Automating the Data Reduction Process for Geosynchronous Satellite Spectra

Katrina Pedersen

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

Joseph Moody, Advisor

Department of Physics and Astronomy

Brigham Young University

August 2016

ABSTRACT

Automating the Data Reduction Process for Geosynchronous Satellite Spectra

Katrina Pedersen
Department of Physics and Astronomy, BYU
Bachelor of Science

Identifying the chemical composition of unknown satellites is of great interest for defense applications. A developing method of accomplishing this takes low-resolution spectra of satellites and deconvolves them using known material reflection and absorption properties. Using a suitable diffraction grating in the filter wheel slot, test spectra have been obtained over a variety of sun angles for various geostationary satellites using the ROVOR 16’’ RC Optical telescope. The data are encouraging but the analysis is cumbersome and time-intensive. We are developing a data analysis package based on the commercial image-processing software package *Mira*, to help automate the data reduction process. Using the *Mira* scripting language and its file event scripting capabilities, data can be automatically processed with limited user interaction. We report on progress to date.

Keywords: Geostationary satellites, spectra, ROVOR, *Mira*

ACKNOWLEDGMENTS

I would like to acknowledge Dr. J. Ward Moody who has given me the opportunity to take on this problem and improve my programming. I am grateful for his support, advice, and guidance. I am also grateful for the continued support and advice of my family. I am also grateful to all of those who have helped make it possible for me to attend Brigham Young University.

Contents

Chapter 1

Introduction

1.1 Importance of satellite identification

Over the past couple decades, various groups have begun analyzing satellites through myriad photometric techniques. There are two predominant motivations for this research. First, we want to identify unknown satellites and space debris to improve our Space Object Identification (SOI). This will aid in detecting possible threats in Low Earth Orbits (LEOs). Second, we want to improve our Space Situational Awareness (SSA). That includes our ability to analyze solar panel offsets to gain information about the possible purpose of a satellite, and to monitor the health of known geostationary satellites (GEOS). Better SSA also allows us to monitor the space in which GEOS orbit to try to prevent collisions and the creation of more space debris. (Payne et al. 2006; Tucker 2015)

1.2 General definitions

Before delving into our research, we will review prior work to improve our identification and understanding of GEOS. Chapter 1 will provide the background necessary to understand the significance of our research and progress. Reflectance spectroscopy is not the only method used for analyzing GEOS, but it is one of the most promising techniques for analyzing space debris (Schildknecht 2009), so it is likely that it is also invaluable to the GEOS identification process. Reflectance spectroscopy allows us to measure changes in the magnitude of reflected light for specific wavelengths. It allows us to learn about satellite composition and size (Bédard 2012).

To avoid any confusion, here is a list of potentially useful definitions:

1.) Active attitude control system (or active ACS): some form of software or hardware that can change or control the orientation of the satellite.
2.) Body reference frame: the reference frame of the body, or bus, of a GEOS. In this frame, the perpendicular direction would be perpendicular to the main bus of the satellite.
3.) Glint: the sudden increase in reflected light that occurs when the phase angle (PA) bisector is approximately parallel to the normal vector to the solar panels. This is similar to a glint caused by reflectance from a mirror when the PA bisector is parallel to the normal of the mirror's surface.
4.) Inertial reference frame: the reference frame of the solar panels of a GEOS. In this frame, zero degrees would be perpendicular to the panels. (The inertial and body reference frames are generally not identical.)
5.) Normal: unless otherwise specified, the normal vector to the solar panels.
6.) Phase angle (PA): the angle between the Sun, GEO, and sensor. It ranges from 0° to 180° generally, but some conventions also use -90° to 90°. At 0°, the Sun, satellite, and sensor are all in the same plane and the sensor and satellite are both on the far side from

the Sun (similar to a full moon). At 180°, they are all in the same plane, but the satellite is on the opposite side of the Earth as the sensor (similar to a new moon).

7.) Solar panel pointing offset: the degree by which solar panels are offset from the sun.

8.) Sun-object-sensor (SOS) geometry: refers to the angles created by sunlight reflecting off various parts of the satellite toward the telescope. This often just refers to the angle created by the sun-solar panel-sensor geometry.

1.3 Satellite identification classes

In a study done by Payne et. al in 2006, a survey of 36 GEOS was taken and their photometric signatures were analyzed over a wide range of PAs. They discovered that all satellites had PA offsets between ±10°, but most were centered near 0°. In addition, all satellites fell into one of five classes, but 78% of these satellites fell into one of two classes (the Canonical and the A2100 Class). These classes help give preliminary information about satellite shape and can be described as follows:

1.) Canonical Class: photometric peak around 0° PA, and no other major peaks. Satellites of this type have solar panels that pivot about the major axis of the satellite. The normal points directly at us at 0° PA, and we see the satellite edge-on at 90° PA.

2.) A2100 Class (named after the A2100 bus type): local minimum at 0° PA. There is a smooth increase in brightness from 0° to approximately ±40° followed by a smooth decrease at higher PAs.

3.) Telstar Class: underlying Canonical signature with secondary peaks around ±40° PA.

4.) BSS702C Class: underlying Canonical signature with secondary peaks around ±60° PA. These secondary peaks are brighter and broader than those of the Telstar Class and are attributed to solar panel concentrators.

5.) Peculiar Class: characterized by a lack of symmetry or sudden changes in brightness or both.

It is clear why many satellites would have 0° PA offsets- to provide maximum energy production from the panels. Payne et. al (2006) also provided possible reasons for the solar panel offsets. Younger satellites may be purposefully offset, so that they do not produce too much electricity early on. Solar panel offsets may also be an attempt at stabilizing the satellite by re-directing radiation pressure forces. Lastly, they suggest that if sensors that control the ACS degrade unevenly, balancing voltages on the panels will naturally lead to offsets.

The most practical result of their work is that only three brightness measurements are required to differentiate the Canonical from the A2100 Class. Small variations in PA brightness may also provide information about details of the bus or panels, or possible malfunctions in the ACS. The classification system is still a work in progress, but measuring PA dependence can be a fast way to determine the general shape and health of a satellite. (Payne et. al 2006)

1.4 Glint observations

Glint observations are extremely useful because they can give insight into the size, segmentation, alignment, and energy generation and consumption capabilities of GEOS. Often, satellites also produce secondary glints that likely come from elements of the bus. In a study done by Hall et. al, glints were found to shift by five minutes over a 14-day period. They found that if they tried graphing the glints with respect to the inertial reference frame, they did not line up. However, when they plotted the glints verses the body-reference frame, they did line up. This suggests that the glints they measured were actually due to a component of the body, rather than the solar panels. This may not be true in all cases, but for their measurements, this appears to be the case. One must first know information about the stabilization method used on the bus to be able to switch to the body reference frame. Even if that information is not available, this adds support to the theory that smaller glints may be caused by elements of the bus. With further research and data, it may be possible to use glints to determine solar panel offsets and bus type. (Hall et. al 2013)

Another study performed by Vrba et. al (2009) used photometric and interferometric data to analyze glints. Although interferometric findings were inconclusive, the authors did provide useful data regarding glints. Many glints are not visible because SOS angles of about ±8.5° are blocked by Earth's shadow. The eclipse season for GEOS lasts about 45 days and is centered on the vernal and autumnal equinoxes, making it difficult to obtain complete data from a glint. The glint season is centered about three weeks after the autumnal equinox and three weeks before the vernal equinox (in the northern hemisphere). Glints generally cover a circular area of about a 320 km diameter, and take about two minutes to pass over that area. This study found that glints tend to be dimmer and last longer than we expect (i.e., longer than two minutes), suggesting that glints are not caused by perfectly flat facets. The glint of DTV9-S lasted about one hour, instead of two minutes. Solar array sub-panels at angles of ±3.75° could have caused this, but the authors did not mention if DTV9-S is known to have sub-panels. Either way, measuring the timing of glints can give insight into the structures responsible for the glint. (Vrba et. al 2009)

In a recent study conducted by the Air Force, it was found that during a glint, more light is absorbed in the red end of the spectrum. Specifically, the peak of the spectra shifted from

~560-570 nm to about 515 nm during the glint. This would support the argument that solar panels are the main cause of the glints because solar panels should absorb more red light than blue light. They are designed to absorb sunlight, which is more towards the red end of the spectrum. So, we should expect to see a glint appear more in the blue. Although several groups have suggested that smaller glints and changes in spectrum brightness may help identify specific features of satellites, there are still a lot of steps that must be taken for us to reach that point. (Tucker 2015)

1.5 Determining shape

Hall et. al (2007) tried to determine a theoretical approach for separating attitude and body parameters, so that we may either use parallel computing or independent formulas to determine the shape and attitude of satellites. Using Mollweide projections of albedo-area product distributions, one can determine how many facets (or flat surfaces) are on the satellite. The projections can also help determine the materials used in building the satellite and how they are aligned with respect to one another. Unfortunately, this method does not work for concavities. However, Mollweide projections still yield useful results and can help us recognize structures that are convex. (Hall et. al 2007)

As noted previously, photometric PA measurements can provide insight into shape and the satellite class (Payne et. al 2002). In addition, glints give us information about the PA bisector, and therefore at least one facet normal. PA brightness distributions can also be used to determine shape, independent of attitude measurements. Every shape has a unique PA distribution, but it is difficult to model a GEO from its PA data. It is easier to compare a GEO's PA data to that of a hypothesized satellite to determine if there is a possible fit. Although PA measurements can help determine shape, they generally do not work as well for satellites with ACS. (Hall et. al 2007)

1.6 Determining attitude

It has been suggested that by inverting observational data and using parallel computing, we may first find facet-area distributions, or the shape of the satellite. Then, we could determine the attitude of the satellite. However, Hall et. al (2007) propose that it is more efficient to only determine one parameter at a time. To find attitude, independent of shape, they make a few suggestions. The first suggestion is to measure variations in synodic brightness to find a sidereal spin rate and spin axis. The periodicity in brightness only depends on spin rate and not shape, so we can find the attitude. Then, it is possible to map out different albedo-areas to determine the shape of the satellite. To use this method however, the satellite must not only be stable or slowly rotating, but it must also spin fast enough that the observation time is not shorter than one rotation period. In addition, the satellite must travel fast enough that synodic and sidereal periods are fairly different. (Hall 2007)

The second method proposed to determine attitude independent of shape is glint analysis. As aforementioned, glints and the timing of glints can help determine the attitude of solar panels and other body elements. Hall et. al also noted that brightening or dimming signatures of single-facets can be used to aid in determining attitude. The brightening or dimming of large facets occurs because of a gradual change in PA. It is suggested that when building a model of attitude parameters, if there are times when the SOS geometry should be the same, brightness measurements should be taken at those times. If those measurements are not very close, the attitude model is likely incorrect. (Hall et. al 2007)

1.7 Additional information

There have been several additional insights regarding general observations of GEOS and their spectra. Several groups of researchers have compared Earth-based spectral reflection measurements to those of satellites in orbit (Abercromby et. al 2006; Guyote et. al 2006). In the lab, backscattering can be measured, but it is unknown whether those results will still be visible from space (Abercromby et. al 2006). The spectral reflection measurements of GEOS in orbit are seen to be darker, and redder (Abercromby et. al 2006). This reddening, or increase in reflectance for redder wavelengths (above 700 nm), is not seen before GEOS leave Earth or after they return. So, reddening must be due to effects of the space environment. Abercromby et. al (2006) found that this reddening is material dependent, but independent of orbit or age. This reddening has still not been fully accounted for.

Another interesting observation is that red/blue ratios are different for each spacecraft (Bédard et. al 2012). This is not altogether surprising since most satellites do not have the exact same shape, attitude, or composition. However, Bédard et. al (2012) suggest that it could depend upon the operational age of the solar cells.

Lastly, some research has been done on three different methods of accounting for extinction. In the first method, the extinction curve was derived from observations of solar analog stars. That curve was used to correct the air mass of the calibration star to the same air mass as the object. The target object was then divided by the calibration star's new spectrum. In the second method, the object and solar analog star were both corrected to zero air mass by using a standard extinction curve. The target object was again divided by the solar analog. In the third method, the difference in air mass between the object and solar analog star was small, so no additional calibration was performed before dividing the two. Although the authors did not compare these methods, they claimed that primary consistency results were promising.

(Schildknecht et. al 2007)

Chapter 2

Methods

2.1 Telescope set-up

All data for this project has been taken using the Remote Observatory for Variable Object Research (ROVOR), which is a 16’’ RC Optical telescope on a Paramount ME. ROVOR is located 12 miles NW of Delta, Utah at 39° 27’ 17.1’’ N and 112° 43’ 01.0’’ W at an elevation of 1396 m. ROVOR uses a FLI Proline 1K 24 μ pixel SITe detector and a CFW-6-6 filter wheel- (see Moody et. al 2012). The dome is a 10’ by 10’ Lifferth design, as can be seen in Fig. 2.1. A Star Analyser 200 (SA-200) diffraction grating was placed in the filter wheel and aligned such that the satellite spectra and star trails are approximately parallel. Because the telescope is tracking with the GEOS, rather than the stars, stars appear elongated. Each image has an exposure time of six seconds and data is usually taken continuously through the night. Data is only taken on photometric nights and seeing of 3’’ is usual. ROVOR works with CCDSoft and TheSkyX to gather data, but *Mira* is used to reduce data.

Figure 2.1 The ROVOR observatory.

Figure 2.2 The Star Analyser 200.

Figure 2.3 A sample image of three satellites.

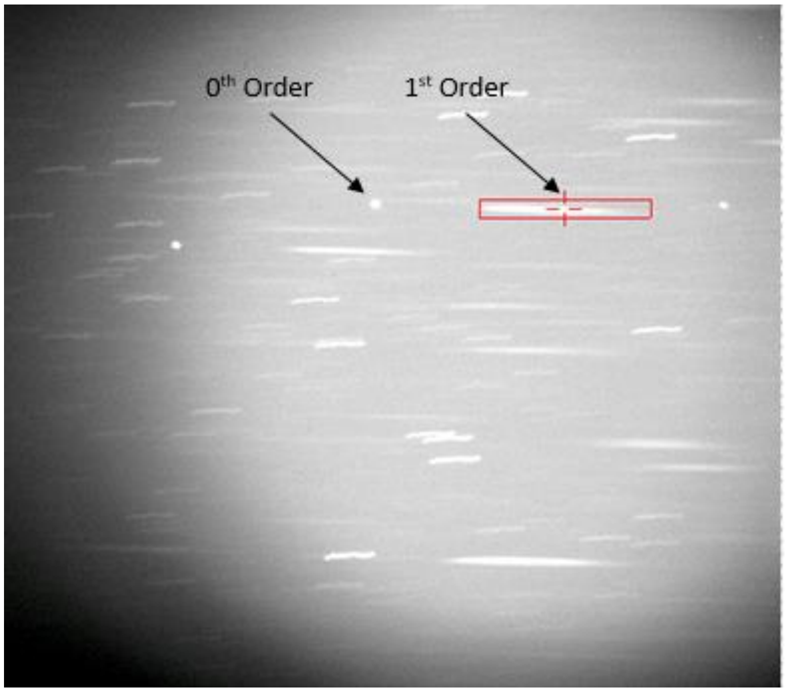

Figure 2.4 A rectangle is placed around the spectrum.

2.2 Process of analyzing data

All data were extracted using the software package *Mira*, but then analyzed using MATLAB. Before using any *Mira* scripts however, all images are reduced using calibration frames (flats, zeros, and darks) in *Mira*. Then, a rectangle is placed over the spectra and all counts are summed up in each column of the rectangle. This rectangle can be clearly seen in Fig. 2.4. The rectangle is over the 1st order spectrum, while the zeroth order spectrum is a little ways to its left. In order to know the wavelength values corresponding to each pixel, we first had to perform a calibration. The company that produces the SA-200 also provides a grating to sensor calculator that provides the wavelength to pixel ratio. The pixel to wavelength ratio can be described as:

$$\lambda=5.187*P+51.431 \tag{2.1}$$

where wavelength is given in nanometers and P is the distance from the zeroth order in pixels. Data is only recorded from 398.96 to 995.465 nm, or pixels values 68 to 183 measured along the x-axis away from the zeroth order.

Next, rectangles of the same size are created 20 pixels below and 20 pixels above the spectra (as seen in Fig. 2.5 and 2.6 respectively), which will later be used to subtract the background values. Again, values were summed up along every column. After obtaining all of these values from *Mira* for all images over one night, I tried to get rid of bad data, particularly where stars passed through the spectrum or background rectangles. To do this, I took the median of every 20 data points of a specific wavelength (or column value) moving forward in time. Any data in that region of 20 values with a relative error (with respect to the median) greater than 0.17, was thrown out and replaced by zero. The relative error limits- both high and low- are easy to change in the program. I generally found that filtering the background data with a relative error of 0.15 did a better job.

Next, the mean and standard deviation were calculated at each column location from every 15 data points (moving forward in time). Tapered values were used at the ends, such that the mean and standard deviation were calculated using 1, 3, 5, 7, 9, 11, 13, and then 15 data points at the beginning, and the reverse of that at the end. All data points were replaced by their mean values. Finally, background values at each x-pixel (or column) location were averaged from 15 values from the upper rectangle and 15 values from the lower rectangle all centered on the same x-pixel value and moving forwards in time. Again, tapered endpoints were used. Background values were then subtracted from any corresponding spectrum values (except those that had already been removed). Any values that would have been negative, are just set at zero, meaning there was no difference between the background and spectrum values at that location.

Following this process, all spectrum values are multiplied by their corresponding flux multipliers. This accounts for the extinction of light using values obtained from a solar analog star. Lastly, a bin value was selected for both wavelength and time. Data was again averaged over these bin values to produce data that could be more easily plotted. Generally, bin values of about five and 30 were used for wavelength and time respectively. If the number of wavelength or time values was not an exact multiple of the bin values, the remainder would be averaged for the last data point.

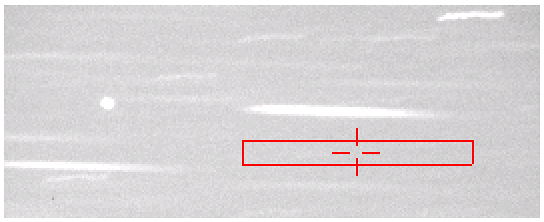
Figure 2.5 The rectangle below the spectra.

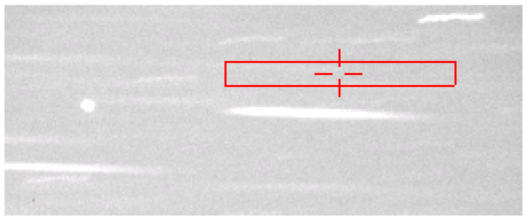
Figure 2.6 The rectangle above the spectra.

2.3 Programming in *Mira*

Mira has built in scripting classes that are specific to image processing, and it primarily uses the Lua programming language. The *Mira*metrics website provides a user's manual for scripting, which explains all of the built-in classes provided. *Mira* has capabilities for file event scripting, in which the user provides an initial text document and chooses an initial, intermediate, and final script. The initial text document gives the program information about the initial location (in pixel values) of the zeroth order satellite. It also tells the program what tracking radius should be allowed when tracking the zeroth order. If for any reason the satellite moves farther than that tracking radius from one frame to the next, *Mira* will not be able to track the satellite. The location of all three rectangles (the spectrum, background above, and background below) is determined from the zeroth order location.

When the user starts file watching, the initial script is run. Following that, the intermediate script will be run on any image added into the watch folder. When the user stops file watching, the final script is run. In the initial script, any values that will be needed through the entire script are initialized. The initial script also checks that the text file has enough entries and then initializes variables in the registry to hold values from the text file. The initial script also initializes variables that will track how many images are skipped and keep a record of their file names. There is also a registry value that holds the previous magnitude of the zeroth order (excluding any skipped images).

In the intermediate, or main, script, the image is processed. If there is a file event, which generally consists of a file being added to a specific folder, then this script will run. It creates an object to hold and manipulate the image, an object to do photometry on the image, and an object to calculate the centroid of the zeroth order. Variables are set from the registry values and if the there are no other problems, the rest of the script runs. If too many images have been skipped in a row, the program will quit because it probably lost track of the zeroth order. The program will also quit if it fails to open an image. An image will be skipped if the change in magnitude of the zeroth order is greater than two from one image to the next. The filename of the skipped image is stored in a registry value.

If the image is not being skipped and the program should not quit, then the script will calculate the values of each column of the rectangle surrounding the spectra. It will also do this for the rectangles above and below the spectra. The program will then output the time at which the exposure started (in seconds since midnight in Greenwich Mean Time), and values corresponding to each column of the spectra rectangle, then the rectangle above, and then the rectangle below. The new centroid coordinates are set, the previous magnitude is updated, and the number and list of skipped images is updated. The image and any unnecessary objects are also closed to free up memory.

The final script is very short, but it does print out the list of skipped images. When file event scripting is stopped, the script messages window contains a time, spectra values, and background values from above and below for each image followed by the skipped images list at the end. It is very easy to copy that data to an excel file, such that every image corresponds to one column. After that point, the data can be analyzed using a MATLAB function.

2.4 Programming in MATLAB

Once the data is in Excel, it just needs to be separated into three sheets. There are two empty rows between the spectrum and background values, and between the different types of background values. So, it should be clear how to separate the data. The background values from above need to be cut and pasted into the second sheet, and the background values from below need to be pasted into the third sheet.

Then, the user needs to run a function from MATLAB, which requires several values, including the name of the file and whether or not it is daylight savings. The MATLAB function analyzes the data according to the process described in Sec. 2.4. It outputs the final values into a new Excel document with the same file name, except with “final” concatenated to the beginning. The user can easily create graphs from this file. Times output from MATLAB are in seconds since noon Mountain Daylight Time, and wavelength values are in nanometers.

2.5 Challenges in automating program

One of the main difficulties in automating the *Mira* scripts is that we do not know if *Mira*'s tracking has failed to track the zeroth order satellite. To fix this, a field was created to store the previous magnitude. If that magnitude changes by two or more, it is likely that *Mira* lost track of the zeroth order. We also did not want the program to stop right away just because there was one bad image. So, the program will only stop if there are five or more tracking failures in a row. Also, it is easy for the user to check if the tracking will work. The user can try to track the zeroth order satellite with the same centroid parameters that are given to the script. If *Mira* doesn't have any trouble with that track, then script should run without any problems. That should take less than five minutes to check. Also, to aid in any trouble shooting, the image path names of any skipped images are printed at the end. If five images are skipped in a row. It is easy to see where the problem happened.

Another difficulty has been getting rid of data corrupted by stars passing through the rectangles. MATLAB should throw out most of those values by getting rid of any values that are greater than the mean added to the standard deviation. We are still working on improving this process however. We want to make sure we don't throw away any valuable data that reflects the true behavior of the satellite while still removing artifacts from passing stars.

The last main challenge was the sheer volume of data. From just one night, we can have about 2,200 frames. *Mira* can take six hours or more processing this data if it tries to process it all at once. We have found that *Mira* seems to work slower if you try to have it work on more than about 300 images at once. I recommend only putting about 250 images into the watch folder at one time, and then leaving it for an hour before putting in the next 250 images.

Also, *Mira* does not always process images in the order one would expect. However, if the user right clicks on the first image that should be processed when copying over the data, the script will process files in the expected order. Otherwise, the last file will generally be processed first. To be extra careful, the user should sort the data based on the time values in the first row after it is all in an Excel file. The MATLAB code only takes about 10 minutes to run on the data from 2,200 images. Although the process of extracting and analyzing the data still takes a lot of time, the user can start the program running and come back later.

In addition to these challenges, I found it difficult to debug code because there was so much data running through it. It takes a lot of time to make sure that the code is running properly on every image in *Mira* and on every matrix of data in MATLAB. I also tried to avoid using more memory than I needed in both *Mira* and MATLAB to avoid slowing the programs down.

I have also done my best to make the code clear to anyone who may need to change it. Any parameter values that are hard coded based on what I found to work best, or what seemed most appropriate have a comment that says “HARD CODED” next to it. For example, I chose to bin the wavelength in sets of five, and the time in sets of 15. If the user would like to change that, they will need to change the MATLAB code. I believe that is easier than requiring the user to provide all of these values every time they would like to call the function however. Any hard coded values are only defined once, so they should be easy to change. I hope that will help make this code more durable, but also flexible if changes need to be made.

Chapter 3

Results

3.1 Glint peaks

When analyzing the glint seen on the night of July 26, 2015, we found that longer wavelengths peaked after shorter wavelengths. In Fig 3.1 and Fig 3.2, it is clear that the longer wavelengths peak later. The vertical black lines are approximately lined up with the peak from the longest wavelength shown in each graph. The difference in timing is particularly clear from the 440-648 nm graph. Although we do not know why this incongruity in timing occurs, it is possible that it is due to a thin film in the surface of the facets. The smaller secondary glints may also be suggestive of multiple reflective surfaces. These may help give insight into the shape or attitude of the satellite, but results are inconclusive at this time. More data is required from SES-1 and other GEOS to discover the complete interpretation of the data.

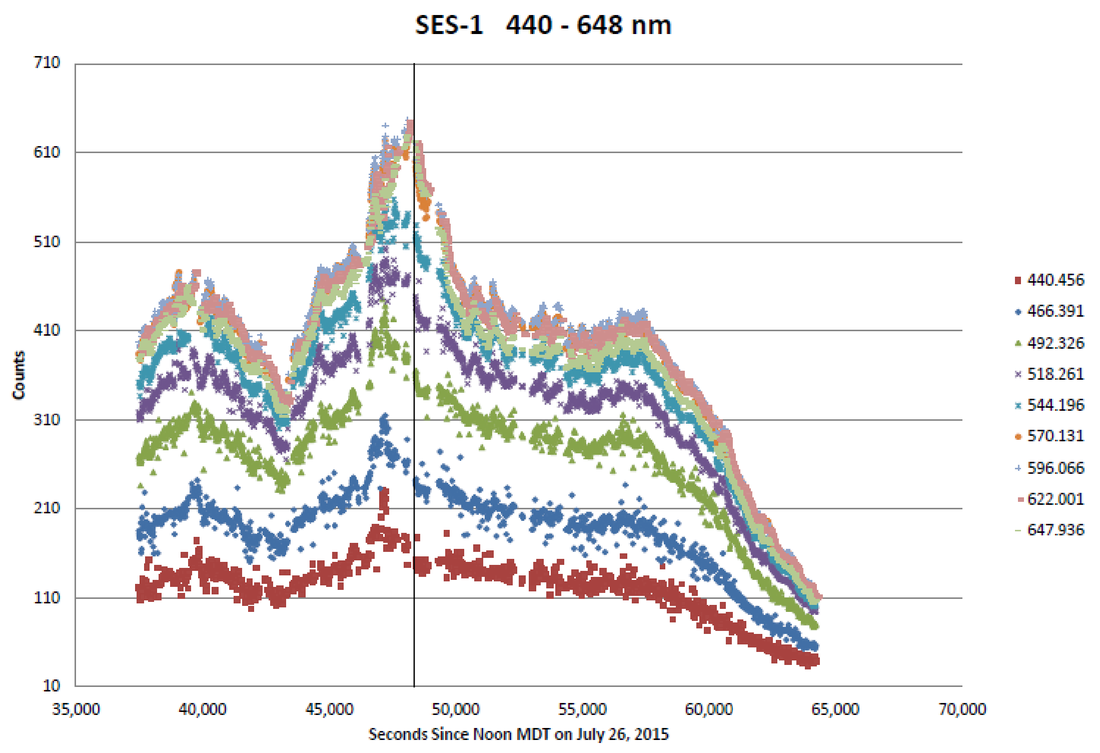

Figure 3.1 A plot of counts over time for wavelengths between 440.456-647.936 nm.

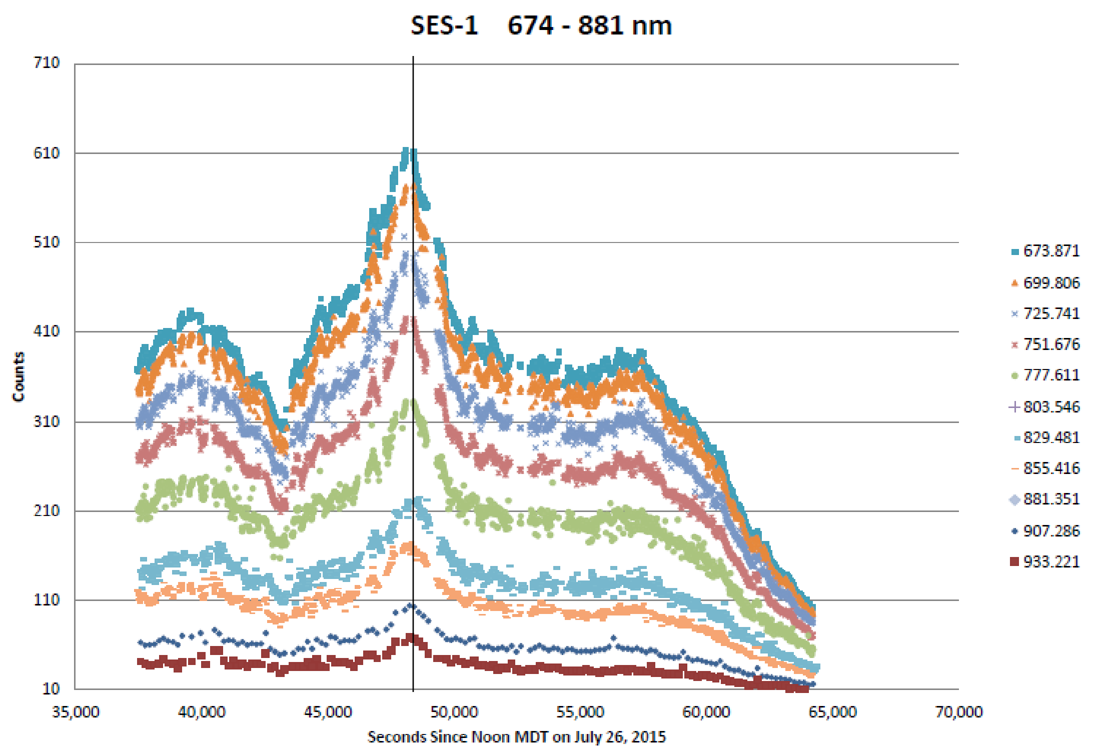

Figure 3.2 A plot of counts over time for wavelengths between 673.871-933.221 nm.

Chapter 4

Conclusion

4.1 Possible programming improvements

Overall, the *Mira* scripts and MATLAB function are reliable. There are still improvements to be made to remove stars from the spectra and background values, but the programs do a fairly good job. In addition, they will not work well if the data given to them was not taken on a photometric night. *Mira* will not be able to track the zeroth order if there are clouds coming in and out of the frame. It is simple to test if the code will accurately track the satellites, however. If *Mira* tracks the zeroth order, then the file event program will also. This only takes a couple minutes to test, so it is worth the time. I have done my best to find any errors that could come up, and stop them from happening, but that does not mean that other errors will not come up as the programs are tested more. I hope that these errors will be easy to fix however because the code is well-commented.

4.2 Future research

Many of the methods being used for gathering information about GEOS are new and still being developed. I think reflectance spectroscopy holds a promising future, but more data is needed to determine what the full capabilities of this method are. Now that the process is mostly automated, it should be easier to analyze large amounts of data quickly. In addition, it would be useful to observe some satellites with known properties, so that we can compare experimental results to physical properties to determine the level of correlation between the two. This is an exciting field currently with many opportunities for research, creativity, and progression.

Appendix A

Mira Code

Initial Script:

```
--//////////////////////////////////////////////////////////////////////////
--//// This script is called once when the “File Event Scripting” is started.

-- The text passed into the initial script is parsed into substrings- str[1],
str[2], etc.
-- These values can change photometry details, set the initial centroid, ….
str  =  StrTok( GetScriptString() )

-- Create a Registry object to hold the values passed in through the text
document.
-- Also, quit the program if there are not enough initial values passed in.
if #str < 9 then Exit( "Not enough parameter strings sent to Initial
Script\n" ) end

Reg = CRegistry:new( "Real-time-phot" ) -- create the Registry
Reg:SetNum( "CoordX", tonumber(str[1]) ) -- initial x coordinate of satellite
Reg:SetNum( "CoordY", tonumber(str[2]) ) -- initial y coordinate of satellite
Reg:SetNum( "Ap1", tonumber(str[3]) ) -- inner aperture radius (in pixels)
Reg:SetNum( "Ap2", tonumber(str[4]) ) -- inner sky sampling radius (in
pixels)
Reg:SetNum( "Ap3", tonumber(str[5]) ) -- outer aperture radius (in pixels)
Reg:SetStr( "ApBg", str[6] ) -- aperture background mode (0 = median)
Reg:SetNum( "CentRadiusX", tonumber(str[7]) ) -- Centroid calculation radius
(in pixels)
Reg:SetNum( "CentRadiusY", tonumber(str[8]) ) -- Centroid tracking radius (in
pixels)
Reg:SetNum( "ApZP", tonumber(str[9]) ) -- photometric zero point

-- The following values are not set by the user in the text passed into this
script.
```

```
Reg:SetNum( "PrevMag", 1 ) -- used to store satellite's previous magnitude,
set to 1 initially
Reg:SetNum( "NumSkipped", 0) -- gives the number of consecutively skipped
images
Reg:SetBool( "QuitNow", false) -- tells the program to quit
Reg:SetStr( "Skipped", "Skipped Images: ") -- string that contains all
skipped image file paths

Reg:SetStr( "Status", "started" ) -- optional, just to keep a record of the
processing

-- end of script
```

Event Script:

```
--////////////////////////////////////////////////////////////////////////
--////  This script is called when a file event occurs- generally when a file
--////  is added to the watch folder.


-- A string is automatically passed to the script for every file event.  The
string gives the file path, file event, ....

str = StrTok( GetScriptString() ) -- Tokenize the input string.

if str[2] ~= "Added" then Exit() end -- If a file was not added, then don't
run the script.

strImageFileName = str[1] -- Save the file name.

-- Open the selected file and then process it
I = CImage:new()  -- Create a CImage object to hold the image.
-- If the image will not open, create an error message and exit the script.
if I:Open( strImageFileName ) == false then
  sErrMsg = Sprintf( "Cannot open '%s' as an image.\n", strImageFileName )
  Exit( sErrMsg )
end

A = CApphot:new() -- Create a CApphot object for doing the photometry.
C = CCentroid:new() -- Create a CCentroid object for the centroid
coordinates.

-- Load aperture photometry parameters from the image (exptime, gain, etc.).
A:GetImageParams( I )

-- Open the Registry for accessing stored parameters.
Reg = CRegistry:new("Real-time-phot")

-- Centroid x and y values:
x = Reg:GetNum( "CoordX", 1 )
y = Reg:GetNum( "CoordY", 1 )
C:SetSample( Reg:GetNum( "CentRadiusX", 5 ), Reg:GetNum( "CentRadiusY", 5 ) )
-- Photometry parameters:
```

```
A.nRadius1 = Reg:GetNum( "Ap1", 5 ) -- inner aperture radius
A.nRadius2 = Reg:GetNum( "Ap2", 10 ) -- inner sky sampling radius
A.nRadius3 = Reg:GetNum( "Ap3", 15 ) -- outer aperture radius
A:SetBgMethod( Reg:GetStr( "ApBg", "mode" ) ) -- set background method
nZeroPt = Reg:GetNum( "ApZP", 22 ) -- set zero point
prevMag = Reg:GetNum( "PrevMag", 1 ) -- gives the magnitude of the satellite
in the previous image
numSkipped = Reg:GetNum( "NumSkipped", 0) -- gives the number of skipped
images
quitNow = Reg:GetBool( "QuitNow", false) -- tells the program to quit if
there are too many skipped images
skipped = Reg:GetStr( "Skipped", "Skipped Images: ") -- saves file paths of
skipped images


-- Only runs the script if you don't want to quit
if quitNow == false then
  -- update the object coordinates using the centroid
  bSuccess = C:Calc( I, x, y )  -- compute the centroid position

  A:Measure( I, C.x, C.y )  -- does aperture photometry on satellite 0th order
  futMag = A.nMag+nZeroPt -- gives the magnitude of the current image
  -- Printf("Prev: %lg  Future: %lg\t", prevMag, futMag)  -- just for
debugging purposes

  -- preMag is set to 1 at beginning, so we can redefine it (there shouldn't
be a tracking problem on the first image because the centroid coordinates
should be set).
  if prevMag == 1 then
    bSuccess = true
    prevMag = futMag
  end

  -- If the change in magnitude from one frame to the next is greater than
1.5, then something went wrong probably.
  -- Note: 1.5 is hard coded...it may need to be changed
  -- HARD CODED
  if math.abs(prevMag-futMag) > 1.5 then
    bSuccess = false
  end

  -- Note: the rectangle position is hard coded based on the position of the
satellite (centroid).
  -- HARD CODED
  xi = x + 67; xf = x + 182; -- setting x initial and x final
  yi = y - 10; yf = y + 3; -- setting y initial and y final

  if bSuccess == false then  -- if something went wrong, skip the image
    numSkipped = numSkipped + 1 -- keeps a count of consecutively skipped
images
    skipped = skipped .. I:Path(30) .. "\n" -- keeps track of skipped images
  end

  if bSuccess == true and quitNow == false then -- if everything seems okay
    numSkipped = 0 -- reset the number skipped
    -- total = 0 -- counts the total number of counts in the rectangle- used
previously to give counts at each wavelength as a percentage of the total.
```

```
    tab = {} -- initialize a table to hold the values of each column
    tabU = {} -- initialize a table for the rectangle above
    tabD = {} -- initialize a table for the rectangle below

    -- run through each column and add up all of the counts in that column in
the rectangle
    for i = 1, (xf - xi + 1)
    do
      colSum = 0
      colSumU = 0
      colSumD = 0

      for j = yi, yf
      do
        colSum = I:Val(i + xi - 1, j) + colSum
        colSumU = I:Val(i + xi -1, j + 20) + colSumU
        colSumD = I:Val(i + xi -1, j - 20) + colSumD
      end

      -- total = total + colSum - Not needed anymore.
      tab[i] = colSum
      tabU[i] = colSumU
      tabD[i] = colSumD
    end

    time = I:Time()
    time = HmsToHr(time)
    Printf( "%lg \t", time) -- print the time when the exposure began (in a
decimal number of hours since midnight GMT)

    -- print all column values from the spectrum
    for index, value in next, tab do
      Printf("%lg \t",tab[index])
    end

    -- print all column values from the rectangle above after leaving some
blank space
    Printf("\t\t")

    for index, value in next, tabU do
      Printf("%lg \t",tabU[index])
    end

    -- print all column values from the rectangle below after leaving some
blank space
    Printf("\t\t")

    for index, value in next, tabD do
      Printf("%lg \t",tabD[index])
    end

    Printf("\n")


    -- Possibly useful values to print:
```

```
    -- Printf( "%-30.30s\tMag = %lg +/- %lg, BG = %lg (sdev=%lg), SNR
= %lg\n", I:Path(30), A.nMag+nZeroPt, A.nMagErr, A.nBgValue, A.nBgValueErr,
A.nSnRatio )

    -- update the object coordinates in the Registry for the next image
    Reg:SetNum( "CoordX", C.x )
    Reg:SetNum( "CoordY", C.y )
    Reg:SetNum( "PrevMag", A.nMag+nZeroPt )
  end

  -- If more than five images have been skipped, tell the user and quit.
  -- HARD CODED
  if numSkipped > 5 then
    Printf("%lg consecutive images have been skipped, so the program has been
terminated.  \n", numSkipped)
    quitNow = true
  end
end

-- Finish updating the registry
Reg:SetNum( "NumSkipped", numSkipped)
Reg:SetBool( "QuitNow", quitNow)
Reg:SetStr( "Skipped", skipped)

I:Close()  -- Close this image to free memory.
-- Delete objects to free memory:
A:delete()
C:delete()
Reg:delete()

-- end of script
```

Final Script:

```
--///////////////////////////////////////////////////////////////////////////
--////  Called once from the "File Event Scripting" task when the file
--////  watching loop is terminated.

-- Print out the list of skipped images and do any needed clean-up.

Reg = CRegistry:new("Real-time-phot")
skipped = Reg:GetStr( "Skipped", skipped)
Printf("%s", skipped)
Reg:SetStr( "Status", "completed" )
Reg:delete()

-- end of script
```

Appendix B

MATLAB Code:

Function satCalc.m:

```
function [] = satCalc(workbookFile, sheetNameAllData, sheetNameBackUp,
sheetNameBackDown, rangeAll,rangeBg,dayLSavings)

% Import data from a spreadsheet
%% Input handling

% If no sheet is specified, read the first, second, and third sheets
if nargin < 7 || isempty(sheetNameAllData) || isempty(sheetNameBackUp) ...
        || isempty(sheetNameBackDown)
    sheetNameAllData = 1;
    sheetNameBackUp = 2;
    sheetNameBackDown = 3;
end

% If ranges are not specified, read all data
if nargin < 7 || isempty(rangeAll) || isempty(rangeBg)
    rangeAll = '';
    rangeBg = '';
end

% If it is not specified in day light savings is on/off, assume it is off
if nargin < 7 || isempty(dayLSavings)
   dayLSavings = 'Off';
end

%% Import the data
[~, ~, raw] = xlsread(workbookFile, sheetNameAllData, rangeAll);
raw(cellfun(@(x) ~isempty(x) && isnumeric(x) && isnan(x),raw)) = {''};
[~, ~, raw2] = xlsread(workbookFile, sheetNameBackUp, rangeBg);
raw2(cellfun(@(x) ~isempty(x) && isnumeric(x) && isnan(x),raw2)) = {''};
[~, ~, raw3] = xlsread(workbookFile, sheetNameBackDown, rangeBg);
```

```
raw3(cellfun(@(x) ~isempty(x) && isnumeric(x) && isnan(x),raw3)) = {''};

%% Replace non-numeric cells with 0
R = cellfun(@(x) ~isnumeric(x) && ~islogical(x),raw); % Find non-numeric
cells
raw(R) = {0}; % Replace non-numeric cells
R2 = cellfun(@(x) ~isnumeric(x) && ~islogical(x),raw2);
raw2(R2) = {0};
R3 = cellfun(@(x) ~isnumeric(x) && ~islogical(x),raw3);
raw3(R3) = {0};

%% Create output variables
I = cellfun(@(x) ischar(x), raw);
raw(I) = {0};
data = reshape([raw{:}],size(raw));
I2 = cellfun(@(x) ischar(x), raw2);
raw2(I2) = {0};
backUp = reshape([raw2{:}],size(raw2));
I3 = cellfun(@(x) ischar(x), raw3);
raw3(I3) = {0};
backDown = reshape([raw3{:}],size(raw3));

%% This code should get rid of any obvious problems or stars for allData,
%  backUp, and backDown.

% I'll try to do it in terms of percentage change ... that way it doesn't
% matter if the input is in counts or counts/(total counts in rectangle),
% etc.

% Set the number of rows and columns and remove the time data.
[numRows, numCols]=size(data);
time=data(1,1:numCols);
allData(1:numRows-1,1:numCols)=data(2:numRows,1:numCols);
numRows=numRows-1;

%% Getting rid of bad data: using medians and then percent error.

numMed=20; % The number of values that we are taking the median of in a given
row.
med=zeros(numRows,ceil(numCols/numMed)); % Creating a matrix to hold the
median values.
medU=med;
medD=med;

% Finding and saving the median values:
for j=1:numRows
    for k=1:ceil(numCols/numMed)
      if k~=ceil(numCols/numMed)
        med(j,k)=median(allData(j,(k-1)*numMed+1:k*numMed));
        medU(j,k)=median(backUp(j,(k-1)*numMed+1:k*numMed));
        medD(j,k)=median(backDown(j,(k-1)*numMed+1:k*numMed));
      else
        med(j,k)=median(allData(j,(k-1)*numMed+1:numCols));
        medU(j,k)=median(backUp(j,(k-1)*numMed+1:numCols));
        medD(j,k)=median(backDown(j,(k-1)*numMed+1:numCols));
      end
    end
end

% Setting the relative error values:
% HARD CODED:
errA=.17;
errU=.15;
errD=.15;

% If the values are outside of the relative error compared to the median
values, replace them with 0.
for j=1:numRows
    for k=1:numCols
      if abs(med(j,ceil(k/numMed))-allData(j,k))/med(j,ceil(k/numMed))>errA
         allData(j,k)=0;
      end
      if abs(medU(j,ceil(k/numMed))-backUp(j,k))/medU(j,ceil(k/numMed))>errU
         backUp(j,k)=0;
      end
      if abs(medD(j,ceil(k/numMed))-
backDown(j,k))/medD(j,ceil(k/numMed))>errD
         backDown(j,k)=0;
      end
    end
end

%% Taking the Standard Deviation and Mean in sets of 15.  Near the end, the
%  standard deviation is taken for the last 14, 13, 12, 11, ... 1 data
%  points.  The opposite is true at the beginning.

% Initialize variables:
stdvAll=zeros(numRows,numCols);
stdvUp=stdvAll;
stdvDown=stdvAll;

meanAll=zeros(numRows,numCols);
meanUp=meanAll;
meanDown=meanAll;

% Unfortunately, MATLAB will count the zeros when it takes the standard
% deviation and mean, so we have to manually skip over those.  The following
% code calculates the standard deviation and mean in sets of 15.

for j=1:numRows
   for k=1:numCols
      if k <= numCols-7 && k >= 8 % If there are at least 7 values on either
        % side of the current column.
        vAll=allData(j,k-7:k+7);
        vAll=vAll(vAll~=0); % Get rid of 0's.
        stdvAll(j,k)=std(vAll); % Calculate the standard deviation.
        meanAll(j,k)=mean(vAll); % Calculate the mean.

        vUp=backUp(j,k-7:k+7);
        vUp=vUp(vUp~=0);
        stdvUp(j,k)=std(vUp);
        meanUp(j,k)=mean(vUp);

        vDown=backDown(j,k-7:k+7);
        vDown=vDown(vDown~=0);
        stdvDown(j,k)=std(vDown);
        meanDown(j,k)=mean(vDown);
      elseif k < 8 % Run for first seven columns.
        vAll=allData(j,1:2*k-1);
        vAll=vAll(vAll~=0);
        stdvAll(j,k)=std(vAll);
        meanAll(j,k)=mean(vAll);

        vUp=backUp(j,1:2*k-1);
        vUp=vUp(vUp~=0);
        stdvUp(j,k)=std(vUp);
        meanUp(j,k)=mean(vUp);

        vDown=backDown(j,1:2*k-1);
        vDown=vDown(vDown~=0);
        stdvDown(j,k)=std(vDown);
        meanDown(j,k)=mean(vDown);
      else % Run for the last seven columns.
        vAll=allData(j,2*k-numCols:numCols);
        vAll=vAll(vAll~=0);
        stdvAll(j,k)=std(vAll);
        meanAll(j,k)=mean(vAll);

        vUp=backUp(j,2*k-numCols:numCols);
        vUp=vUp(vUp~=0);
        stdvUp(j,k)=std(vUp);
        meanUp(j,k)=mean(vUp);

        vDown=backDown(j,2*k-numCols:numCols);
        vDown=vDown(vDown~=0);
        stdvDown(j,k)=std(vDown);
        meanDown(j,k)=mean(vDown);
      end
   end
end

%% Getting the total background values by averaging backUp and backDown.
%  Also, subtracting them from all data points in meanAll.

background=zeros(numRows,numCols);

% Average 15 values from backUp and the corresponding values from backDown.
for j=1:numRows
   for k=1:numCols
      if k <= numCols-7 && k >= 8 % Run for values in the middle.
        vBd=zeros(15,1); % Initialize values.
        vBu=vBd; % Initialize values.
        vBd(1:15,1)=meanDown(j,k-7:k+7)'; % Get the values from meanDown.
        vBd=vBd(vBd~=0); % Remove 0's.
        vBu(1:15,1)=meanUp(j,k-7:k+7)'; % Get the values from meanUp.
        vBu=vBu(vBu~=0); % Remove 0's.
        vBg=[vBd;vBu];
        background(j,k)=mean(vBg);
      elseif k < 8 % Run for values at the beginning.
        vBd=zeros(2*k-1,1);
        vBu=vBd;
        vBd(1:2*k-1,1)=meanDown(j,1:2*k-1)';
        vBd=vBd(vBd~=0);
        vBu(1:2*k-1,1)=meanUp(j,1:2*k-1)';
        vBu=vBu(vBu~=0);
        vBg=[vBd;vBu];
        background(j,k)=mean(vBg);
      else % Run for values at the end.
        vBd=zeros(2*numCols-2*k+1,1);
        vBu=vBd;
        vBd(1:2*numCols-2*k+1,1)=meanDown(j,2*k-numCols:numCols)';
        vBd=vBd(vBd~=0);
        vBu(1:2*numCols-2*k+1,1)=meanUp(j,2*k-numCols:numCols)';
        vBu=vBu(vBu~=0);
        vBg=[vBd;vBu];
        background(j,k)=mean(vBg);
      end
      if meanAll(j,k) - background(j,k) > 0
          meanAll(j,k) = meanAll(j,k)-background(j,k);
      else % If meanAll was going to be negative, replace it with 0.
          meanAll(j,k) = 0;
      end
   end
end

%% Setting the time to be in the form of seconds since noon in MDT.

if strcmp(dayLSavings,'On')== 1
   time(1,1:numCols)=time(1,1:numCols)-3600*6;
else
    time(1,1:numCols)=time(1,1:numCols)-3600*7;
end

time(1,1:numCols)=time(1,1:numCols)+12*3600;

%% Flux Multipliers

% HARD CODED
fluxMult=[6.85,7.28,6.72,5.53,4.90,4.39,2.53,4.00,3.95,3.55,3.35,2.38, ...
    2.02,1.57,1.42,1.25,1.27,1.02,1.15,1.25,1.16,1.18,1.14,0.97,0.98, ...
    0.91,0.82,0.80,0.83,0.78,0.80,0.80,0.90,0.93,1.01,1.03,0.96,0.87, ...
    0.79,0.68,0.70,0.68,0.65,0.67,0.74,0.85,0.97,1.05,1.05,1.00,0.91, ...
    0.83,0.81,0.75,0.76,0.76,0.78,0.82,0.90,1.05,1.12,1.19,1.12,1.10, ...
    1.09,1.05,1.02,1.01,1.02,1.06,1.11,1.25,1.51,1.82,1.79,1.61,1.40, ...
    1.39,1.41,1.38,1.37,1.42,1.53,1.64,1.79,1.98,2.16,1.98,2.25,2.56, ...
    2.24,2.64,2.62,2.63,2.59,2.60,2.69,2.83,3.11,3.24,3.30,3.76,4.32, ...
    4.76,5.44,5.86,6.54,6.73,7.09,7.26,7.01,7.25,7.76,8.56,9.35,9.90];

% I only want 68-183 (in pixel values). Create a vector of wavelength values.
lambda=zeros(numRows,1);
% HARD CODED
lambdaMin=398.96; h=5.187; lambdaMax=995.465;
```

```
lambda(1:numRows,1)=lambdaMin:h:lambdaMax;

% Multiply by the corresponding flux multipliers.
for j=1:numRows
    meanAll(j,1:numCols)=meanAll(j,1:numCols)*fluxMult(j);
end

% These bin values are HARD CODED- you can change them if you want...
wlBinVal=5;
timeBinVal=30;

wlavg=zeros(wlBinVal,1); % holds data values until they need to be averaged
in wavelength
timeavg=zeros(timeBinVal,1); % holds data values until they need to be
averaged in time
wlBinAvg=wlavg; % holds wavelength values until they need to be averaged
timeBinAvg=timeavg; % holds time values until they need to be averaged
wlBin=zeros(ceil(numRows/wlBinVal),1); % holds all final wavelength values
timeBin=zeros(1,ceil(numCols/timeBinVal)); % holds all final time values

% I'll do the wavelength binning first, and then in time.
binValuesIn=zeros(ceil(length(lambda)/wlBinVal),numCols); % stores values
from first binning the wavelength
binValuesFin=zeros(ceil(length(lambda)/wlBinVal),ceil(length(time)/timeBinVal
)); % stores values from final binning of time
for k=1:numCols
   for j=1:numRows
       if mod(j,wlBinVal)==0 || j==numRows % only run if we need to set a
value in binValuesIn
         wlavg(length(wlavg)+1,1)=meanAll(j,k);
         wlavg=mean(wlavg(wlavg~=0)); % average all values, taking out 0's
         binValuesIn(ceil(j/wlBinVal),k)=wlavg; % set the new binned value
         wlavg=zeros(wlBinVal,1); % reinitialize the vector

         % the following code does the same process for actual wavelengths:
         wlBinAvg(length(wlBinAvg)+1,1)=lambda(j,1);
         wlBinAvg=mean(wlBinAvg(wlBinAvg~=0));
         wlBin(ceil(j/wlBinVal),1)=wlBinAvg;
         wlBinAvg=wlavg;
       else % if we aren't done gathering data, add values to be averaged
         wlavg(mod(j,wlBinVal),1)=meanAll(j,k);
         wlBinAvg(mod(j,wlBinVal),1)=lambda(j,1);
       end
   end
end

% the following loop should now bin the binValuesIn in time:
for j=1:ceil(numRows/wlBinVal)
   for k=1:numCols
       if mod(k,timeBinVal)==0 || k==numCols % only run if we need to set a
value in binValuesFin
         timeavg(length(timeavg)+1,1)=binValuesIn(j,k);
         timeavg=mean(timeavg(timeavg~=0)); % average values, taking out 0's
         binValuesFin(j,ceil(k/timeBinVal))=timeavg;
         timeavg=zeros(timeBinVal,1);
         timeBinAvg(length(timeBinAvg)+1,1)=time(1,k);
         timeBinAvg=mean(timeBinAvg(timeBinAvg~=0));
         timeBin(1,ceil(k/timeBinVal))=timeBinAvg;
         timeBinAvg=timeavg;
       else % if we aren’t done gathering data, add values to be averaged
         timeavg(mod(k,timeBinVal),1)=binValuesIn(j,k);
         timeBinAvg(mod(k,timeBinVal),1)=time(1,k);
       end
   end
end

[binR, binC]=size(binValuesFin); % set the new row and column sizes
timePlots=zeros(binR+1,binC+1); % initialize a matrix for all final values
timePlots(2:binR+1,2:binC+1)=binValuesFin(1:binR,1:binC); % put the final
binned values in the matrix
timePlots(2:binR+1,1)=wlBin(1:binR,1); % put the wavelengths along the left
side
timePlots(1,2:binC+1)=timeBin(1,1:binC); % put the times along the top


%% Export the data to a new Excel file with the same name, except with
“final” before it.

filename = strcat('final',workbookFile);
xlswrite(filename,timePlots,1)

% end of script
```

Additional Possibly Useful Code:

```
%%
% Loops to get rid of any value that has a relative change of 20% or
% higher.  That value can be changed, by altering the "relCh" variable.
% This also relies on all of the values in the first column being "good"
% values.  The problem with this code is that if values jump up by 19% and
% then down by 21% and stay there or slowly get lower, the program will
% continue to skip over those values (because the relative change was greater
% than 20%), even though they are probably okay.  Basically, relying on just
% the previous data point is not good enough.

% relCh IS HARD CODED- you can change it...
relCh=.2; % the allowed relative change up
relChD=.25; % the allowed relative change down
minCountCh=40; % the minimum change in counts required to get rid of data

for j=1:numRows-1
    numSkippedAll=0; % initialize values to hold the number of consecutively
skipped values- for debugging purposes
    numSkippedUp=0;
    numSkippedDown=0;

    errA=0; % initialize values to hold errors
    errU=0;
    errD=0;
    for k=1:numCols-1
       % set the relative difference of the spectrum data
       diffA=abs(allData(j,k+1)-allData(j,k-numSkippedAll));
       relA=diffA/allData(j,k-numSkippedAll);
       % I tried using a bunch of conditions to avoid throwing out good data,
       % but ultimately failed.  I won't enumerate the conditions because
       % they are fairly straightforward.  If the conditions are met, replace
       % the value by 0 and increase the number skipped by 1.
       if relA>relCh && (allData(j,k+1)>allData(j,k-numSkippedAll) ||
       allData(j,k+1)==0 || ...
       (allData(j,k+1)<allData(j,k-numSkippedAll) && ...
       relA>relChD)) && diffA>minCountCh && (allData(j,k+1)>150 ...
       || allData(j,k+1)==0) && ((allData(j,k+1)<320 && diffA>100) ||
       allData(j,k+1)>320)
           allData(j,k+1)=0;
           numSkippedAll = numSkippedAll + 1;
       else
           numSkippedAll = 0;
       end

       % Run through the same process for the values above the spectrum.
       diffU=abs(backUp(j,k+1)-backUp(j,k-numSkippedUp));
       relU=diffU/backUp(j,k-numSkippedUp);
       if relU>relCh && (backUp(j,k+1)>backUp(j,k-numSkippedUp) ||
       backUp(j,k+1)==0 || ...
       (backUp(j,k+1)<backUp(j,k-numSkippedUp) && relU>relChD)) ...
       && diffU>minCountCh && (backUp(j,k+1)>150 || backUp(j,k+1)==0) ...
       && ((backUp(j,k+1)<320 && diffU>100) || backUp(j,k+1)>320)
           backUp(j,k+1)=0;
           numSkippedUp = numSkippedUp + 1;
       else
           numSkippedUp = 0;
       end

       % Run through the same process for the values above the spectrum.
       diffD=abs(backDown(j,k+1)-backDown(j,k-numSkippedDown));
       relD=diffD/backDown(j,k-numSkippedDown);
       if relD>relCh && (backDown(j,k+1)>backDown(j,k-numSkippedDown) ||
       backDown(j,k+1)==0 || ...
       (backDown(j,k+1)<backDown(j,k-numSkippedDown) ...
       && relD>relChD)) && diffD>=minCountCh && (backDown(j,k+1)>150 ...
       || backDown(j,k+1)==0) && ((backDown(j,k+1)<320 && diffD>100) ||
       backDown(j,k+1)>320)
           backDown(j,k+1)=0;
           numSkippedDown = numSkippedDown + 1;
       else
           numSkippedDown = 0;
       end

       % Create an error message if more than 8 values are consecutively
skipped.
       if numSkippedAll>8
           errA=1;
       end
       if numSkippedUp>8
           errU=1;
```

```
            end
            if numSkippedDown>8
                errD=1;
            end
        end

        if errA==1
            fprintf('Possible error in allData row %g',j)
        end
        if errU==1
            fprintf('Possible error in backUp row %g',j)
        end
        if errD==1
            fprintf('Possible error in backDown row %g',j)
        end
end
```

Additional Possibly Useful Code:

```
%% Trying to remove bad data.  Data is replaced by the mean if the value from
% allData is greater than the value from the mean + mult * stdev or less than
% the value from mean - mult * stdev.  Otherwise, it becomes the value from
% the mean.  The problem with this code is similar to that of the previous
% section.  In short, it just didn't work and would throw out too much good
% data and not get rid of all of the bad data, particularly when values were
% extremely close to the acceptable threshold. Considering how simple it is,
% it did a decent job sometimes.

% HARD CODED-Multiplier by stdv
mult=2; % changes the amount of variance you will allow
for j=1:numRows
   for k=1:numCols
      % If the data point is outside of the acceptable range, replace it by
0.  Otherwise, replace the data value by the mean at that point.
      if allData(j,k) > meanAll(j,k)+mult*stdvAll(j,k) || allData(j,k) <
      meanAll(j,k)-mult*stdvAll(j,k)
          allData(j,k)=0;
      else
          allData(j,k)=meanAll(j,k);
      end

      % Do the same process for the rectangles above and below.
      if backUp(j,k) > meanUp(j,k)+mult*stdvUp(j,k) || backUp(j,k) <
      meanUp(j,k)-mult*stdvUp(j,k)
          backUp(j,k)=0;
      else
          backUp(j,k)=meanUp(j,k);
      end

      if backDown(j,k) > meanDown(j,k)+mult*stdvDown(j,k) || backDown(j,k) <
      meanDown(j,k)-mult*stdvDown(j,k)
          backDown(j,k)=0;
      else
          backDown(j,k)=meanDown(j,k);
      end
   end
end
```

Additional Possibly Useful Code:

```
%% The following code create and saves an animation of the wavelength values
% as they progress in time.  You must first import an Excel file with the
% data you want to animate.  It takes about 2 minutes to load the data if
% there are about 2000 entries.

%% Import the data using the file path name (this happened to be the file I
was using)
 [~, ~, raw] = xlsread('\\physics\Shares\Users\klpdance\My
Documents\MATLAB\SES 1 to 500.xlsx','Mean Data','J2:CFI187');

%% Replace non-numeric cells with 0.0 and create an output variable.
R = cellfun(@(x) ~isnumeric(x) || isnan(x),raw); % Find non-numeric cells
raw(R) = {0.0}; % Replace non-numeric cells
data = cell2mat(raw);

[numRows,n]=size(data); % Find how many rows and columns are in the data.

% HARD CODED - change name of file
writerObj = VideoWriter('peaksMean.avi'); % what you want to name the file
open(writerObj);
set(gca,'nextplot','replacechildren');
set(gcf,'Renderer','zbuffer');

total = zeros(numRows,1);
% HARD CODED bin number
binNum = 5; % how many columns you want to average before plotting
switchVal = n - mod(n, binNum) + 1; % where the last set of averaging starts
for i = 1:n
    if n - i >= binNum && i < switchVal  % for most columns ...
        total = data(:,i)./binNum + total;
    else % for the last set ...
        total = data(:,i)./(n - switchVal + 1) + total;
    end

    if mod(i, binNum) == 0 || i == n % only plot when total is ready
      plot(1:numRows,total(:,1))
      ylim([0 1100]) % sets the y axis - HARD CODED
      % HARD CODED - speed
      pause(.3) % determines speed of video - may want to change
      frame = getframe;
      writeVideo(writerObj,frame);
      total = zeros(numRows,1);
   end
end

%% Clear temporary variables
clearvars data raw R columnIndices;
```

Bibliography

Abercromby, K. J., Hamada, K., Okada, J., Guyote, M., & Barker, E. "Comparisons of Ground Truth and Remote Spectral Measurements of the FORMOSAT and ANDE Spacecraft," Proceedings of the 2006 AMOS Technical Conference (The Maui Economic Development Board, Inc., Kihei, Maui, HI, 2006).

Bédard, D., Monin, D., Scott, R. & Wade, G. "Spectrometric Characterization of Active Geosynchronous Satellites," Proceedings of the 2012 AMOS Technical Conference (The Maui Economic Development Board, Inc., Kihei, Maui, HI, 2012).

Chun, F.K., Tucker, R.M., Weld, E.M., & Tippets, R.D. "Spectral Measurements of Geosynchronous Satellites During Glint Season." Proceedings of the 2015 AMOS Technical Conference (The Maui Economic Development Board, Inc., Kihei, Maui, HI, 2015).

Guyote, M., Abercromby, K. J., & Okada, J. "Using Space Weathering Models to Match Observed Spectra to Predicted Spectra," Proceedings of the 2006 AMOS Technical Conference (The Maui Economic Development Board, Inc., Kihei, Maui, HI, 2006).

Hall, D., Calef, B., Knox, K., Bolden, M., & Kervin, P. "Separating Attitude and Shape Effects for Non-resolved Objects," Proceedings of the 2007 AMOS Technical Conference (The Maui Economic Development Board, Inc., Kihei, Maui, HI, 2007).

Hall, D., & Kervin, P. "Analysis of Faint Glints from Stabilized GEO Satellites," Proceedings of the 2013 AMOS Technical Conference (The Maui Economic Development Board, Inc., Kihei, Maui, HI, 2013).

Mira Pro Script Module User's Guide, (Mirametrics, Inc., 2012). Retrieved from http://mirametrics.com/help/mira_pro_script_7/

Mira Pro x64 User's Guide, (Mirametrics, Inc., 2016). Retrieved from http://mirametrics.com/help/mira_pro_x64_8/

Moody, J.W., Boizelle, B., Bates, K., Little, B., McCombs, T., Nelson, J., Pace, C., Pearson, R. L. III, Harrison, J., Brown, P. J., and Barnes, J. "Remote Observatory for Variable Objects Research (ROVOR)", PASP, 124, 956, 2012.

Payne, T. E., Gregory, S.A., & Luu, K. "SSA Analysis of GEOS Photometric Signature Classifications and Solar Panel Offsets," Proceedings of the 2006 AMOS Technical Conference (The Maui Economic Development Board, Inc., Kihei, Maui, HI, 2006).

Schildknecht, T., Vananti, A., Krag, H., & Erd, C. "Reflectance Spectra of Space Debris in GEO," Proceedings of the 2009 AMOS Technical Conference (The Maui Economic Development Board, Inc., Kihei, Maui, HI, 2009).

Vrba, F.J., Hutter, D.J., Shankland, P.D., Armstrong, J.T., Schmitt, H.R., Hindsley, R.B., DiVittorio, M.E., & Benson, J.A. "A Survey of Geosynchronous Satellite Glints." Proceedings of the 2009 AMOS Technical Conference (The Maui Economic Development Board, Inc., Kihei, Maui, HI, 2009).

Index