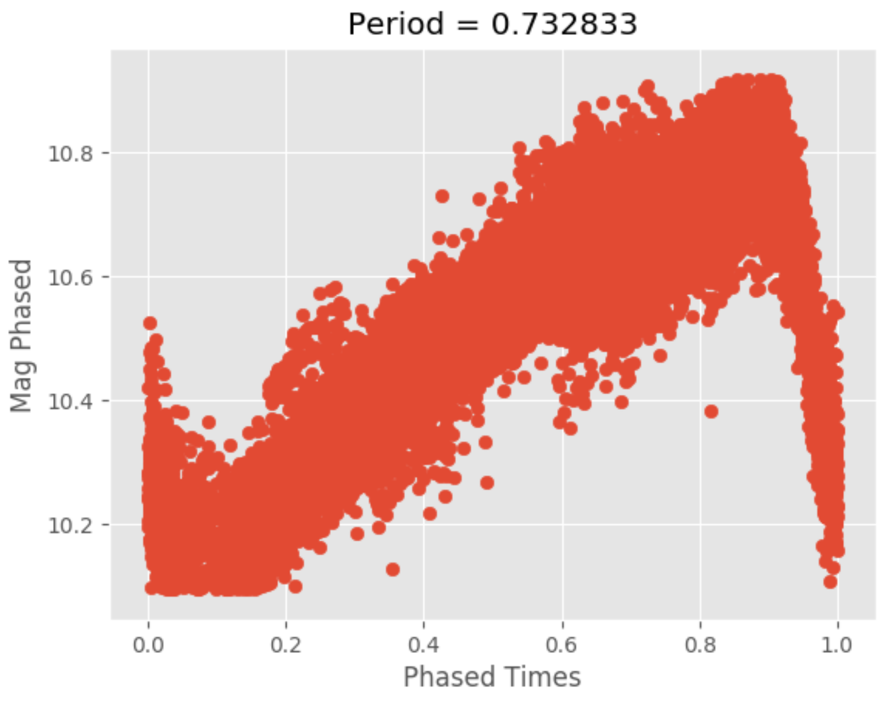
EVR_57068836_178.225952148_-10.2506151199.txt
Period = 8.28




EVR_57068177_177.949142456_-10.8753728867.txt
Period = 7.48
Mag Phased
13.0
12.8
12.6
12.4
12.2
0.0
0.2
0.4
0.6
0.8
1.0
Phased Times


EVR_57067780_177.777145386_-11.2078266144.txt
Period = [ 9.01]
Magnitude
8.10
8.05
8.00
7.95
7.90
0.0
0.2
0.4
0.6
0.8
1.0
Phased Times





**2/10/18:**

This week we simulated a light curve of eclipsing binaries in order to test the supersmoother function's limitations (see above). This was done with the code below

```
EPOCH_NUM             = int(3*525600/2)
TEN_HOUR_PERIOD       = 60*10
THOUSAND_HOUR_PERIOD  = 60*1000
ECLIPSE_LENGTH        = 1.5*60
```

```
def create_lightcurve():
    MIN_STEP      = 2
    STEADY_MAG    = 12
    MAG_DROP      = 0.5

    TWENTY_FOUR_HOURS = 24*60
    SNR_ARR       = [0.001,2,10]


    for snr in SNR_ARR:
        noise = 0.5/snr
        epoch_arr     = np.arange(0,EPOCH_NUM,MIN_STEP)
        mag_arr       = np.ones(EPOCH_NUM // MIN_STEP)*STEADY_MAG

        for i in range(0,len(mag_arr),int(TEN_HOUR_PERIOD // MIN_STEP)):
            mag_arr[i:i + int(ECLIPSE_LENGTH // MIN_STEP)] += MAG_DROP

        for i in range(0,len(mag_arr),int(TWENTY_FOUR_HOURS // MIN_STEP)):
            mag_arr[i:i + int(TEN_HOUR_PERIOD // MIN_STEP)] = 'nan'


        plt.plot(epoch_arr,mag_arr)
        plt.xlim(0,6000)
        plt.xlabel("Time (minutes)")
        plt.ylabel("Magnitude")
        plt.title("Magnitude Over 100 Hours Without Any Noise")
        plt.savefig("./eclipse_magnitudes_day_night_without_noise.png")
        plt.show()

        mag_arr              = np.array([i + np.random.normal(scale = noise) for i in mag_arr])

        period               = int(10*60)

        _, _, phased_epoch_sorted,mag_sorted = supersmoother(epoch_arr,mag_arr,period=period)


        plt.scatter(phased_epoch_sorted,mag_sorted, marker='o',edgecolors='blue')
        plt.ylabel("Magnitude")
        plt.xlabel("Time (minutes)")
        ax = plt.gca()
        ax.set_ylim(ax.get_ylim()[::-1])
        plt.title("Mock Data Light Curve with SNR = %s " % snr)
        plt.savefig("./phased_up_ten_hour_period_%s_SNR.png" % snr)
        plt.show()
```

This creates a supersmoother periodogram for a 10 hour period. This can be done for an 1000 period replacing TEN_HOUR_PERIOD with THOUSAND_HOUR_PERIOD in the create_lightcurve() function above. All plots use a epoch_arr with one point every two minutes for 3 years.

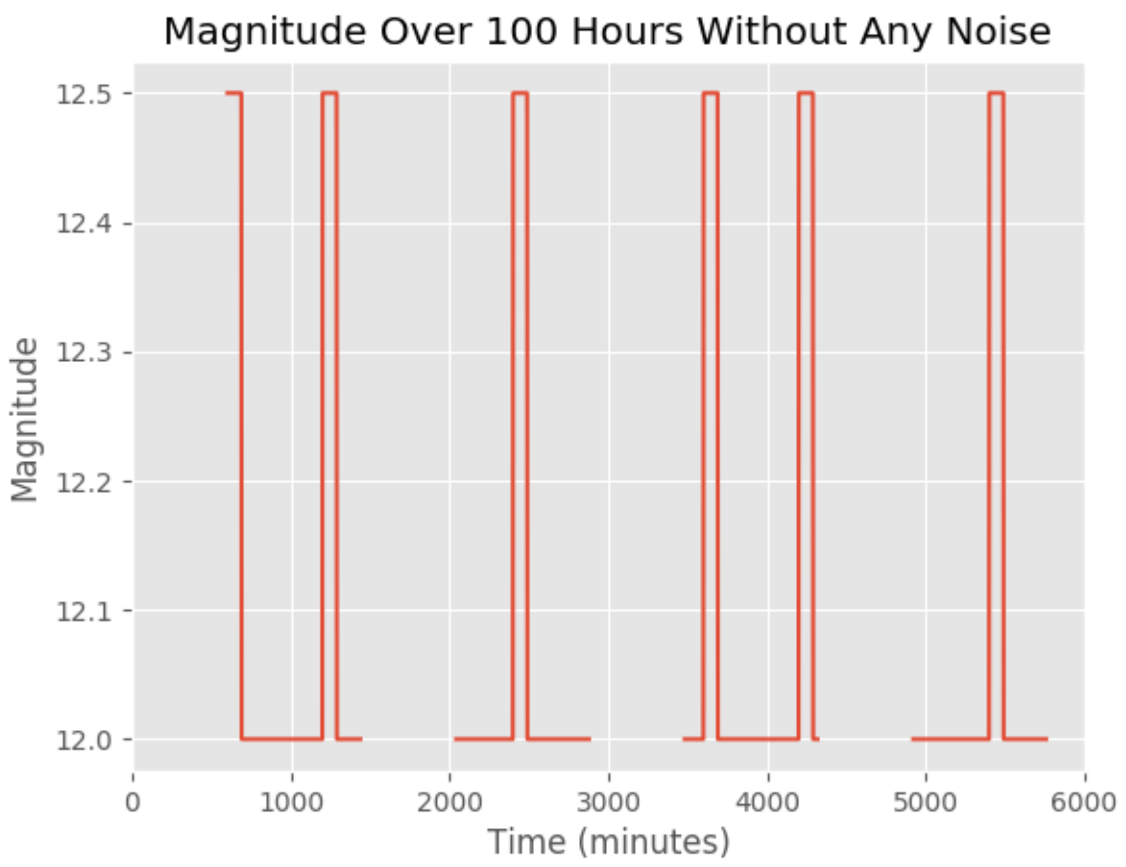
First, the magnitude of eclipsing binaries was plotted without any noise. The length of the eclipse is made to be 1.5 hours and the period is 10 hours. This was done through:

```
for i in range(0,len(mag_arr),int(TEN_HOUR_PERIOD // MIN_STEP)):
            mag_arr[i:i + int(ECLIPSE_LENGTH // MIN_STEP)] += MAG_DROP
```

In addition, day-night cycles were taking into consideration via

```
for i in range(0,len(mag_arr),int(TWENTY_FOUR_HOURS // MIN_STEP)):
            mag_arr[i:i + int(TEN_HOUR_PERIOD // MIN_STEP)] = 'nan'
```

where the telescope is considered to be on 10 hours per day. This yielded the plot below:



In order to test how the supersmoother function upheld when noise was added, noise was added to every element in the magnitude array using:

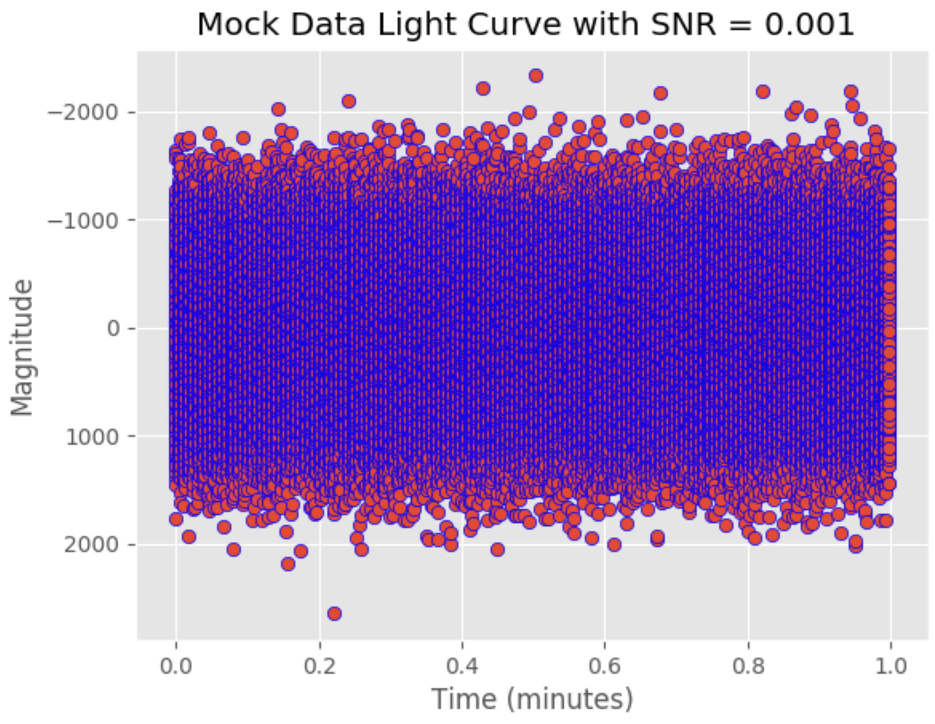
```
mag_arr             = np.array([i + np.random.normal(scale = noise) for i in mag_arr])
```

In addition, an array of SNR values (`SNR_ARR       = [0.001,2,10])` ) was created and looped through to calculate the noise values that yielded these SNR vales with:

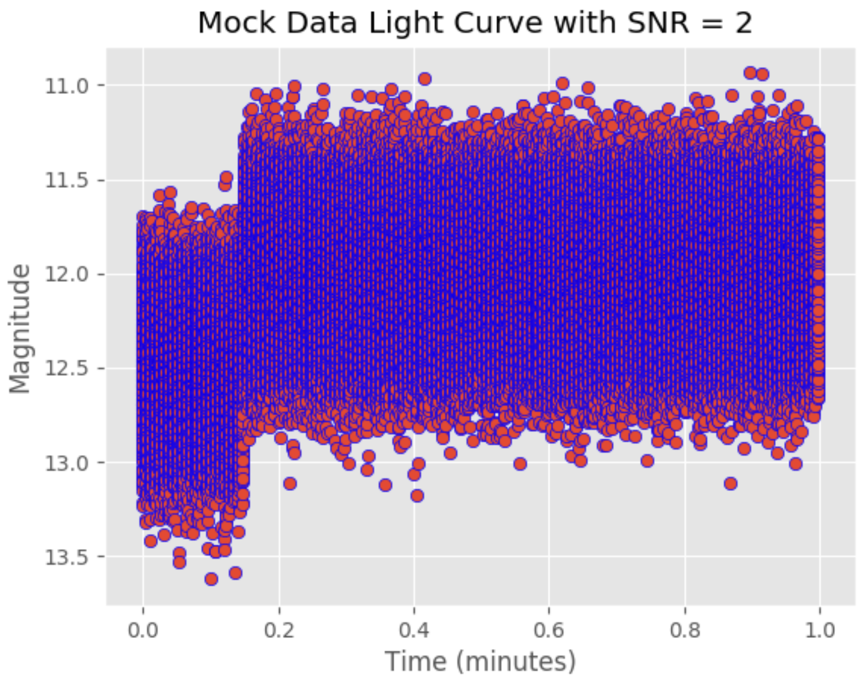
```
for snr in SNR_ARR:
        noise = 0.5/snr
```

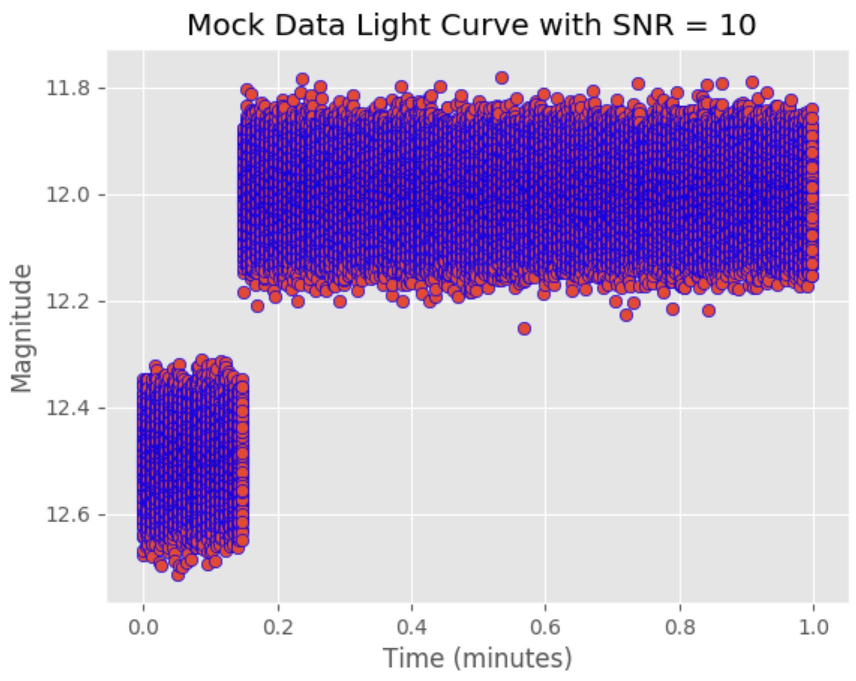
This noise was then used as the covariance of the random number generator as evident from np.random.normal(scale = noise) above.

This yielded the plots below for different SNR values:

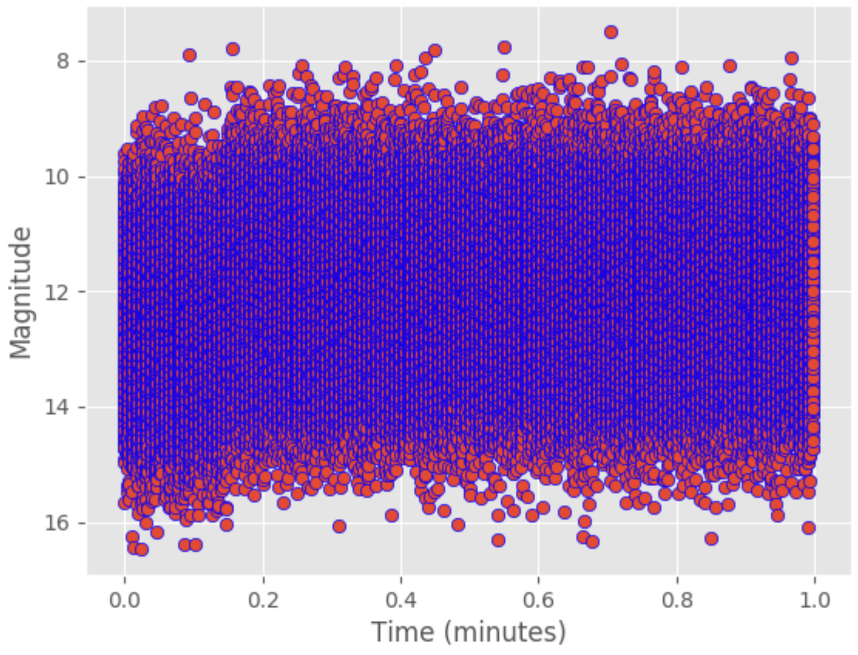


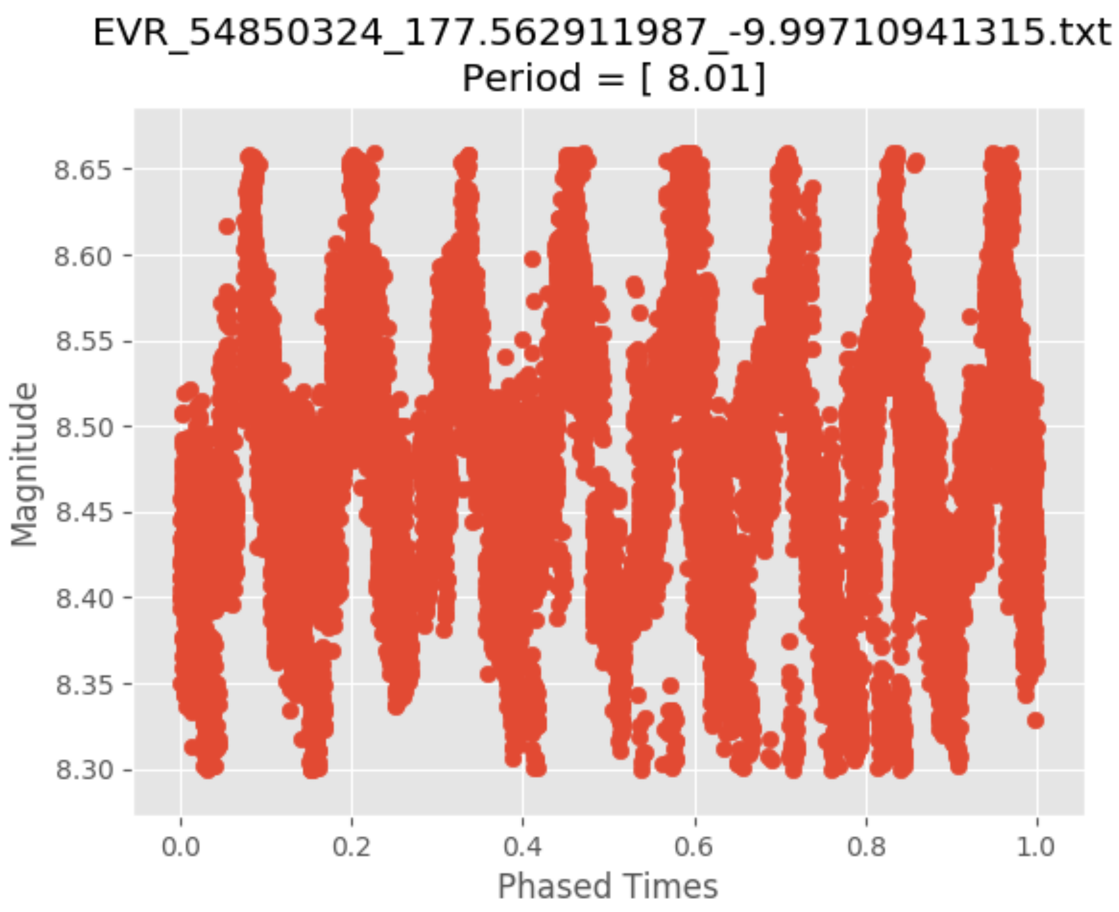
Where the SNR ~ 0.





When the standard deviation was just 1:





**2/18/19:**

Dr. Law – I am aware something is wrong with my BLS function. I would have come see you this morning but I had an interview. I worked for hours last night to try and figure out what was wrong but I can't seem to figure it out. I'm not sure why the plot isn't trending upwards.

This week the Box Least Squares method (BLS) was applied to the simulated data, evr_samples, and var_examples – a new dataset.

The BLS function is defined below:

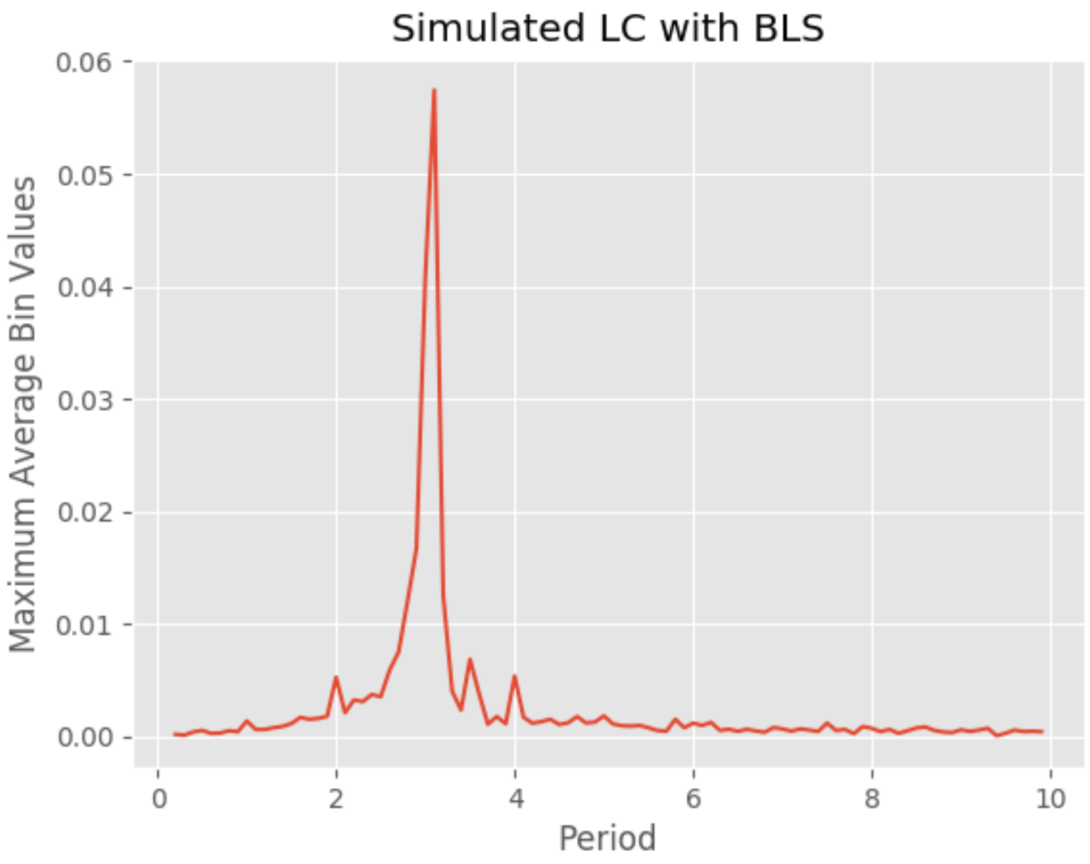
```python
def BLS(epoch_arr,mag_arr,period=None,period_min=None,period_max=None,stepsize=None):

    if period:
        period_arange = np.arange(period,period+1,1) # list of size 1 so it works with loop below
    if stepsize:
        period_arange = np.arange(period_min,period_max,stepsize)

    mag_arr -= np.median(mag_arr)

    best_chunks = []
    for period in period_arange:
        phased_epoch        = np.fmod(epoch_arr,period)/period
        phase_sorted_ind    = np.argsort(phased_epoch)
        mag_sorted          = mag_arr[phase_sorted_ind]
        mag_binned = []
        for i in range(0,len(mag_arr),int(period * TWENTY_FOUR_HOURS)):
            mag_binned.append(mag_sorted[i:i+int(ECLIPSE_LENGTH)])



        bin_means = []
        for bin in mag_binned:
            weighted_mean = np.sum(bin)/(len(bin)*(len(mag_binned)-len(bin)))
            bin_means.append(weighted_mean)

        best_chunks.append(np.max(bin_means))

    return period_arange, best_chunks

def main()
       data  = np.loadtxt(filepath)
       epoch, mag, mag_err, flag, median_mags, stds = manipulate_data(data)


       img = (filepath.split("/")[2]).split(".txt")[0]

       period_arange, best_bins = BLS(epoch,mag,period_min=0.2,period_max=10.0,stepsize=0.01)

       plt.figure()

       plt.plot(period_arange,best_bins)
       plt.xlabel("Period")
       plt.ylabel("Maximum Average Bin Value (Dimmest Magnitude)")
       plt.title("%s \n Using BLS" % img)
       plt.savefig("./images/%s_imgs/%s/BLS_%s.png" % (TEST_DATA,img,img))
main()
```

where first the median of the light curve was subtracted from the light curve using

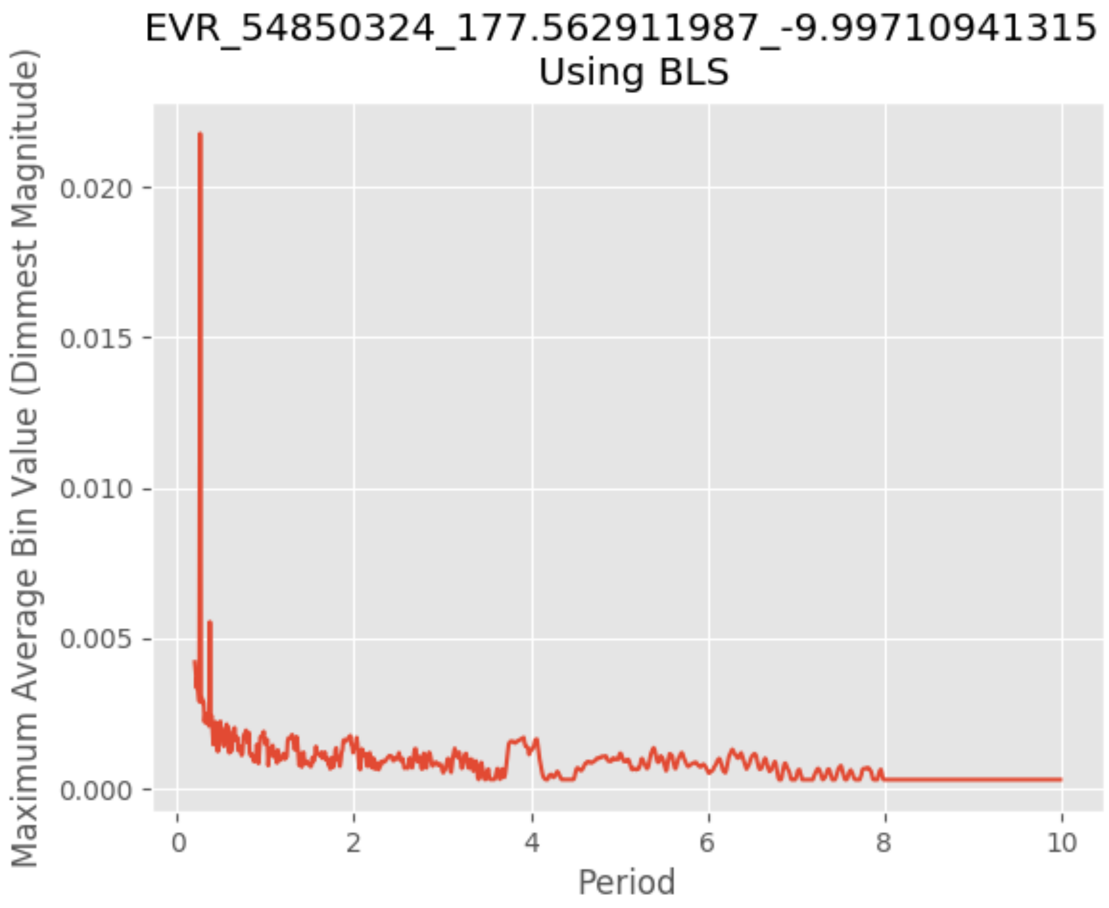
```
mag_arr -= np.median(mag_arr)
```

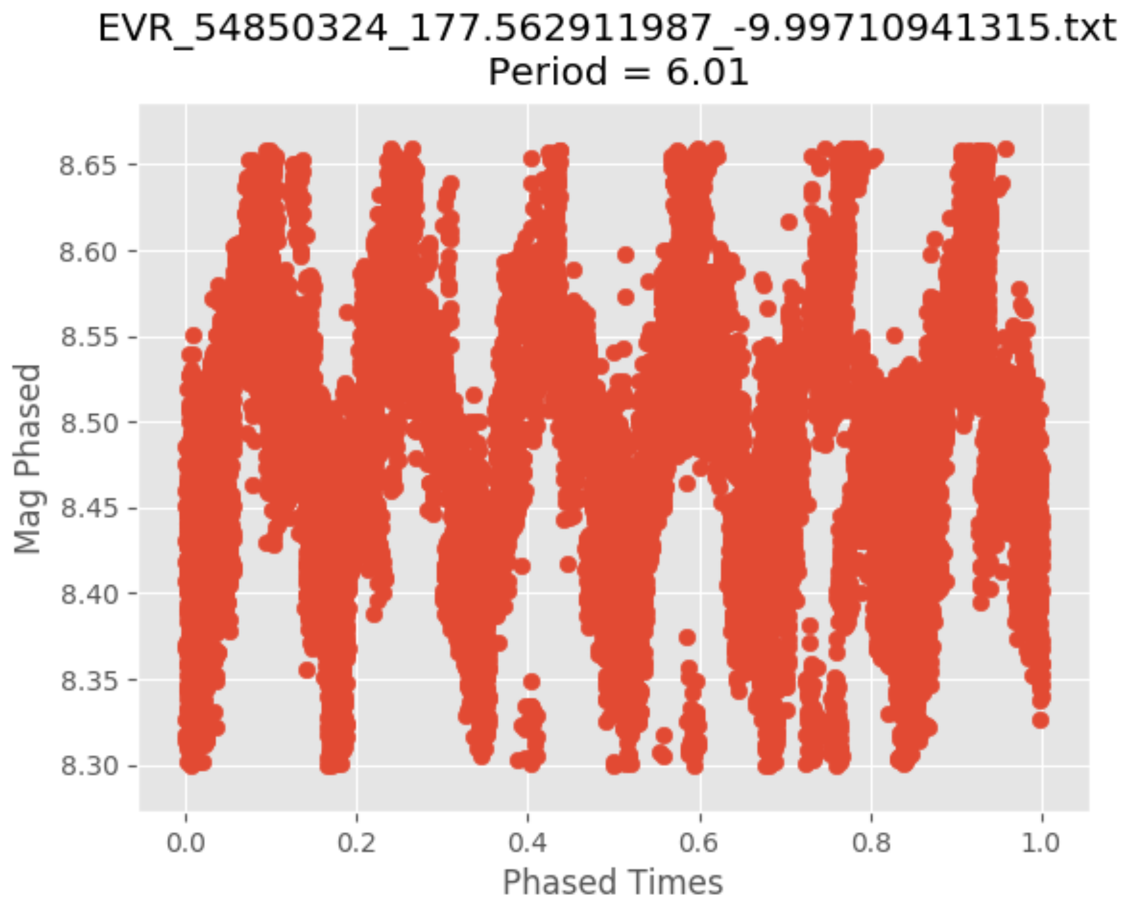
Then, the light curve was phased up and split into bins of the length of one eclipse. (Note: the length of the eclipse is only known for the simulated data) After binning the magnitudes, the bin with the smallest magnitude weighted mean was chosen and added to bin_means – a list holding the smallest magnitude weighted means for each period.

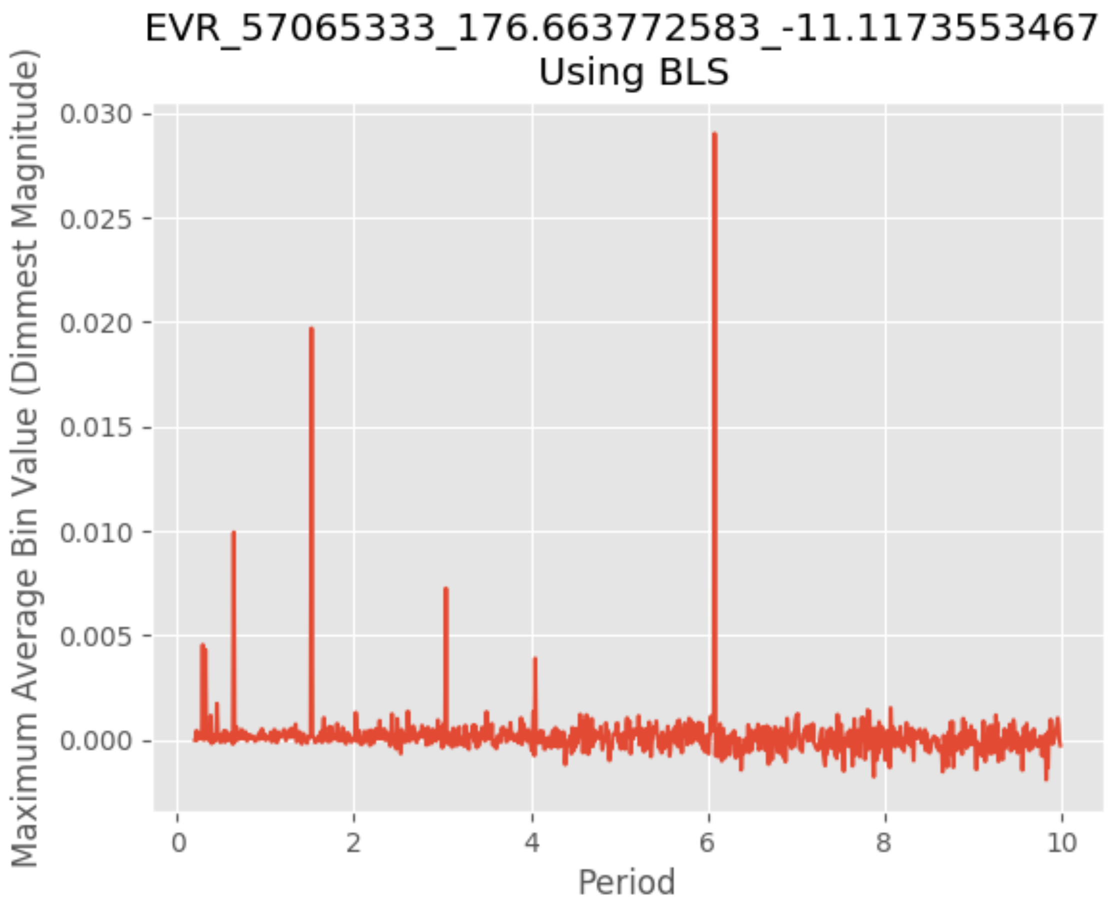
When this BLS was applied to the simulated data, the following plot was generated:

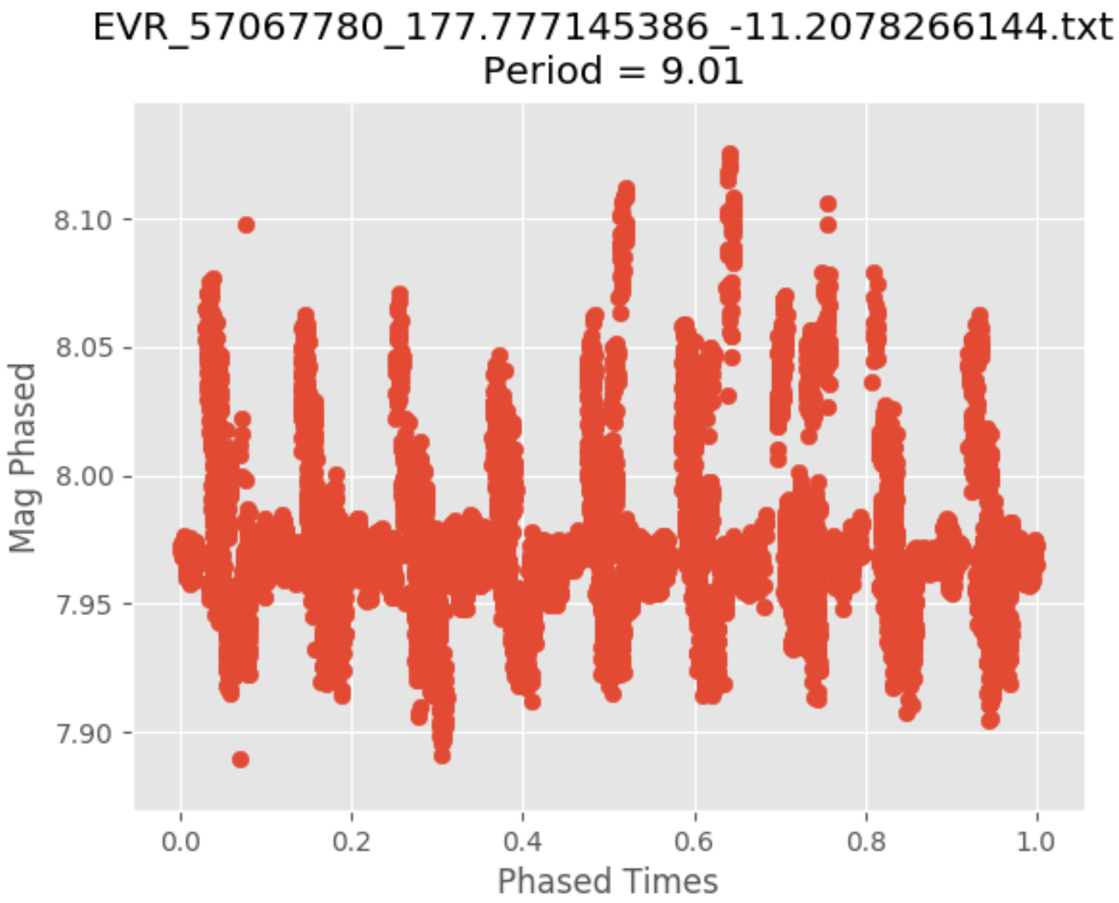


When the BLS was applied to evr_sample images, the following plots were generated: (Note: the light curves after being put through the super smoother are also shown)

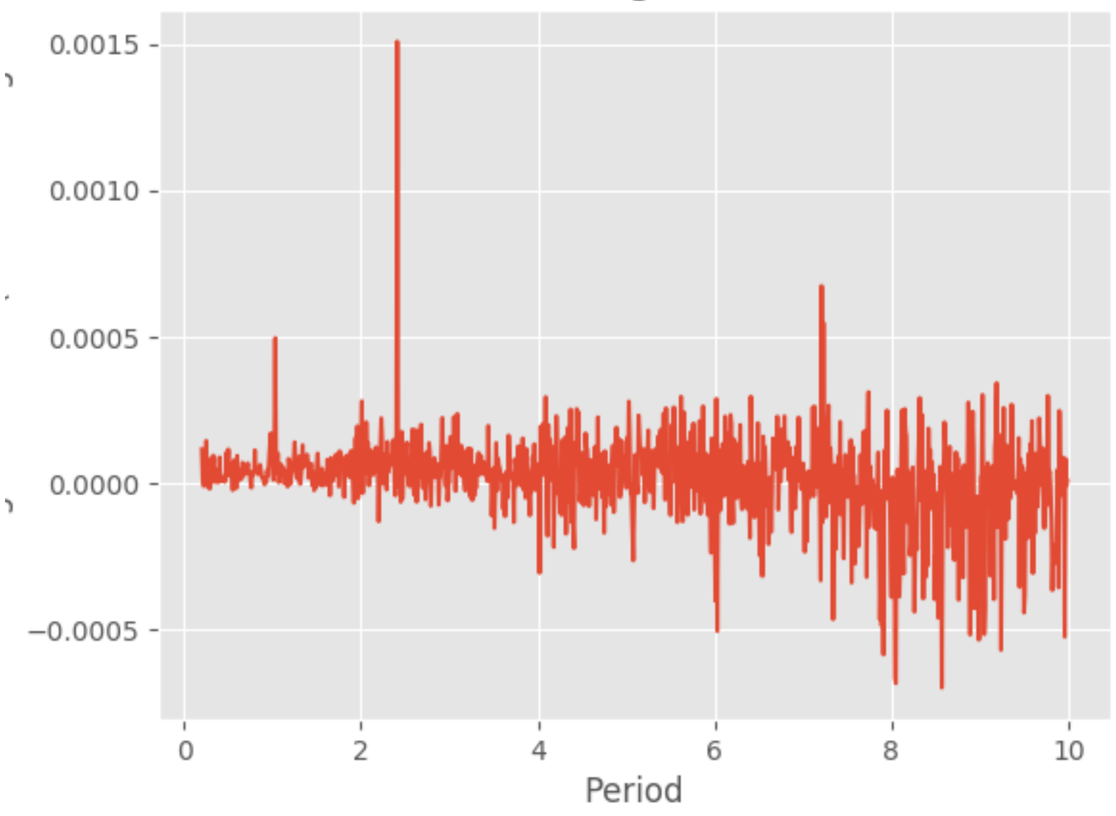

EVR_54850324_177.562911987_-9.99710941315
Using BLS
Maximum Average Bin Value (Dimmest Magnitude)
0.000
0.005
0.010
0.015
0.020
0
2
4
6
8
10
Period


EVR_54850324_177.562911987_-9.99710941315.txt
Period = 6.01
Mag Phased
8.30
8.35
8.40
8.45
8.50
8.55
8.60
8.65
0.0
0.2
0.4
0.6
0.8
1.0
Phased Times

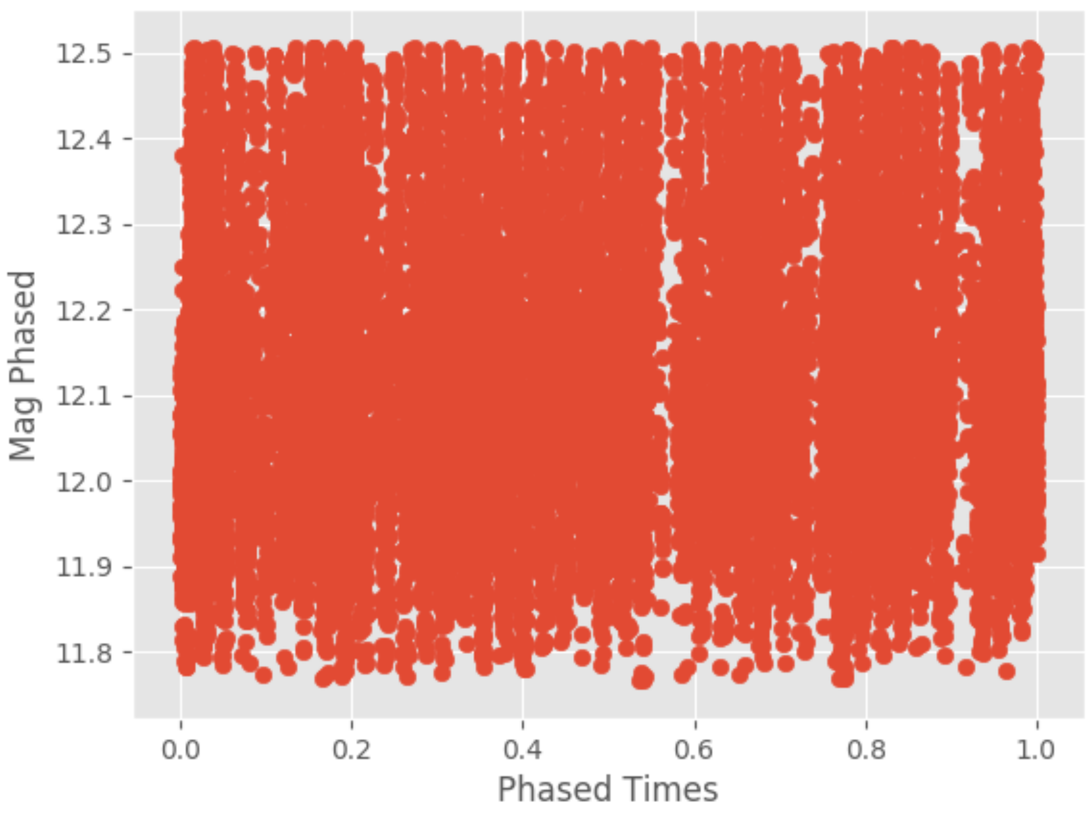

EVR_57065333_176.663772583_-11.1173553467
Using BLS
Maximum Average Bin Value (Dimmest Magnitude)
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
2
4
6
8
10
Period

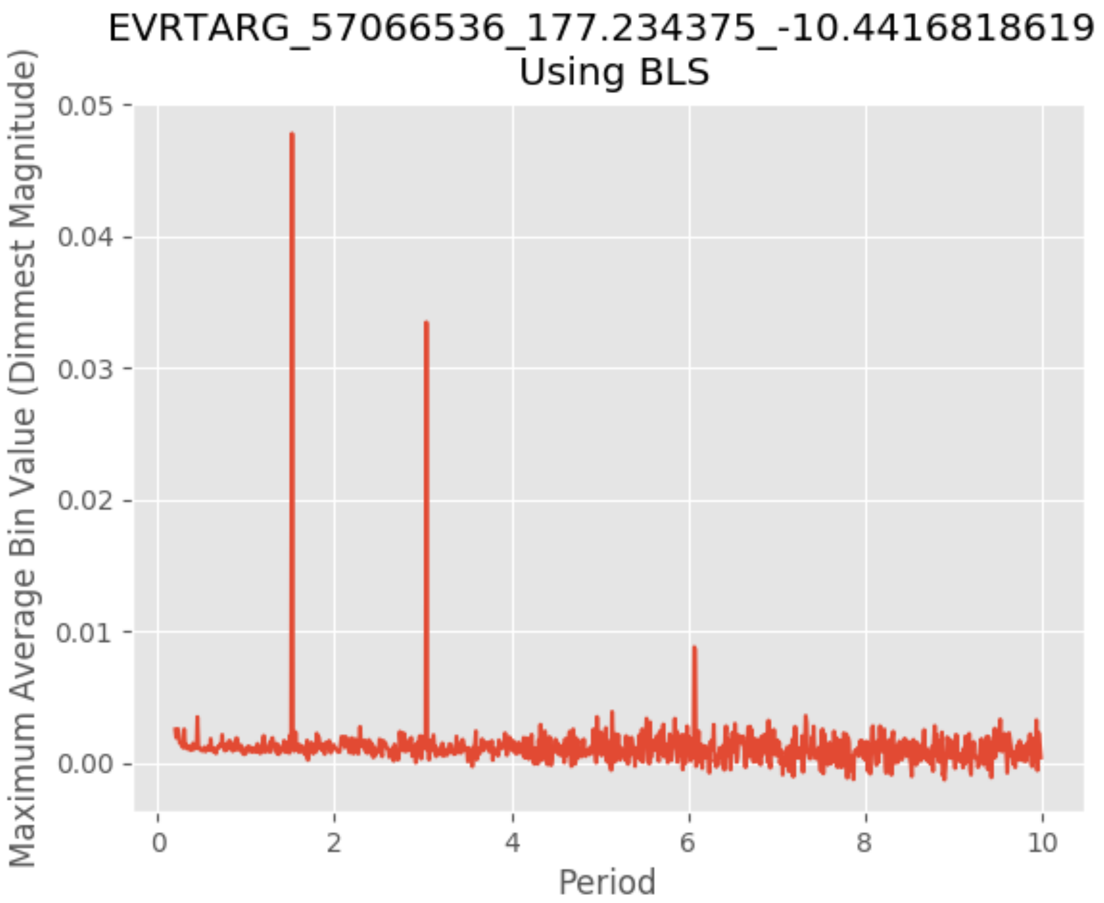

EVR_57067780_177.777145386_-11.2078266144.txt
Period = 9.01
Mag Phased
8.10
8.05
8.00
7.95
7.90
0.0
0.2
0.4
0.6
0.8
1.0
Phased Times

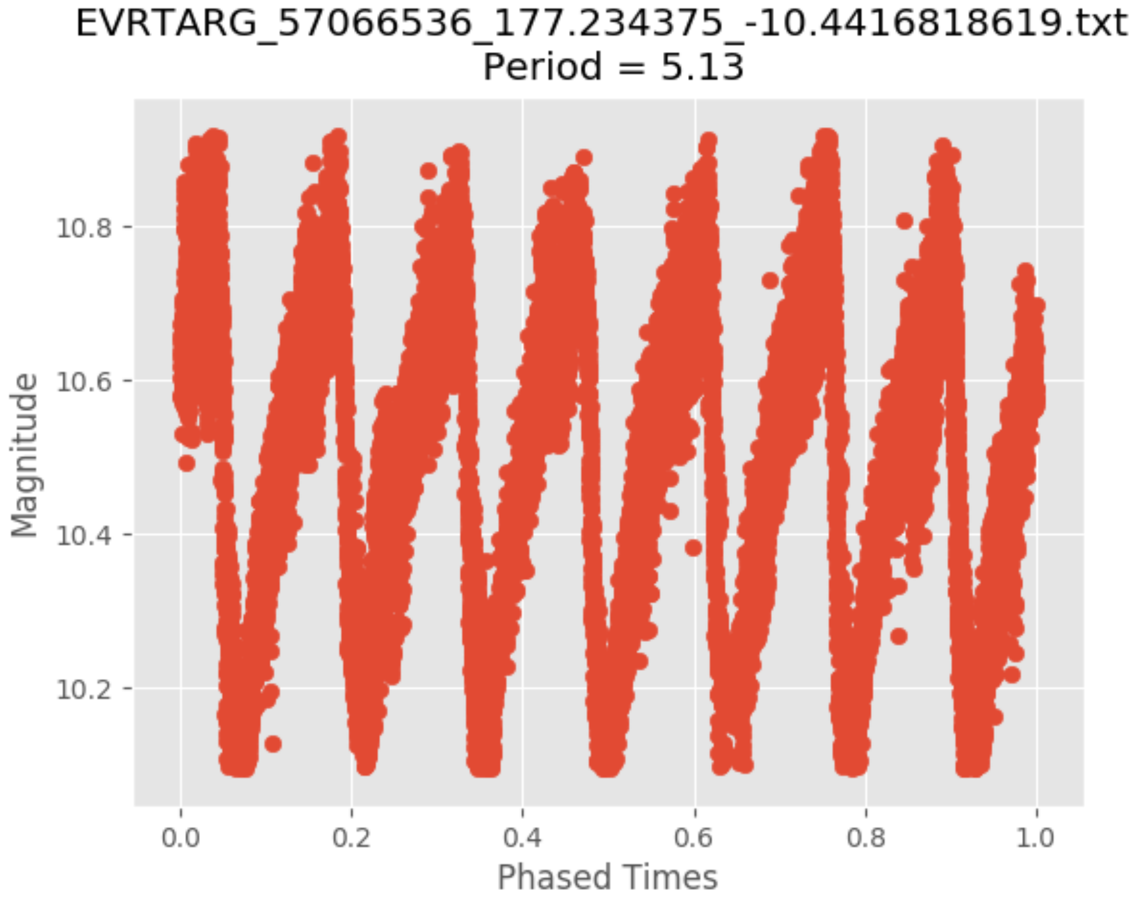
EVR_57067780_177.777145386_-11.2078266144
Using BLS



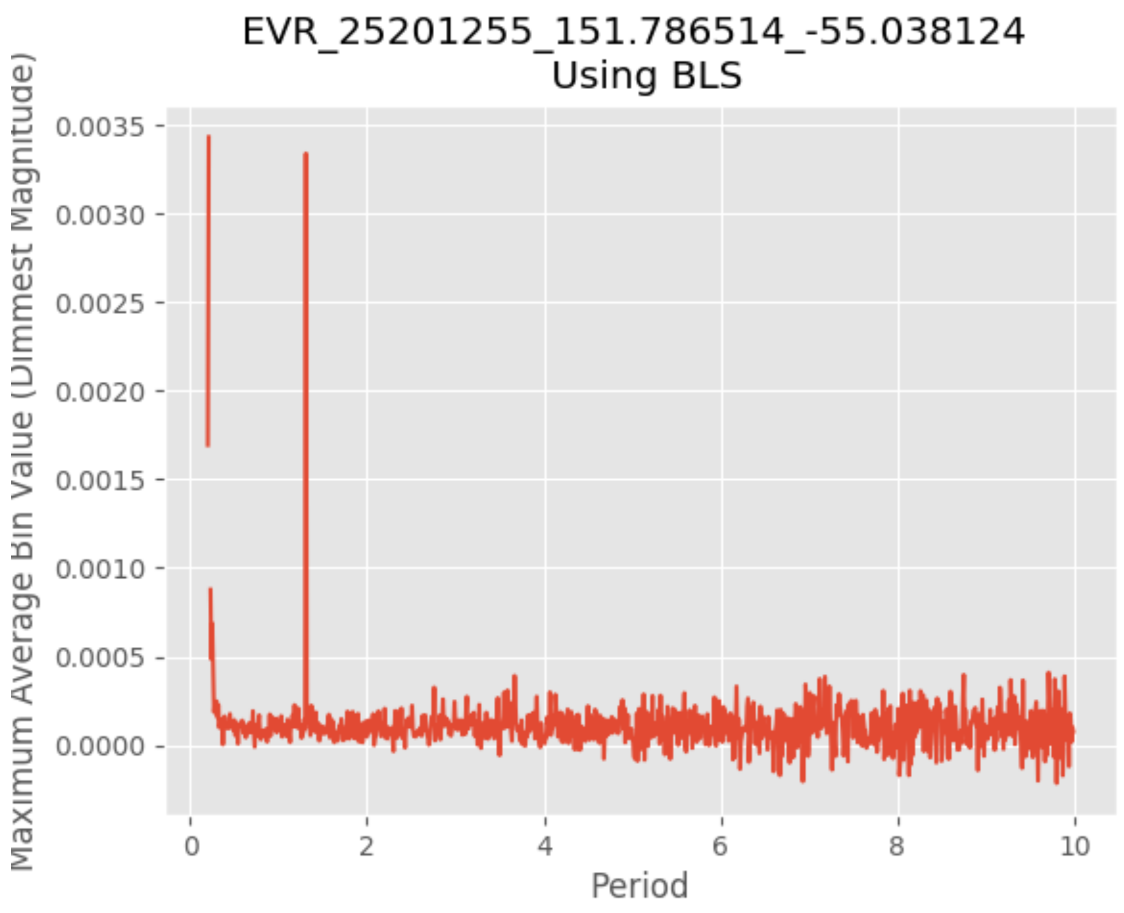
EVR_57065333_176.663772583_-11.1173553467.txt
Period = 5.79

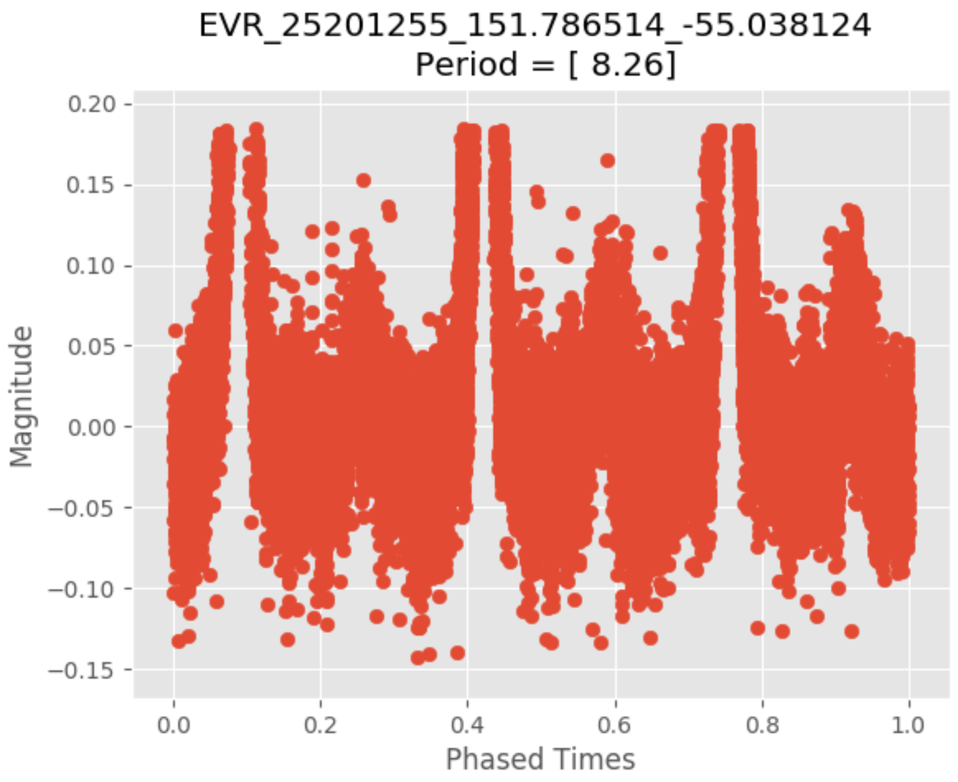


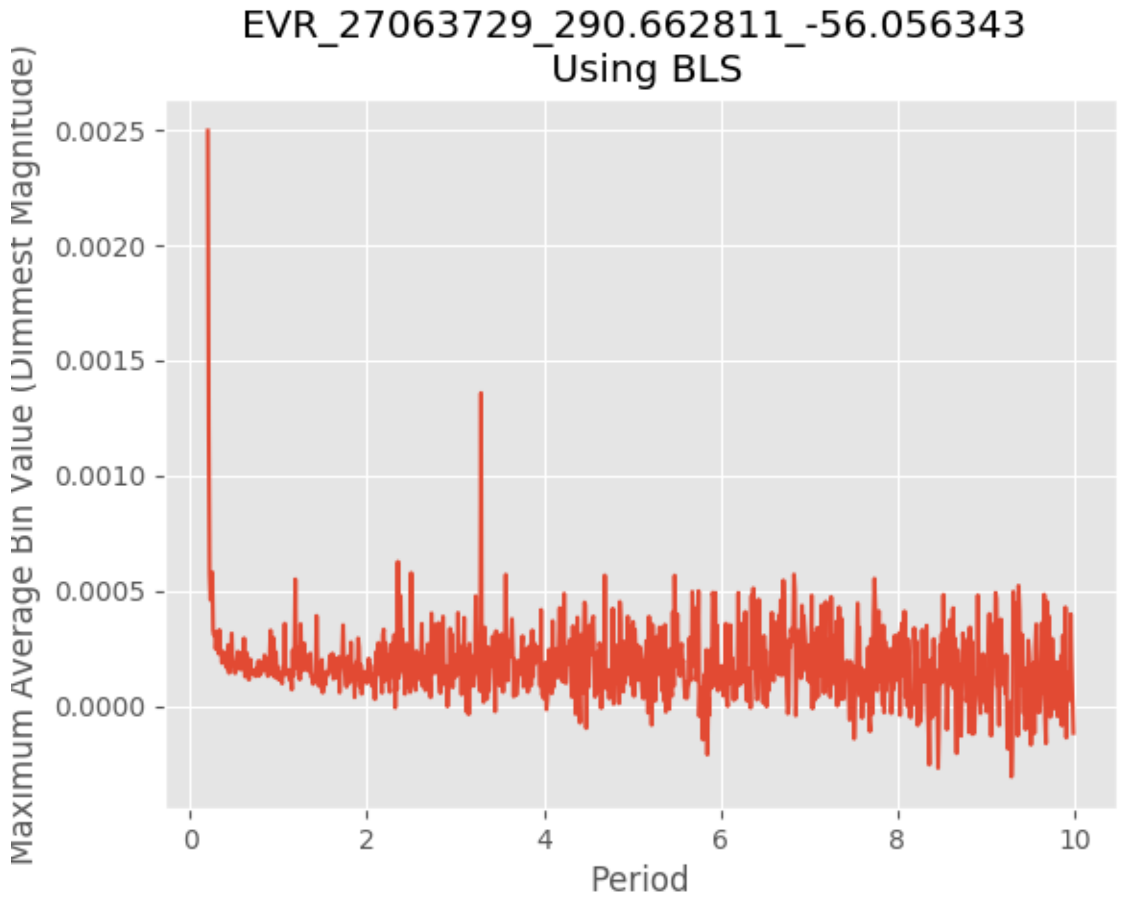

EVRTARG_57066536_177.234375_-10.4416818619
Using BLS
Maximum Average Bin Value (Dimmest Magnitude)
Period
0.05
0.04
0.03
0.02
0.01
0.00
0
2
4
6
8
10

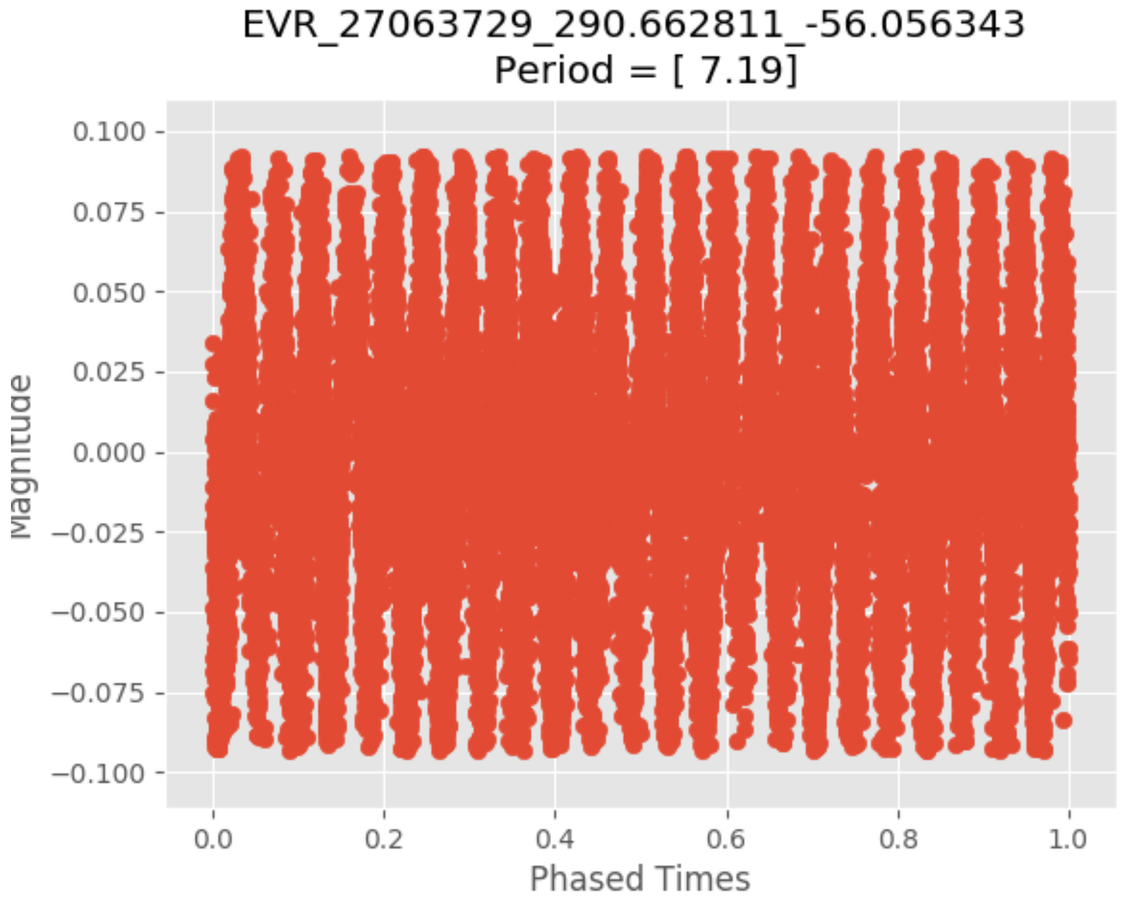

EVRTARG_57066536_177.234375_-10.4416818619.txt
Period = 5.13
Magnitude
Phased Times
10.8
10.6
10.4
10.2
0.0
0.2
0.4
0.6
0.8
1.0

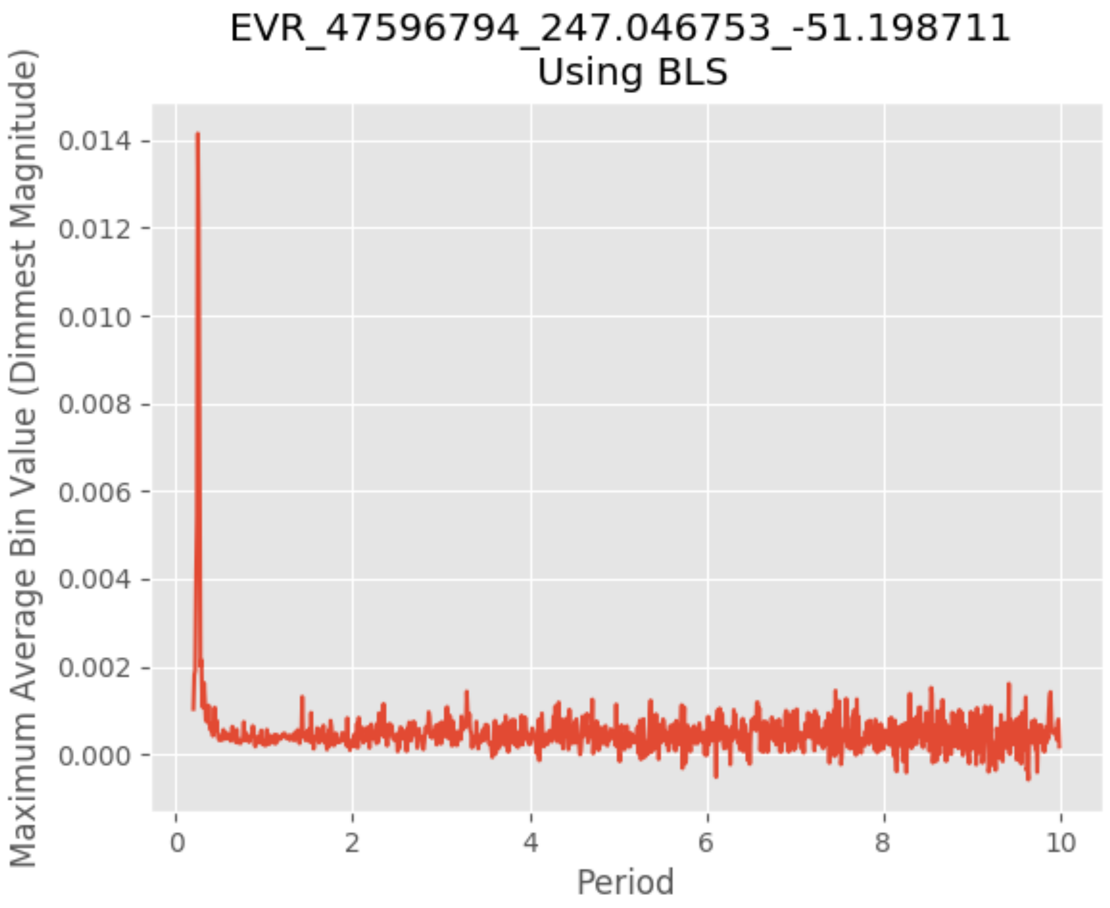
When the BLS was applied to var_example images, the following plots were generated: (Note: the light curves after being put through the super smoother are also shown)

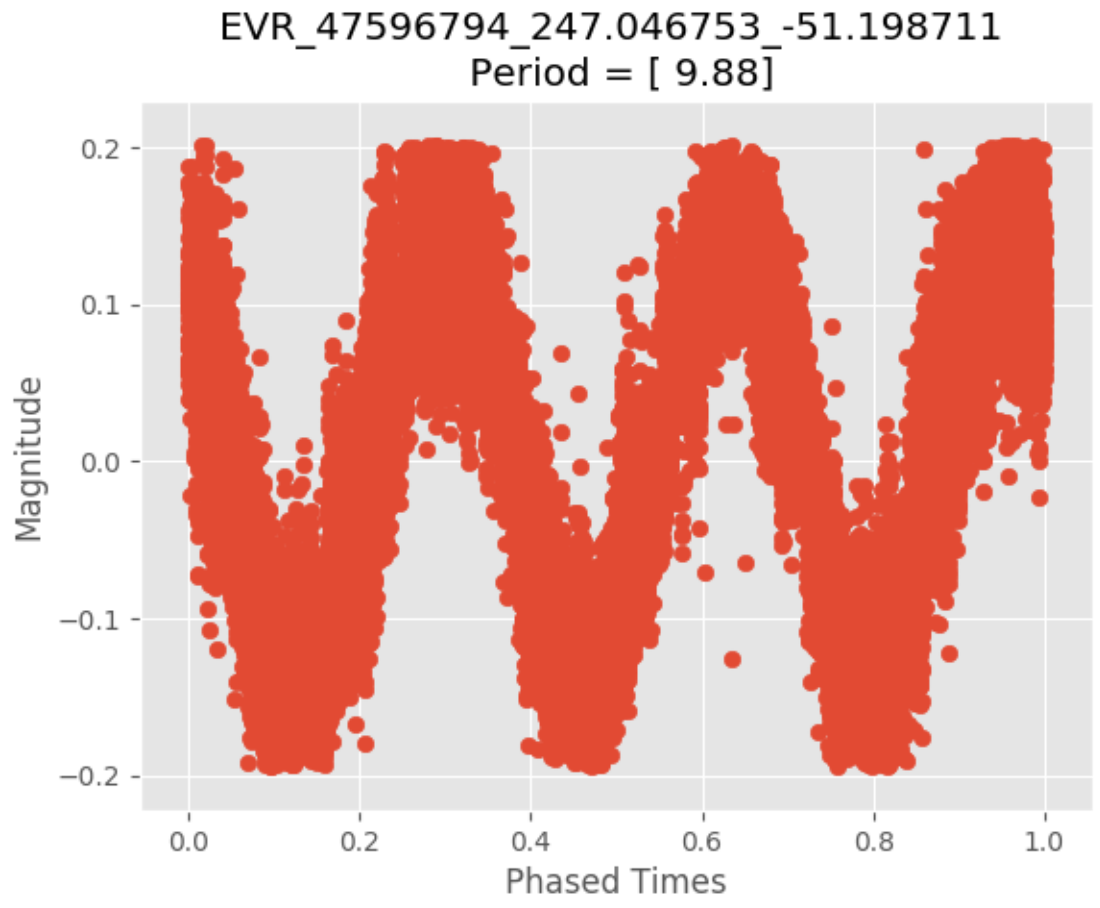


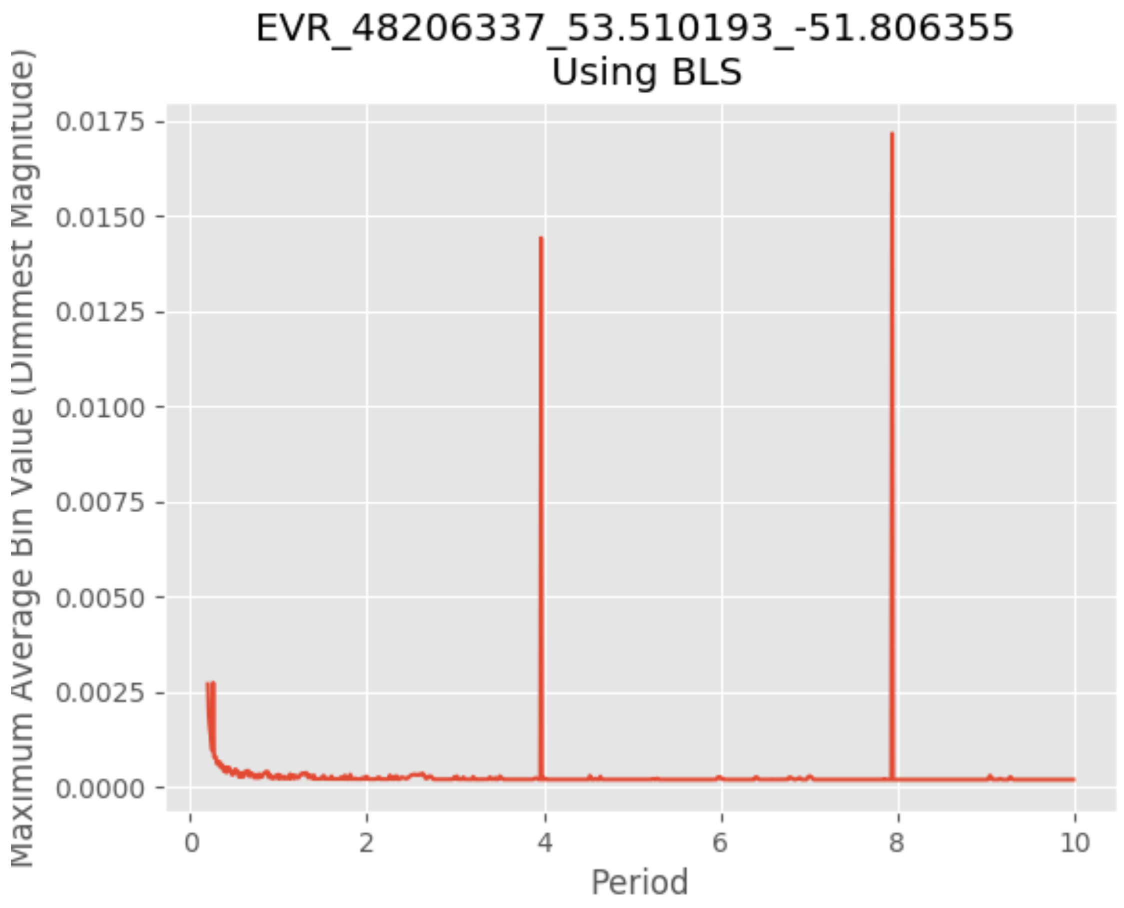


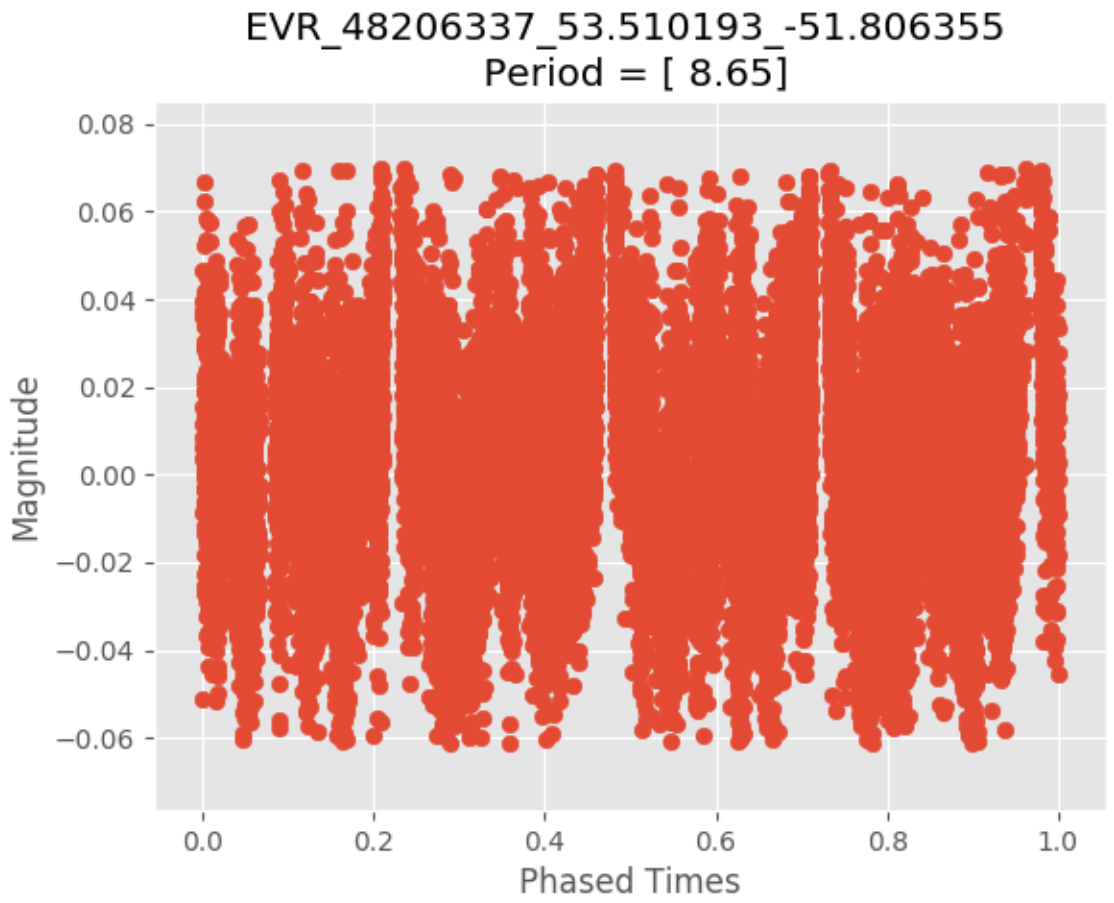

EVR_27063729_290.662811_-56.056343
Using BLS
Maximum Average Bin Value (Dimmest Magnitude)
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
2
4
6
8
10
Period

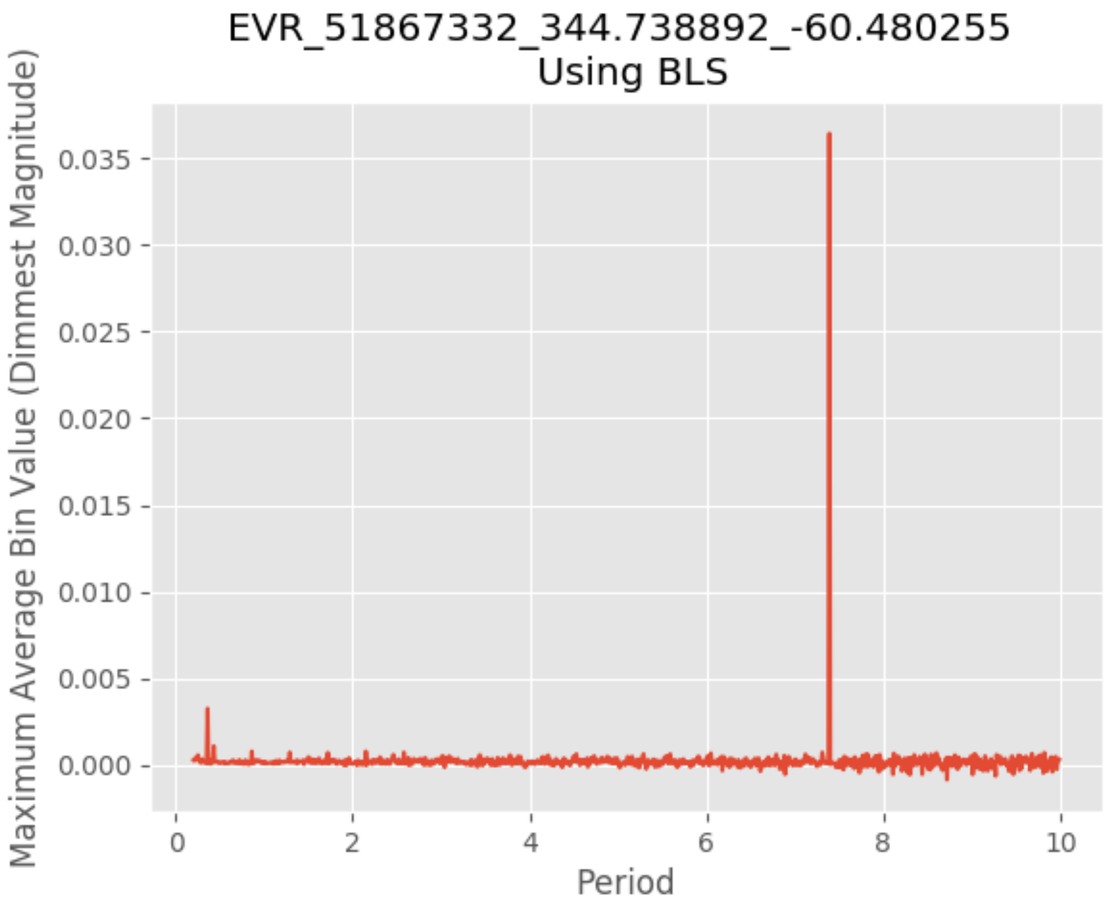

EVR_27063729_290.662811_-56.056343
Period = [ 7.19]
Magnitude
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
0.0
0.2
0.4
0.6
0.8
1.0
Phased Times

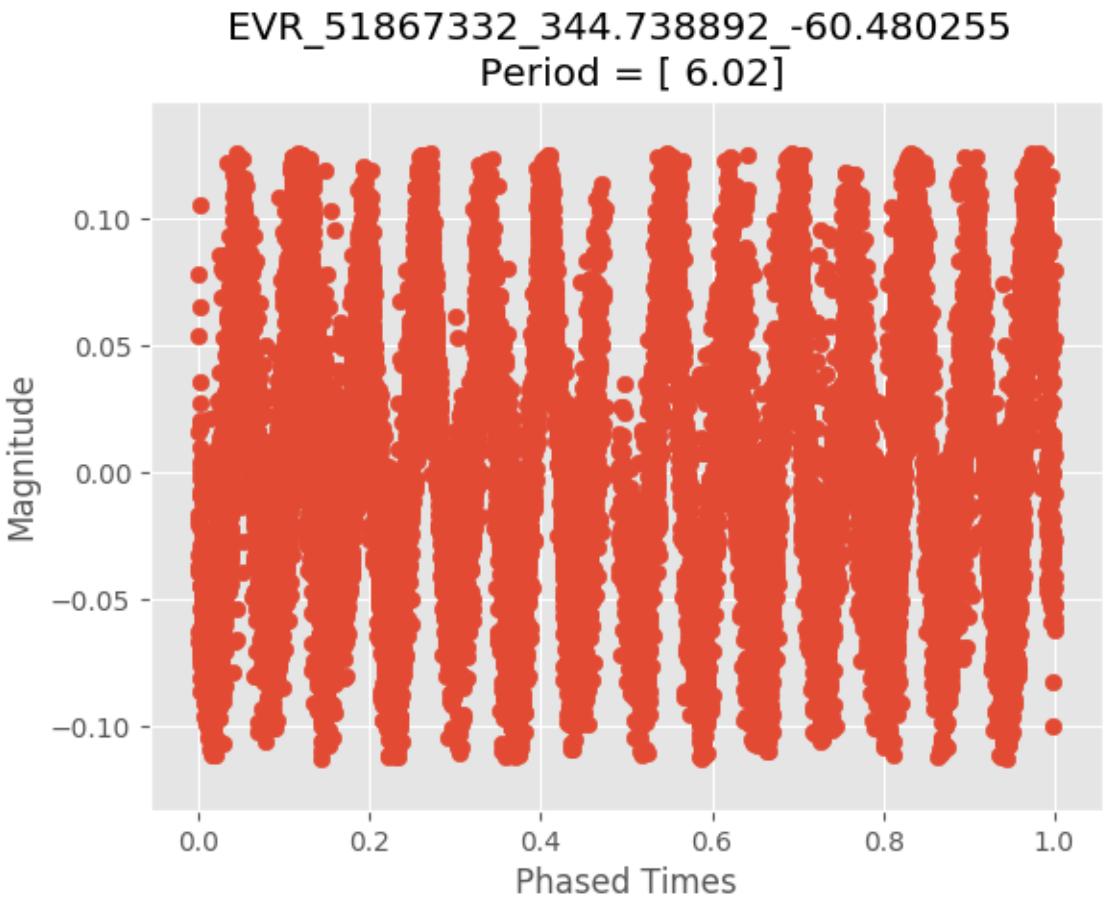

EVR_47596794_247.046753_-51.198711
Using BLS
Maximum Average Bin Value (Dimmest Magnitude)
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
2
4
6
8
10
Period


EVR_47596794_247.046753_-51.198711
Period = [ 9.88]
Magnitude
0.2
0.1
0.0
−0.1
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
Phased Times


EVR_48206337_53.510193_-51.806355
Using BLS
Maximum Average Bin Value (Dimmest Magnitude)
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0
2
4
6
8
10
Period


EVR_48206337_53.510193_-51.806355
Period = [ 8.65]
Magnitude
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
0.0
0.2
0.4
0.6
0.8
1.0
Phased Times


EVR_51867332_344.738892_-60.480255
Using BLS
Maximum Average Bin Value (Dimmest Magnitude)
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
2
4
6
8
10
Period


EVR_51867332_344.738892_-60.480255
Period = [ 6.02]
Magnitude
0.10
0.05
0.00
−0.05
−0.10
0.0
0.2
0.4
0.6
0.8
1.0
Phased Times

**2/25/19:**

First, a function that directly measures the SNR was scripted. This is shown below:

```
def direct_measurement(period_arange, BLS_bins):
    DIRECT_DEBUG = 0
    stepsize = period_arange[1]-period_arange[0]

    peaks_sort_ind = np.argsort(BLS_bins)
    max_peak_ind = peaks_sort_ind[-1]
    max_peak = BLS_bins[max_peak_ind]

    # Getting local region of bin values
    max_peak_period = period_arange[peaks_sort_ind][0]
    period_limit = max_peak_period * 0.1
    period_arange_cut = np.arange(max_peak_period - period_limit, max_peak_period +
period_limit,stepsize)
    if len(period_arange_cut) % 2 != 0:
        period_arange_cut = period_arange[(max_peak_ind - len(period_arange_cut)//2): max_peak_ind
+ len(period_arange_cut)//2]
    BLS_bins_cut = BLS_bins[(max_peak_ind - len(period_arange_cut)//2): max_peak_ind +
len(period_arange_cut)//2]

    if DIRECT_DEBUG:
        print("len bins = ", len(BLS_bins_cut))
        print("len period = ", len(period_arange_cut))


    if (max_peak_ind - len(period_arange_cut)//2) < 0:
        period_arange_cut = period_arange[:max_peak_ind + len(period_arange_cut)//2]
        BLS_bins_cut      = BLS_bins[     :max_peak_ind + len(period_arange_cut)//2]

    if DIRECT_DEBUG:
        print("len bins = ", len(BLS_bins_cut))
        print("len period = ", len(period_arange_cut))
        plt.plot(period_arange_cut,BLS_bins_cut)
        plt.show()
    local_rms = np.sqrt(np.mean(BLS_bins_cut))
    local_avg = np.mean(BLS_bins_cut)

    peak_snr = (max_peak - local_avg)/local_rms
    print(peak_snr)
    return peak_snr
```

In this function, a peak was selected through:

```
    max_peak_ind = peaks_sort_ind[-1]
    max_peak = BLS_bins[max_peak_ind]
```

The peak was arbitrarily chosen to be the maximum peak. This could be changed to the second or third highest peak through using [-1] or [-2], respectively. Then, the local region around the peak (10% on either side) in the periodgram was extracted with:

```
max_peak_period = period_arange[peaks_sort_ind][0]
period_limit = max_peak_period * 0.1
```

```
    period_arange_cut = np.arange(max_peak_period - period_limit, max_peak_period +
period_limit,stepsize)
    if len(period_arange_cut) % 2 != 0:
        period_arange_cut = period_arange[(max_peak_ind - len(period_arange_cut)//2): max_peak_ind
+ len(period_arange_cut)//2]
    BLS_bins_cut = BLS_bins[(max_peak_ind - len(period_arange_cut)//2): max_peak_ind +
len(period_arange_cut)//2]

    if DIRECT_DEBUG:
        print("len bins = ", len(BLS_bins_cut))
        print("len period = ", len(period_arange_cut))


    if (max_peak_ind - len(period_arange_cut)//2) < 0:
        period_arange_cut = period_arange[:max_peak_ind + len(period_arange_cut)//2]
        BLS_bins_cut      = BLS_bins[     :max_peak_ind + len(period_arange_cut)//2]
```

In order to make the local period arrange (period_arange_cut) and the local bin values (BLS_bins_cut) have the same length, the local region had to be found differently if the amount of periods in period_arange_cut was odd or even. This is because if period_arange_cut has a length of 9, half this array is 4.5 and int(4.5) is 4 but we want to have 5 so that we can add 5 values onto either side of the peak.  If the length of the periods 10% on either side of the peak was even, then

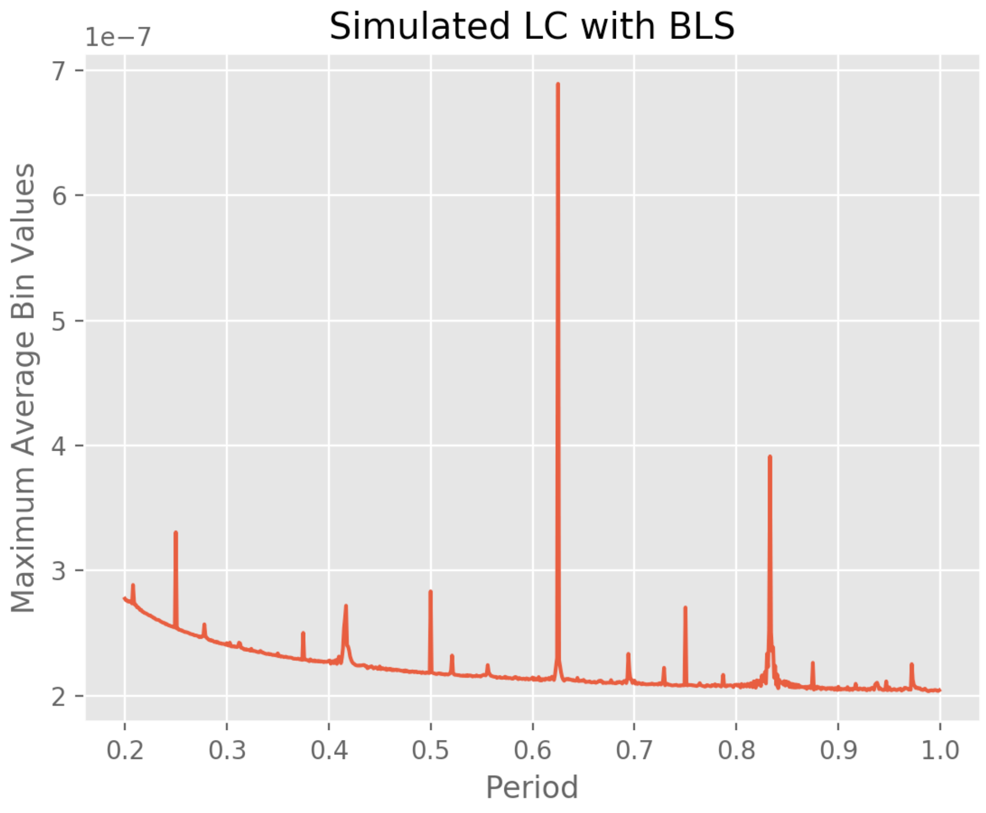
```
period_arange_cut = np.arange(max_peak_period - period_limit, max_peak_period +
period_limit,stepsize)
```

Otherwise,

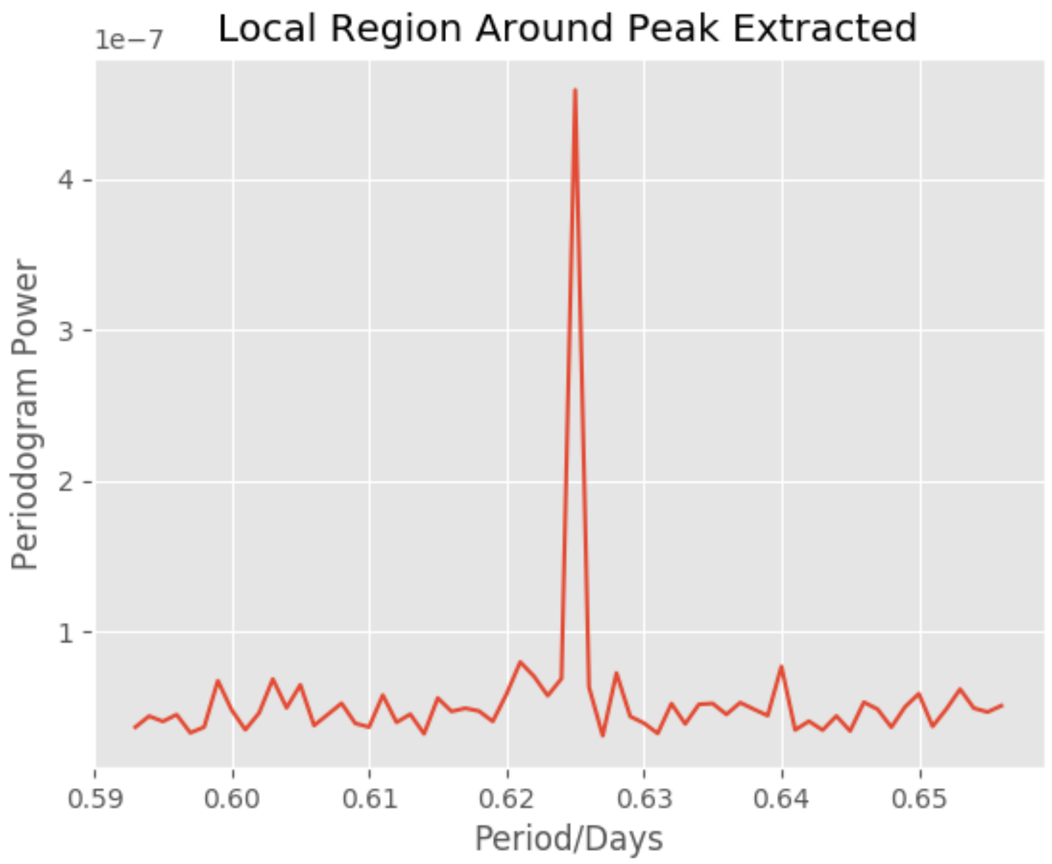
```
if len(period_arange_cut) % 2 != 0:
        period_arange_cut = period_arange[(max_peak_ind - len(period_arange_cut)//2): max_peak_ind
+ len(period_arange_cut)//2]
```

However, period_arange_cut was just used to validate that the function was working. Therefore, this function can be simplified for future uses.

In order to validate the function was working properly, the periodogram using the BLS method was plotted:



and then the local region around the peak was plotted:

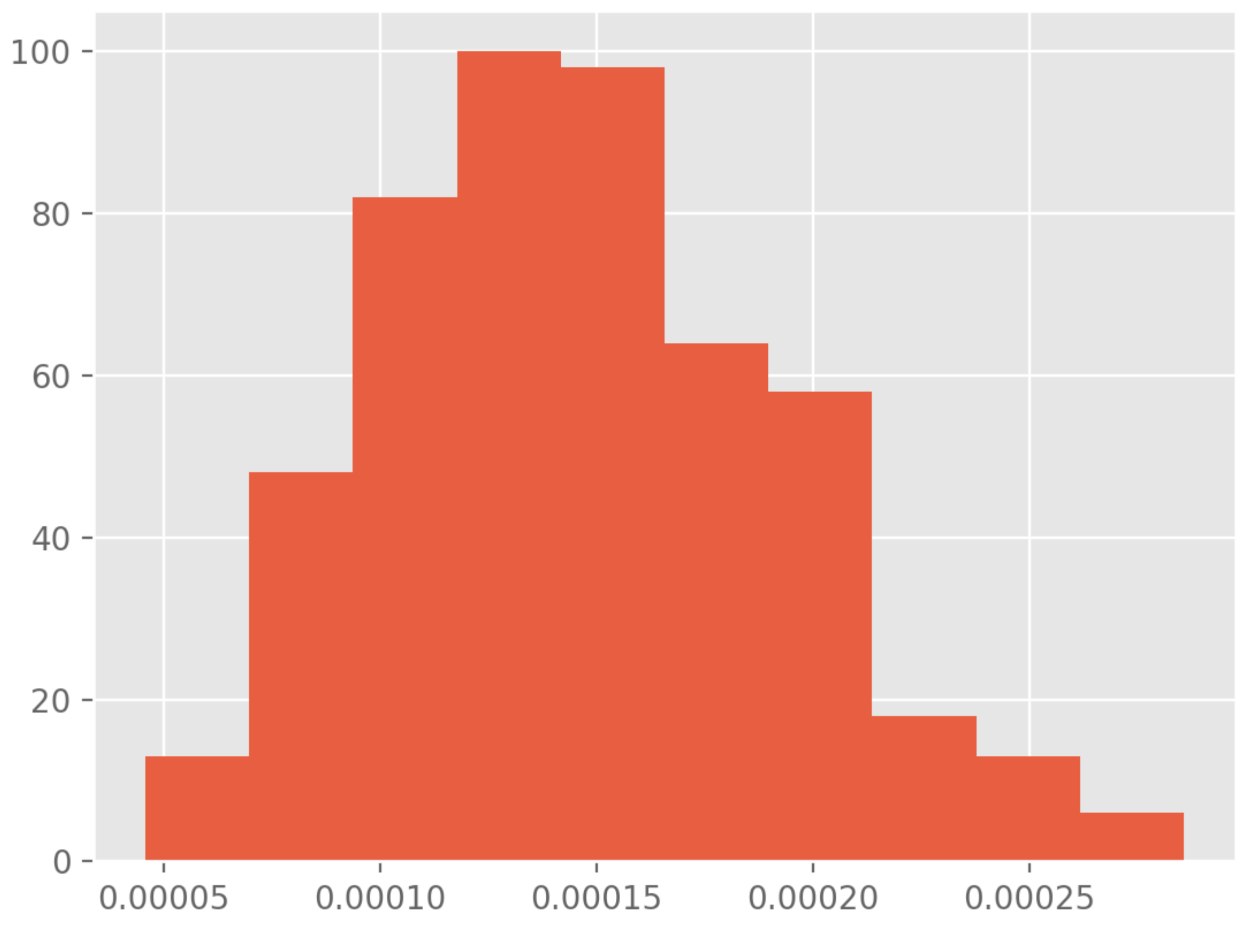


After the correct local region was verified, the local rms and local average was calculated in order to calculate the snr of peak. The SNR found was `0.000104776222181`.

To test the probability of this snr value being due to noise, Monte Carlo simulations were used:

```
def monte_carlo_simulations(epoch_arr, mag_arr):
    NUM_TRIALS    = 500
    PERIOD_MIN    = 0.2
    PERIOD_MAX    = 3
    STEPSIZE      = 0.01
    SNR_THRESHOLD = 0.05

    # Original Result for non-shuffled mag_arr
    sim_period_arange, sim_best_bins = 
BLS(epoch_arr,mag_arr,period_min=PERIOD_MIN,period_max=PERIOD_MAX,stepsize=STEPSIZE)
    peak_snr = direct_measurement(sim_period_arange,sim_best_bins)
    #pdb.set_trace()
    reproduced_results     = []
    #non_reproduced_results = []
    for i in range(NUM_TRIALS):
        mag_shuffled = np.array(random.sample(list(mag_arr), len(mag_arr)))
        _, sim_best_bins_shuffled = 
BLS(epoch_arr,mag_shuffled,period_min=PERIOD_MIN,period_max=PERIOD_MAX,stepsize=STEPSIZE)
        shuffle_peak_snr = direct_measurement(sim_period_arange,sim_best_bins_shuffled)
        reproduced_results.append(shuffle_peak_snr)
        #pdb.set_trace()
        #if peak_snr < shuffle_peak_snr * (1 + SNR_THRESHOLD) and peak_snr > shuffle_peak_snr * (1
- SNR_THRESHOLD):

        #reproduced_results.append()
        #non_reproduced_results.append(0)
        #else:
        #non_reproduced_results.append(1)
        #reproduced_results.append(0)
        if i % 50 == 0:
            print("%s Complete" % (i/NUM_TRIALS * 100))
    plt.hist(reproduced_results)
    plt.show()
```

Here, the mag_arr of simulated data was shuffled 500 times. The BLS method and direct measurement function shown above were run on each shuffled array. The histogram of output SNR values in shown below:



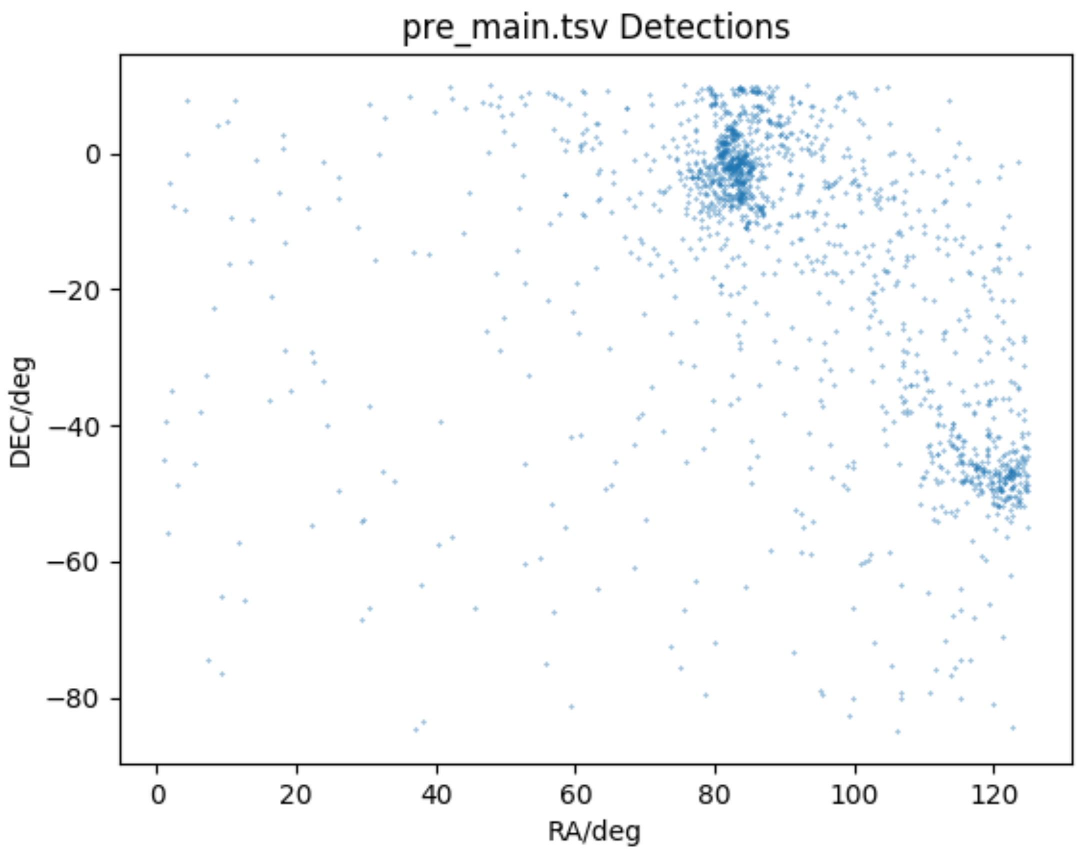
(I apologize for lack of title and axis labels). This histogram shows that the SNR value of `0.000104776222181` was not the most probable SNR value when the arrays were shuffled. Instead, the most probable value appears to be more around 0.00013. Therefore the SNR value found above was most likely due to noise.

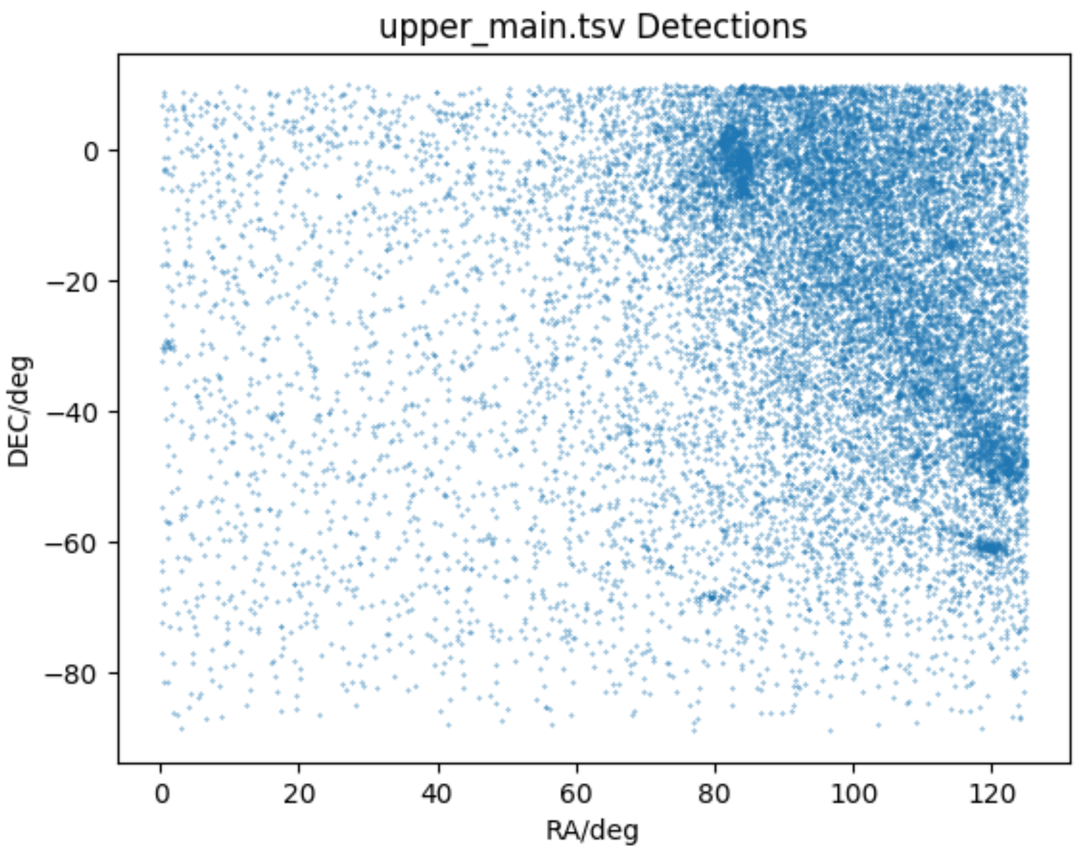
This histogram took about an hour or two to run so this function must be optimized.

**3/6/18:**

With the following code, the following plots were generated for the upper main sequence stars and pre main sequence stars from an RA of 0-125 degrees and then the RA and DEC values were exported as csv files:

```
def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        return False

def get_data(filename):

    ra_lst,dec_lst = [],[]
    with open(filename,"r") as f:
        for line in f:

            first_isnum_bool = is_number(line[0])
            if first_isnum_bool:

                    line     = line.split(";")
                    line[-1] = line[-1].strip("\n")

                    last_isnum_bool = is_number(line[-1])
                    if last_isnum_bool:
                        ra_val  = float(line[0])
                        dec_val = float(line[1])
                        if ra_val < 125.:
                            ra_lst.append(ra_val)
                            dec_lst.append(dec_val)

    return ra_lst, dec_lst

def create_dataframe(ra_arr,dec_arr,filename):
    df = pd.DataFrame(OrderedDict( (('RA',pd.Series(ra_arr)), ('DEC',pd.Series(dec_arr)) )))
    filename=filename.split(".tsv")[0]
    print(filename)
    df.to_csv("./%s_ra_dec_coords.csv" % filename, index=False)
    return df



if __name__ == "__main__":
    parser = argparse.ArgumentParser(formatter_class=RawTextHelpFormatter)
    parser.add_argument("file", type=str,
                        help = "data to be evaluated: pre_main.tsv or upper_main.tsv")
    args     = parser.parse_args()
    filename = args.file


    ra_lst, dec_lst = get_data(filename)

    plt.figure(1)
    plt.scatter(ra_lst,dec_lst, s = 0.1)
    plt.title("%s Detections" % filename)
    plt.ylabel("DEC/deg")
    plt.xlabel("RA/deg")
    plt.savefig("%s_0_to_125_deg_RA.png" % filename)
    plt.show()

    df= create_dataframe(ra_lst,dec_lst,filename)
    print(df)
```





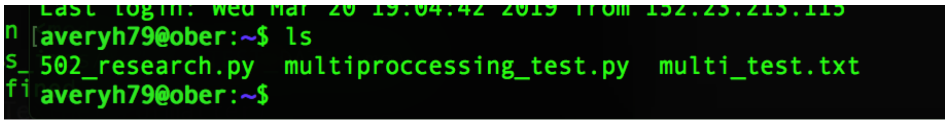
Using the coordinates on the plots above and then using Vizier, the following clusters were found:

| RA/deg | DEC/deg | Radius/deg |
|---|---|---|
| 85.43 | -1.842 | 0.3 |
| 114.15 | -14.05 | 0.4 |
| 84.6 | -67.5 | 0.5 |
| 47.704 | -57.766 | 0.1 |
| 119.53 | -61.16 | 0.5 |
| 116.288 | 38.015 | 0.7 |
| 118.042 | -38.53 | 0.8 |
| 86.066 | -62.785 | 0.1 |
| 80.626 | -70.155 | 0.1 |
| 122.479 | -49.205 | 0.3 |
| 1.029 | -19.833 | 1 |
| 125.06 | -37.354 | 0.5 |
| 83.01 | -3.04 | 0.5 |
| 78.99 | 3.043 | 0.8 |
| 79.38 | 4.25 | 0.6 |
| 124.75 | -47.47 | 0.5 |
| 81.75 | -4 | 0.9 |

**3/25/2019: MULTIPROCESSING**

This week I was able to ssh into the server using: scp script.py averyh79@152.2.38.186:

And I was able to upload files to the server:



```
Last login: Wed Mar 20 19:04:42 2019 from 152.23.213.115
averyh79@ober:~$ ls
502_research.py  multiproccessing_test.py  multi_test.txt
averyh79@ober:~$
```

I was able to get the multiprocessing working both locally and on the server. The following results were obtained:

```
Doing something 0
Doing something 10
Doing something 30
Doing something 40
Doing something 20
Doing something 11
Doing something 1
Doing something 41
Doing something 31
Doing something 21
Doing something 12
Doing something 2
Doing something 22
Doing something 42

Doing something 32
etc
```

Note how the parameters are not in order. The parameters were output to a .txt file which is shown below:

0

30

40

10

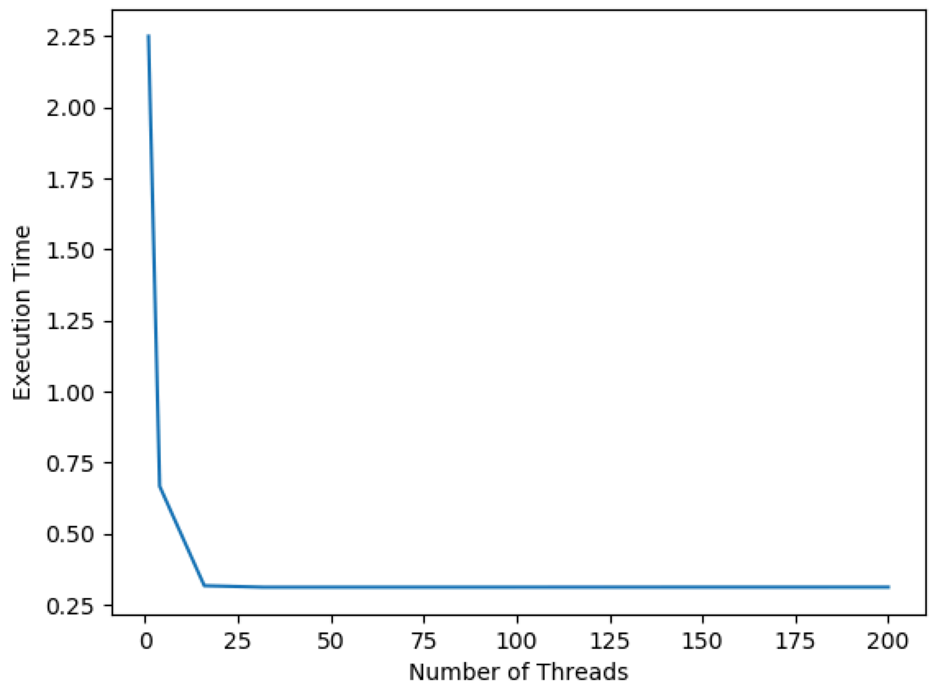
20

41

31

52…etc.

The execution time vs. number of threads plot is shown below:



The multiprocessing was then applied to my supersmoother & BLS code, which was very tricky. I had to write a new main function and call my old one “do_work”. My new main function is shown below:

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(formatter_class=RawTextHelpFormatter)
    parser.add_argument("data", type=str,
                            help = "directory to data .txt files or single .txt file to be
evaluated: ex) bls_test_lc.txt or evr_samples")
    parser.add_argument("-m", type=str, nargs = "+",
                            help = "method, ex) '-m supersmoother bls' or just '-m supersmoother'
or '-m bls'  ")
    parser.add_argument("-snr", type=str, nargs = "+", action ='store',
                            help = "method, ex) '-snr direct mc' or just '-m direct' or '-m mc'  ")
    args   = parser.parse_args()

    if not os.path.isdir(args.data):
        filename  = args.data
        filelist = glob.glob('./%s' % filename)
        TEST_DATA = ((filelist[0].split('.txt')[0]).split('/')[1]).split('./')[-1]

        if filename   == "bls_test_lc.txt" or filename == "supersmoother_test_lc.txt":
            TEST_DATA       = "debug_imgs"

    else:
        filelist  = glob.glob('./%s/*.txt' % args.data)
        TEST_DATA = ((filelist[0].split('.txt')[0]).split('/')[1]).split('./')[-1]


    pool = multiprocessing.Pool()
    func = partial(do_work,TEST_DATA,args)
    pool.map(func,filelist)
    pool.close()
    pool.join()
```

This is a little different than the example shown in class – I had to import

```
from functools import partial
```

because my do work function takes multiple parameters (TEST_DATA,args), not just one like the example shown in class. My file was uploaded to the server and tested.

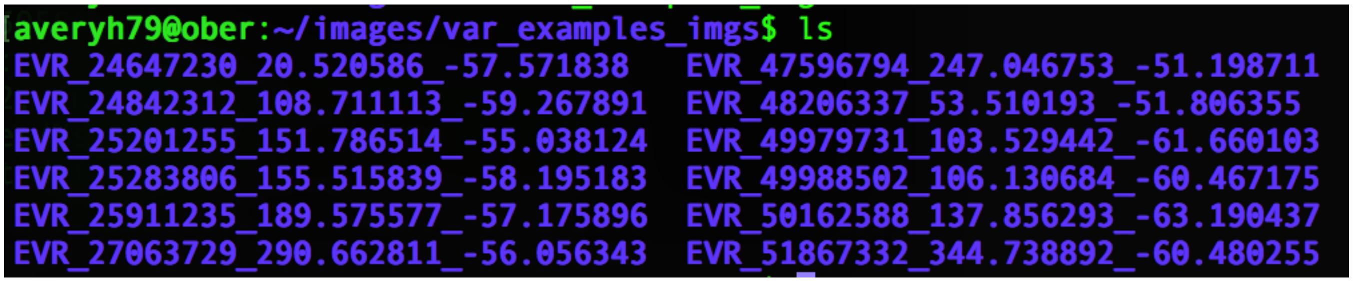
Another way to do this is using starmap:

```
    pool = multiprocessing.Pool()
    input_list = []
    for filepath in filelist:
        input_list.append([TEST_DATA,args,filepath])
    print(input_list)
    #func = partial(do_work,TEST_DATA,args)
    pool.starmap(do_work,input_list)
    pool.close()
    pool.join()
```

This was tested remotely using var_examples and was successful. The var_examples were uploaded using:

```
scp -r var_examples averyh79@152.2.38.186:
```

PNG images were saved to the server:

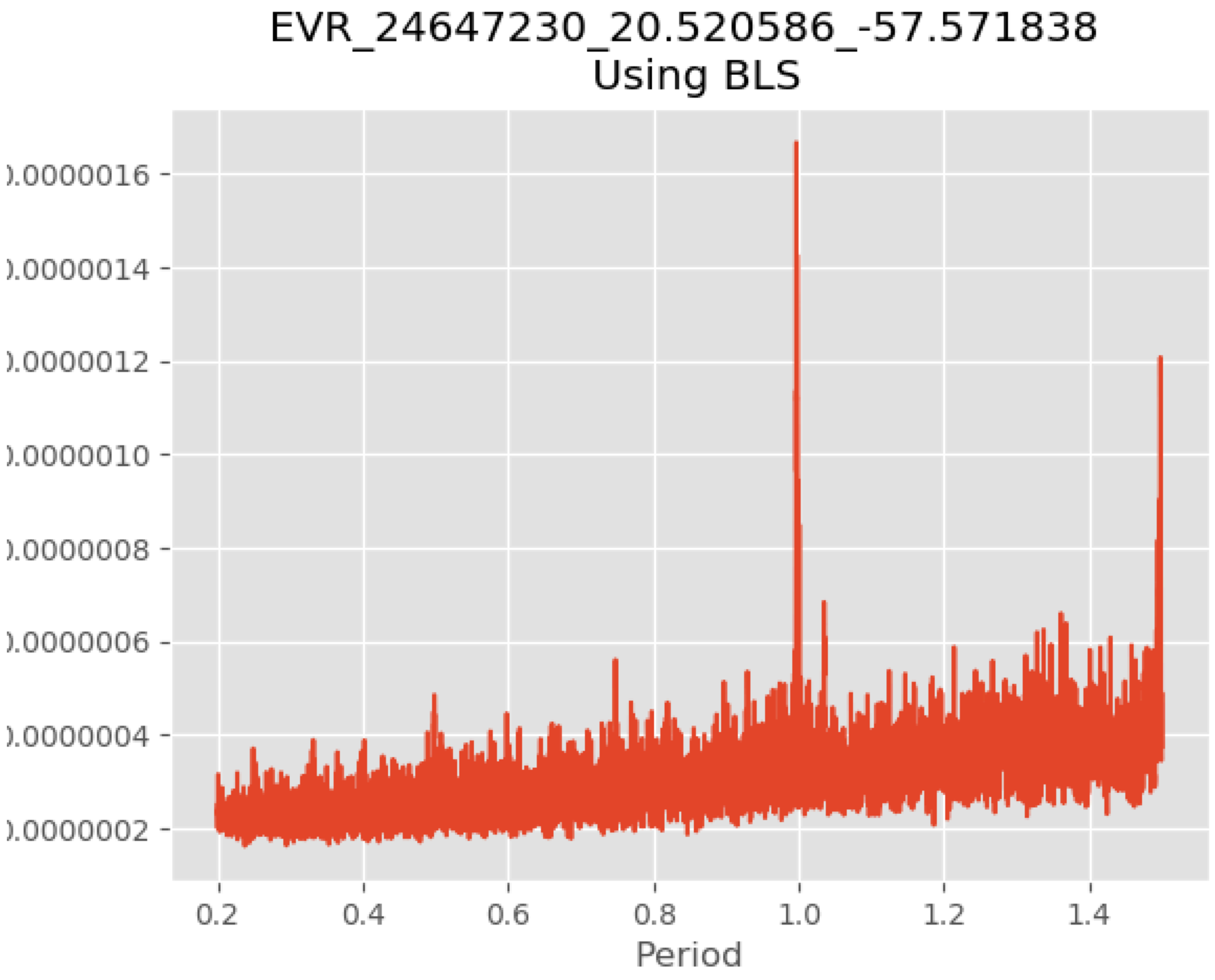


```
averyh79@ober:~/images/var_examples_imgs$ ls
EVR_24647230_20.520586_-57.571838   EVR_47596794_247.046753_-51.198711
EVR_24842312_108.711113_-59.267891  EVR_48206337_53.510193_-51.806355
EVR_25201255_151.786514_-55.038124  EVR_49979731_103.529442_-61.660103
EVR_25283806_155.515839_-58.195183  EVR_49988502_106.130684_-60.467175
EVR_25911235_189.575577_-57.175896  EVR_50162588_137.856293_-63.190437
EVR_27063729_290.662811_-56.056343  EVR_51867332_344.738892_-60.480255
```

Using

```
averyh79@ober:~/images/var_examples_imgs$ display
EVR_24647230_20.520586_-57.571838/BLS_EVR_24647230_20.520586_-
57.571838.png
```

The following image was shown:



The metrics were also output to a csv file using:

```
csv_params = {"Type":None,"Star": None, "RMS" : None, "Median Mag": None, "SS Best Metric" : None,
"SNR at Peak": None, "Peak Period": None }

csv_params['Type'] = img.split("_")[0]
csv_params['Star'] = img.split("_")[1]

csv_params["RMS"]         = np.std(mag)
csv_params["Median Mag"] = np.median(mag)
csv_params["Peak Period"] = peak_period
csv_params["SNR at Peak"] = peak_snr
csv_params['SS Best Metric'] = test_period
```

```
with open("./star_data.csv", "wb") as f:
            w = csv.DictWriter(f,csv_params.keys())
            w.writeheader()
            w.writerow(csv_params)
```

**4/1/2019:**

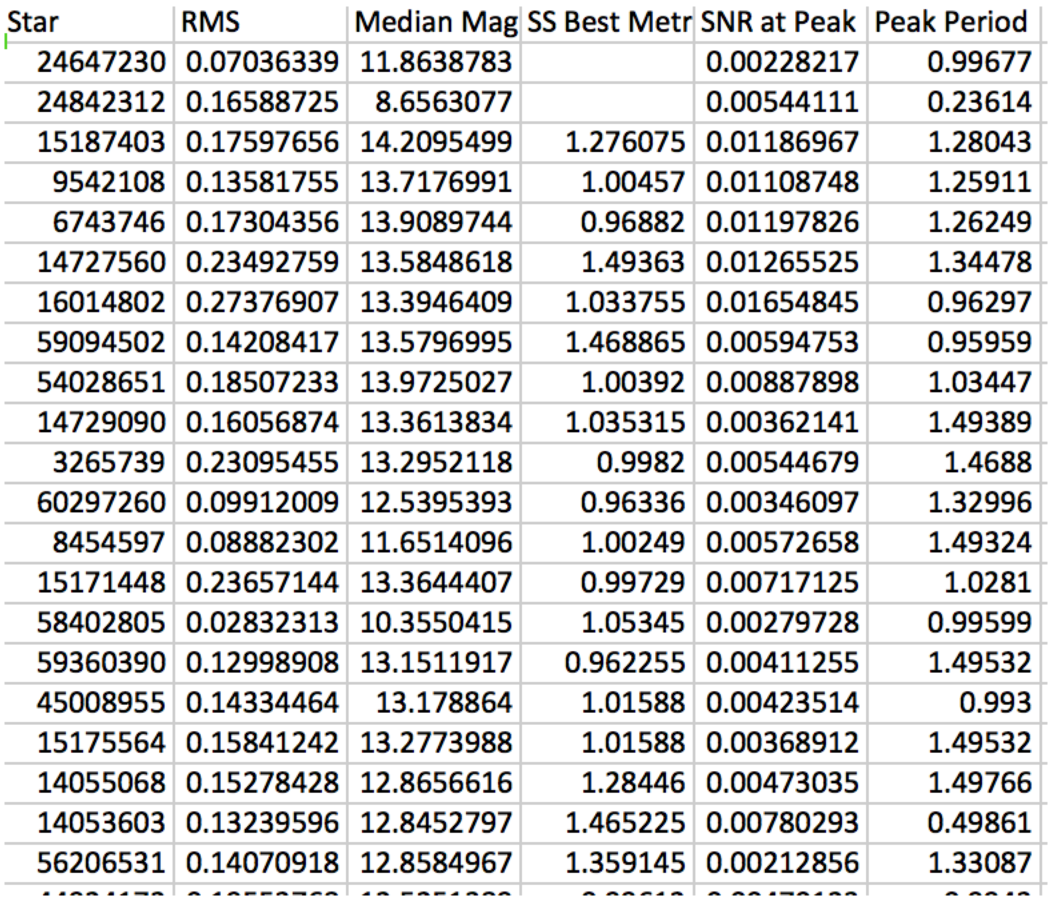
This week the code was run on 2000 pms_stars on the server.

- Maggie, Tyler, and I are all running the code and exporting the parameters below

```
csv_params = {"Star": None, "RMS" : None, "Median Mag": None, "SS Best Metric" : None, "SNR at
Peak": None, "Peak Period": None }
```

for each star into a csv file on the server.

- This csv file will then be copied onto our local machine and we can compare values. Below is a section of the csv file that was created when the code was run on 2000 files.



| Star | RMS | Median Mag | SS Best Metr | SNR at Peak | Peak Period |
|---|---|---|---|---|---|
| 24647230 | 0.07036339 | 11.8638783 | | 0.00228217 | 0.99677 |
| 24842312 | 0.16588725 | 8.6563077 | | 0.00544111 | 0.23614 |
| 15187403 | 0.17597656 | 14.2095499 | 1.276075 | 0.01186967 | 1.28043 |
| 9542108 | 0.13581755 | 13.7176991 | 1.00457 | 0.01108748 | 1.25911 |
| 6743746 | 0.17304356 | 13.9089744 | 0.96882 | 0.01197826 | 1.26249 |
| 14727560 | 0.23492759 | 13.5848618 | 1.49363 | 0.01265525 | 1.34478 |
| 16014802 | 0.27376907 | 13.3946409 | 1.033755 | 0.01654845 | 0.96297 |
| 59094502 | 0.14208417 | 13.5796995 | 1.468865 | 0.00594753 | 0.95959 |
| 54028651 | 0.18507233 | 13.9725027 | 1.00392 | 0.00887898 | 1.03447 |
| 14729090 | 0.16056874 | 13.3613834 | 1.035315 | 0.00362141 | 1.49389 |
| 3265739 | 0.23095455 | 13.2952118 | 0.9982 | 0.00544679 | 1.4688 |
| 60297260 | 0.09912009 | 12.5395393 | 0.96336 | 0.00346097 | 1.32996 |
| 8454597 | 0.08882302 | 11.6514096 | 1.00249 | 0.00572658 | 1.49324 |
| 15171448 | 0.23657144 | 13.3644407 | 0.99729 | 0.00717125 | 1.0281 |
| 58402805 | 0.02832313 | 10.3550415 | 1.05345 | 0.00279728 | 0.99599 |
| 59360390 | 0.12998908 | 13.1511917 | 0.962255 | 0.00411255 | 1.49532 |
| 45008955 | 0.14334464 | 13.178864 | 1.01588 | 0.00423514 | 0.993 |
| 15175564 | 0.15841242 | 13.2773988 | 1.01588 | 0.00368912 | 1.49532 |
| 14055068 | 0.15278428 | 12.8656616 | 1.28446 | 0.00473035 | 1.49766 |
| 14053603 | 0.13239596 | 12.8452797 | 1.465225 | 0.00780293 | 0.49861 |
| 56206531 | 0.14070918 | 12.8584967 | 1.359145 | 0.00212856 | 1.33087 |

- From here, we will order the csv file by highest SNR values at the peak.
- Then, we will look at the BLS and supersmoother images sitting on the server of those stars with the highest SNR. In other words, storing the data in a csv file prevents us from having to look at the plots for every star. Instead, we can look at the plots for the ones that are most likely variable stars.

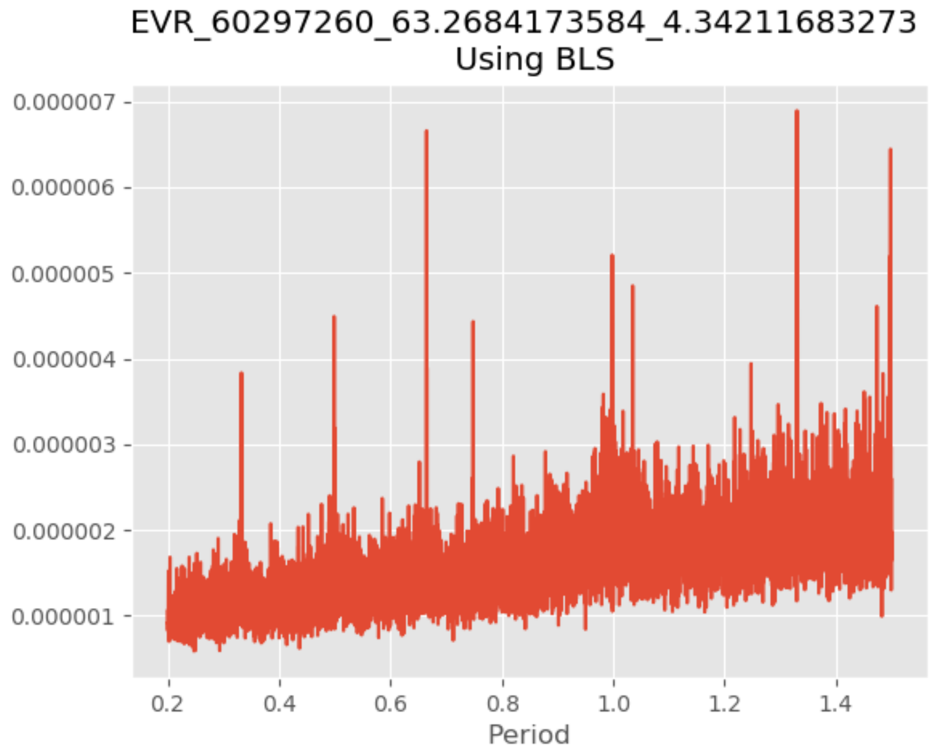
Note: While writing this I realized I should make an RA and DEC column as well.

My display "___".png is not working on the server anymore so I will work on that tomorrow. But, when the excel sheet was sorted these stars have the highest SNR at the peak:

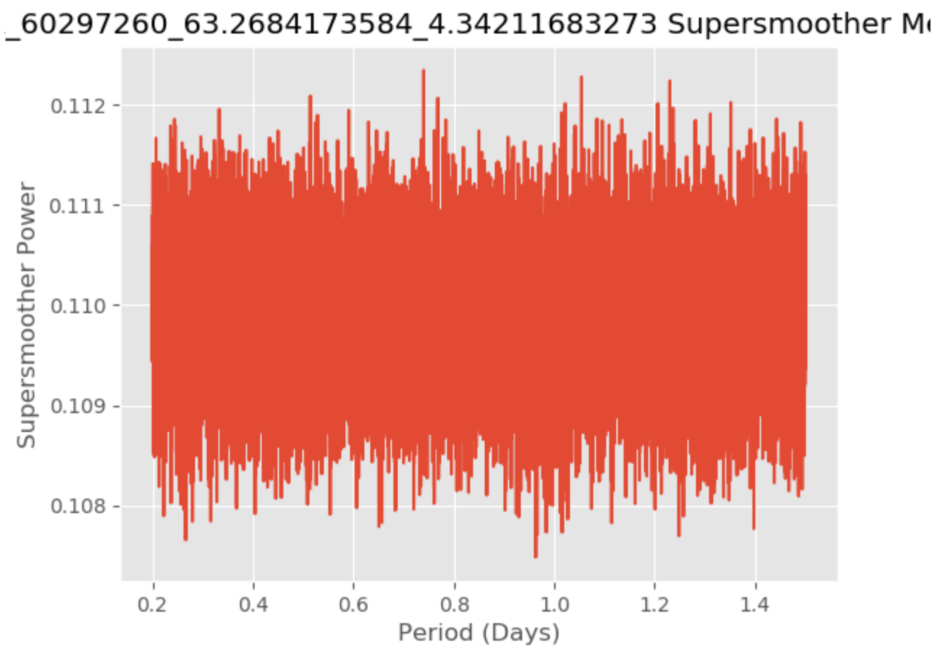
| Type | Star | RMS | Median Mag | SS Best Metric | SNR at Peak | Peak Period |
|---|---|---|---|---|---|---|
| EVR | 60297260 | 0.099120089 | 12.53953934 | 0.96336 | 0.211551148 | 1.32996 |
| EVR | 8454597 | 0.088823019 | 11.65140963 | 1.00249 | 0.123653298 | 1.49324 |
| EVR | 15171448 | 0.236571441 | 13.36444068 | 0.99729 | 0.12131212 | 1.0281 |

| | | | | | | |
|---|---|---|---|---|---|---|
| EVR | 58402805 | 0.028323129 | 10.3550415 | 1.05345 | 0.070627471 | 0.99599 |
| EVR | 59360390 | 0.129989076 | 13.15119171 | 0.962255 | 0.064161247 | 1.49532 |
| EVR | 45008955 | 0.143344643 | 13.178864 | 1.01588 | 0.041950351 | 0.993 |
| EVR | 15175564 | 0.158412422 | 13.27739882 | 1.01588 | 0.03156391 | 1.49532 |

Therefore, I will look at the plots of these stars first once I get my display working again. When the BLS plot for 60297260 was copied to my local machine, I got the following plot:



With supersmoother:



**4/8/19:**

This week came with many struggles. I had trouble implementing the polynomial fit and subtract checkpoint. The problem was that I misspelled a variable in my code but the debugger with multiprocessing is very bad. It returns this unhelpful error quite a lot:

```
Traceback (most recent call last):
  File "/usr/lib/python3.6/multiprocessing/pool.py", line 119, in worker
    result = (True, func(*args, **kwds))
  File "/usr/lib/python3.6/multiprocessing/pool.py", line 47, in 
starmapstar
    return list(itertools.starmap(args[0], args[1]))
  File "502_research.py", line 387, in do_work
    if "supersmoother" in args.m:
TypeError: argument of type 'NoneType' is not iterable
"""

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "502_research.py", line 503, in <module>
    pool.starmap(do_work,input_list)
  File "/usr/lib/python3.6/multiprocessing/pool.py", line 296, in starmap
    return self._map_async(func, iterable, starmapstar, chunksize).get()
  File "/usr/lib/python3.6/multiprocessing/pool.py", line 670, in get
    raise self._value
TypeError: argument of type 'NoneType' is not iterable
```

So I added a debugger state variable in my code that when it is true, I don't use multiprocessing:

```
if DEBUG:
        for filepath in filelist:
            if not os.path.isdir(args.data):
                do_work(TEST_DATA,args,filepath)
            else:
                ra = np.float((filepath.split('/')[-1]).split('_')[2])
                if ra < 125.:
                    do_work(TEST_DATA,args,filepath)
    else:
        pool = multiprocessing.Pool(32)
        input_list = []
        for filepath in filelist:
            if not os.path.isdir(args.data):
                input_list.append([TEST_DATA,args,filepath])
            else:
                ra = np.float((filepath.split('/')[-1]).split('_')[2])
                if ra < 125.:
                    input_list.append([TEST_DATA,args,filepath])
```

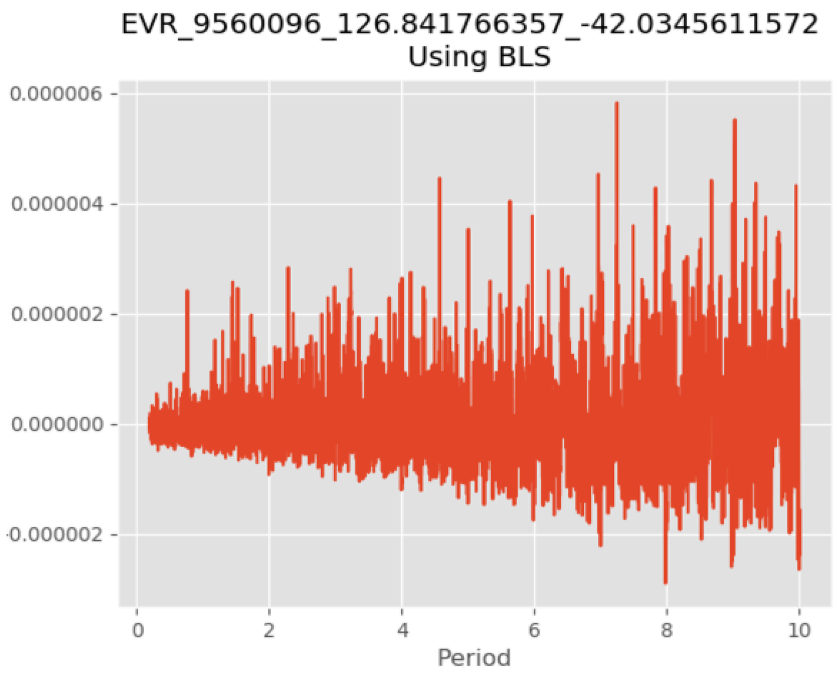
I would recommend others to do the same because I found the error the first time I ran the code after adding this.

So after adding the polynomial fit and subtract below:

```
def flatten_BLS(period_arange, best_bins):
    poly_coeffs = np.polyfit(period_arange, best_bins, 2)

    # Sigma Clipping
    rms = np.std(best_bins)                        # measure the standard deviation of the periodogram
                                                   # this includes the slope, but we'll assume that's
ok
    clip_mask =    best_bins <  rms * 3.0          # make a mask for the points we want to keep
    periods_polyfit = period_arange[clip_mask]     # apply the mask to periods
    pgram_polyfit = best_bins[clip_mask]           # and the periodogram

    polynomial_func = np.poly1d(poly_coeffs)
    y_fit_values = polynomial_func(period_arange)

    return y_fit_values

if "bls" in args.m:
            period_arange, best_bins =
BLS(epoch,mag,period_min=PERIOD_MIN,period_max=PERIOD_MAX,stepsize=STEPSIZE)

            y_fit_values = flatten_BLS(period_arange, best_bins)

            best_bins_flat = best_bins - y_fit_values
```

Note: best_bins should be an array. I had trouble with this for awhile because my best_bins was a list.

As a result, I got the BLS plot:



This is a little different than the one on the checkpoint page on Sakai. However I still need to mask out my day periods while plotting the BLS.

I spent about two hours searching for variable stars via the method described above (sorting csv by maximum snr) and did not find anything. However, my maximum period plotted was only 2 days so this will be increased to 20 days and I will run my code on 2000 stars again.

I also started to look for stars where the peak period (found by BLS) was the same as the SS best metric (best period found by supersmoother). I was thinking about also maybe trying to fit a sine curve to the phased up light curve and sort the files by the lowest standard deviation from the fit curve. But, I would worry that the algorithm would fit a messy light curve with a sine wave with an amplitude of 0 (so a line) and say that it is a good fit. So, a minimum amplitude would need to be set. Also, perhaps a low RMS is essentially telling us a phased up light curve follows a pattern?

This past week I copied the excel files manually to my local machine but next week I will write a csv processing script so I can do everything on the server.

I'm sorry I didn't get results but I definitely put in a lot of effort.

**4/15/19:**

I ran my code again on 1000 lightcurves for pms_stars and ums_stars. The good news is is that my function to remove the nans that I added this week is working:

```
def delete_nans(arr_cut_nans, arr_to_shorten):
    arr_cut_nans   = np.array(arr_cut_nans)
    arr_to_shorten = np.array(arr_to_shorten)

    cond = np.array(np.where(arr_cut_nans==arr_cut_nans)).flatten()
    return arr_to_shorten[cond]
```

I figured out this was a problem when I sorted my peak SNR from highest to lowest values but the top 5 were all nan values. This is no longer a problem!

I also masked out the integer periods and the periods within 5% of .5 with:

```
def day_mask(period_arange):
    remainder            = np.remainder(period_arange,1)
    period_arange_masked = np.array(np.where((remainder < 0.95) & (remainder > 0.05))).flatten()
    return period_arange_masked
```

Finally, I have added a cvs processing script to sort the csv file I output with all the data on the server. This way I do not have to copy csv file to my local machine and sort it myself.  This code is shown below:

```
#################################################################
######
import matplotlib
matplotlib.use("Agg")
import numpy as np
import math
import matplotlib.pyplot as plt
import pdb
import argparse
import csv
#################################################################
```

```
######


########################################################################
######
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('logfile',help='CSV file w/output from 
lc_multiproc.py')
    args = parser.parse_args()

    data = np.loadtxt(args.logfile,delimiter=',',skiprows=3)
    # star ID, RA, DEC, RMS, Median Mag, SS period, SNR at Peak, BLS Peak 
period
    star_num  = data[:,0]
    ra        = data[:,1]
    dec       = data[:,2]
    rms       = data[:,3]
    med_mag   = data[:,4]
    SSperiod  = data[:,5]
    peak_snr  = data[:,6]
    BLSperiod = data[:,7]


    print(star_num[np.argsort(peak_snr)[::-1]])
```

I will add to this csv file today to first sort by stars where the best supersmoother period is within 5% of the period of the peak of the BLS.

I still am frustrated I have not found anything. At this point I have looked through about 40 stars and still have not found anything. I need to filter my results better.

**4/22/18:**

This week I figured out why my phased up light curves were always off → my epochs were in minutes while everything else was in days. When I figured this out, I reran my code on the pms stars and ums stars. I also wrote a script to display 50+ plots at once so I could look at the phased up light curves at once. This code is shown below:

```
import numpy as np
import os
import os.path
import os
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import matplotlib.image as mpimg


folder_path = "images/pms_stars_imgs"
```

```
images = []
for dirpath,dirnames, filenames in os.walk(folder_path):
    for filename in [f for f in filenames if f.startswith("EVR_")]:
        images.append(os.path.join(dirpath, filename))
```

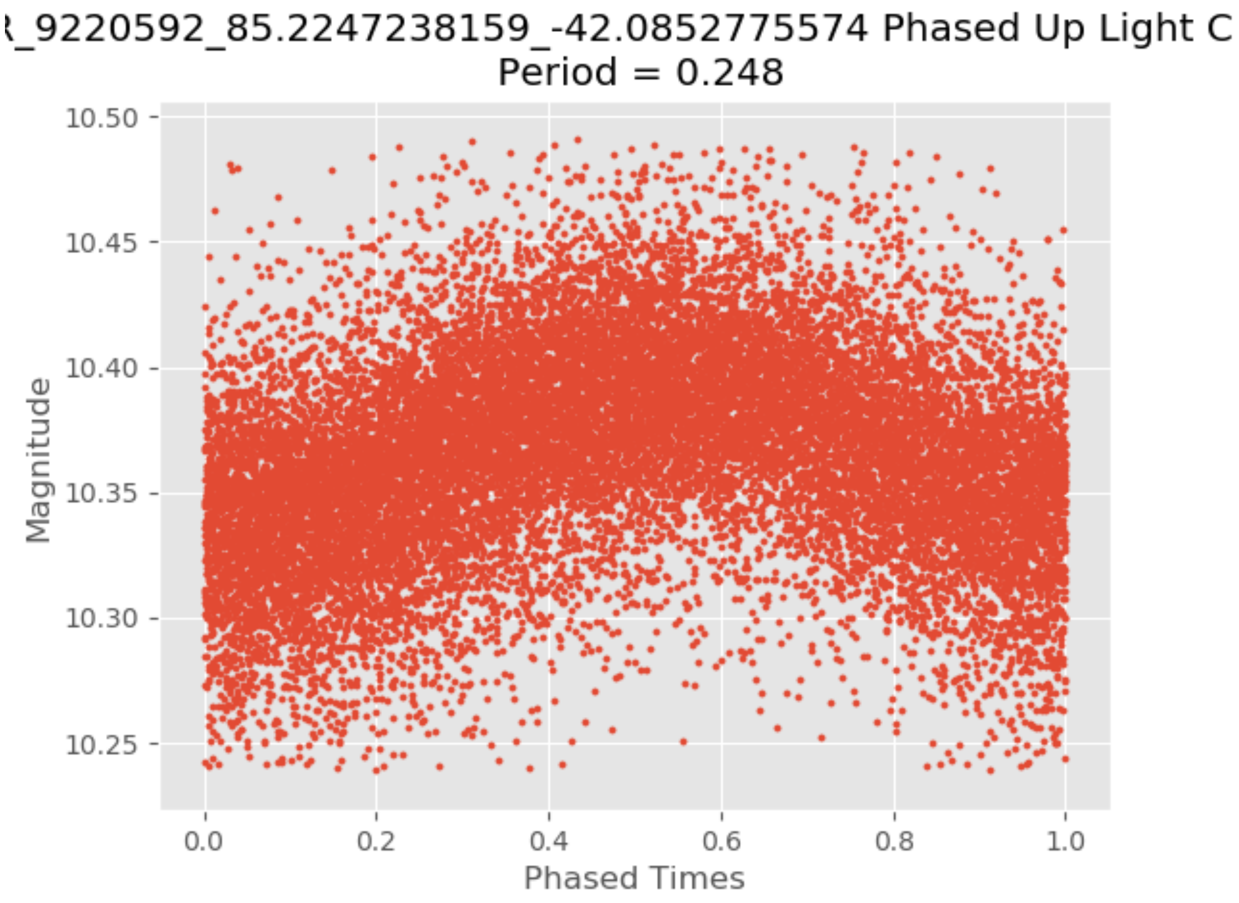
```
images = images[350:400]
n = len(images)
gs1 = gridspec.GridSpec(n//5,5)

for i,img_path in enumerate(images):
    plt.subplot(gs1[i])
    img = mpimg.imread(img_path)
    plt.axis('off')
    plt.imshow(img)
plt.show()
```

Using this and other techniques, Maggie, Tyler, and I found the following pms stars to appear to be variable based on the phased up light curves:

| | | | | | | |
|---|---|---|---|---|---|---|
| 9220592 | 0.248 | | a star | Y | Y | Y |
| 9434604 | 0.744 | | n/a | Y | Y | Y |
| 9461085 | 0.892 | | rotationally variable | Y | Y | Y |
| 11531982 | 0.14233 | eclipsing binary | n/a | Y | Y | Y |
| 12176863 | 2.07 | | either this star or the one next to it | Y | Y | Y |
| 14051010 | 0.782 | | n/a | Y | Y | Y |
| 14053932 | 0.832 | | a star | Y | Y | Y |
| 14055068 | 1.63 | | n/a | Y | Y | Y |
| 14059062 | 1.429 | | eclipsing binary candidate | Y | Y | Y |
| 14061212 | 0.912 | | emission-line star | Y | Y | Y |
| 16796220 | 1.631 | | flare star | Y | Y | Y |
| 33940047 | 1.397 | | n/a | Y | Y | Y |
| 37470621 | 2.188 | | x-ray source | Y | Y | Y |
| 37480680 | 0.748 | | TYC 8143-395-1 | Y | Y | ? |
| 37496046 | 2.07 | | TYC 8140-1395-1 | Y | ? | Y |
| 37522904 | 0.2493 | Good undiscovered candidate! | TYC 8149-2713 | Y | ? | ? |
| 44829290 | 3.175 | | X-ray source | Y | ? | ? |
| 47852173 | 0.376 | | n/a | Y | N | N |
| 53970999 | 4.395, 9.95198 | | X-ray source(but a star very nearby) | Y | Y | Y |
| 53989609 | 0.748, 9.272? | | n/a (but near a VERY bright star) | Y | Y | Y |
| 53998446 | 2.748 | | TYC 5331-24-1 | Y | Y | Y |
| 37494010 | | | | | | |

The Y/N/? resemble whether Maggie, myself, or Tyler, respectively, found the same thing from their phased up light curves.

An example of a light curve that we saw is shown below:


_9220592_85.2247238159_-42.0852775574 Phased Up Light C
Period = 0.248
Magnitude
Phased Times