Evaluation of the EEBLS Algorithm in STExTS Exoplanet Searches

Matt Heuser
University of Dallas
Department of Physics
May 2013

Thesis Advisors:
Richard Olenick, Ph.D.
Arthur Sweeney

Submitted in partial fulfillment of
the Bachelor of Science degree
at the University of Dallas

Abstract

Transiting exoplanets are too small to be observed by the naked eye or by telescope. To detect these transits, other methods must employed in place of direct observation. The method used to conduct this research was photometry. The magnitude of approximately 3000 stars was measured over several weeks and then each star's magnitude was analyzed for changes in magnitude. Many times, these changes in magnitude are caused by a planet in orbit around the star passing between the star and the observer. Stars which appear to show this phenomenon are transit candidates.

In order to analyze the large amount of data gathered, a computer algorithm was employed to automate this process. The algorithm was a subroutine called EEBLS written in FORTRAN'77 by Géza Kovács at the Hungarian Academy of Sciences' Konkoly Observatory. The algorithm was used to develop a list of transit candidates.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I wish to thank the University of Dallas and the Donald A Cowan Institute for their support in this work.

1 Introduction

A transit occurs when a planet crosses in front of a star in the field of view of an observer on earth. When this happens, a small portion of the light emitted from the star is blocked from reaching the observer. Consequently, the brightness of the star decreases as the planet moves past the star, and then increases as the star exits, passing the star. A star's brightness is measured and recorded over a period of time. The processed data is then analyzed for episodes of reduction in the brightness. These “dips” can indicate that a planet orbits the star, as shown in Figure 1.1.

The first extrasolar planet was discovered in this manner in 1995, orbiting a star similar to our sun. Since then, many improvements have been made in the astronomical equipment used to search for exoplanets. Currently, there are over 880 known extrasolar planets. It is estimated that 100 to 400 billion exoplanets exists in the milky way galaxy alone. [1]

In March of 2009, the Kepler Project was launched by NASA for the purpose of detecting potential life-supporting planets, orbiting stars outside the solar system. Kepler is a space-based telescope that utilizes the transit method of planet detection. The Kepler telescope has a diameter of 0.95 meters and a field of view of approximately 10 degrees square. The satellite simultaneously monitors 100,000 stars brighter than 14th magnitude. [2]

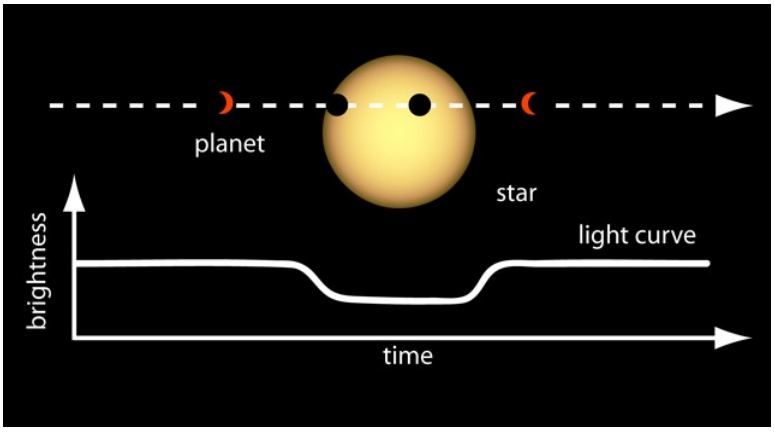

Figure 1.1: *The brightness vs. time graph for a star while a planet is transiting.*

Eclipsing binaries can be detected in a similar way. When one of the stars passes in front of the other one, the detected total magnitude appears to decrease to an observer on earth. However, there are several differences between the light curve of a binary and that of a transit: the decrease in magnitude in the light curve of a binary star will be be much greater than a transit. The depth of the "drop" in magnitude for an exoplanet transit is typically about 50 millimagnitudes. The binary light curve will usually resemble the shape of the letter V whereas the transit light curve will have a box-like shape. Figure 1.2 shows a comparison of an eclipsing binary and an exoplanet transit.

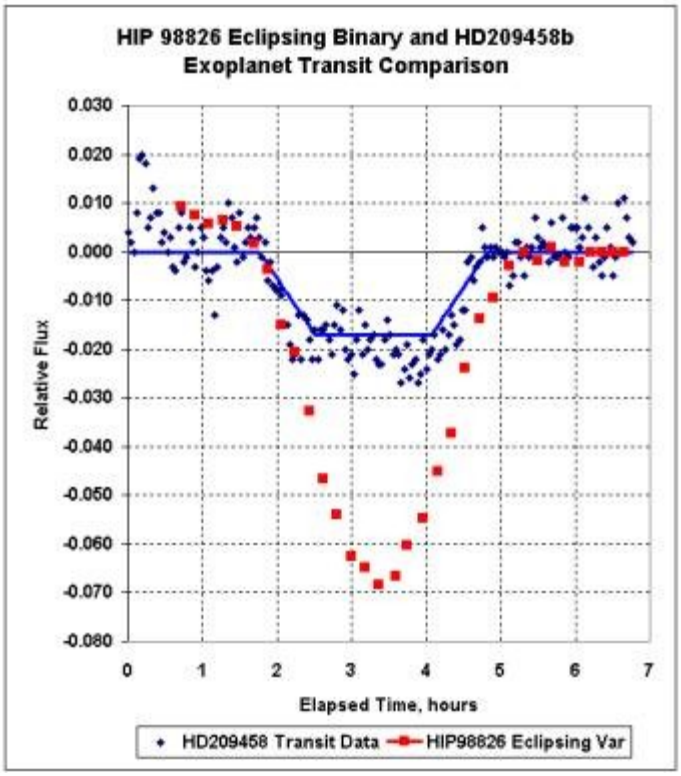

Figure 1.2: *Comparison of light curves for a binary and a transit. The transit magnitude decrease (blue) is more shallow and square than the binary light curve (red).*

2 Equipment

The primary telescope had a 6 inch diameter objective lens with a focal length of 200 millimeters and an aperture of f/1.5 stopped down to f/2.8. An R-band filter was used.

The secondary telescope, mounted on top of the main telescope, was an Orion telescope which had a diameter of 80 millimeters. The secondary telescope was used to guide the main scope with PHD Guiding, as shown in Figure 2.4, while the main scope simultaneously acquired the data images using CCD Soft, as shown in Figure 2.2.

The primary CCD was a SBIG-ST10 which had a resolution of 3.2 mega-pixels which produced images with a full frame resolution of 2184 x 1472 pixels. The SBIG CCD included a temperature regulator which was controlled through CCD Soft. Communication to the computer was through a USB 2.0 connection. Each telescope also included a heater to prevent dew from forming on the lens. A German equatorial mount was used.

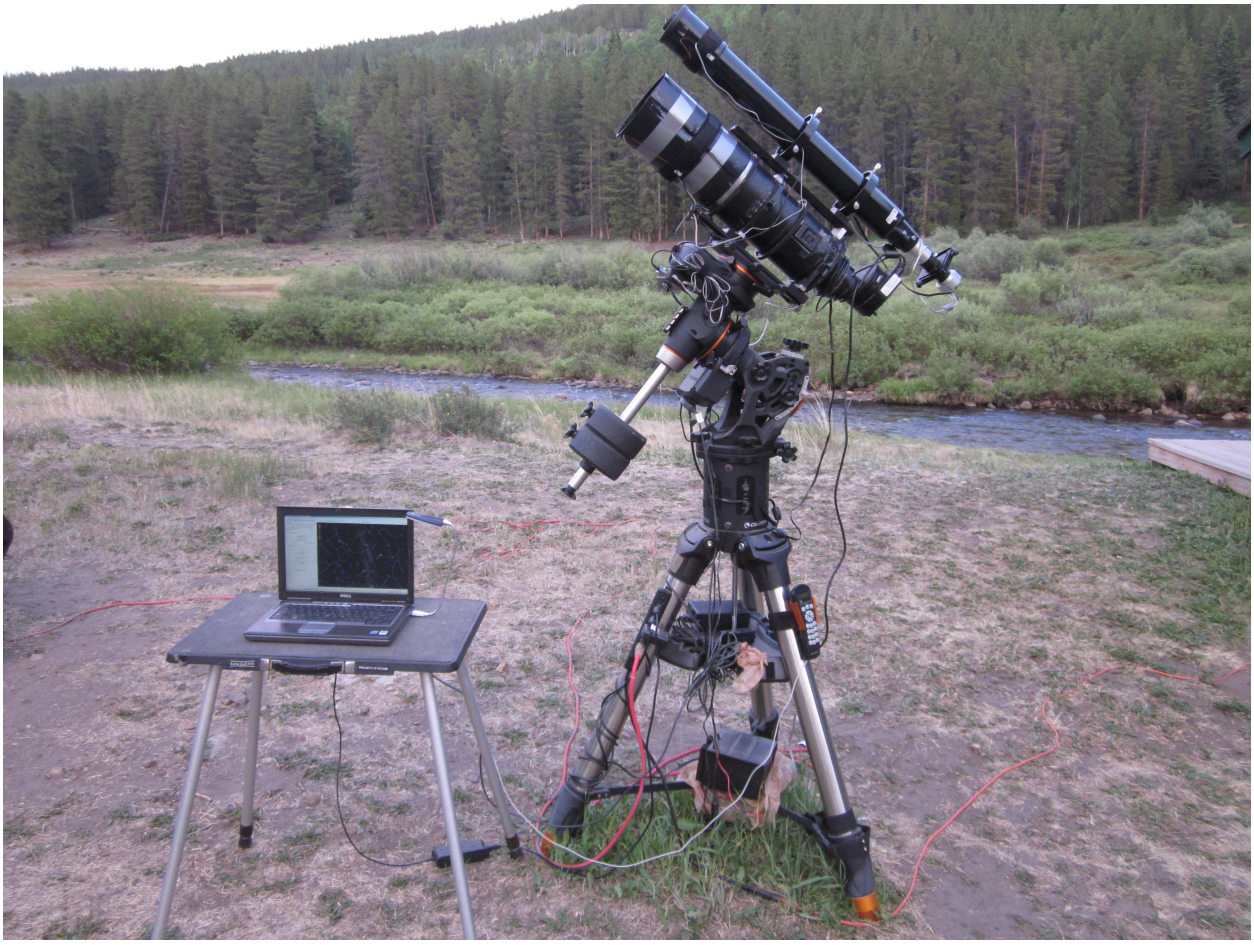

Figure 2.1: *Telescope setup showing the guide scope mounted on top of the main scope and controlled by a German equatorial mount. The laptop (left) was used to retrieve and store the images from the main telescope.*

CCD Soft was used to capture images from the main scope camera and save them to the hard drive of the computer. It was also used to control the temperature of the camera, which was cooled to -20.0 degrees Celsius.

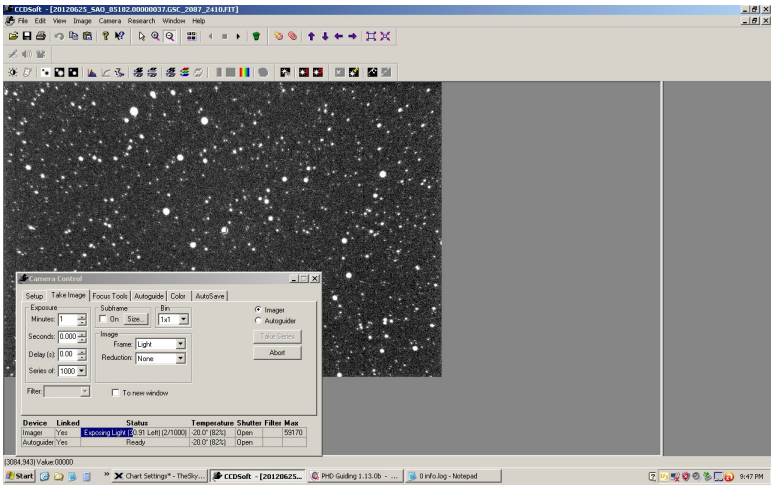

Figure 2.2: *Screenshot of CCD Soft showing an image being captured from on the main telescope.*

The Sky X provided relevant star charts and was used to control the mount and point the telescope to the correct area of the sky.

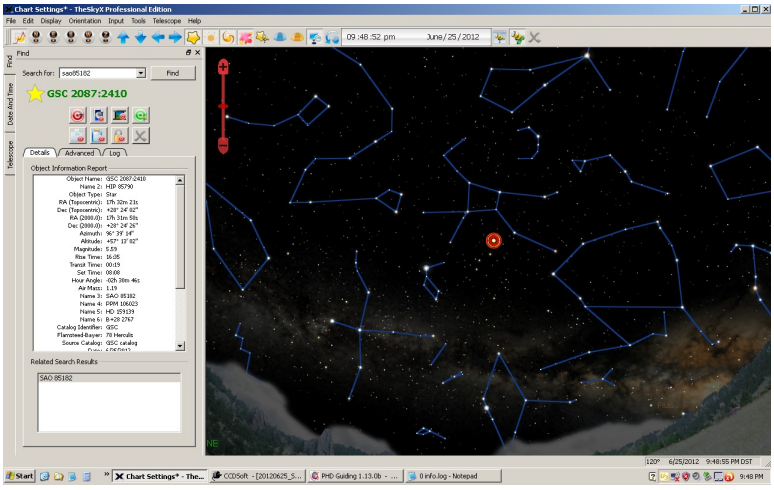

Figure 2.3: *Screenshot of The Sky X showing a star map centered on SAO 85182.*

Once the constellation Hercules was found, the telescope was centered on SAO 85182 and PHD Guiding, shown in Figure 2.4, was used to guide on that star during the night.

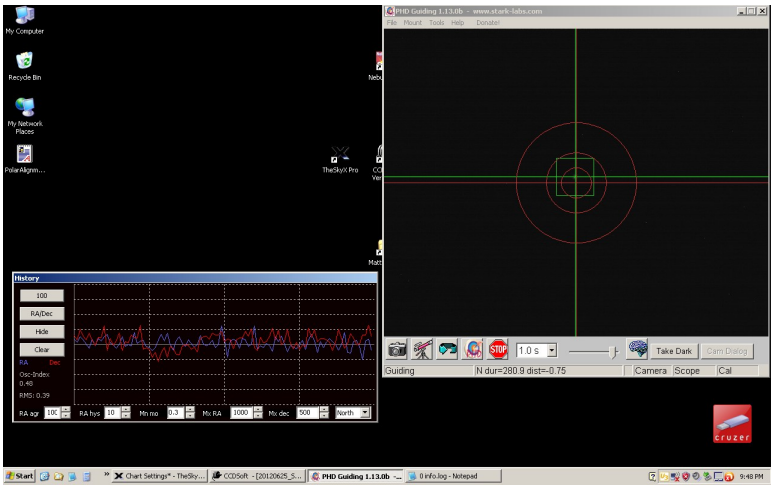

Figure 2.4: *Screenshot of PHD Guiding showing the telescope being guided on the star SAO 85182.*

Nebulosity, shown in Figure 2.5, was used to focus the main camera. The camera was focused at least once every night to ensure clear images.

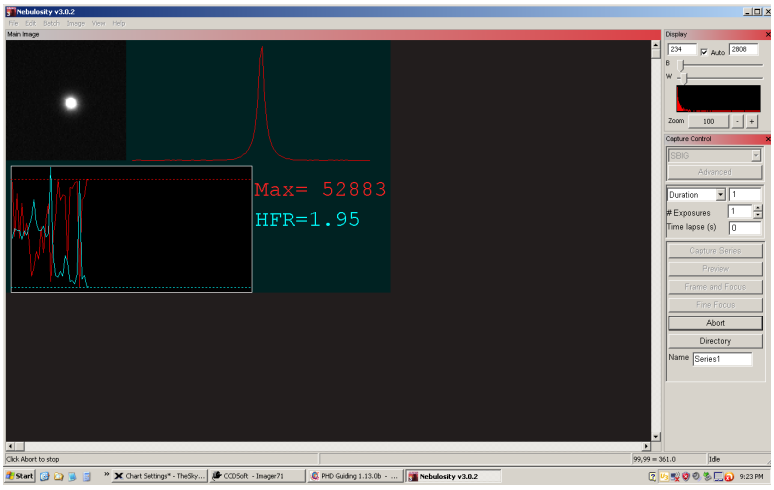

Figure 2.5: *Screenshot of Nebulosity just after being focused on the center star, SAO 85182.*

3 Data Acquisition

The research for this project took place in Pitkin, Colorado between May 21, 2012 and June 25, 2013. Pitkin is located on the western slopes of Colorado at an elevation of 2809 meters. Data was taken every night that the weather permitted from astronomical twilight until dawn. The relatively high altitude helped to reduce air mass and the remote location allowed for minimized light pollution.

A total of 37 nights of data were taken, resulting in a total of 8535 data images. Each image was taken with a 60 second integration time. The signal to noise ratio varied for each image peaking at 100. Calibration images were taken at the beginning and end of each night.

Date	Begin	End	Images	Exposure (s)
05-21-12	0	3:50	220	60
05-22-12	22:56	5:28	321	60
05-23-12	00:39	5:11	205	60
05-24-12	22:48	5:20	213	60
05-25-12	22:38	4:55	308	60
05-27-12	22:59	5:01	297	60
05-28-12	22:55	4:59	303	60
05-29-12	23:45	5:40	292	60
05-30-12	23:12	5:39	309	60
05-31-12	23:12	5:29	177	60
06-01-12	23:04	5:51	337	60
06-05-12	23:06	5:27	317	60
06-06-12	22:45	4:05	266	60
06-07-12	22:26	5:09	343	60
06-08-12	22:50	4:58	306	60
06-09-12	22:32	5:27	344	60
06-10-12	22:38	5:23	339	60
06-11-12	22:31	5:21	342	60
06-12-12	22:30	4:50	317	60
06-13-12	22:27	5:31	352	60
06-14-12	23:25	5:03	273	60
06-15-12	23:02	4:59	298	60
06-16-12	22:33	5:05	323	60
06-17-12	23:28	5:00	286	60
06-18-12	22:33	5:00	324	60
06-19-12	22:50	5:07	315	60
06-20-12	22:54	5:18	320	60
06-21-12	23:15	5:36	318	60
06-22-12	23:07	4:59	292	60
06-23-12	23:06	5:40	306	60
06-24-12	23:57	4:48	243	60
06-25-12	22:46	5:31	331	60

Table 2.1: *Log of observations.*

Date	Condition
05-21-12	01:00 The sky is clear and there is almost no wind 04:00 The sky is clear and there is almost no wind
05-22-12	22:00 Clear sky with no wind
05-23-12	22:00 Sky is cloudy with a slight gusty wind 23:30 Clear sky with no wind 03:30 There are several large clouds in the sky but no wind 03:45 PHD Guiding has stopped tracking due to clouds
05-24-12	21:38 Clear sky no wind 11:00 No wind but clouds are starting to form 11:30 Clouds are blocking the stars but still no wind 01:40 Clouds have cleared and the wind is calm
05-25-12	21:30 A few scattered clouds in the sky with a slight wind
05-27-12	22:00 Sky is clear with a slight wind
05-28-12	22:00 Sky is clear with no wind
05-29-12	21:30 Thin clouds moving in and out of frame
05-30-12	22:00 Sky is partially covered with clouds. No wind 22:15 One large cloud is moving toward data star 22:20 Cloud wisps are present in the frame and a large cloud is moving toward field
05-31-12	22:00 A few scattered clouds, no wind 00:10 A large cloud is moving toward data star 00:20 Cloud has covered data star. Stopping images until sky clears
06-01-12	22:02 Clear sky
06-05-12	22:00 Clear sky, no wind
06-06-12	21:30 Clear Sky, no wind 02:00 A few small scattered clouds in the sky 03:10 Clouds have entered the field of view. Stopped taking images.
06-08-12	21:45 Clear sky, no wind
06-09-12	21:30 Clear Sky, no wind 01:45 Scattered clouds in the sky
06-11-12	21:30 Clear sky, no wind
06-12-12	21:00 Clear sky, no wind
06-14-12	21:00 Clouds cover most of the sky 22:30 Clear sky, no wind 00:55 Clouds moving into frame. Stopped taking images
06-15-12	21:30 Clouds covering the sky 22:00 Sky cleared, no wind 22:45 Scattered clouds, light wind
06-17-12	22:30 Clear sky, no wind
06-18-12	21:30 Clear sky, no wind
06-20-12	22:00 Clear sky, no wind
06-21-12	22:00 A few scattered clouds, no wind
06-23-12	22:00 Clear sky, no wind 22:35 Clouds entering frame. Stopping until it clears
06-24-12	21:30 Clouded sky. waiting until it clears 23:00 Clear sky, no wind
06-25-12	21:45 Clear sky, no wind

Table 2.2: *Log of atmospheric conditions.*

The field of view of the telescope used was about 3 degrees, allowing data to be captured on 2500 to 3000 stars. The center star in this field was SAO 85182 which is within the Hercules constellation.

Figure 3.1: *Sample Data Image.*

4 Methods

In order to identify possible transits in the images gathered, photometric reduction of each image was completed. In photometry, the flux from a star is measured from an image. Curves of magnitude versus time, known as light curves, are then constructed and examined.

The period for a transit is typically several hours in length. With our telescope and CCD, we were able to capture and record data points for the light curves of 2500 to 3000 stars approximately once per minute. Before photometric measurements were made, the images were calibrated using SYSREM (see appendix C) and the stars were extracted using SExtractor. Once the coordinates of the star were found, they were compared with the Guide Star Catalogue (GSC) database to determine whether or not they had been previously recorded. Figure 4.1 shows details of the data pipeline. [3]

With light curves constructed for nearly 3000 stars, an algorithm was needed to search for signals of possible transits. We used the EEBLS algorithm. [4]

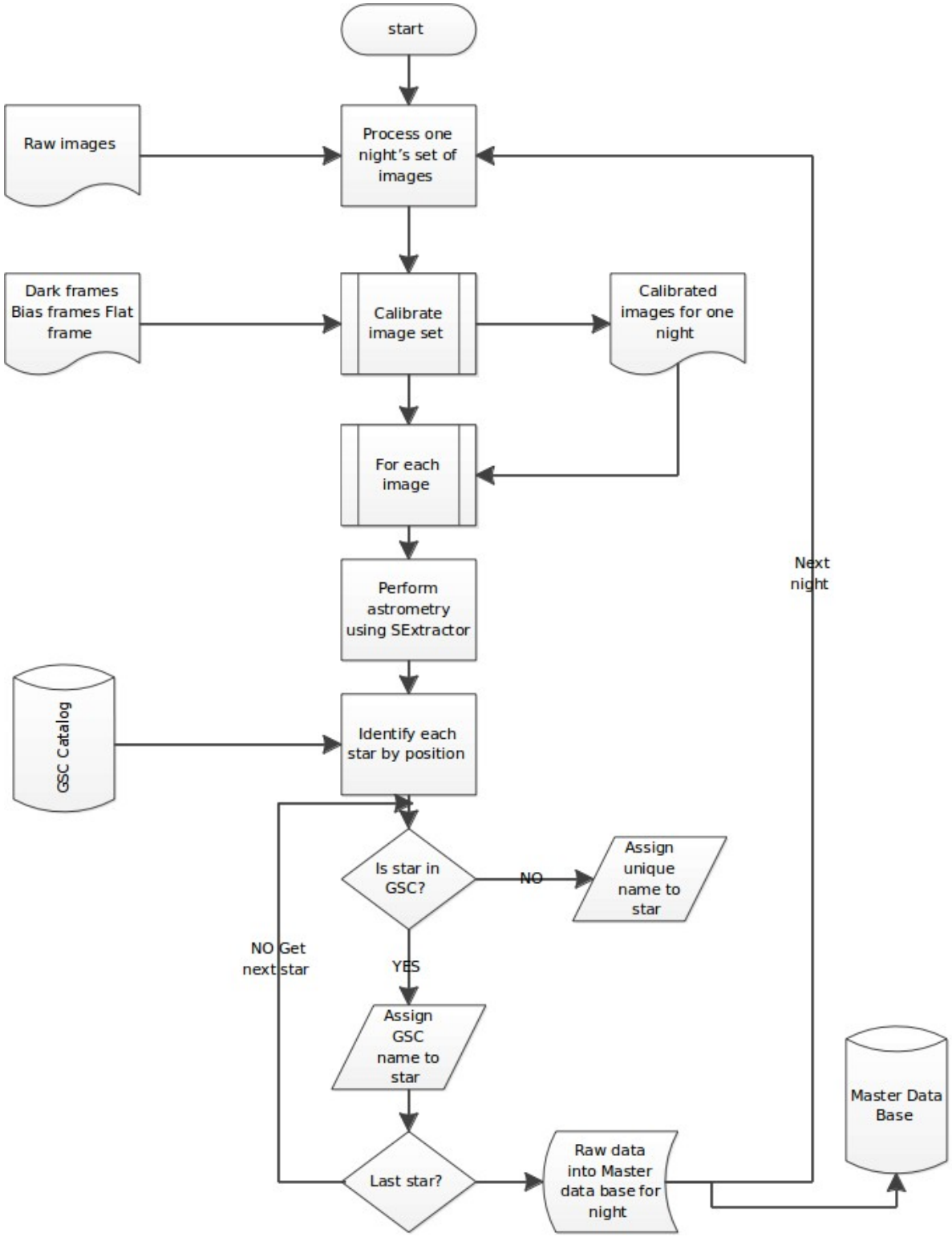

Figure 4.1: *Flow Chart of Image Reduction Process, Phase I.*

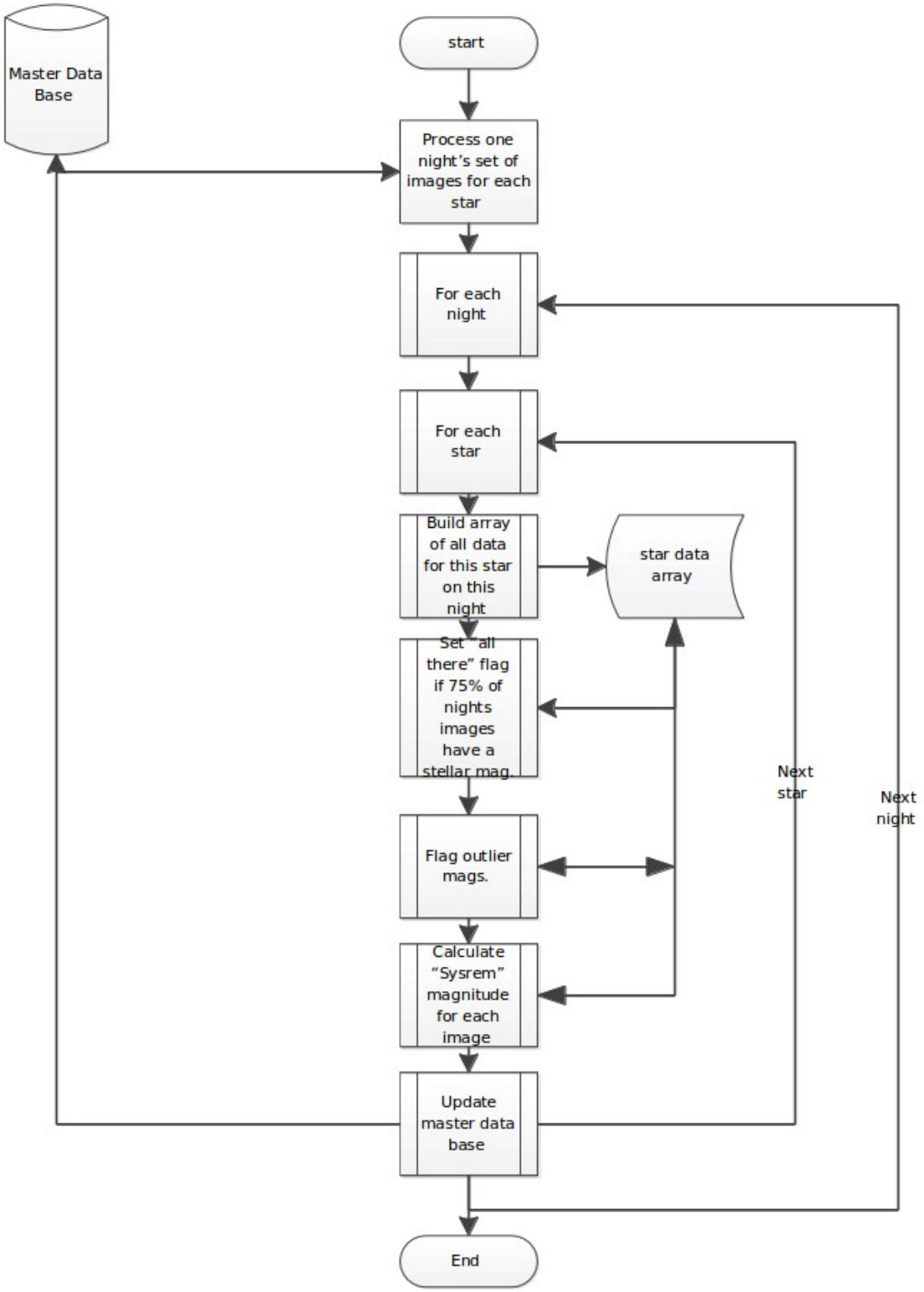

Figure 4.2: *Flow Chart of Image Reduction Process, Phase II.*

5 Algorithm

The algorithm used was the EEBLS, Edge Enhanced Box Least Squares. It is a box-fitting least squares algorithm developed by Kovács at the Konkley Observatory in Hungary. The algorithm analyzes the light curve data points for periods of reduced magnitude and attempts to fit a box to any periodicity found. These “box” shaped functions are composed of two step functions, that is, they alternate between a high value and a low value. The low value represents the time in which the planet is blocking some of the light from the star and the high value represents the time in which no light is blocked.

This is in contrast to Fourier analysis where sinusoidal functions are fit to curves. For a transit, the time it takes the planet to make the transition from no magnitude reduction to full magnitude reduction is relatively small compared to the total time it spends in full magnitude reduction. This characteristic makes a box least squares algorithm more suitable for finding transits than Fourier analysis does.

In order to fit a box to a period, the magnitude data points are “folded” to the period creating phase diagram. If a transit is observed, some fraction of the period will have a lower magnitude than the rest of the period. The minimum and maximum limits of this fraction are specified as input parameters.

The EEBLS algorithm developed by Kovács is a subroutine written in FORTRAN'77. It was necessary to develop an additional driver program to read the data files and input it into the subroutine. The decision was made to port the EEBLS

algorithm to C and write the driver program in C because the team was more familiar with C than FORTRAN. This program can be found in Appendix B.

The driver program (see Appendix B) written for the EEBLS subroutine reads a text file containing a list of recorded star brightnesses and the Modified Julian Date (MJD) when the brightness was recorded. The data files are stored in CSV format with a single header row.

The driver program transfers the entire data file into arrays in memory, one array for the MJD and arrays for as many columns as contain recorded magnitudes for a star. The EEBLS subroutine is then called and these arrays are passed to it as input. The output from the EEBLS subroutine is returned to the driver program and is subsequently written to a text file in CSV format.

6 Results

Table 6.1 is a sample of the output of the EEBLS algorithm. The data is sorted by the depth of the magnitude reduction in the light curve of the transits candidate, shown in Column D. Only candidates with negative depth are considered.

	A	B	C	D	E	F	G	H
1	Star	bper	bpow	depth	qtran	T	IP	EP
2	UD17.595704+26.626751	33.201714	1.743014	-8.244696	0.046893	3.113877	0.106625	0.63975
3	GSC 2087-0323	11.611849	0.227872	-1.86007	0.01524	0.353937	0.673937	0.488144
4	GSC 2087-0108	22.666009	0.229341	-1.858042	0.015475	0.701504	0.244795	0.611263
5	GSC 2605-1636	14.467817	0.043933	-0.442313	0.009965	0.288339	0.544496	0.544496
6	GSC 2083-1870	1.983659	0.069403	-0.241622	0.090739	0.359989	0.456091	0.300688
7	GSC 2087-1908	7.805248	0.05145	-0.216802	0.059906	0.935166	0.520855	0.296033
8	GSC 2083-1636	4.461917	0.045424	-0.179866	0.068464	0.610964	0.908882	0.082611
9	GSC 2087-1579	7.805248	0.038312	-0.161439	0.059906	0.935166	0.520855	0.296033
10	GSC 2083-0661	10.296654	0.040188	-0.160804	0.06694	1.37852	0.576157	0.298708
11	GSC 2087-0343	7.805248	0.034396	-0.144938	0.059906	0.935166	0.520855	0.296033
12	GSC 2088-0814	15.596019	0.037547	-0.143876	0.073505	2.292779	0.737719	0.027339
13	GSC 2087-0469	19.952542	0.041417	-0.138202	0.099766	3.981152	0.504219	0.921205
14	GSC 2088-0134	10.191714	0.022895	-0.106419	0.048652	0.991691	0.361182	0.035178
15	GSC 2088-0560	11.478562	0.025331	-0.090608	0.085463	1.961986	0.7997	0.645825
16	UD17.612972+27.116050	10.191714	0.018063	-0.083958	0.048652	0.991691	0.361182	0.035178
17	GSC 2087-1127	10.403778	0.019959	-0.083195	0.061313	1.275774	0.42421	0.204952
18	GSC 2087-0218	11.3483	0.015447	-0.079637	0.039156	0.888706	0.99724	0.744433
19	GSC 2605-1950	10.191714	0.015428	-0.07171	0.048652	0.991691	0.361182	0.035178
20	GSC 2083-2131	31.134291	0.020389	0.078072	0.073623	4.584369	0.434542	0.796661
21	GSC 2087-0790	13.67635	0.016277	0.078699	0.044783	1.224939	0.79467	0.336766
22	GSC 2087-2090	31.134291	0.020781	0.079572	0.073623	4.584369	0.434542	0.796661

Table 6.1: *Sample EEBLS Output.*

In Table 6.1, column A indicates the name of the star, usually from the Guide Star Catalog (GSC). If the star was not found, a name was assigned beginning with UD. Column B is the period of the transit at the maximum power. Column C gives the power of the signal at the period in column B. Column D signifies the depth of the reduction of the flux. Column E is the fraction of the period that was captured in our

data. Column F is the length of the period. Columns G and H represent the the ingress and egress phases.

Because the EEBLS algorithm attempts to fit boxes to the data points, it will detect any periodicity found. Consequently, not every star listed in the output will have a transit. One criterion used to eliminate stars from the output that are not transits is positive depth. For a transit or a binary star, the magnitude of the star will decrease for a period of time, not increase.

As seen on Line 6 of Table 6.1, the algorithm has found the the star GSC 2083-1870. As shown in the following graphs, this star is most likely a binary star rather than a transit. Figure 6.1 is the light curve for GSC 2083-1870.

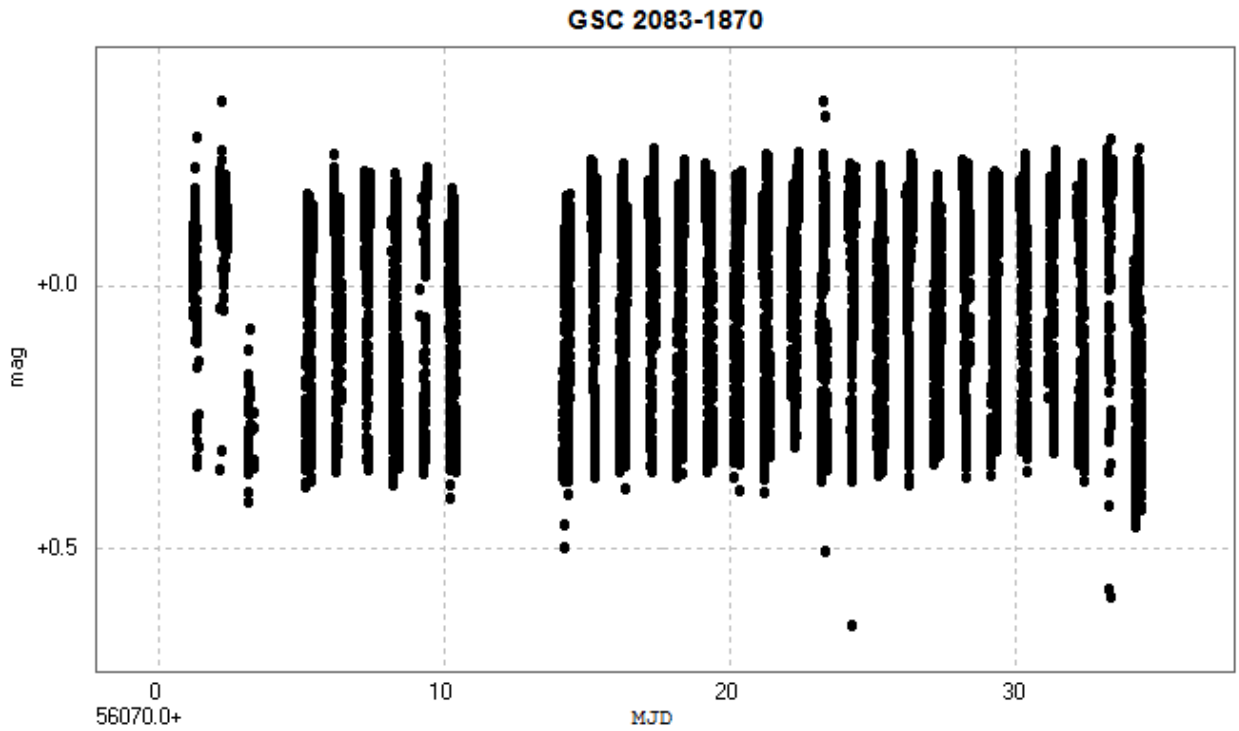

Figure 6.1: *Light curve for GSC 2083-1870.*

Figure 6.2 is the EEBLS periodigram for GSC 2083-1870. The spikes indicate values of the frequency of detected signals.

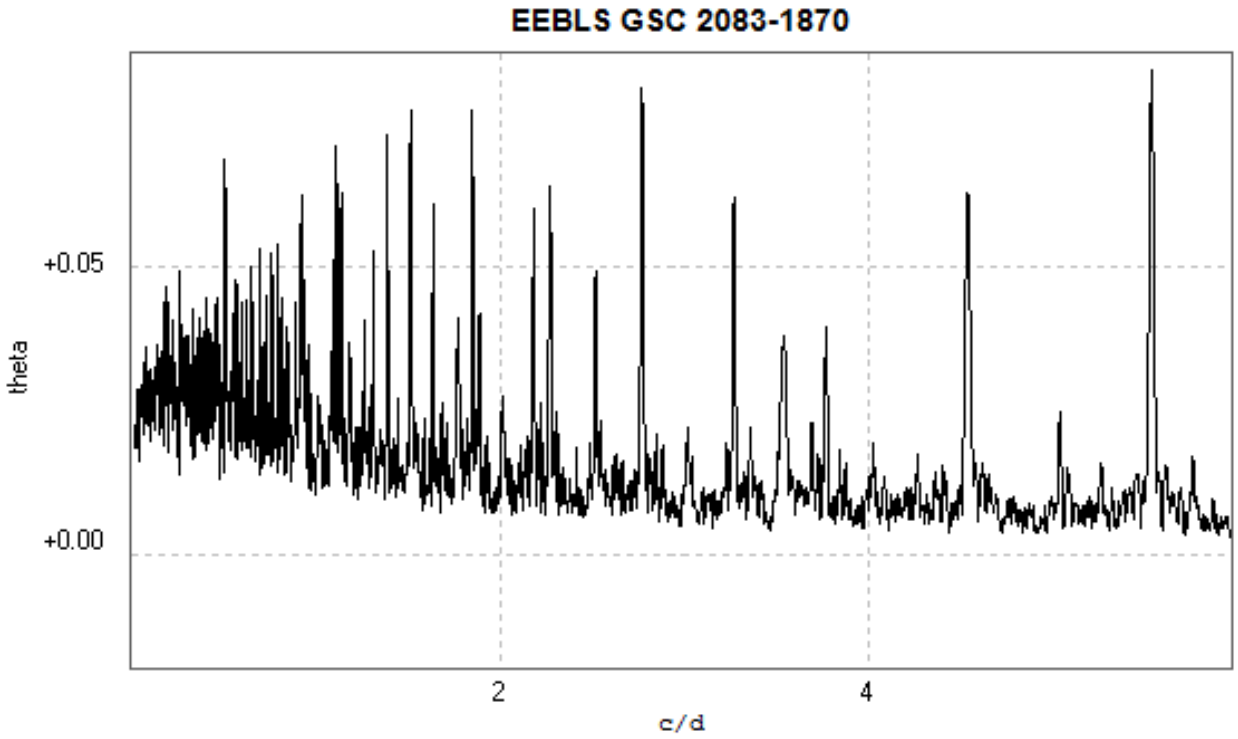

Figure 6.2: *Perodigram for GSC 2083-1870.*

In order to more easily distinguish repeated changes in brightness, the data points are “folded” to the period. This creates a phase diagram. If a transit is observed, part of the curve will show a lower magnitude than the rest. Figure 6.3 illustrates the phase diagram for GSC 2083-1870 for a period of 0.3608 days. It is a binary star because of its depth and the “V” shape of its magnitude reduction.

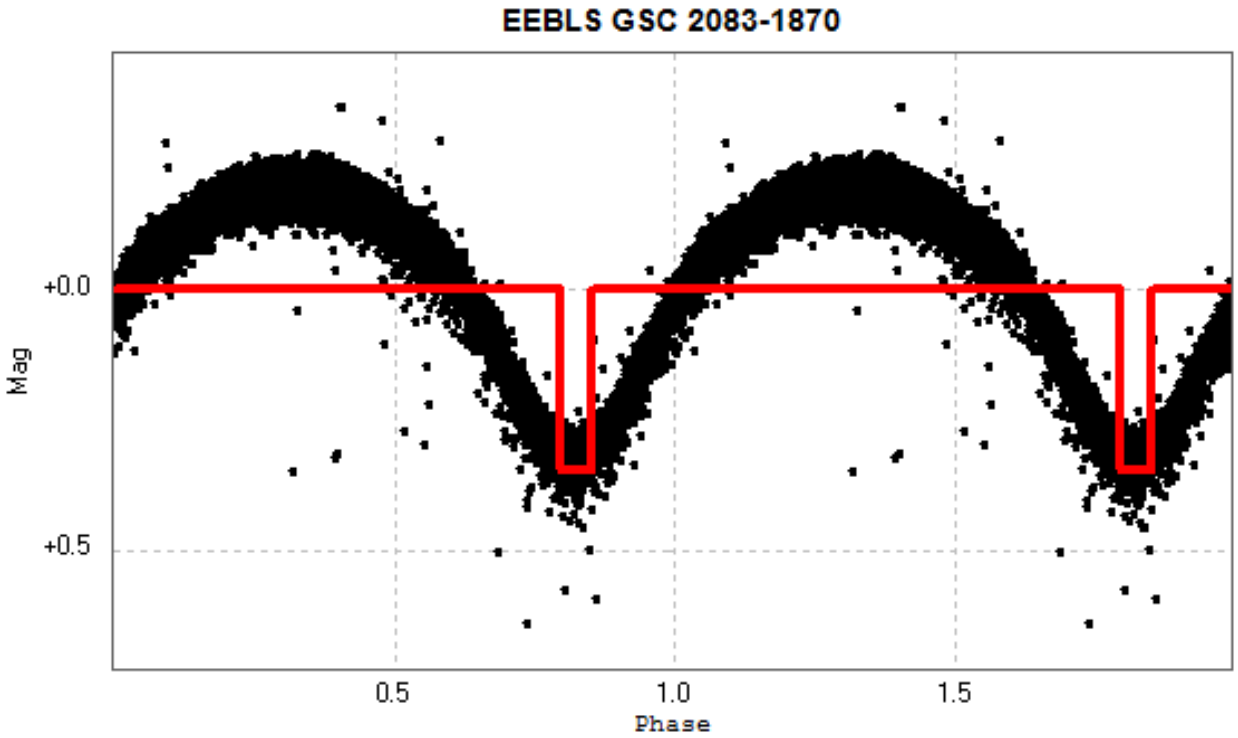

Figure 6.3: *Phase diagram for GSC 2083-1870 with a box superimposed on it.*

Figure 6.4 is the light curve for GSC 2084-0455. It is apparent that there are variations in the brightness of the star.

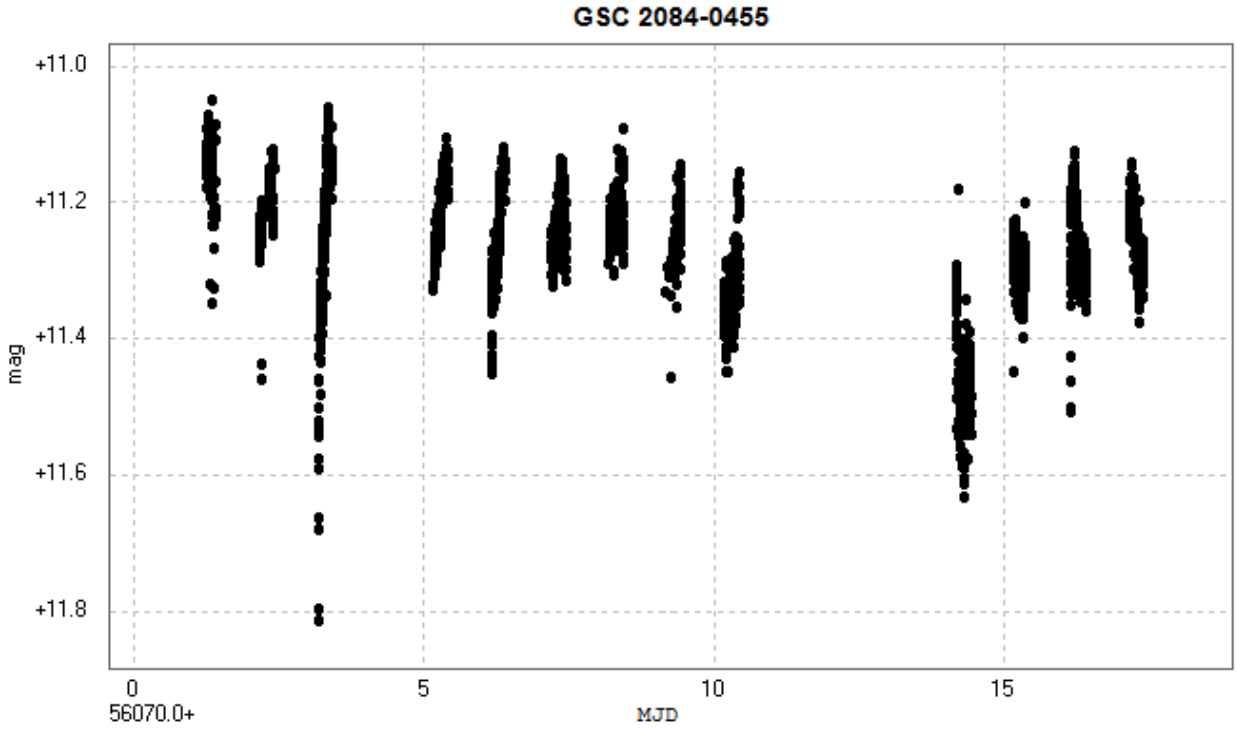

Figure 6.4: *Light curve for GSC 2084-0455.*

Figure 6.5 is the EEBLS periodigram for GSC 2084-0455 showing a signal at a frequency of 0.1703 c/d or a period of 5.6085 days.

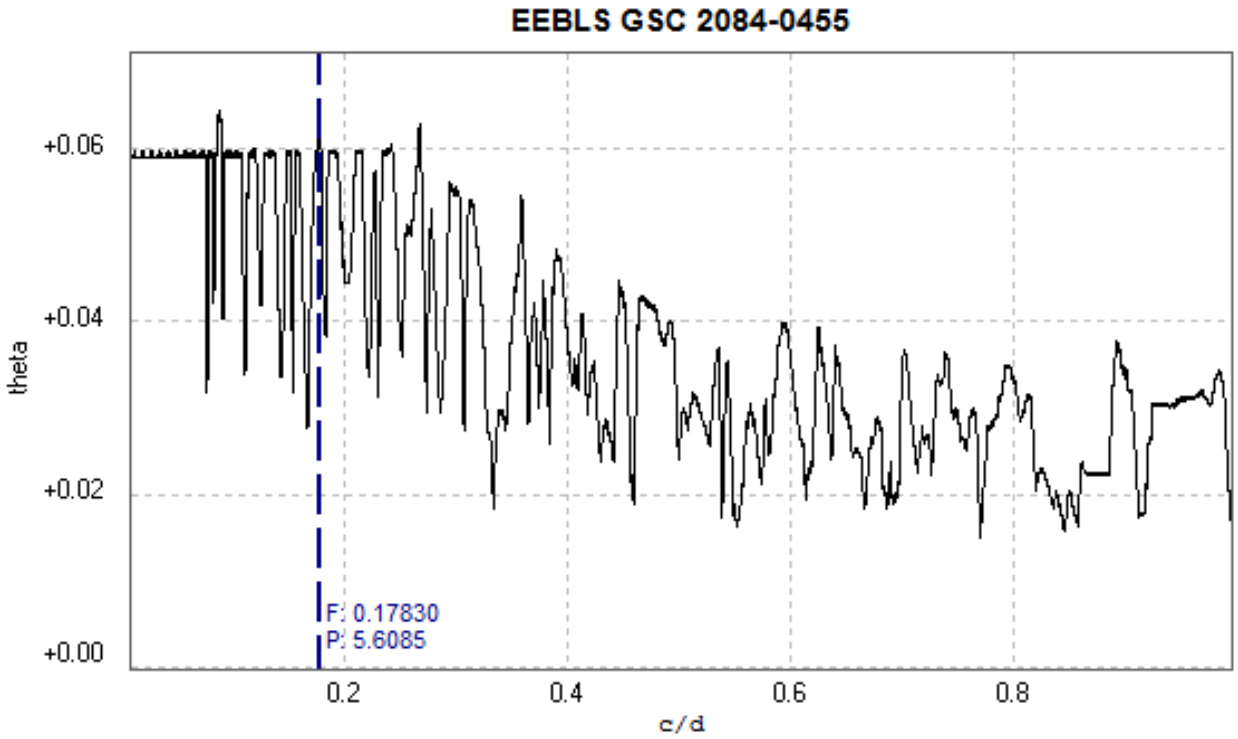

Figure 6.5: *Periodigram for GSC 2084-0455.*

Figure 6.6 is the phase diagram for GSC 2084-0455 with a period of 5.6085 days. It is a binary star because of its depth and the “V” shape of its magnitude reduction. EEBLS was able to fit a box to the major decrease in magnitude near 0.3 phase, however, it did not locate the two “dips” near 0.7 and 0.9 phase.

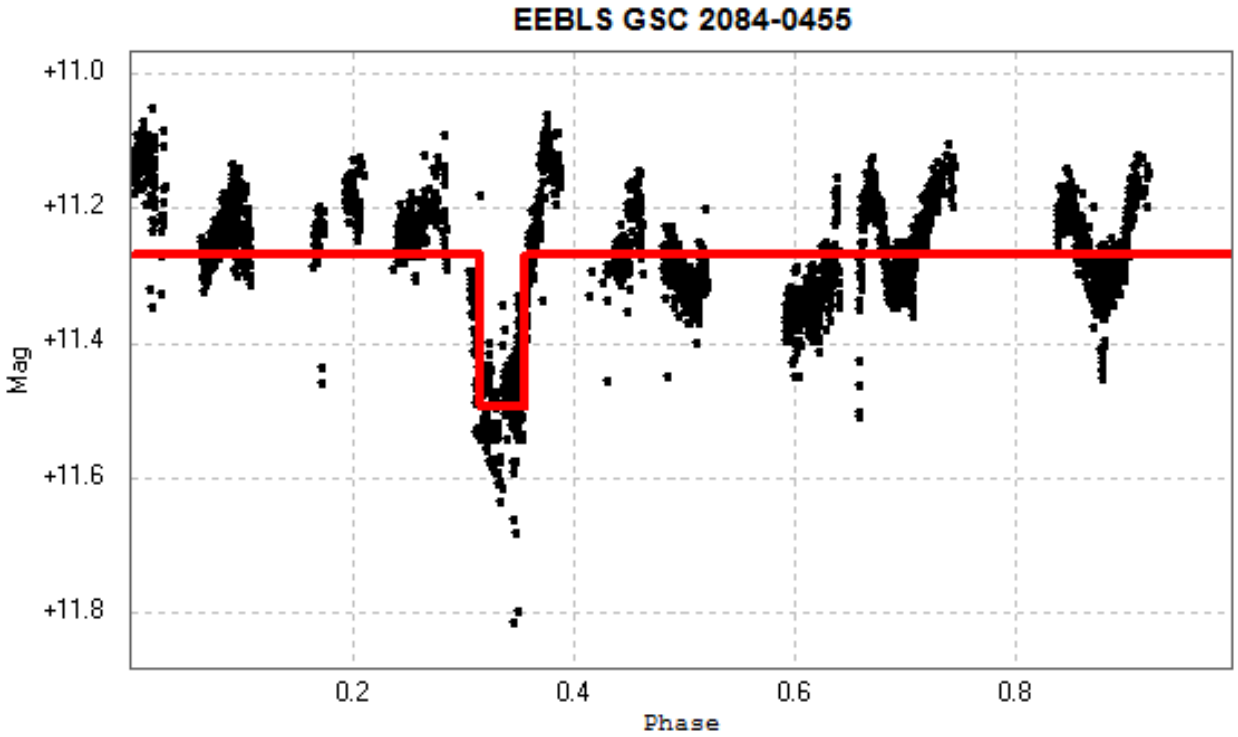

Figure 6.6: *Phase diagram for GSC 2084-0455 with a box superimposed on it.*

Figure 6.7 is the light curve for GSC 2083-0557. Major changes in the star's brightness can be seen near MJD 17 and 32.

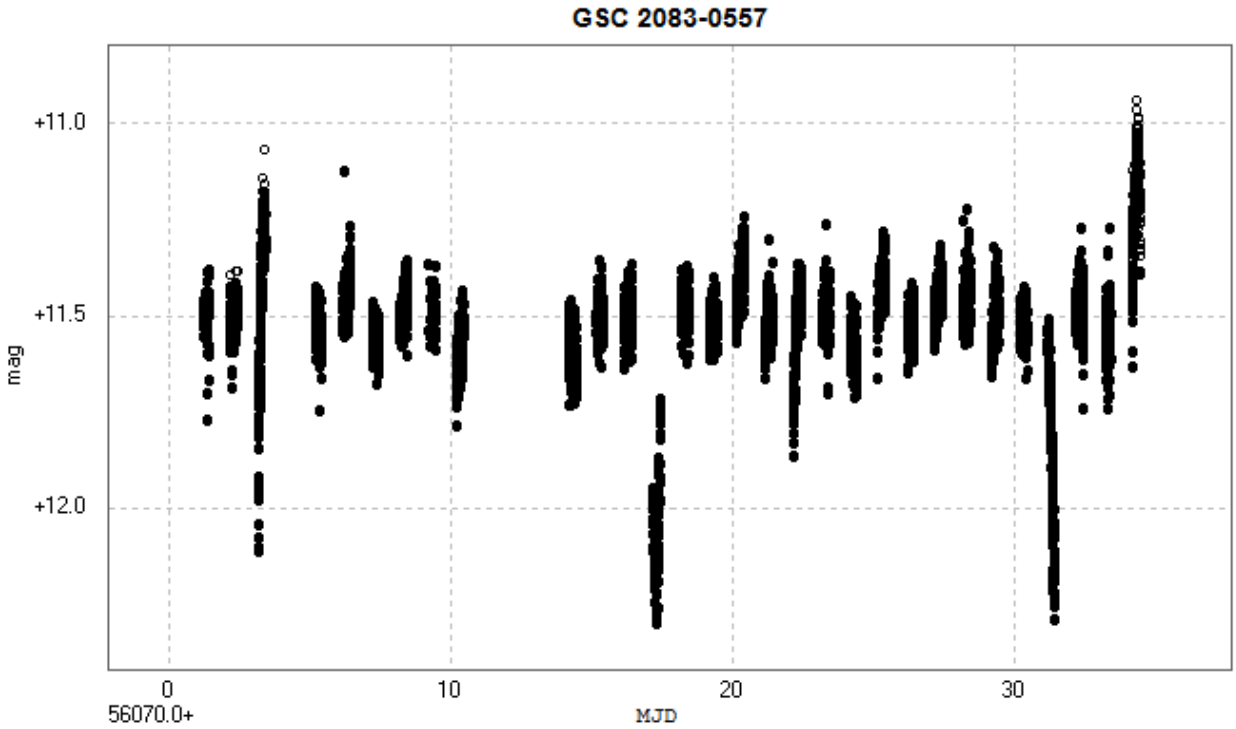

Figure 6.7: *Light curve for GSC 2083-0557.*

Figure 6.8 is the periodigram for GSC 2083-0557 showing a signal at a frequency of 0.21142 c/d or a period of 4.7299 days.

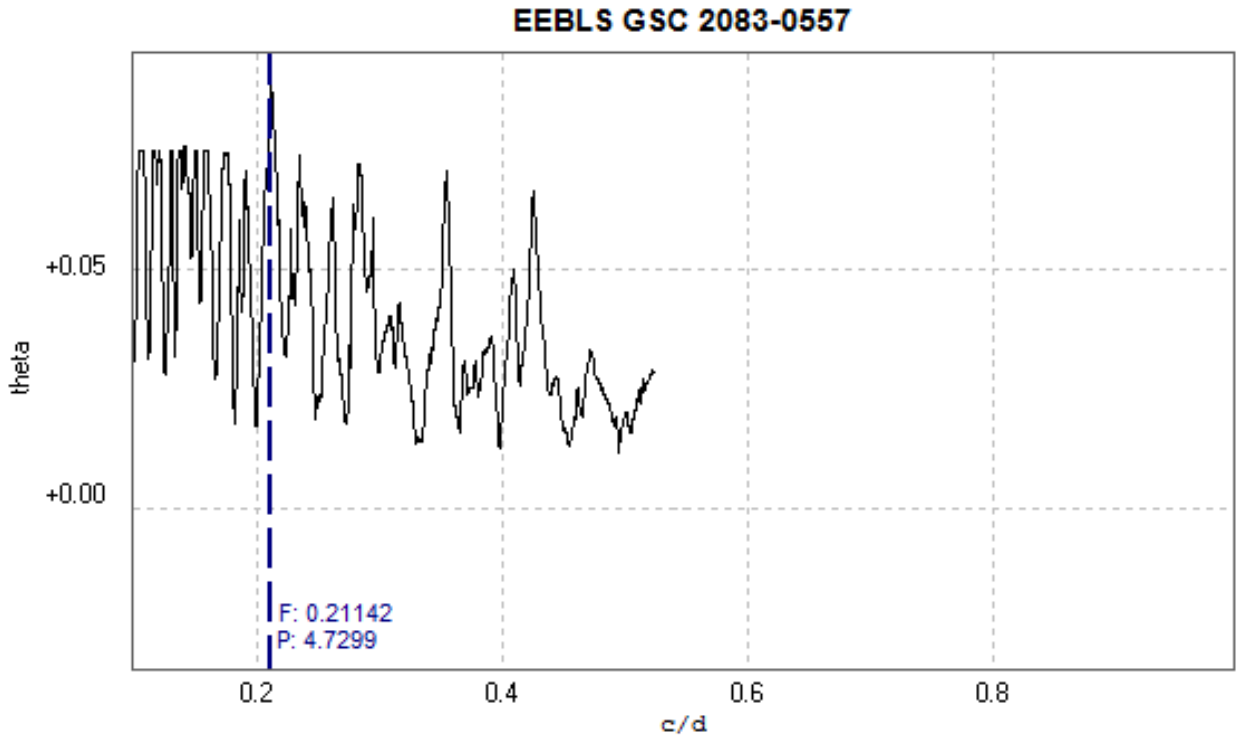

Figure 6.8: *Periodigram for GSC 2083-0557.*

Figure 6.9 is the phase diagram for GSC 2083-0557. This diagram most likely represents a binary star because of its "V" shape and because its depth is much greater than 50 millimagnitudes. The superimposed box has a depth of 0.5 magnitudes and width of 0.2 of the orbital period.

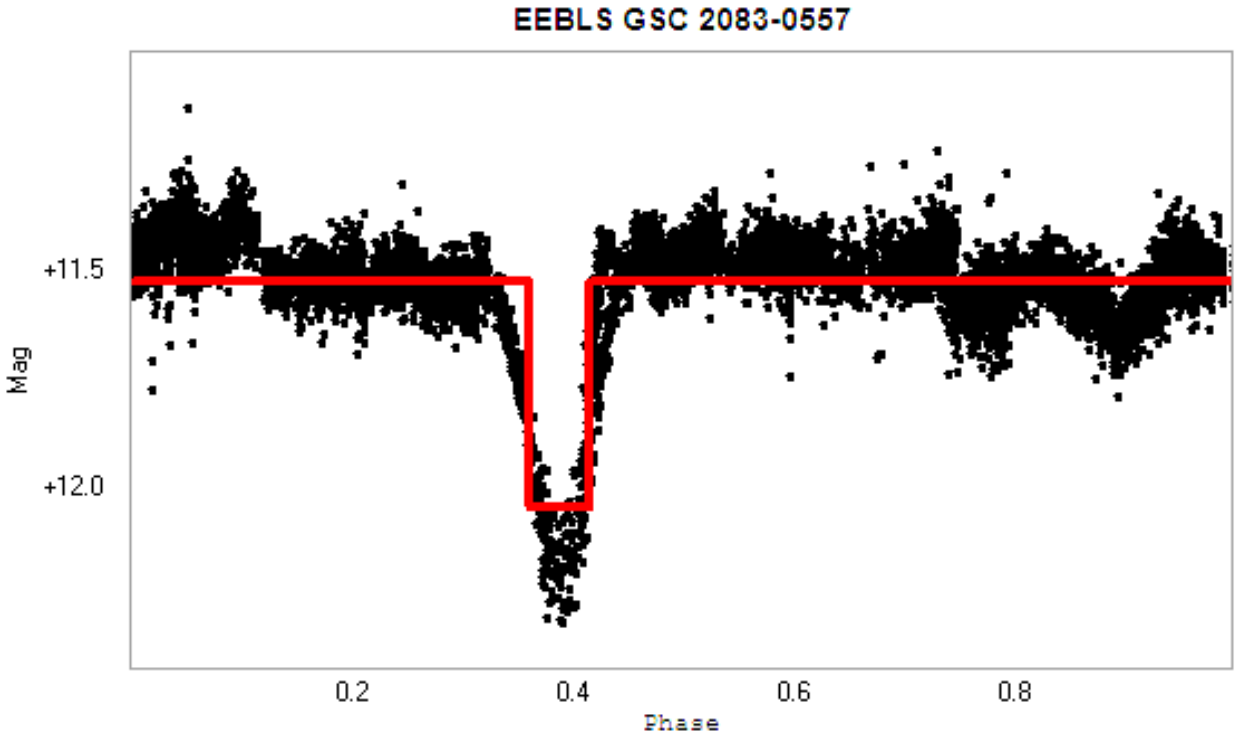

Figure 6.9: *Phase diagram for GSC 2083-0557 at a period of 4.7299 days with a box superimposed on it.*

7 Conclusion

The EEBLS algorithm has aided us in finding at least 3 binary candidates, GSC 2083-1870, GSC 2084-0455, and GSC 2083-0557. The output for these are shown and discussed in the results section.

From this evaluation, we learned that the EEBLS algorithm is useful as a guide. It significantly narrows down the number of possible transit candidates. Many more can be eliminated by hand from the output of the algorithm. For instance, the output of the depth of the superimposed box on the data can be used to eliminate transit candidates. Positive depths are not considered transit candidates because they signal brightness increases rather than decreases.

However, the EEBLS algorithm does not produce a definitive list of transits. The output must be further examined by hand and graphs of the data from transit and binary candidates must be visually inspected.

One of the things that can be done to further this research is automate some of the inspection of the EEBLS algorithm that is currently done by hand. A script can be written to remove stars from the output with positive depth. A program could also be written to eliminate groups of stars whose period exactly match. This is usually caused by events on earth such as dust, clouds, or telescope imperfections.

8 References

[1] Haswell, Carole A. *Transiting Exoplanets*. Cambridge: Cambridge UP, 2010. Print.

[2] http://www.nasa.gov/mission_pages/kepler/overview/index.html

[3] http://www.astromatic.net/software/sextractor

[4] A box-fitting algorithm in the search for periodic transits

Kovács, G.; Zucker, S.; Mazeh, T.

Astronomy and Astrophysics, v.391, p.369-377 (2002)

9 Appendix A: EEBLS

```
//
//
//
//--------------------------------------------------------------------
//     >>>>>>>>>>>> This routine computes BLS spectrum <<<<<<<<<<<<<
//
//         [ see Kovacs, Zucker & Mazeh 2002, A&A, Vol. 391, 369 ]
//
//     This is the slightly modified version of the original BLS routine
//     by considering Edge Effect (EE) as suggested by
//     Peter R. McCullough [ pmcc@stsci.edu ].
//
//     This modification was motivated by considering the cases when
//     the low state (the transit event) happened to be devided between
//     the first and last bins. In these rare cases the original BLS
//     yields lower detection efficiency because of the lower number of
//     data points in the bin(s) covering the low state.
//
//     For further comments/tests see  www.konkoly.hu/staff/kovacs.html
//--------------------------------------------------------------------
//
//     Input parameters:
//     ~~~~~~~~~~~~~~~~~
//
//     n    = number of data points
//     t    = array {t(i)}, containing the time values of the time series
//     x    = array {x(i)}, containing the data values of the time series
//     u    = temporal/work/dummy array, must be dimensioned in the
//            calling program in the same way as  {t(i)}
//     v    = the same as  {u(i)}
//     nf   = number of frequency points in which the spectrum is computed
//     fmin = minimum frequency (MUST be > 0)
//     df   = frequency step
//     nb   = number of bins in the folded time series at any test period
//     qmi  = minimum fractional transit length to be tested
//     qma  = maximum fractional transit length to be tested
//
//     Output parameters:
//     ~~~~~~~~~~~~~~~~~~
//
//     p    = array {p(i)}, containing the values of the BLS spectrum
//            at the i-th frequency value -- the frequency values are
//            computed as  f = fmin + (i-1)*df
//     bper = period at the highest peak in the frequency spectrum
//     bpow = value of {p(i)} at the highest peak
//     depth= depth of the transit at   *bper*
```

```
//     qtran= fractional transit length  [ T_transit/bper ]
//     in1  = bin index at the start of the transit [ 0 < in1 < nb+1 ]
//     in2  = bin index at the end   of the transit [ 0 < in2 < nb+1 ]
//
//
//     Remarks:
//     ~~~~~~~~
//
//     -- *fmin* MUST be greater than  *1/total time span*
//     -- *nb*   MUST be lower than  *nbmax*
//     -- Dimensions of arrays {y(i)} and {ibi(i)} MUST be greater than
//        or equal to  *nbmax*.
//     -- The lowest number of points allowed in a single bin is equal
//        to   MAX(minbin,qmi*N),  where   *qmi*  is the minimum transit
//        length/trial period,   *N*  is the total number of data points,
//        *minbin*  is the preset minimum number of the data points per
//        bin.
//
//=======================================================================
//

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>

void eebls(int n, double t[], double x[], double u[], double v[], int nf,
         double fmin, double df, int nb, double qmi, double qma,
         /* Output variables */ double p[], double* bper, double* bpow,
         double* depth, double* qtran, int* in1, int* in2){

  // variables starting with a-h or o-z are "double"
  // implicit real*8 (a-h,o-z)
  //
  // dimension t(*),x(*),u(*),v(*),p(*)
  //   These were declared arrys inside the function prototype

  double y[2000];
  int ibi[2000];
  int i, jf;                 /* Counters I've added, used later */
  int minbin;
  int nbmax;
  double tot;
  double rn;
  int kmi;
  int kma;
  int kkmi;
  int nb1;
  int nbkma;
  double s;
  double t1;
```

```
double f0;
double p0;
int j;
double ph;
int jnb;
double power;
int k;
int kk;
int nb2;
double rn1;
double pow;
int jn1;
int jn2;
double rn3;
double s3;

minbin = 5;
nbmax  = 2000;
if(nb > nbmax)
  printf(" NB > NBMAX !!\n");
if(nb > nbmax)
  exit(0);

tot = t[n] - t[1];
if(fmin < 1.0/tot)
  printf(" fmin < 1/T !!\n");
if(fmin < 1.0/tot)
  exit(0);
//------------------------------------------------------------------------
//
rn = (double)n;
kmi = (int)(qmi*((double)nb));
if(kmi < 1)
  kmi = 1;
kma = qma * ((double)nb)+1;
kkmi = (int)(rn * qmi);
if(kkmi < minbin)
  kkmi = minbin;
*bpow=0.0;
//
//     The following variables are defined for the extension
//     of arrays  ibi()  and  y()  [ see below ]
//
nb1   = nb+1;
nbkma = nb+kma;
//
//================================
//     Set temporal time series
//================================
//
s = 0.0;
```

```
t1 = t[1];
for(i = 1; i <= n; i++){
  u[i] = t[i] - t1;
  s = s + x[i];
}

s = s / rn;

for(i = 1; i <= n; i++){
  v[i] = x[i] - s;
}
//
//*****************************
//     Start period search     *
//*****************************
//

for(jf=1; jf <= nf; jf++){
  f0 = fmin + df * (double)(jf - 1);
  p0 = 1.0 / f0;
  //
  //=======================================================
  //     Compute folded time series with  *p0*  period
  //=======================================================
  //
  //############################################################

  for(j = 1; j <= nb; j++) {
    y[j] = 0.0;
    ibi[j] = 0;
  }

  for(i = 1; i <= n; i++) {
    ph = u[i]*f0;
    ph = ph - floor(ph);
    j = 1 + floor(nb*ph);
    ibi[j] = ibi[j] + 1;
    y[j] = y[j] + v[i];
  }

  //-----------------------------------------------
  //     Extend the arrays  ibi()  and  y() beyond
  //     nb   by  wrapping

  for(j = nb1; j <= nbkma; j++) {
    jnb = j - nb;
    ibi[j] = ibi[jnb];
    y[j] = y[jnb];
  }
  //-----------------------------------------------
```

```
//=============================================
//     Compute BLS statistics for this period
//=============================================

power = 0.0;

for(i = 1; i <= nb; i++) {
  s = 0.0;
  k = 0;
  kk = 0;
  nb2 = i + kma;
  for(j = i; j <= nb2; j++) {
    k = k + 1;
    kk = kk + ibi[j];
    s = s + y[j];

    if(k < kmi) {
      continue;
    }

    if(kk < kkmi) {
      continue;
    }

    rn1 = (double)(kk);
    pow = s*s/(rn1*(rn-rn1));

    if(pow < power) {
      continue;
    }

    power = pow;
    jn1 = i;
    jn2 = j;
    rn3 = rn1;
    s3 = s;
  }
}

power = sqrt(power);
p[jf] = power;

if(power < *bpow) {
  continue;
}

*bpow = power;
*in1 = jn1;
*in2 = jn2;
*qtran = rn3/rn;
*depth = -s3*rn/(rn3*(rn - rn3));
```

```
    *bper = p0;

  }

  //     Edge correction of transit end index

  if(*in2 > nb) {
    in2 = in2 - nb;
  }
}
```

10 Appendix B: EEBLS Driver

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<omp.h>

#define MAXLEN 10000
#define MAX_NUM_STARS 10000
#define MAX_NUM_READINGS 10000

void eebls(int n, double t[], double x[], double u[], double v[], int nf,
         double fmin, double df, int nb, double qmi, double qma,
         /* Output variables */ double p[], double* bper, double* bpow,
         double* depth, double* qtran, int* in1, int* in2);

int main(int argc, char* argv[]){

  FILE *fptr = fopen(argv[1], "r");
  char headline[MAXLEN];
  char starNames[MAX_NUM_STARS][50];
  int  numStars = 0;
  printf("Opening %s\n", argv[1]);

  // Read the first line of the file, the tops of the columns
  fgets(headline, MAXLEN, fptr);
  if(headline[strlen(headline) - 1] == '\n') /* strip trailing newline */
    headline[strlen(headline) - 1] = '\0';

  printf("Opened file: %s\n", argv[1]);

  // Read the names of the stars.
  char* data;
  int counter = 0;
  data = strtok(headline, ",");
  data = strtok(NULL, ",");
  while(data != NULL){
    strcpy(starNames[counter], data);
    counter++;
    data = strtok(NULL, ",");
  }
  numStars = counter;
  printf("Found %d stars.\n", numStars);

  // Now read all the time and brightness data;
  int numDataPoints = 0;      /* Number of timeslices */
  char dataLine[MAXLEN];      /* Holds one timeslice of data */
```

```
  double timeStamps[MAX_NUM_READINGS];
  double dataValues[MAX_NUM_STARS][MAX_NUM_READINGS];
  int i;                /* Counters */
  while(fgets(dataLine, MAXLEN, fptr) != NULL){
    timeStamps[numDataPoints] = atof(strtok(dataLine, ","));
    for(i = 0; i < numStars; i++)
      dataValues[i][numDataPoints] = atof(strtok(NULL, ","));
    numDataPoints++;
  }
  int numReadings = numDataPoints;
  printf("Read %d time-values for each star.\n", numReadings);

  /*
   * At this point in the code:
   * numStars    holds the correct number of stars
   * numReadings holds the correct number of readings (rows of the CSV)
   * timeStamps  holds the times, the first column of the CSV
   * dataValues  holds the brightness values: dataValues[i] for ith star.
   *
   * Uncomment the lines below if you want to see a printout
   */

  /*
    int j;
    for(i = 0; i < numReadings; i++){
    for(j = 0; j < numStars; j++){
    printf("%1.20lf ", dataValues[i][j]);
    }
    printf("\n");
    }
  */

  /*
   * Now we will apply the eebls function to each star.
   */
  // Set / Allocate input structures and values
  double* u = (double*)malloc(numReadings * sizeof(double));
  double* v = (double*)malloc(numReadings * sizeof(double));
  int     numFreqPts = 1000; // # freq. points in which the spectrum is
computed
  double  tot        = timeStamps[numReadings-1] - timeStamps[0];
  double  freqMin    = 1.0/tot;
  double  freqStep   = 0.001;
  int     numBins    = 100;
  double  qMin       = 0.01;
  double  qMax       = 0.04;

  // Set / Allocate output structures and values
  double* p = (double*)malloc(numReadings * sizeof(double));
  double bper  = 0;
  double bpow  = 0;
```

```
  double depth = 0;
  double qtran = 0;
  int    in1   = 0;
  int    in2   = 0;

  printf("Star\tbper\tbpow\tdepth\tqtran\tin1\tin2\tT\tIP\tEP\n");
#pragma omp parallel for num_threads(8)
  for(i = 0; i < numStars; i++){
    eebls(numReadings, timeStamps, dataValues[i], u, v, numFreqPts,
        freqMin, freqStep, numBins, qMin, qMax,
        /* Output variables */ p, &bper, &bpow, &depth, &qtran, &in1, &in2);

    double T = 2.0*bper*qtran;
    double dummy;
    printf("%s\t%f\t%f\t%f\t%f\t%d\t%d\t%f\t%f\t%f\n",
         starNames[i], bper, bpow, depth, qtran, in1, in2,
         T, modf(in1 * tot / numBins / T, &dummy), modf(in2 * tot / numBins /
T, &dummy));

  }

  return 0;
}
```

11 Appendix C: SYSREM

```
###############################################################################
#sysrem.py
#Application of the SYSREM by Tamuz et al.
###############################################################################

##
##  Modified to analyze a directory of CSV files rather than a single file
##
##  Modified by Matt Heuser
##

## Import necessary libraries

import pylab
from pylab import*
import scipy
import scipy.interpolate
import numpy
from numpy import*
from math import*
import csv
import os
import string

## Read the CSV file into an array

indirname = raw_input("Please specify the name of the input directory: ")
print "Please wait..."

indir = "./" + indirname + "/"
dirlist = os.listdir(indir)

for filename in dirlist:
    sfilename = filename.rstrip(".csv")
    v = genfromtxt(indir + filename, delimiter = ",", filling_values = 0.0,
skip_header = 1)


    ## Use the length of v to set the number of stars N-3  (first three
columns have time, airmass, zeropoint
    N = len( v[0] ) #We want the length of one row of v to get the number of
columns (i.e. stars)

    ## Take a slice (second column) which contains all the air masses.  These
are functions of date but are the same for each star N
    a = v[:,2]
```

```
    ## Pull out MJD column using a slice of v:
    MJD = v[:,0]

    ## Use the length of a to set the number of measurements M per star (i.e.
rows)
    M = len(a)

    ## Read the zero points for each time into an array by slicing v
    zp = v[:,1]

    ## Create arrays to handle the calculations

    c_old = zeros(N, float)     #This is the extinction coefficient
    c_new = zeros(N, float)     #This is the redundant extinction coefficient
used iteratively
    c_dif = zeros(N, float)
    a_old = zeros(M, float) #airmass used for iteration
    a_new = zeros(M, float) #airmass used for iteration
    a_dif = zeros(M, float)
    s = zeros((M,N), float) #This contains the errors (standard deviations)
for each measurement so it is an MxN array
    m = zeros((M,N), float)     #This contains the observed magnitudes for
each time
    m2 = zeros((M,N), float)    #This will contain the corrected magnitudes
for each time
    r = zeros((M,N), float) #These are the residuals from fitting the data
(magnitude vs. airmass) to a straightline
    mean = zeros(N, float)

    ## Calculate the error in the measurements for each star and place them in
the NxM array
    ##I simplified this so it didn't create an intermediate array
    for i in range(0,M): #Switched N and M in this nested loop (I think our
array is the transpose of the article's). For M dimension, we don't want to
skip the first three elements, changed below, too
        for j in range(3,N):
            if v[i,j]>0 or v[i,j]<0: s[i,j] =
2.5/log(10.0)/sqrt(60.0*pow(10.0, -(v[i,j]-zp[i])/2.5))  #calculate
uncertainty in star i measurement
            else: s[i,j] = 999999 #Since nan was not being handled, I set the
empty values to 999999 and just skip that whenever it comes up later

    ##I think we need the residuals before doing the fitting
    for j in range(3, N):
        sum_v=0.
        count=0.
        for i in range(0,M):
            if v[i,j]>0 or v[i,j]<0:
                m[i,j] = v[i,j]
                sum_v+=v[i,j]
```

```
                count+=1.
        mean[j]=sum_v/count
        for i in range(0,M):
            r[i,j]=v[i,j]-mean[j] #subtracts the mean magnitude from each
measurement, giving residual

    for j in range(3, N):
        a_temp=[] #create temp arrays so as not to include the empty elements
of r
        r_temp=[]
        for i in range (0,M):
            if r[i,j]>0 or r[i,j]<0:
                a_temp.append(a[i])
                r_temp.append(r[i,j])
        polycoeffs = scipy.polyfit(a_temp, r_temp, 1)   #makes a straight line
fit to the data f(a,r)
        c_old[j] = polycoeffs[0] #stores the slope of the line as the
extinction coefficient


    ##Set differences to be large before entering interation loop
    for j in range(3,N): c_dif[j]=1e9
    for i in range(0,M): a_dif[i]=1e9

    avg_c_dif=sum(c_dif)/len(c_dif)
    avg_a_dif=sum(a_dif)/len(a_dif)
    a_old=a
    count=0

    ##Iterative process to minimize airmass and coefficient of extinction
    while avg_c_dif>0.001 or avg_a_dif>0.001: #calculates the new extinction
coefficients and airmasses until there is little change
        for j in range(3, N): #extinction coefficients
            denom = 0.
            coeff = 0.
            for i in range(0, M):
                if s[i,j] != 0 and s[i,j] != 999999: denom +=
a_old[i]*a_old[i]/(s[i,j]*s[i,j])
            for i in range(0, M):
                if s[i,j] != 0 and s[i,j] != 999999:
                    coeff += r[i,j]*a_old[i]/(s[i,j]*s[i,j])
            c_new[j]=coeff/denom
            c_dif[j]=abs(c_new[j]-c_old[j]) #determines the amount of change
in this iteration
            c_old[j]=coeff/denom

        for i in range(0,M): #airmasses
            denom = 0.
            coeff = 0.
            for j in range(3, N):
                if s[i,j] != 0 and s[i,j] != 999999:
```

```
                    denom += c_old[j]*c_old[j]/(s[i,j]*s[i,j])
            for j in range(3, N):
                if s[i,j] != 0 and s[i,j] != 999999:
                    coeff += r[i,j]*c_old[j]/(s[i,j]*s[i,j])
            a_new[i]=coeff/denom
            a_dif[i]=abs(a_new[i]-a_old[i])
            a_old[i]=coeff/denom
        avg_c_dif=sum(c_dif)/(N-3) #average change per element in this
iteration
        avg_a_dif=sum(a_dif)/len(a_dif)
        print count,avg_c_dif,avg_a_dif
        count+=1


    for i in range(0,M):
        for j in range(3,N):
            r[i,j]=r[i,j]-c_old[j]*a_old[i]#subtracting the systematic error
from the residual
            m2[i,j] =  r[i,j] + mean[j]  #Corrected 27 Nov. 2011



###############################################################################
    for j in range(3, N):
        d = v[:,0]
        m3 = m2[:,j]
        pylab.subplot(211)
        pylab.figure(1)
        #spl = scipy.interpolate.splrep(d, m3, s=0.4*std(m3)) #reduced from
0.5 on 27 Nov. 2011
        #y = scipy.interpolate.splev(d, spl)
        #m2[:,j] = y
        y = m2[:,j]
        #pylab.plot(d, y ,'bo')   #plot magnitude as a function of time
        #pylab.errorbar(d,y, yerr = std(y), fmt = 'bo')
        #pylab.plot(v[:,0], m[:,j], 'ro')
        pylab.errorbar(v[:,0],m2[:,j], yerr = std(m2[:,j]), fmt = 'bo')
        #pylab.ylim(mean[j] - 0.05, mean[j] + 0.05)
        ax = gca()
        ax.set_ylim(ax.get_ylim()[::-1])   #reverse the y-axis direction
        pylab.grid(True)
        f = file(indir + filename)
        reader = csv.reader(f)
        name = reader.next()[j]
        f.close()
        pylab.title("SYSREM " + str(name) )
        pylab.xlabel("MJD")
        pylab.ylabel("magnitude")
        pylab.grid(True)
        pylab.subplot(212)
        #pylab.plot(v[:,0], m[:,j], 'ro')
```

```
    #pylab.plot(v[:,0], m2[:,j],'bo')
    #pylab.errorbar(v[:,0], m2[:,j], yerr = std(m2[:,j]), fmt = 'bo')
    pylab.errorbar(v[:,0], m[:,j], yerr = std(m[:,j]), fmt = 'ro')
    ax = gca()
    ax.set_ylim(ax.get_ylim()[::-1])   #reverse the y-axis direction
    pylab.xlabel("MJD")
    pylab.ylabel("magnitude")
    pylab.grid(True)
    if not os.path.exists("./sysrem/" + sfilename + "/"):
        os.makedirs("./sysrem/" + sfilename + "/")
    pylab.savefig("./sysrem/" + sfilename + "/" + "SYSREM-" + str(name))
    #pylab.show()
    pylab.clf()


## Save data to a csv file
if not os.path.exists("./sysrem/"):
    os.makedirs("./sysrem/")
out = open('./sysrem/' + filename, "w+")
print >> out, "MJD",",", "\t",
for j in range(3,N):
    f = file(indir + filename)
    reader = csv.reader(f)
    name = reader.next()[j]
    f.close()
    print >> out, "\t", name, ","."\t",
print >> out, "\n",
for i in range(0, M):
    print >> out, MJD[i], ",",
    for j in range (3, N):
        #if j != 1 or j!= 2:
        print >> out,   m2[i,j], ",",
    print >> out, "\n",
out.close()
print filename + "...done"
```