An Analysis of the Efficacy of the CSIM Cubesat for SSI Continuity

by

Ansh Desai

A thesis submitted to the

Faculty of the

University of Colorado in partial fulfillment

of the requirements for

departmental honors in the

Department of Physics

March 24, 2021

Desai, Ansh (B.A. Physics)

An Analysis of the Efficacy of the CSIM Cubesat for SSI Continuity

Thesis directed by Dr. Erik Richard (LASP)

One of the many data-sets required by scientists seeking to understand Earth's climate is a long term record of solar variability over a wide range of spectral regions. Measurements of solar spectral irradiance (SSI) have been taken by two space-based missions: the SOlar Radiation and Climate Experiment (SORCE) and the Total and Spectral Solar Irradiance Sensor (TSIS), both primarily relying on two different implementations of the Spectral Irradiance Monitor (SIM) instrument. While large spacecraft like SORCE (and TSIS) are effective, there are problems associated with these types of missions; they are expensive and an instrument failure can put a continuous data record at risk. A potential solution to this problem can be found in a new emergent technology in space-based observation, miniaturized single-instrument spacecraft colloquially known as CubeSats. In order to explore this alternative, the Compact Spectral Irradiance Monitor (CSIM) was built at LASP by professionals working with undergraduate students as a flight demonstration mission. Because of the relatively cheap cost and higher acceptable risk, CSIM could also afford to test new technologies. TSIS SIM measures SSI from 200nm to 2400nm ($\sim$96% of the total solar irradiance, TSI), and CSIM is the first to make measurements from 200 nm all the way to 2800 nm (97.4% of TSI). To maintain accuracy of solar measurements, the SIM instrument needs to be corrected for optical degradation (common throughout all instruments measuring the sun directly). In developing a correction method for CSIM, slightly different analysis methods were required compared to TSIS and SORCE. Corrected data revealed that while this instrument is far from perfect, it performed well enough to merit further investigation into this technology.

Acknowledgements

I would like to thank my PI, Dr. Erik Richard for taking the time to mentor and support me throughout this process. I would also like to thank Michael Chambliss for guiding me through my data analysis process. I thank the scientists and engineers that worked for over a decade to make TSIS and CSIM a reality and the scientists and engineers that made the NRLSSI2 datasets available for comparison. I would also like to thank Dr. Cumalat for his guidance throughout the honors process, as well as Dr. Wilcox and Dr. Hamilton for taking the time out of their busy schedules to serve on my committee.

Contents

Figures

Chapter 1

Introduction

While a long term data record of Total Solar Irradiance (TSI) exists, such a record cannot capture any information about the spectral composition of solar variability. Therefore in order to study the solar variability of the sun as a function of wavelength, a stable long term Solar Spectral Irradiance (SSI) data set is required. This data set is crucial for understanding Earth's climate and atmosphere, which is the primary motivation for NASA missions taking these measurements. Measurements of SSI are traditionally done via high budget missions such as SORCE and TSIS. While effective thus far, there are potential problems associated with relying too heavily on large scale missions. Since these missions depend on huge spacecraft buses to accommodate many instruments, they are very expensive to build and launch, which poses financial problems as well as the risk of loss of continuity. For example, NASA's original followup mission to SORCE, GLORY, was lost in a launch failure, putting the continuation of the solar data record at risk. This is a critical point; overlap analysis is key to reducing uncertainty of past measurements as well as for maintaining the record. Therefore, a large gap between missions is unacceptable. A potential solution to these problems may exist. With the advent of satellite miniaturization, known colloquially as CubeSats, spending hundreds of millions of dollars on multi instrument satellite buses can be avoided by launching cheap miniature satellites with individual instruments instead. If able to perform to the same stability requirements as high budget missions, they would reduce the cost of making these measurements, open up high budget missions for other space experimentation, and provide a solution for the data gap problem. The relatively cheap cost of these instruments makes launching a CubeSat constellation much more feasible, allowing for replaceability if a single instrument fails. Thus there is significant interest in determining whether these CubeSats can meet long-term stability requirements. This is the purpose behind CSIM, which was launched December 3rd, 2018 with the goal of advancing new technology and demonstrating unique capabilities of a complete SSI mission with compact design and low mass [30].

Measurements are taken using a relatively simple instrument design. CSIM performs daily solar SSI observations from 200-2800nm. The instrument design consists of two separate channels. Each channel is exposed by opening a slit, which allows incoming solar radiation to hit a collimating mirror, reflecting the radiation onto a prism,which disperses solar radiation onto a photodiode detector focal plane. CSIM also contains an Electronic Substitution Radiometer (ESR), which is a slow, robust absolute detector that periodically calibrates the photodiodes and provides a way to track the degradation of the photodiode itself. To record an ESR measurement, solar radiation follows the same optical path as described above, except the light from the prism is reflected onto a separate focusing mirror (rather than the photodiode), which in turn points towards a bolometer. Spectral resolution of CSIM is extremely similar to TSIS SIM [30]. The spectral resolution of TSIS SIM ranges from 1 nm in the UV to 35 nm in NIR [26].

Measuring solar spectral variability over extremely long time scales requires continuous space-based measurements with a high degree of stability. The primary issue with solar viewing instruments is degradation [2]. Instrument degradation can have multiple causes with potentially unpredictable effects. An example of a potential degradation source might be high energy proton radiation, which would result in the degradation of the instrument electronics as well as the photodiode. This would be modeled as a function of mission time. Another example might be the polymerization of outgassed molecules onto optical surfaces. This may result in a buildup on surfaces, which in turn absorbs a wavelength dependent fraction of incoming solar radiation[2],[23],[25]. This buildup changes over the lifetime of the mission, and is most highly correlated with instrument exposure time. This latter form of degradation is observed to outweigh all other aging effects and in certain wavelength regions can result in measurements many orders of magnitude removed from stability requirements. This means that correcting this exposure dependent form of degradation is essential for recording SSI measurements over long periods of time.

This degradation can be tracked by duty cycling multiple identical channels on the same instrument [6],[13],[15]. A primary channel, sometimes referred to as channel 1, other times referred to as channel A, is exposed as frequently as is necessary to meet science requirements, approximately once a day. A secondary channel (channel 2 or channel B) is exposed significantly less frequently (approximately 10% of channel 1). This second instrument channel is thought to degrade less, based on the theory that this degradation is primarily a function of exposure time. The best method for correction has not been concretely identified [26]. A simple solution might be to assume channel 2 does not degrade at all; and thus a comparison of channel 1 measurements to channel 2 measurements would allow for the adjustment of channel 1 data. A more realistic method is to assume both are degrading at different orders of magnitude, thus requiring a model relating all possible factors relating to instrument degradation. These factors may include accumulated exposure time, mission time, exposure frequency, instrument temperature, solar variability, distance between the instrument and the Sun, and instrument pointing [26]. Quantifying these factors would be difficult even in a laboratory setting; the fact that these instruments are in space operating for potentially longer than a decade makes the problem even more challenging. This issue notwithstanding, because degradation exceeds solar variability by potentially many orders of magnitude, any errors in correction could potentially propagate and result in an inaccurate representation of solar behavior over time. The fact that long-term trends identified by independent solar observing instruments all differ suggest the scale of the challenge [7], [10],[20], [23], [26],[32].

There exist multiple missions measuring SSI at different resolutions and over different time scales [7], but the primary mission of comparison for CSIM is TSIS. TSIS is the current gold standard for SSI measurements; corrected integrated TSIS SIM SSI agrees with independent observations of TSI within 0.0045% [26]. In order to compare CSIM measurements with TSIS measurements, a similar correction methodology is required for CSIM data. However with CSIM there exists a problem previously unobserved on solar instruments; CSIM exhibited apparent data recovery rather than data degradation in the first few months of the mission. This recovery was also wavelength dependent; regions that usually don't degrade 'recovered' much more than regions with high degradation, potentially implying that the high degradation present at smaller wavelengths overcame this recovery. This region of recovery was then followed by a pattern that appeared similar to the degradation exhibited by SORCE and TSIS.

In this work I present a correction of degradation for CSIM based on a simplified version of the method used on the TSIS SIM instrument, and with two different methods for dealing with the recovery exhibited by CSIM. While the mission is still ongoing, preliminary conclusions suggest that although the spacecraft has had many issues associated with the particular implementation of the spacecraft bus design (such as pointing, bus voltage drops, loss of contact) the instrument itself is still able to take SSI data to high precision, suggesting future CubeSat missions are potentially a long term viable alternative to continuing this data record.

Chapter 2

Data and Method

2.1 Measurement Protocol

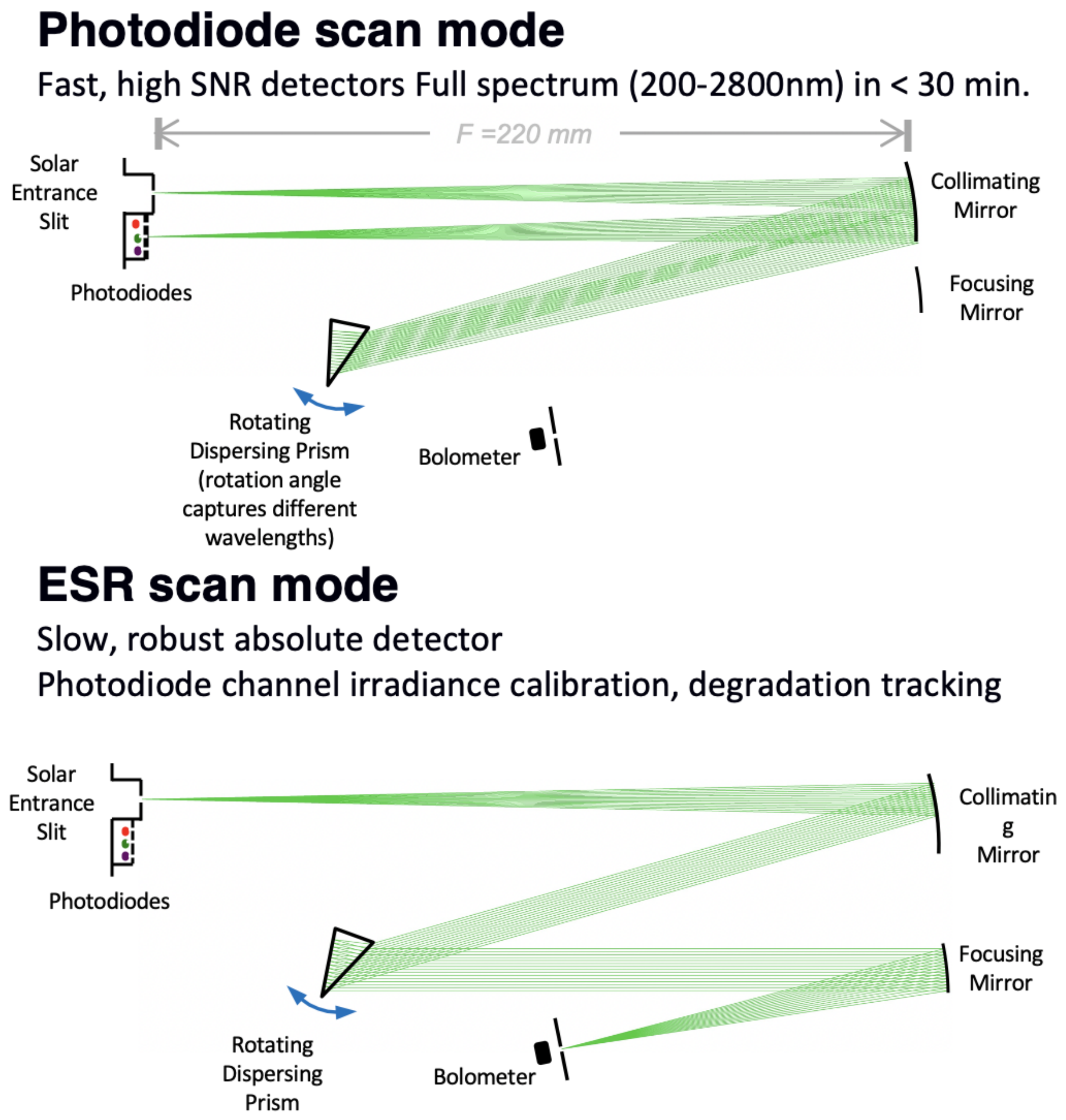
CSIM uses fast, high signal-to-noise ratio detectors that are able to measure the full spectrum (200-2800 nm) in around 25 minutes. This translates to roughly half a second per data point with photodiodes. The ESR measurement system is much slower, requiring about 10-50 seconds per point, meaning measuring the full spectra takes about 14 orbits. In this paper data ranged from March 25th, 2019 to November 22nd, 2020. For this range, accumulated exposure on CSIM Channel 1 was roughly 388 hours, whereas accumulated exposure on CSIM Channel 2 was around 43 hours. For a diagram of this measurement process, see figure 2.1.

2.2 Data Processing

Although data underwent filtering in the initial processing pipeline, there were additional steps taken in order to make spectral comparisons feasible. The initial data processing was performed by LASP scientists. My work used Python to analyze the processed data retrieved from LASP's internal server.

Scans are separated by observation ID, and cover different wavelength regions. These scans are mostly consistent in the wavelength region they cover, but due to the space-based nature of the instrument, data is less perfect than would be collected in a lab, i.e. scan gaps and incompleteness are more frequent. The primary observation IDs I worked with were 1,2,3,4, and 5, because they covered the full spectrum. These wavelength regions were trimmed in order to avoid inconsistency.

Figure 2.1: Here is a simplified diagram demonstrating measurement protocol for both photodiode detectors and the ESR detector. The photodiode is able to complete a full scan in about 25 minutes (0.5s per point) and is used for primary science. The ESR detector is an absolute detector, but is slow and sporadic, taking about 10-50s per point. A full ESR scan is obtained only after 14 points. This data is used to re-calibrate the photodiodes, since they aren't absolute detectors. Of course the actual instrument has many more components (dealing with operations and spacecraft health), but this simplified measurement diagram is sufficient for the purposes of this analysis.


Photodiode scan mode
Fast, high SNR detectors Full spectrum (200-2800nm) in < 30 min.
F =220 mm
Solar Entrance Slit
Collimating Mirror
Photodiodes
Focusing Mirror
Rotating Dispersing Prism (rotation angle captures different wavelengths)
Bolometer
ESR scan mode
Slow, robust absolute detector
Photodiode channel irradiance calibration, degradation tracking
Solar Entrance Slit
Collimatin g Mirror
Photodiodes
Focusing Mirror
Rotating Dispersing Prism
Bolometer

ID 1 was designated as the region covering 200-286nm, ID 2 = covered 286-387nm, ID 3 covered 387-900nm, ID 4 covered 900-1500nm, and ID 5 covered 1500-2300nm. Although CSIM is supposed to be able to take data up to 2800 nm, data beyond 2300nm was omitted due too inconsistency. Moreover, for the purposes of comparison this is a moot point; data from 2400-2800nm cannot be compared to TSIS because TSIS SIM only goes to 2400nm. As data from 2300nm -2400nm exhibits hardly any variability, this can be omitted from comparisons safely. Due to the precision level of the instrument, the scans cannot measure the exact same regions every time, so in order to account for this, each scan was interpolated to a standard wavelength grid (characterized by the longest scan in any given wavelength region/ID). Doing so ensured that integrals over these regions were accurate; if these integrals were performed on inconsistent wavelength grids they would be inaccurate. Kappa calculations (described in detail in Chapter 3) were done by considering each of these IDs independently. The full scan was built by taking the above data separated by ID and combining them on the Julian day they were measured. Only scans that covered this entire region were included in plots of all scans. This allowed for a complete set of all scans that covered the entire region. Storing these values in a data frame allowed for easy access and manipulation. Solar variability plots were generated using these full scans. Data was not filtered or removed from the final data set, but in kappa calculations obvious outliers from the integrated spectra, determined by a statistical z score greater than 3, were removed when determining kappa. This was done to provide a cleaner fit, and the corrections were applied to the final spectra without the removing or filtering of any data. The Python analysis program is attached in Appendix A with functions and explanations.

The CSIM data set was retrieved from a LASP server, imported to a .csv file using IDL, and then opened with a Python program. TSIS TSI, TSIS SIM, and NRLSSI2 (solar model) data was taken from LISIRD (https://lasp.colorado.edu/lisird/), where solar data sets are available to download in .csv format to the general public.

2.3 Spacecraft Performance

CSIM experienced quite a few problems that negatively impacted its ability to carry out its primary goal. There were three primary issues: SD Card failure, period noise interference, and pointing offsets.

There was a delay in mission start due to the primary SD card failing. This occurred in mid-January 2019 and resulted in loss of communication. It was several weeks before communications were restored, and new programs and scripts were uploaded. These new commands switched data storage to the redundant SD card, as well as building in new fail-safes to prevent SD card failure from causing lock-out again. A second communications lock-out anomaly was experienced in November 2019 (the same problem as the initial lock-out). However, the correct software patches were in place that allowed for this SD card to be reset, and since then CSIM has not experienced any anomalies of this type.

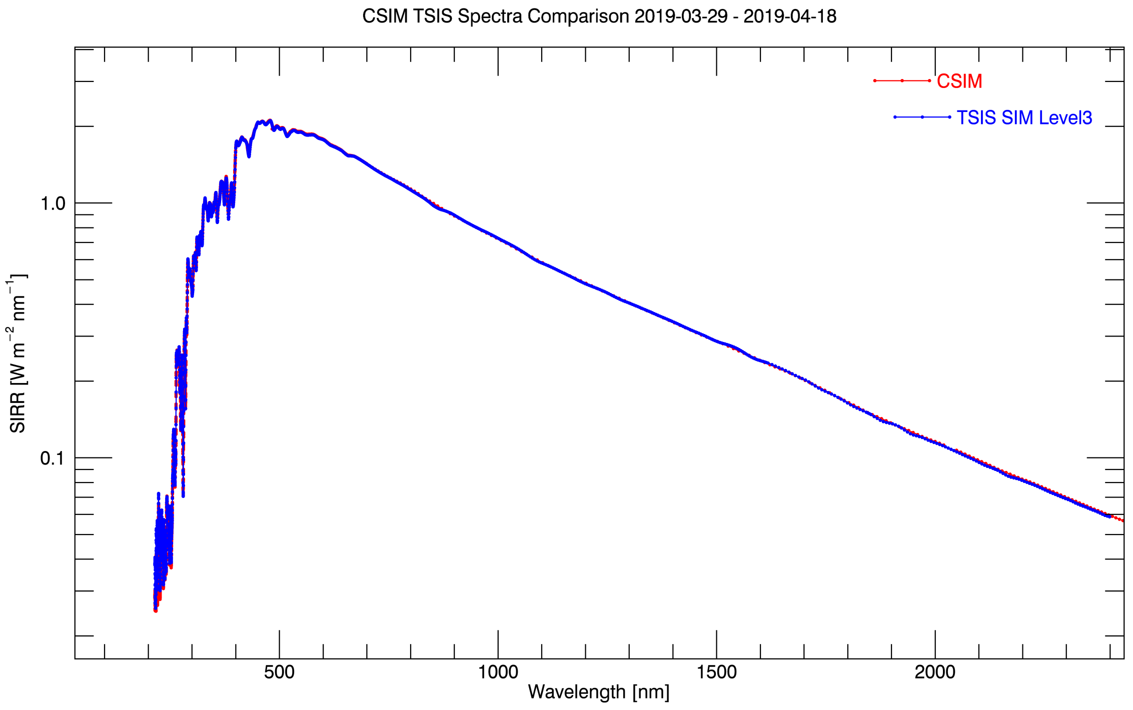
Another issue during the early time period was attempting to understand periodic noise interference seen in the photodiode scans. The cause of this noise appeared to be related to the battery heater circuit turning on and off, producing a slight DC voltage offset in the signal amplifiers. Fortunately, since dark currents were monitored in the redundant channel, a DC correction was applied to the measured signals, accounting for the periodic noise interference occurring during channel 1 diode scans. This was implemented in data processing software and now has been automated.

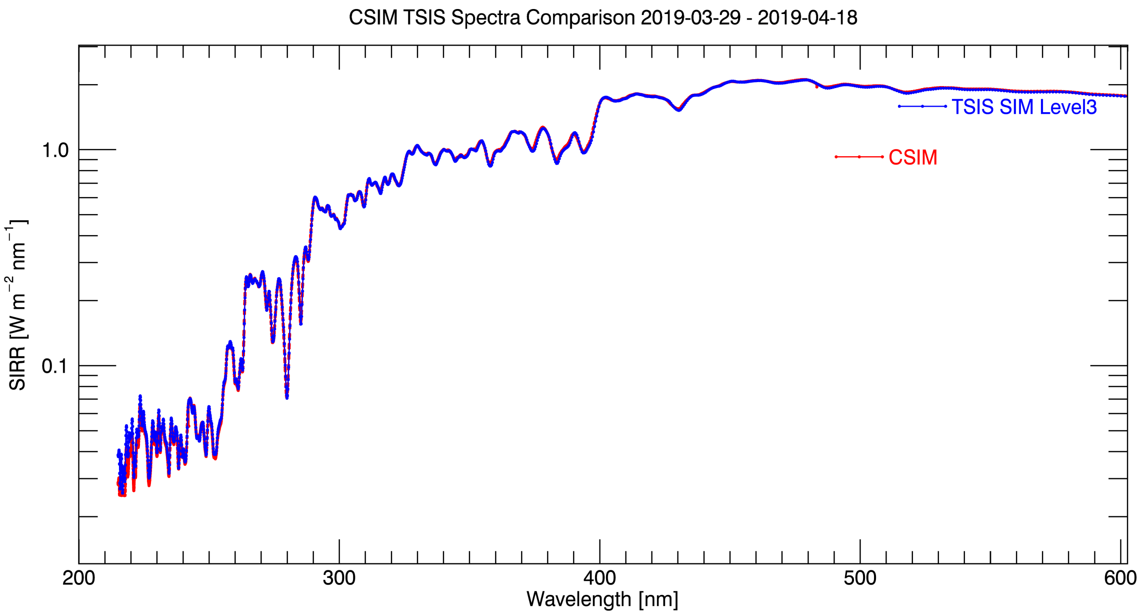
An issue that has proven more difficult to correct is CSIM's poor solar pointing stability; CSIM pointing offsets are larger than expected. The BCT pointing stability turned out to be poorer than pre-launch estimates, resulting in poor azimuthal pointing. This is the most unfavorable direction as this is the dispersion direction of solar radiation. In September 2019 a BCT star tracker calibration scan was done to reset the instrument fine sun sensor to star tracker pointing. While this initially worked, since then it has drifted back. Noise spikes in pointing data appear to correlate with noise spikes in diode data.

2.4 Solar Spectral Measurement Achievements

CSIM had excellent overlap with TSIS in the first three of weeks of its mission (see figure 2.1). Instruments agreed to about 1% in absolute scale, which was a major achievement. The challenge was then to test if CSIM could capture solar variability, which can vary greatly in magnitude over long time scales. Solar variability can range up to 10% in the 200-300nm region, 1% in the 300-400nm region, 0.1% in the 400-700nm (the bulk of the spectra) and 0.01% beyond 700nm. Stability of 0.05% is required a year to quantify the spectral partitioning of irradiance variability, and this must be maintained over solar timescales (i.e. a minimum of several years). Even though CSIM performed very well in this short time period, the challenge is measuring extremely small differences over long timescales, requiring an effective correction method.

Figure 2.2: Here three weeks of continuous SSI observations of CSIM (blue) and TSIS (red) are shown overplotted ($\sim$ 21 daily spectra). All spectra are plotted on their native irradiance scale, with no offsets or scaling applied. The left axes is shown on a log scale to demonstrate the magnitude differences of the measurements. This is shown to demonstrate that CSIM and TSIS data measure the full spectra to nearly the same precision level, an achievement in an of itself.





Chapter 3

Data Analysis

3.1 Kappa Method

The methodology used for correcting CSIM is an on-orbit analytic method similar to methods for degradation correction on SORCE SIM and TSIS SIM, and involves duty cycling multiple identical instrument channels to provide a mechanism for tracking optical degradation. SORCE SIM only had two channels, and it was assumed that these identical channels degraded at the same rate In order to test if this assumption was valid, TSIS was launched with three channels: channel 1 exposed daily, channel 2 exposed at 10% the frequency of channel 1, and a third channel exposed twice a year [26]. Comparisons of different degradation rates of these channels resulted in significantly more precise corrections. CSIM, only having two channels, cannot perform this analysis to the same level as TSIS, and thus is not expected to match TSIS requirements on its own. Therefore assumptions and methods are more similar to those used on SORCE SIM. This method for correction has been nicknamed "The Kappa Method."

The Kappa Method for correction relies on a simplified model: the degradation is purely from the reduced transmissivity of the optical system and as such is only a function of exposure time and wavelength. .

On any given solar day, a scan taken at the same time by channel 2 and channel 1 would theoretically measure the same irradiance.

$$I_{correctedCH1} = I_{correctedCH2} \tag{3.1}$$

The corrected irradiance would have the following form

$$I_{corrected} = \frac{I_{uncorrected}}{degradation_{optics}} \tag{3.2}$$

A key assumption here is that the thickness of the layer on the prism is increasing linearly with exposure time, thus through Beer's Law, the light transmitted through the layer is attenuated exponentially. Thus a simplified expression for degradation is:

$$Degradation_{optics} = e^{-\kappa(\lambda)*t_{exposure}} \tag{3.3}$$

Another assumption made with this analysis is that $\kappa(\lambda)$ is the same regardless of instrument channel (i.e. has the same degradation rate with exposure time) Through an analysis of TSIS data, this is known to not be the case; however, as there is no third channel on CSIM to track this, kappa is assumed to be the same for both channels. From coinciding measurements of channel 1 and channel 2 :

$$I_{uncorrectedchannel1} * e^{-\kappa(\lambda)*t_{exposurechannel1}} = I_{uncorrectedchannel2} * e^{-\kappa(\lambda)*t_{exposurechannel2}} \tag{3.4}$$

$$\frac{I_{uncorrectedCH1}}{I_{uncorrectedCH2}} = \frac{e^{-\kappa(\lambda)*t_{-exposureCH2}}}{e^{-\kappa(\lambda)*t_{-exposureCH1}}} \tag{3.5}$$

$$\ln(\frac{I_{uncorrectedCH11}}{I_{uncorrectedCH2}}) = -\kappa(\lambda) * \Delta t_{exposure} \tag{3.6}$$

Thus by plotting the natural log of the ratio of irradiances (recorded simultaneously) against the differences in exposure time, a kappa function can be determined. In order to account for the fact that kappa is a function of wavelength, kappa was calculated for several wavelength regions, integrated over small bands. The choice to integrate bands rather than perform this analysis at each individual wavelength is due to two reasons, (1) integrated wavelength bands instead of SSI minimizes error in ratios from both instrument noise and the fact that SSI measurements are interpolated to a common wavelength grid, and (2) the degradation function (as seen on SORCE and TSIS) is known to be a smooth function of wavelength. In this process, very small regions were specifically chosen in the high energy region (200-286nm, where kappa has significantly greater magnitude of variability), and larger regions were chosen in the near infrared region (because here kappa is smooth and close to zero). Choosing these wavelength regions accordingly allowed for the generation of a kappa curve dependent on wavelength. This was then used to correct the solar spectra (through Equation 3.6 above). Because CSIM exhibited data recovery in early regions, this posed a problem to a standard kappa analysis. This problem was explored through two methods. The first involves the standard kappa calculation over time with no attempt to treat the recovery region as separate. The second involves a piecewise correction method for two independent regions, the recovery region, spanning from mission start time to about January 2020, and the post-recovery region, spanning from January 2020 to present. This was chosen precisely in order to obtain better fits for kappa, but has the unfortunate problem of the first region having significantly fewer points between channel 1 and 2 to compare. Another potential solution to this problem is to simply set the mission start time to after the recovery disappeared. This could work, as overlap analysis can still be performed with TSIS using the second region, but both regions are included in this paper. To perform this analysis, 40 wavelength regions in between 200-286nm (since the degradation is significantly greater in this area), and 10 wavelength regions each in the other primary four regions (286-387nm, 387-900nm, 900-1500nm, 1500-2300nm) were chosen. Note how naturally this implies larger wavelength regions as you move towards the less energetic regions of the spectrum. A process was then automated to integrate these regions, find a slope for a each kappa, and calculate a kappa curve.

3.2 Overall Kappa

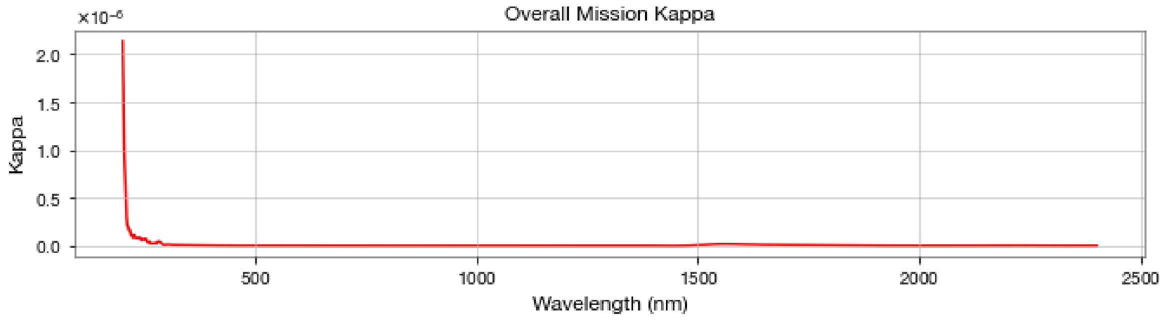
An overall mission kappa was calculated using the method described above without any attempt to account for the data recovery exhibited in the first few months of the mission. A plot of overall kappa as a function of wavelength is shown in figure 3.1. As a sample, plots over two wavelength bands are provided, one in the UV region (high energy) and one in the visible region (low energy), to demonstrate the effectiveness of an overall kappa method (Figures 3.2 and 3.3).

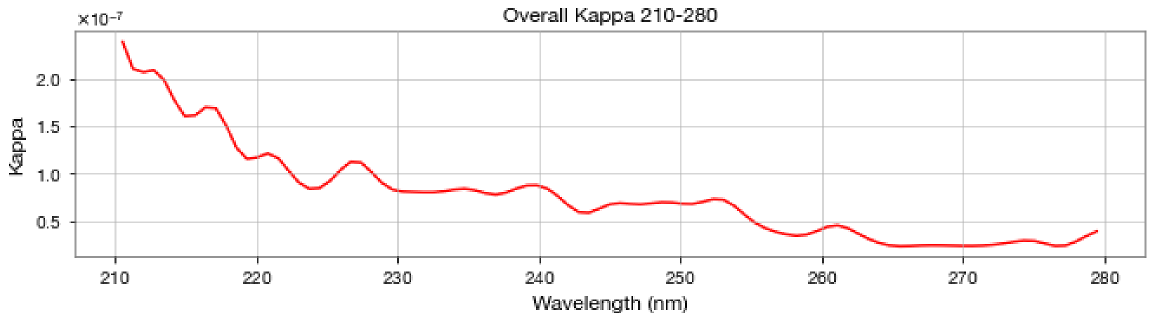
The overall mission kappa performs well in the UV, but not in the visible region. In the latter region, a clear split is observed where recovery appears to stop, resulting in a poor average fit.

3.3 Two Stage Kappa

As an attempt to account for the problems associated with an overall kappa, a two stage piecewise kappa function was calculated. This was done by treating the solar spectral measurements prior to January 2020 as the recovery region, and spectral measurements following January 2020 as the post-recovery region. Plots generated of these two kappa functions are shown in figure 3.4. As a sample, plots over the same two wavelength bands are provided for both the recovery and the post-recover region, to demonstrate the effectiveness of a two stage kappa method (Figures 3.5 and 3.6). Overall, fits are improved. Note that fewer points in the recovery region imply less accurate kappas.

Figure 3.1: Plots of the generated kappa function are shown here over multiple wavelength ranges to demonstrate the overall behavior of the degradation over wavelength. Note the degradation follows exponential behavior as it approaches smaller wavelengths, and drops to nearly zero for longer wavelengths.





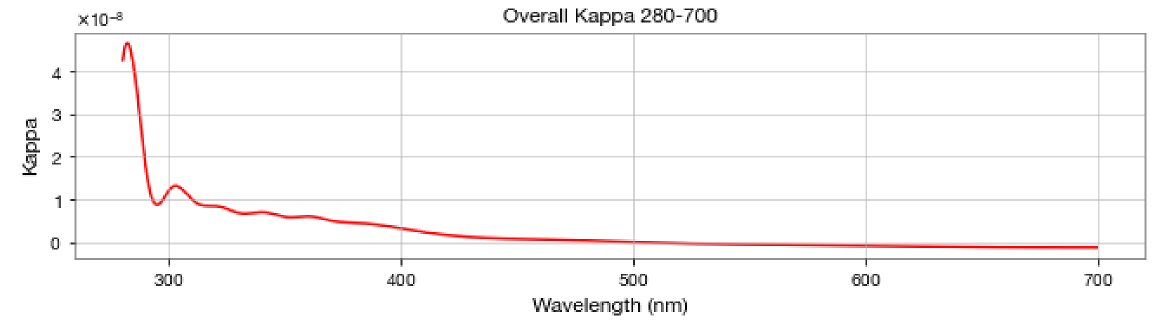


Figure 3.2: The top left (channel 1) and bottom left (channel 2) plots show the solar spectra over this wavelength range. All spectra over full mission time are plotted on top of one another here in order to show the level of degradation present. The top right plot has integrated data of both channel 1 (blue) and channel 2 (red) for this particular wavelength region. Note the relative stability of channel 2 as well as the percentage axis on the right. The bottom right plot shows the natural log of the integrated ratios plotted over the change in exposure time in order to show kappa is fit for this region. For this particular region, a linear fit performs relatively well. This region was chosen in order to show the effects of high degradation, as well as demonstrate that recovery is not visible here.

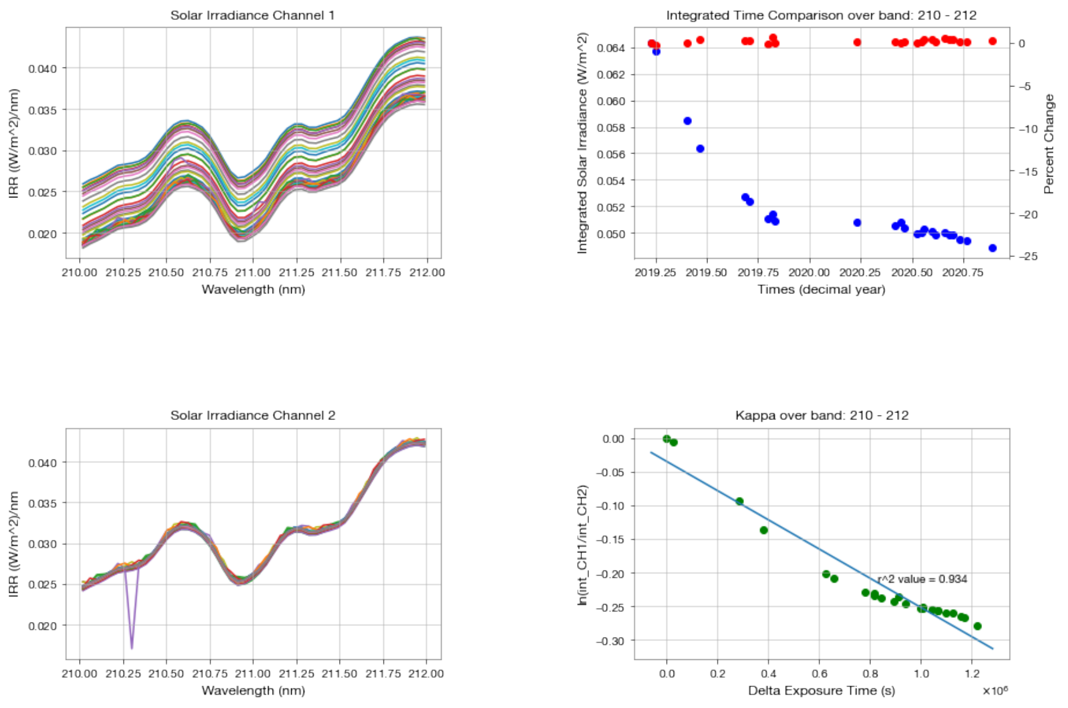


Figure 3.3: The top left (channel 1) and bottom left (channel 2) plots show the solar spectra over this wavelength range. All spectra over full mission time are plotted on top of one another here in order to show the level of degradation present. The top right plot has integrated data of both channel 1 (blue) and channel 2 (red) for this particular wavelength region. Note the relative stability of channel 2 as well as the percentage axis on the right. The bottom right plot shows the natural log of the integrated ratios plotted over the change in exposure time in order to show kappa is fit for this region. A linear trend seems to poorly apply here; corrections are naturally going to propagate a lot of error when attempting to average the recovery and post-recovery regions together. This region was chosen in order to show the effects of mixed degradation and recovery. There is a clear split when recovery stops and data appears to flatten.

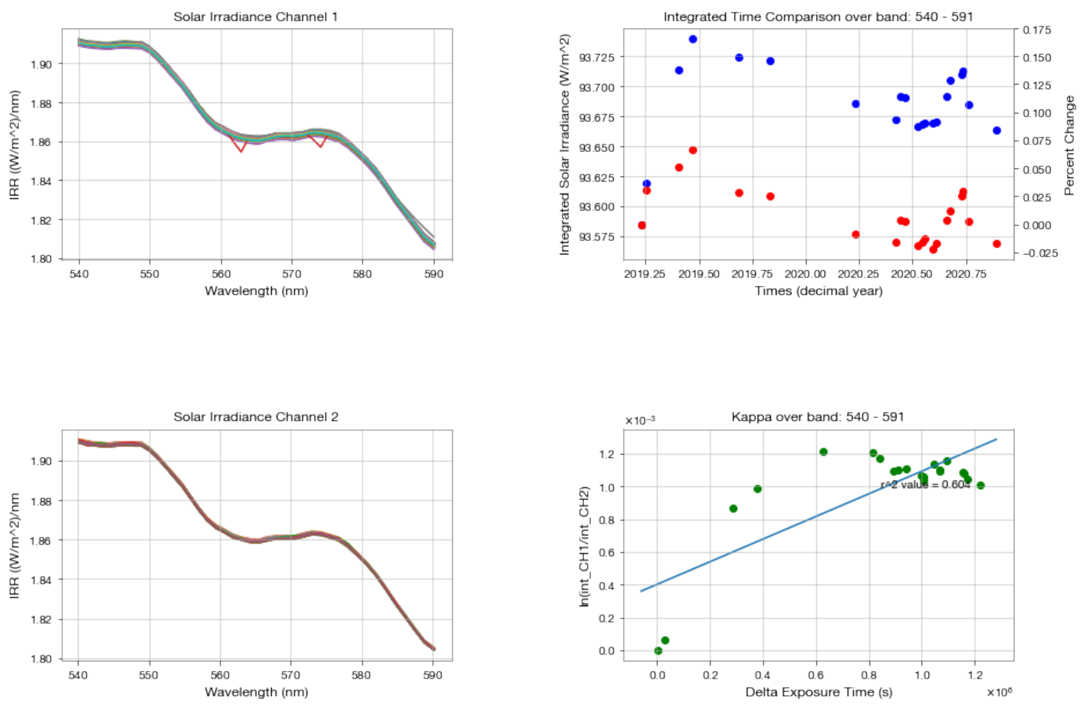


Figure 3.4: Plots of the generated kappa function are shown here over multiple wavelength ranges to demonstrate the overall behavior of the degradation over wavelength. Note the degradation follows exponential behavior as it approaches smaller wavelengths, and drops to nearly zero for longer wavelengths. Also note that while the shapes of the two kappas are similar, the magnitudes differ, which would have a significant impact on corrections.

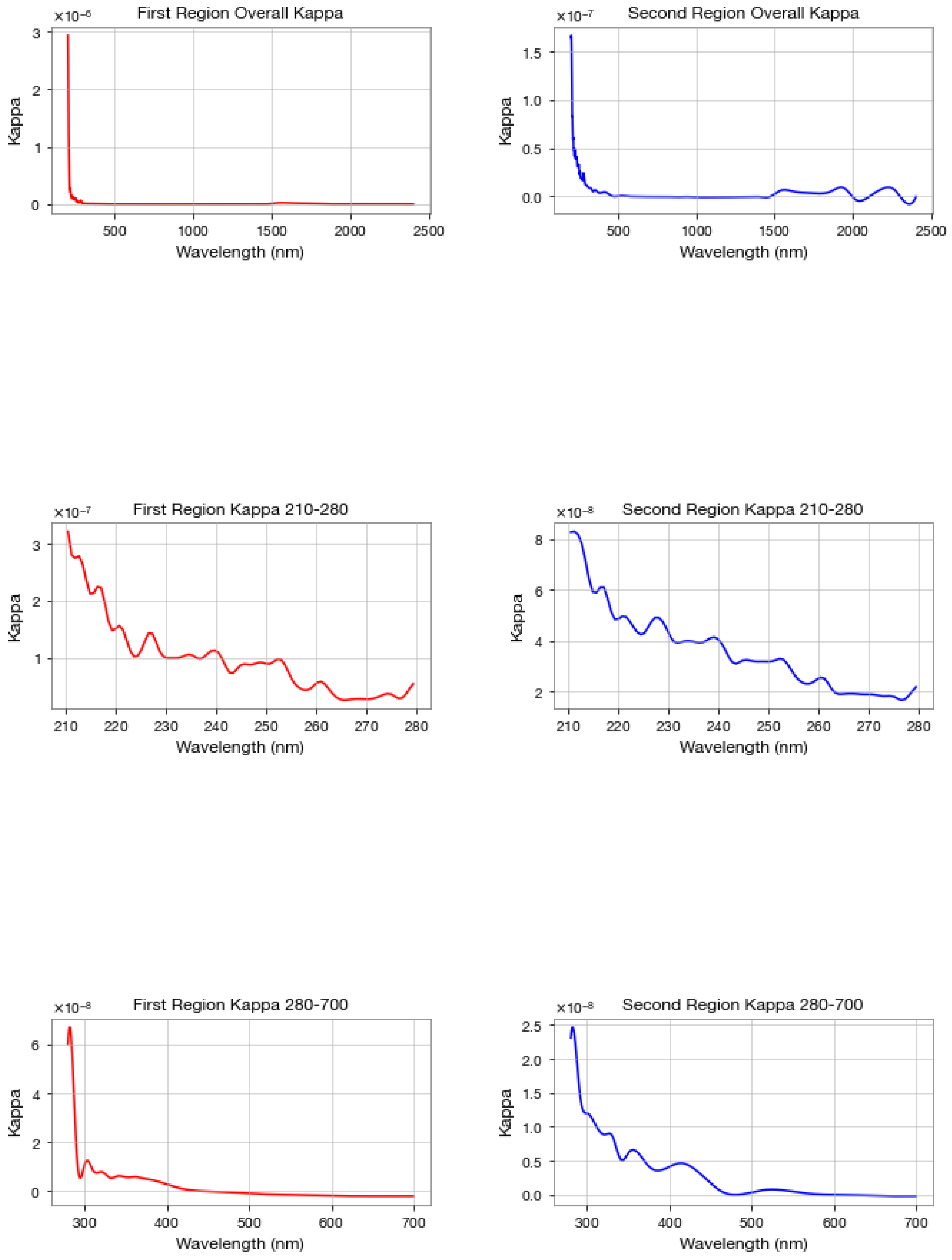


Figure 3.5: Here the performance level of the recovery region is shown. There are fewer comparison points, implying less accuracy, but overall fit is good in both regions.

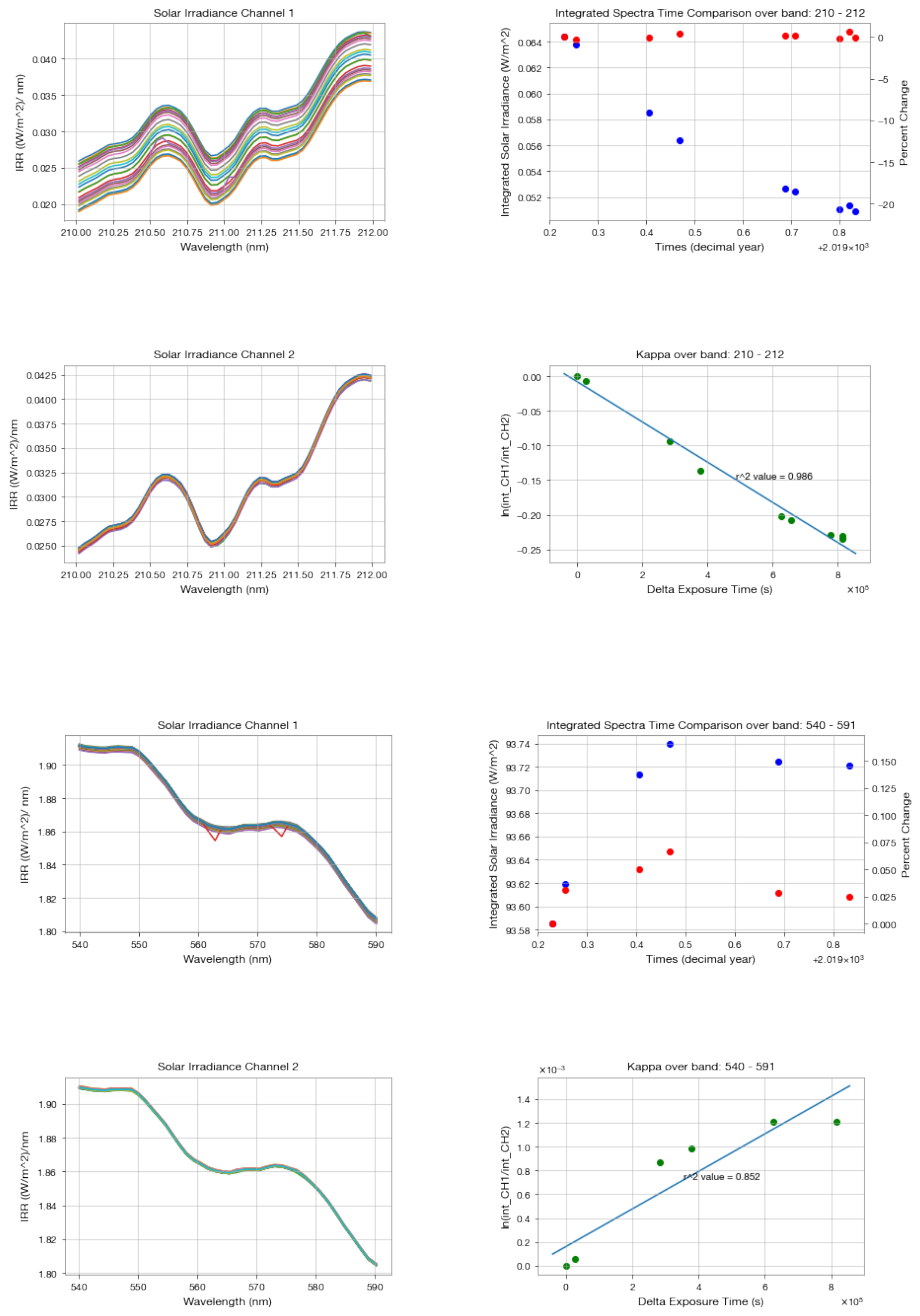
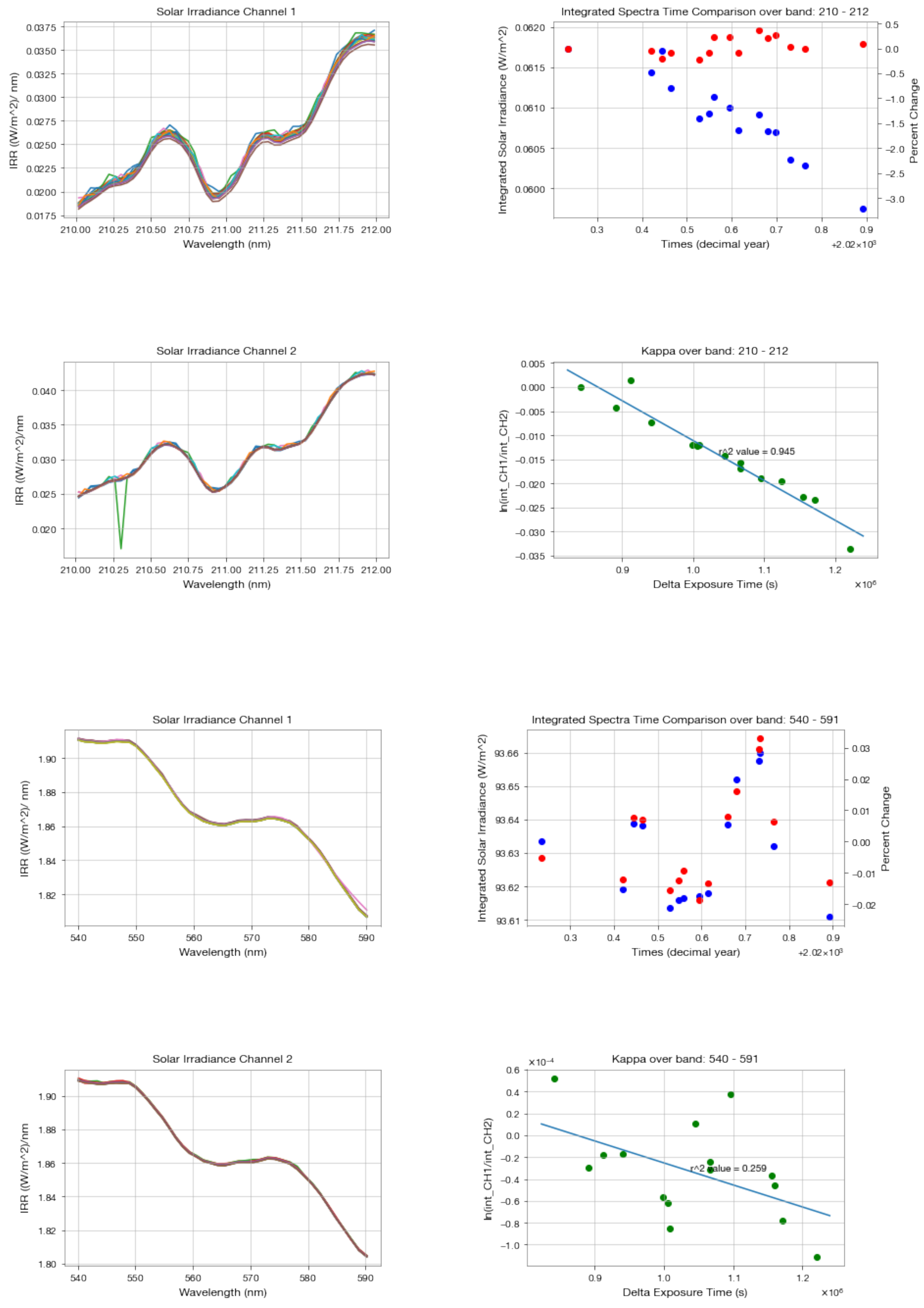


Figure 3.6: Here the performance level of the post-recovery region is shown. There are many more comparison points and overall fit is significantly improved in both regions. In the visible spectra, correlation appears to be poor; however, I would draw attention to the percentage scale of the integrated spectra. These points are hardly degrading, and correlation closer to zero is expected in low degradation regions.



Chapter 4

Corrections

A small aside: on TSIS, instrument degradation is undetectable (<0.01% per year) at wavelengths longer than 1050nm, and so no correction applies at these wavelengths. For CSIM, the recovery problem requires a correction at these wavelengths and so corrections are applied to the entire spectrum.

4.1 Comparisons to TSIS TSI

4.1.1 Overall Kappa

The first check performed to verify the accuracy of a correction is to integrate the total spectra and compare it to TSIS SIM's Integrated Spectra as well as TSIS TIM (Total Irradiance Monitor). The TSIS TIM measures the total solar irradiance (TSI) with extremely high absolute accuracy (<0.02% uncertainty). An offset must be applied to TSIS Integrated SSI and CSIM Integrated SSI to force agreement at the beginning of the mission. This offset comes from the fact that TIM makes measurements of the entire solar spectrum as a single total irradiance value, whereass TSIS SIM and CSIM focus on only a specific region of wavelengths. The measurements from both SIM instruments cover the bulk of the spectra (>96% TSI), and the parts of the spectra missing (Far IR, X-Ray, etc.) do not impact TSI trending significantly. As such, a constant offset is safe to apply. This back-of-the-book type check is a good test to see if corrections are matching solar trending.

Plots of these regions are produced using both an overall correction and a two stage correction. Corrections from the overall kappa method are shown in figure 4.1. CSIM Channel 2 was left uncorrected because its degradation was minimal.

4.1.2 Two Stage

A two stage integrated correction is shown in figure 4.2.

4.2 Wavelength Comparisons to an Empirical Solar Model

TSI comparisons are a necessary condition of agreement, but not a sufficient one. The purpose of the CSIM measurement is not to provide a reference of total solar irradiance, it is to analyze spectra (the spectral decomposition of the TSI). Therefore, it seems more apt to make wavelength region comparisons. CSIM was compared to one extant instrument, TSIS SIM, and the Naval Research Laboratory solar variability model for SSI (NRLSSI2) [5], which estimates daily solar irradiance from 1882 through present day from 115 nm to 100 μm. The model is an updated version of empirical models established in a series of papers [18][19][21]. Solar irradiance is derived from relating the Magnesium II index, that serves as a proxy for facular brightening and photometric sunspot index, to SSI observations on solar rotational timescales.

In order to make wavelength comparisons and simultaneously reduce noise associated with single wavelength measurements over time, five wavelength regions were chosen: 200-240nm, 240-310nm, 310-500nm, 500-700, 700-1800nm [26], integrated, and compared to the same regions on TSIS and from the model.

Overall comparisons reveal CSIM's performance is not great at the edges where the absolute magnitude of solar irradiance is small. The reason why is simple, the smaller the absolute measurement scale; the closer it is to the detection limit of the instrument. For example the UV region composes such a small signal region of the measured spectrum that scattering effects due to pointing issues, and slight errors in the correction are apt to manifest more clearly in this region than the others. On the other hand, the far IR region hardly degrades at all, but it also has a lower irradiance signal that is near the detection limit of the ESR detector, thus resulting in noisier measurements. However, the visible region corrections result in measurements that are quite consistent

Figure 4.1: The first plot contains CSIM Uncorrected Channel 1 data as well to provide a visual for the level of degradation present over the total measured spectra. Note the scale of the axes; the left axis is solar irradiance, the right axis is percentage difference, and the bottom axis is the time in decimal year format. Note that the corrected Integrated SSI, while only differing by 0.1% appears to still be trending negative compared to TSIS data, giving more credence to the idea that an overall kappa will not work for CSIM Data. The second plot contains the same data as the first, but omits the uncorrected spectra to give a sense of scale of the efficacy of the corrections.

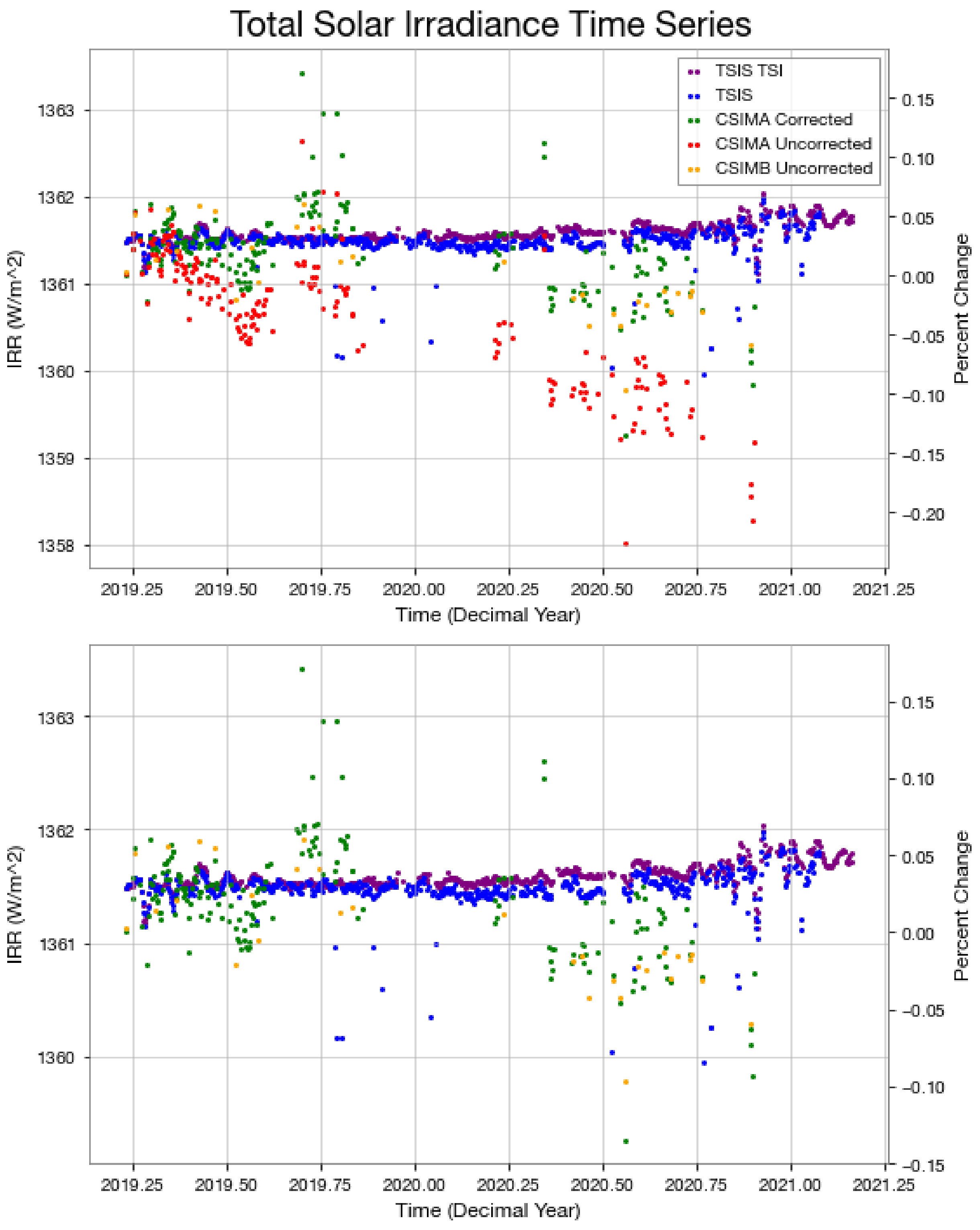
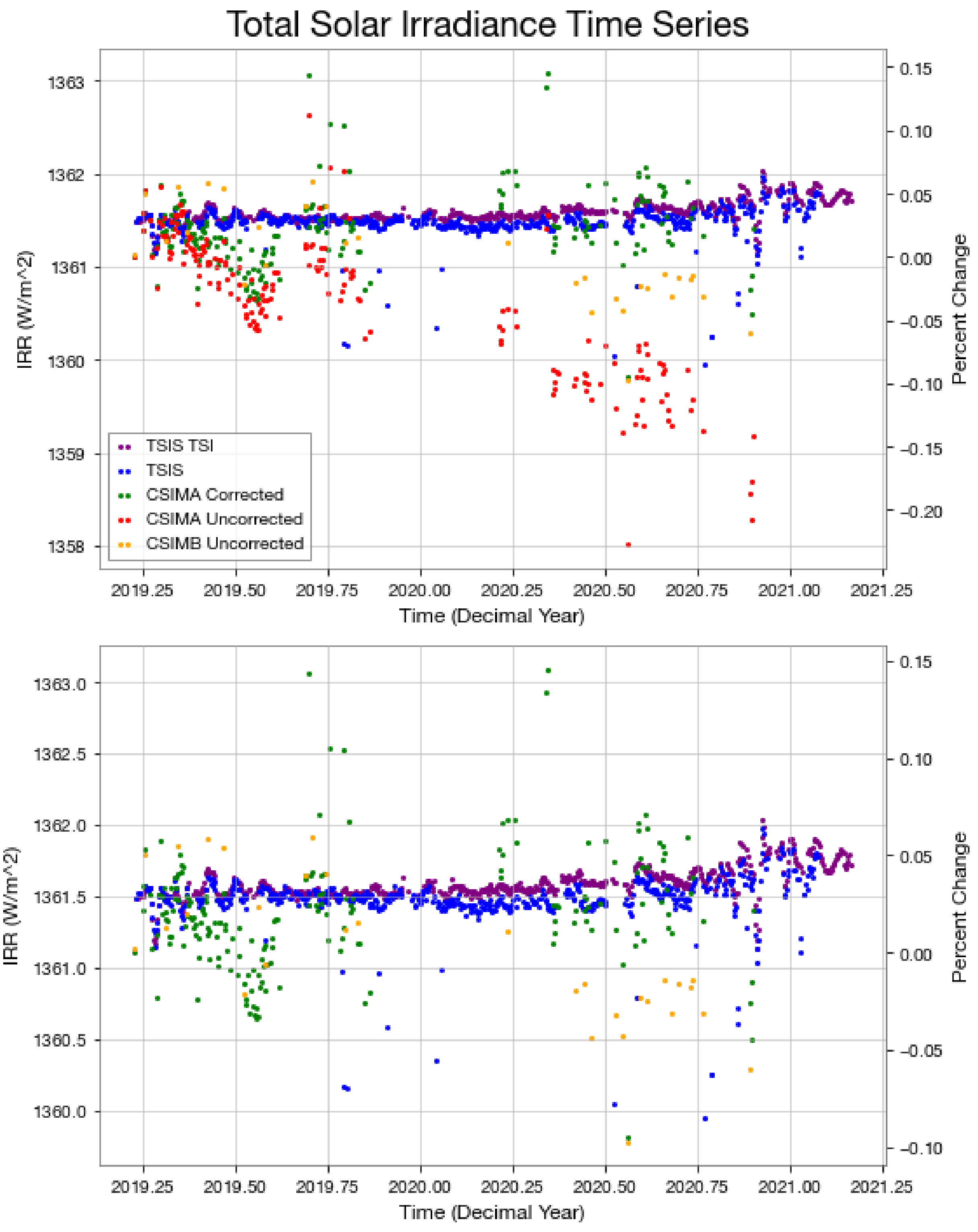


Figure 4.2: These plots contain the same data as before, the only difference being that CSIM Data is corrected with a two-stage kappa. The initial downward trend shows that recovery region data is not well corrected by this method. However, the integrated spectra farther along in mission time no longer exhibits the downward trend that the overall kappa method had issues with. This gives a slight clue that it is performing better over the long term than the overall kappa; however, the noise level of this data suggests high uncertainty. The second plot contains the same data as the first, but omits the uncorrected spectra to give a sense of scale of the efficacy of the corrections.



with TSIS measurements.

The actual comparisons of course differ between the two correction methods which I will detail below. Offsets are applied here to match CSIM CH1, CSIM CH2, TSIS SIM and the model at the measurement start period, and this is due to calibration scale differences between CSIM data and TSIS SIM Data. For comparison purposes, an offset is more than fair for plotting purposes since the concern here is with a relative stability comparison, however do note that absolute measurements do play an important role in an actual long term data record.

4.2.1 Overall Kappa

The SSI for five integrated wavelength bands is shown in figures 4.3, 4.4, 4.5, 4.6, and 4.7 with corrections using an overall kappa method. There is poor agreement in the 200-240nm band; but degradation on the order of 20% coupled with the high sensitivity of this region to both spacecraft problems and errors in corrections means this result is not too surprising. Corrections do not follow the general shape of SSI; this region begins by undercorrecting in the beginning and overcorrecting towards the end, resulting in SSI data that is within a poor 8% margin of TSIS data. There is significant improvement in the 240-300nm regions. Here the agreement in the early part of the mission (prior to August 2019) is excellent (within 0.05%). However, the overcorrection problem occurs again towards the end, resulting in differences of $\sim$0.6%. While not good, this is certainly encouraging. Agreement in the primary visible region (300-500nm) is excellent. The shape matches the shape of TSIS data as well as model data, and differences are on the order of 0.05%. In the secondary visible region (500-700nm) agreement is poorer. While on the order of tenths of a percent, the recovery effect mentioned previously is prominent. This recovery region is dealt with poorly by an overall kappa, but post-recovery the mission agreement is significantly better ($< 0.05\%$). However, given the trending of this correction it must be observed over time to determine whether this correction is actually accurate as the shape is not consistent with the shape of TSIS or model data. In the near IR region, where the primary detector is the ESR and not a photodiode, CSIM is known to be noisy. While within 0.1% of TSIS measurements, it is difficult to see trending.

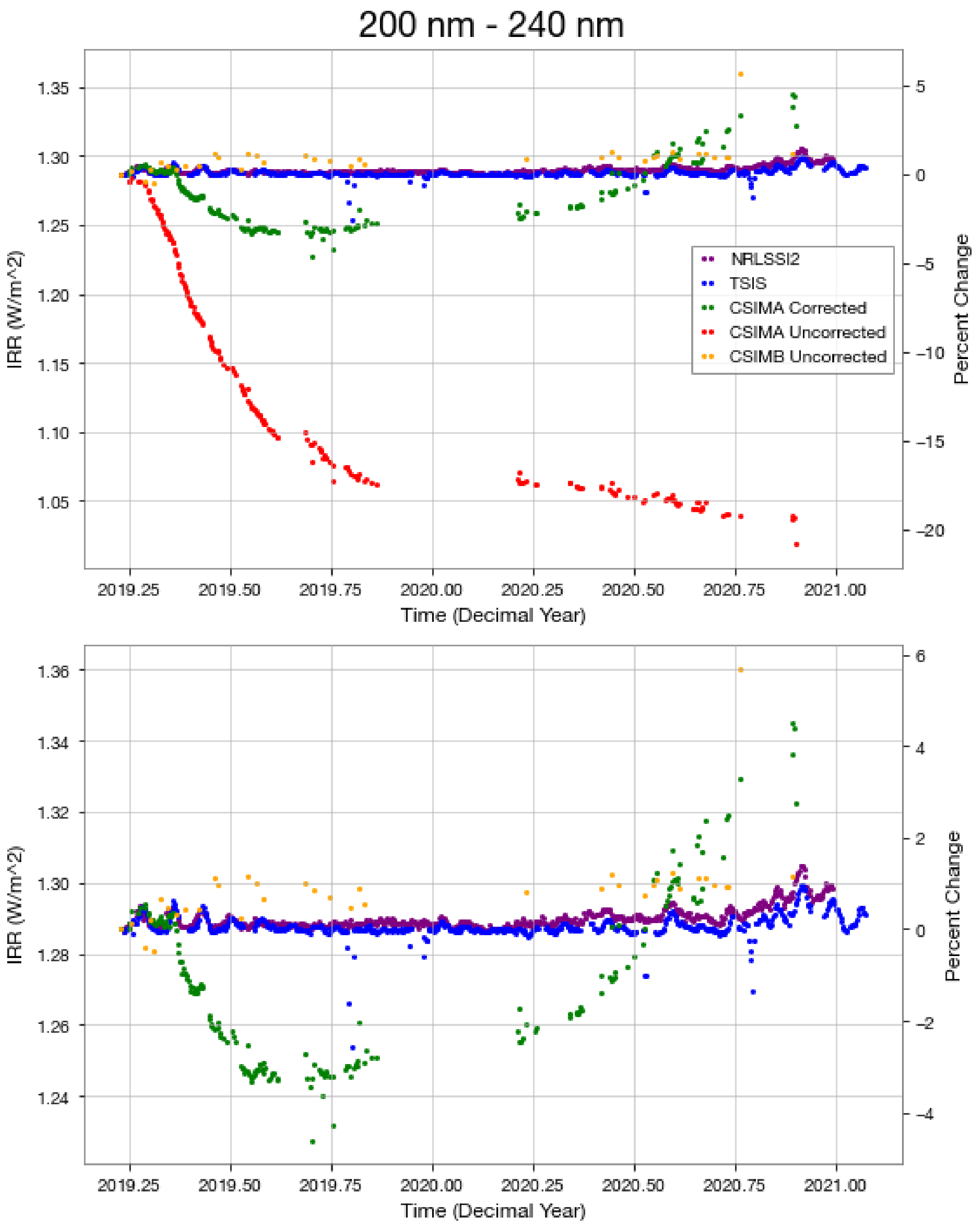
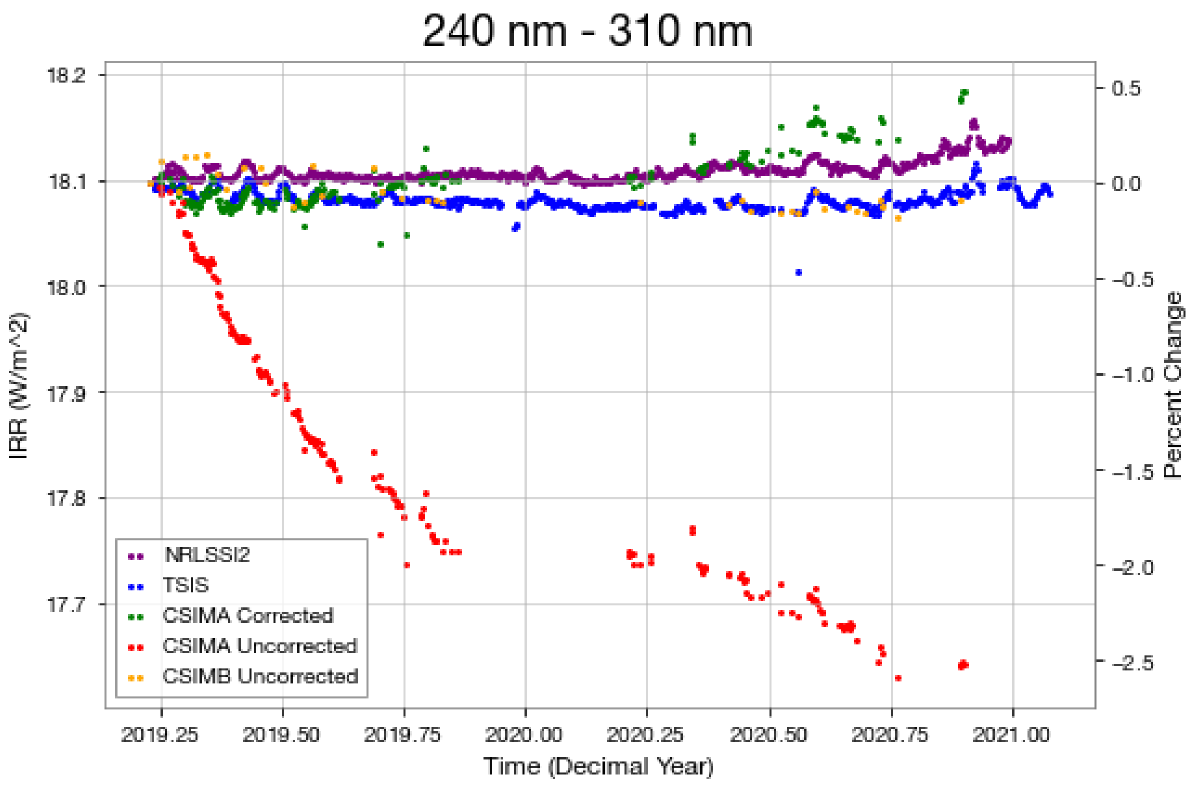


Figure 4.3: Comparison of overall kappa corrected CSIM channel 1 (green) to TSIS (blue), CSIM channel 2 (yellow), uncorrected CSIM channel 1 (red) and NRLSSI2 (purple) for the 200-240nm. Note percentage scale is on the right axis while absolute values are on the left axis. Here an overall kappa performs poorly. The second plot contains the same data as the first, but omits the uncorrected spectra to give a sense of scale of the efficacy of the corrections.



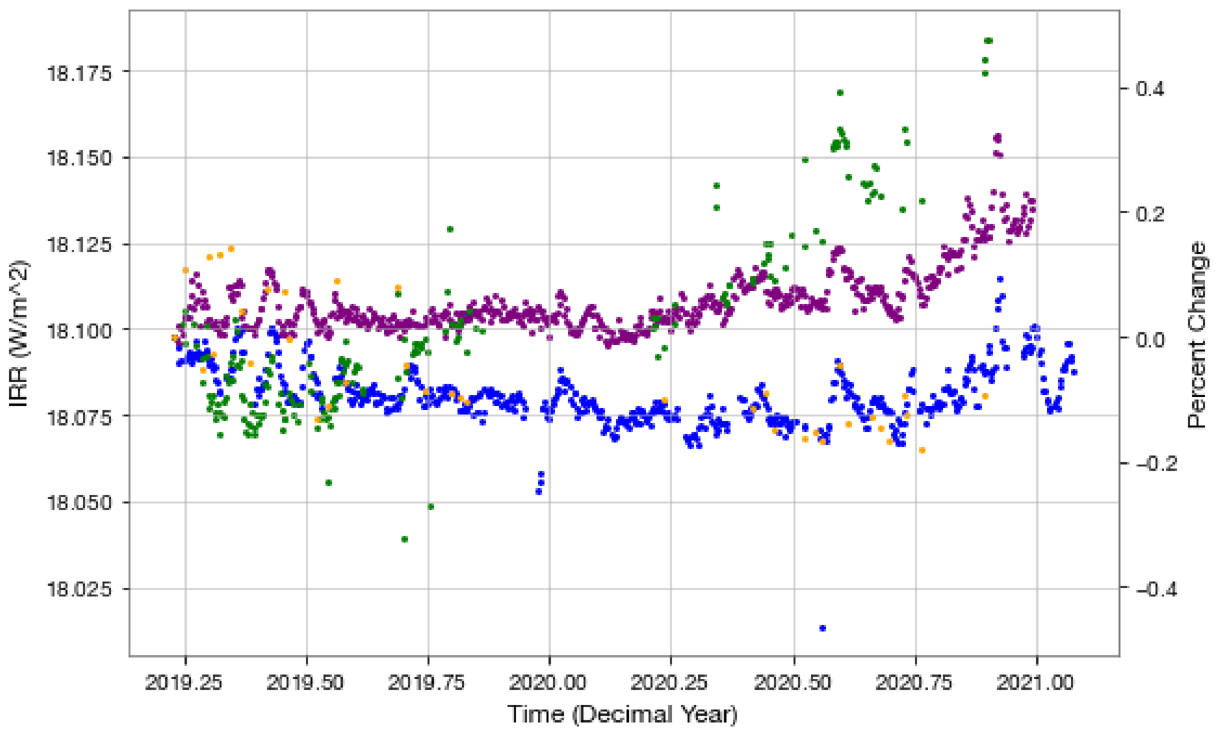


Figure 4.4: Comparison of overall kappa corrected CSIM channel 1 (green) to TSIS (blue), CSIM channel 2 (yellow), uncorrected CSIM channel 1 (red) and NRLSSI2 (purple) for 240-310nm. Note percentage scale is on the right axis while absolute values are on the left axis. Here agreement is excellent in the beginning, but over time becomes more inaccurate. The second plot contains the same data as the first, but omits the uncorrected spectra to give a sense of scale of the efficacy of the corrections.

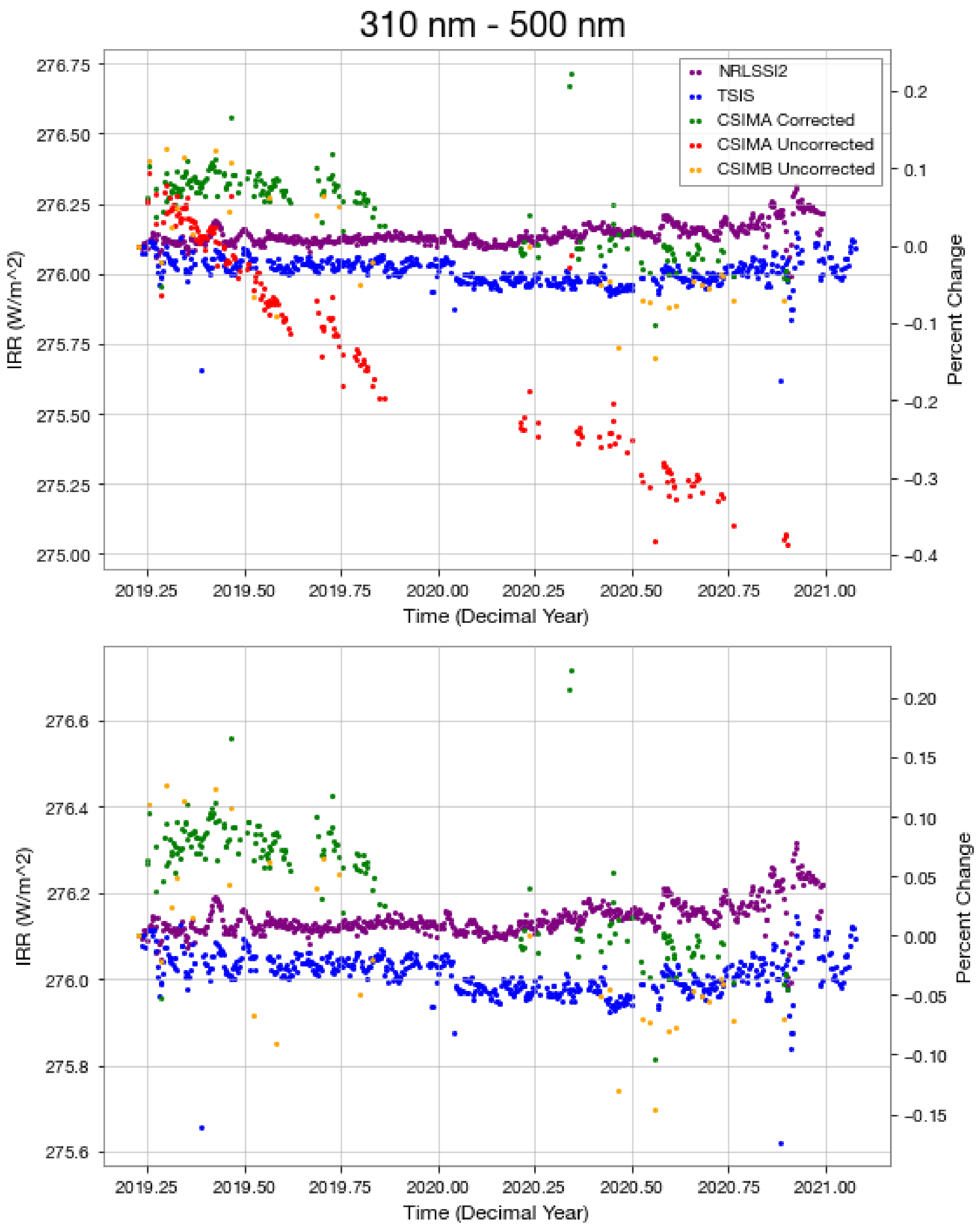
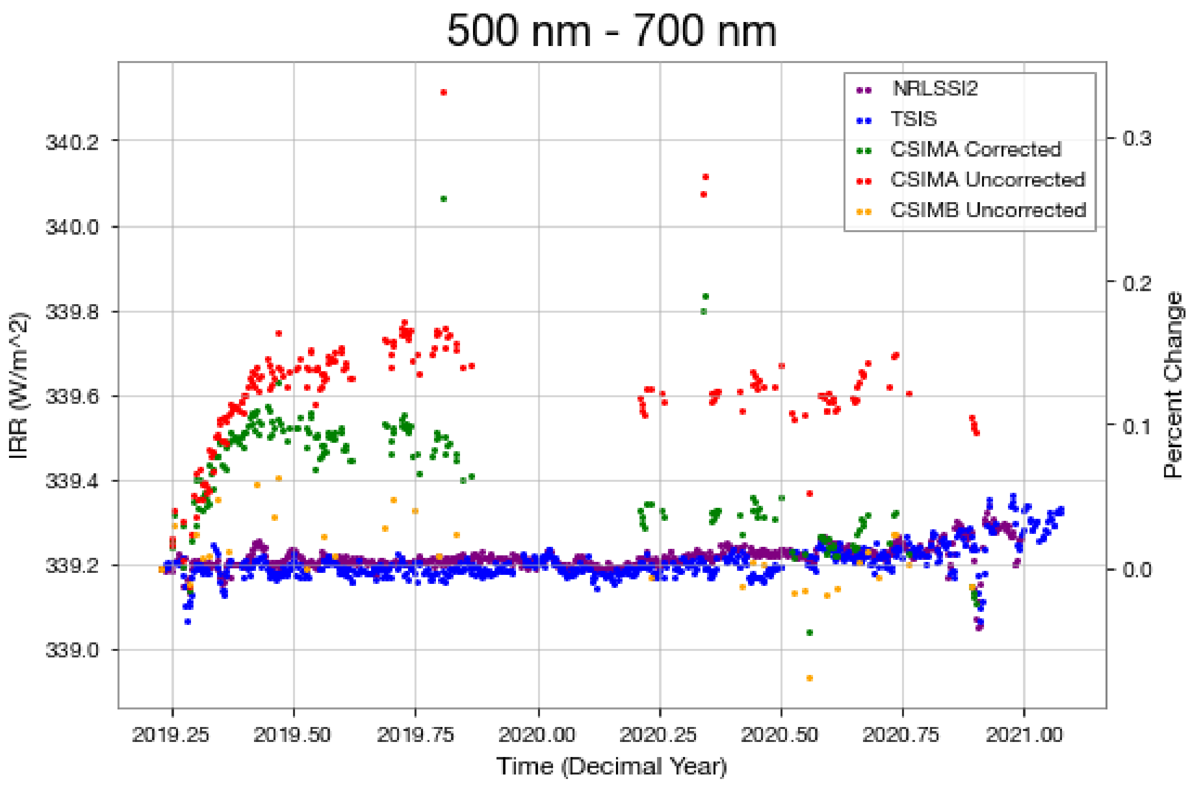


Figure 4.5: Comparison of overall kappa corrected CSIM channel 1 (green) to TSIS (blue), CSIM channel 2 (yellow), uncorrected CSIM channel 1 (red) and NRLSSI2 (purple) for 310-500nm. Note percentage scale is on the right axis while absolute values are on the left axis. Here the recovery region is not well corrected, but data in the post recovery region fits in neatly between model and TSIS data. The second plot contains the same data as the first, but omits the uncorrected spectra to give a sense of scale of the efficacy of the corrections.



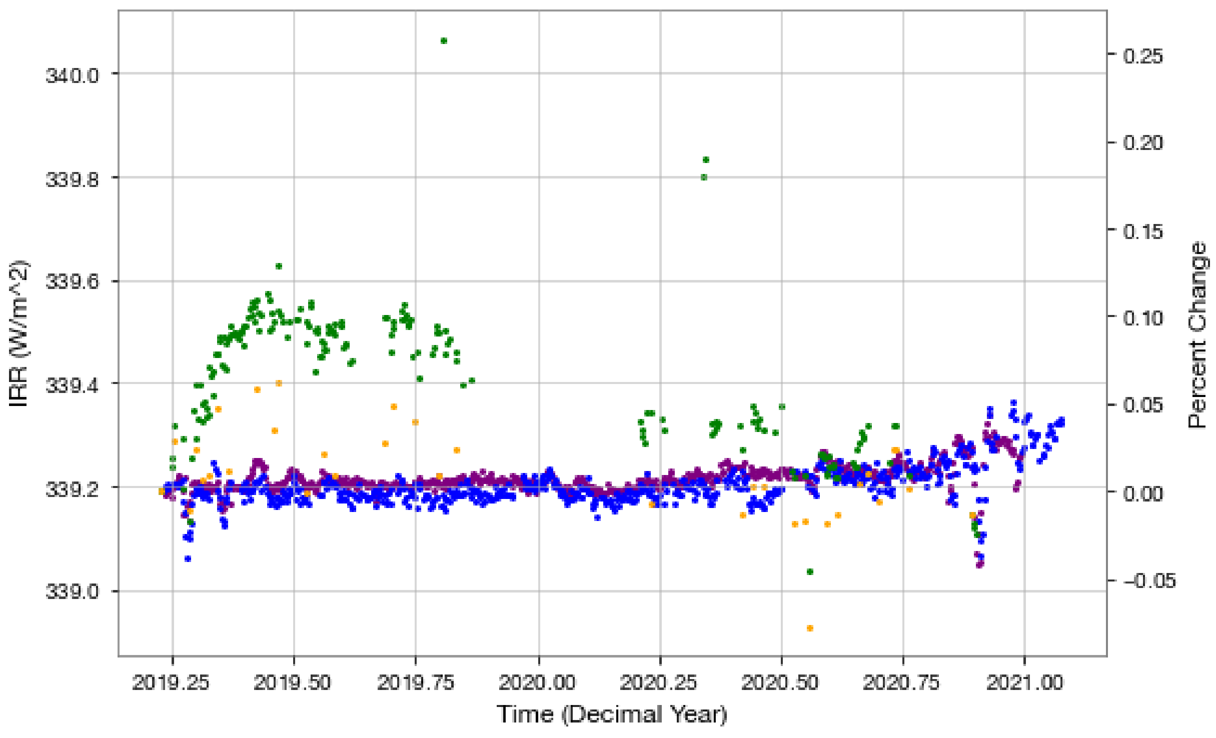
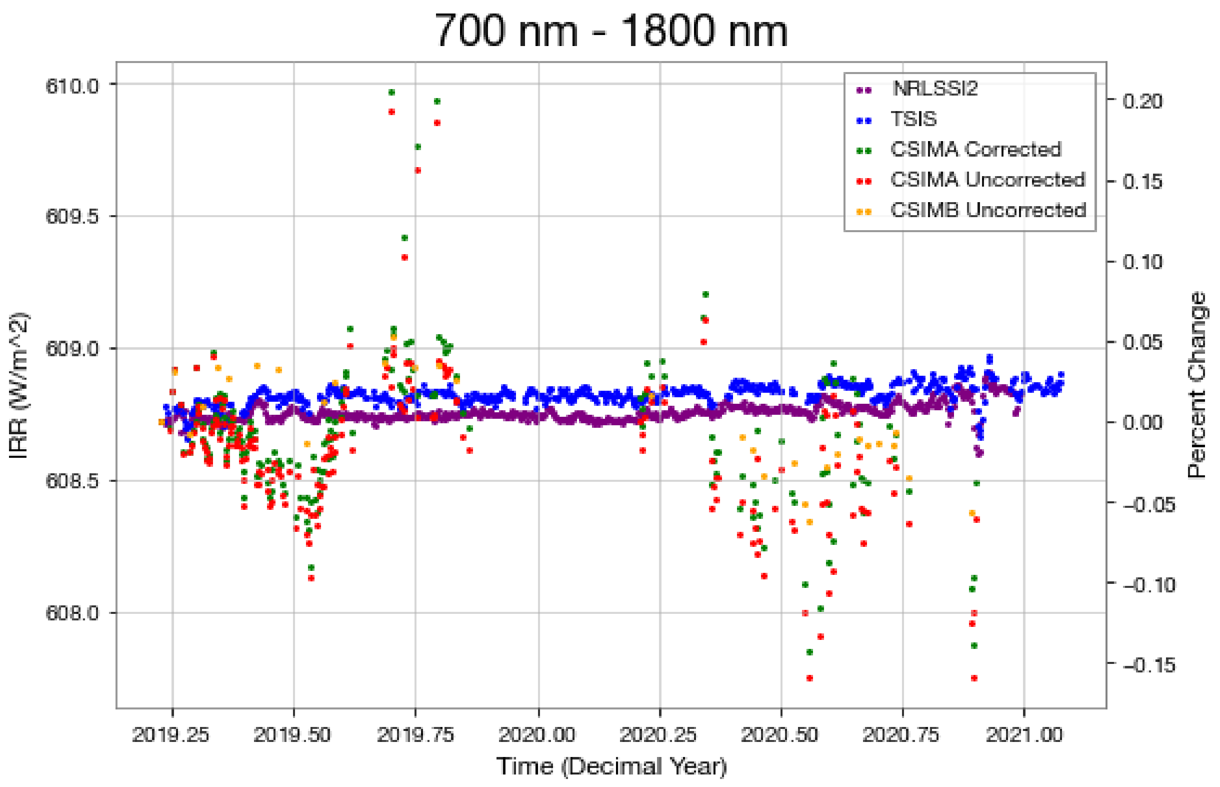


Figure 4.6: Comparison of overall kappa corrected CSIM channel 1 (green) to TSIS (blue), CSIM channel 2 (yellow), uncorrected CSIM channel 1 (red) and NRLSSI2 (purple) for 500-700nm. Note percentage scale is on the right axis while absolute values are on the left axis. Here agreement is poor in both the recovery region as well as the post-recovery region.



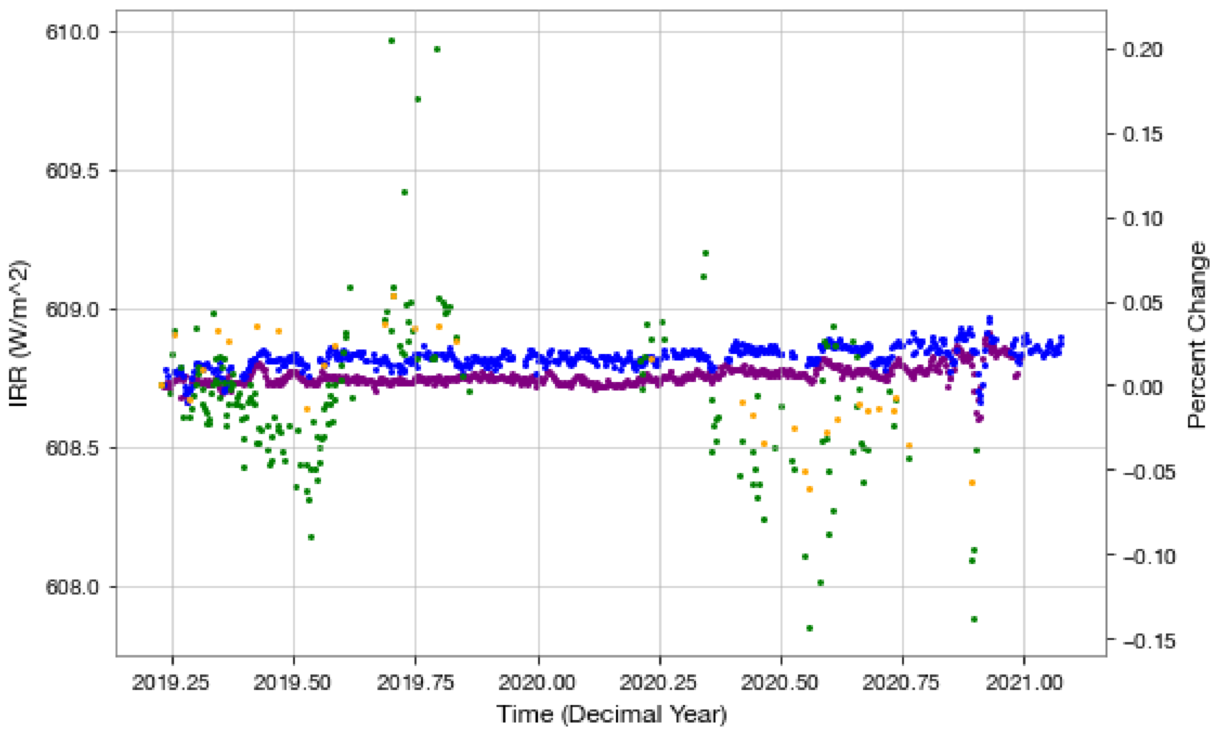
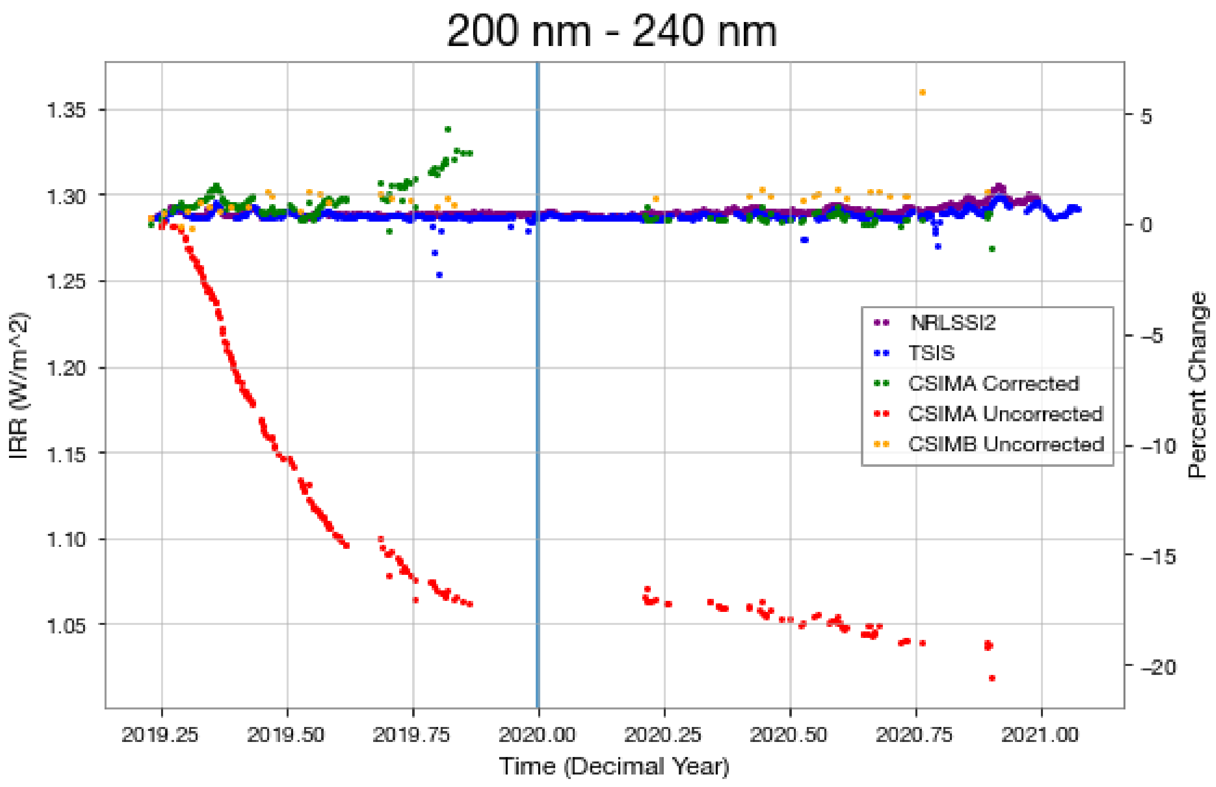


Figure 4.7: Comparison of overall kappa corrected CSIM channel 1 (green) to TSIS (blue), CSIM channel 2 (yellow), uncorrected CSIM channel 1 (red) and NRLSSI2 (purple) for 700-1800nm. Note percentage scale is on the right axis while absolute values are on the left axis. Here data is noisy, and appears to have a periodic trend in the beginning of the mission. The cause for this is as of yet unknown. The second plot contains the same data as the first, but omits the uncorrected spectra to give a sense of scale of the efficacy of the corrections.

4.2.2 Two Stage Kappa

The SSI for the same five integrated wavelength bands is shown in figures 4.8, 4.9, 4.10, 4.11, and 4.12 with corrections using a two stage kappa method. Overall there is significant improvement with this method in the post-recovery region, but the recovery region is poorly corrected. This is not very surprising; degradation coupled with recovery is extremely difficult to separate and quantify, but after this recovery region has passed, a standard correction method works well.

There is poor agreement in the recovery region time period in the 200-240nm band, but following this period, corrections perform extremely well; enough to match TSIS within 0.05%. This gives further credence to the theory that the recovery region causes a majority of the problems with an overall kappa correction. Corrections here also match the solar variability of TSIS and model data extremely well, which is exciting given that this is a region where CSIM was expected to perform poorly. A similar result was observed in 240-300nm regions. Here again the agreement in the recovery region is poor (within 4%), but in the post-recovery region agreement once again is incredible, within 0.05% of TSIS data. It captures the solar variability of the model and TSIS measurements, thus boosting its credibility. Agreement in the primary visible region (300-500nm) is excellent. Again the variability matches that of the TSIS data as well as model data, and differences are between 0.05% in the pre-September 2019 period and within 0.025% in the post September 2019 period. In the secondary visible region (500-700nm) agreement is also excellent. The recovery region is partially corrected, and data following this appears to match TSIS and model data while being within 0.05%. In the near IR region unfortunately the noise of CSIM dominates. Data remains within 0.1% of TSIS measurements. This region also appears to be following a periodic trend, meriting further investigation of this region and the initial assumptions about the degradation function.



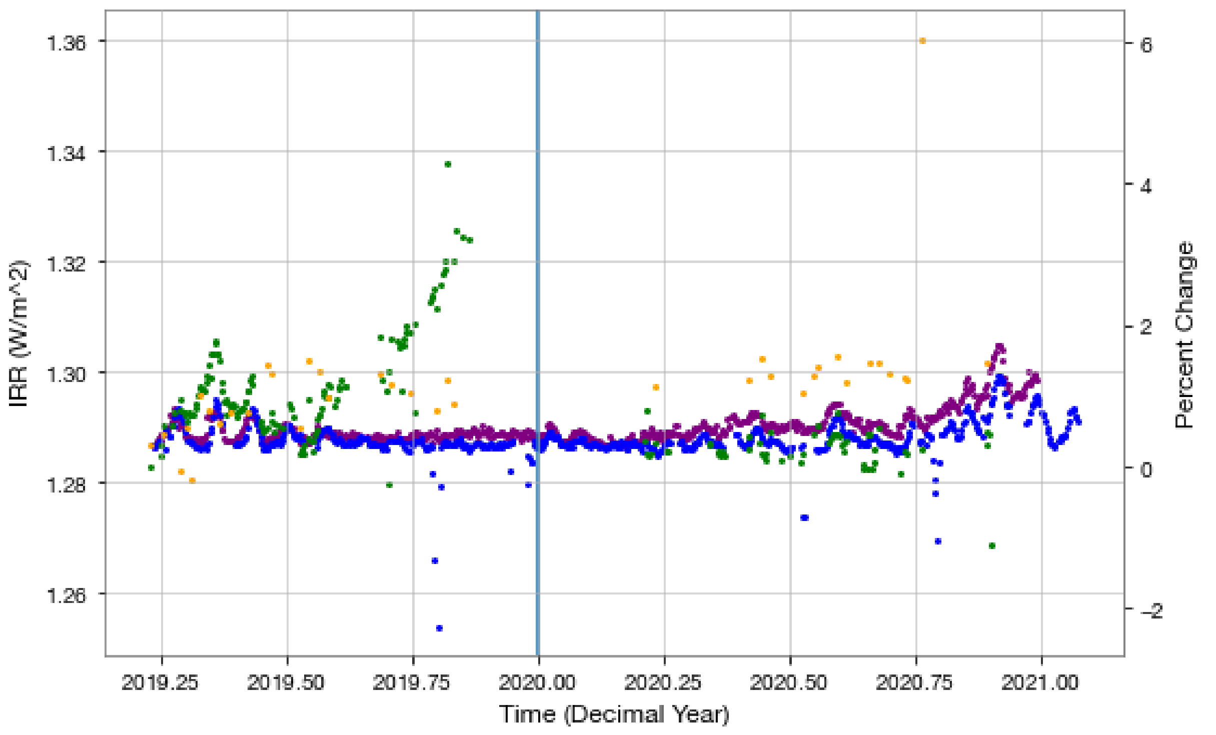
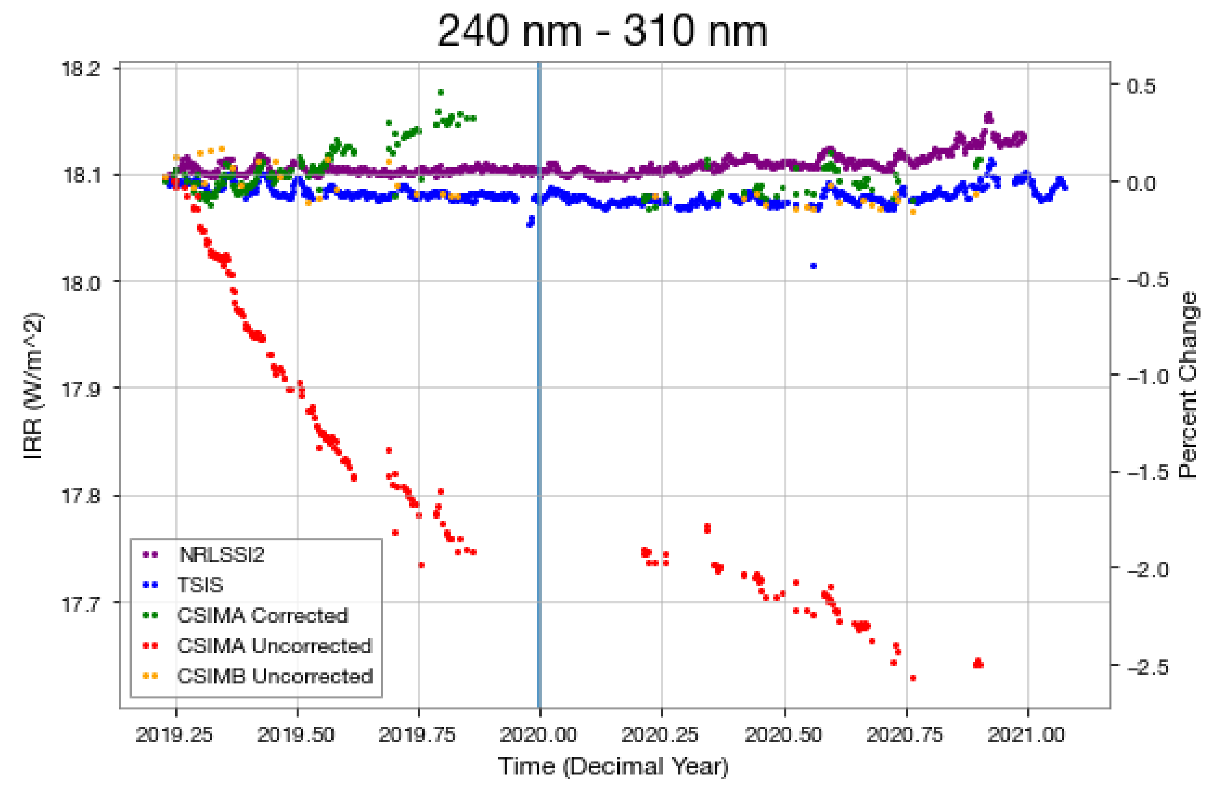


Figure 4.8: Comparison of two-stage corrected CSIM channel 1 (green) to TSIS (blue), CSIM channel 2 (yellow), uncorrected CSIM channel 1 (red) and NRLSSI2 (purple) for 200-240nm. The blue line here indicates the divide in the recovery and post-recovery regions. Note percentage scale is on the right axis while absolute values are on the left axis. Here initial trends are far over-corrected, but post-recovery data matches TSIS and model data extremely well. The second plot contains the same data as the first, but omits the uncorrected spectra to give a sense of scale of the efficacy of the corrections.



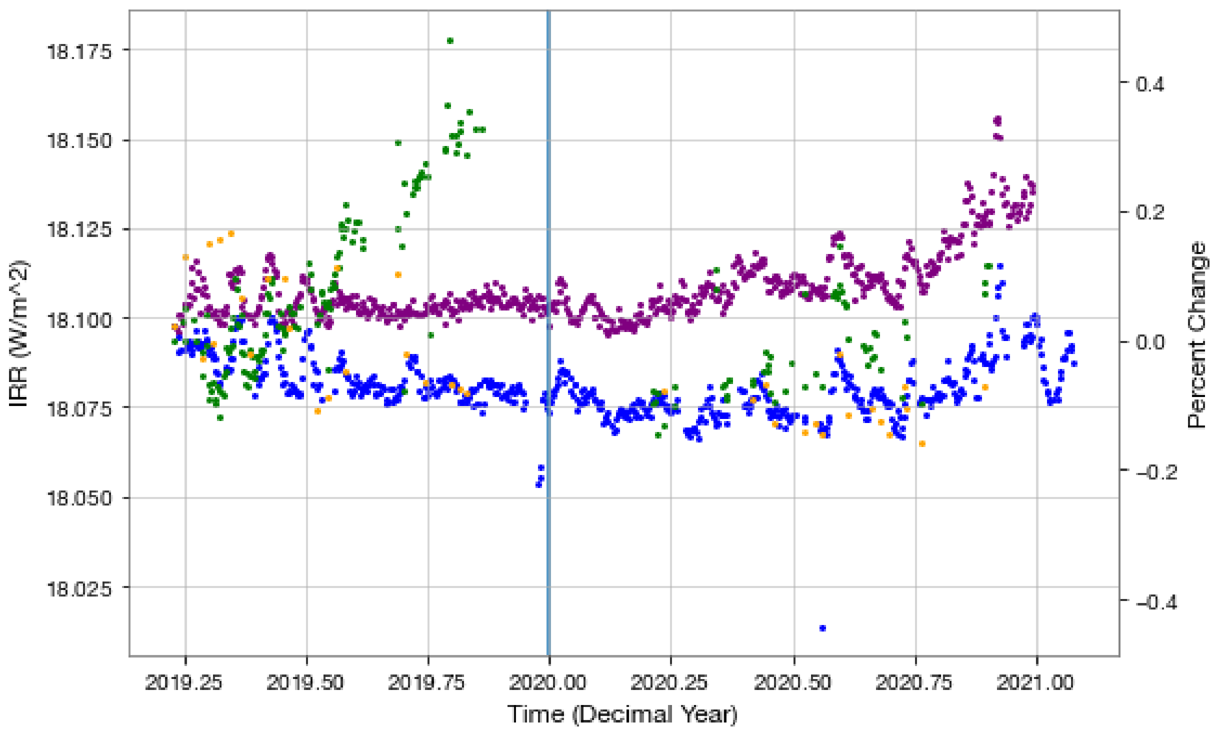
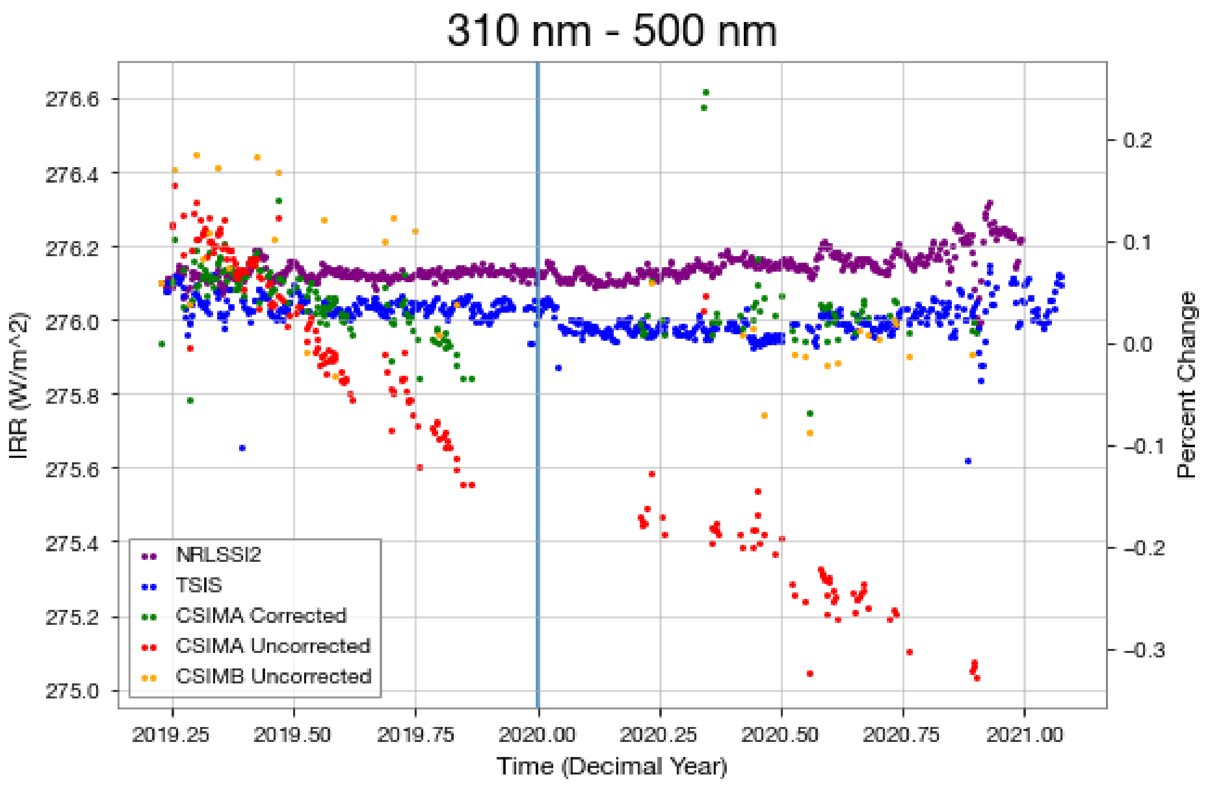


Figure 4.9: Comparison of two-stage corrected CSIM channel 1 (green) to TSIS (blue), CSIM channel 2 (yellow), uncorrected CSIM channel 1 (red) and NRLSSI2 (purple) for 240-310nm. The blue line here indicates the divide in the recovery and post-recovery regions. Note percentage scale is on the right axis while absolute values are on the left axis. Here initial trends are over-corrected, but post-recovery data matches TSIS and model data extremely well. The second plot contains the same data as the first, but omits the uncorrected spectra to give a sense of scale of the efficacy of the corrections.



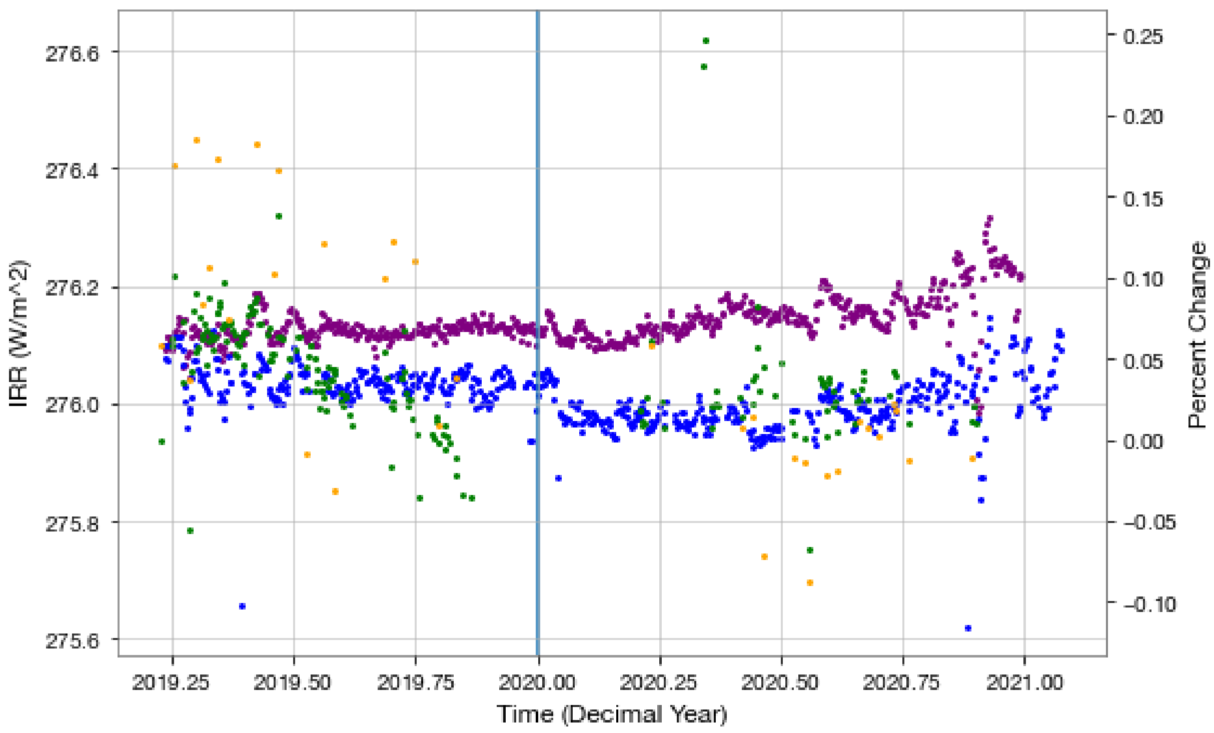


Figure 4.10: Comparison of two-stage corrected CSIM channel 1 (green) to TSIS (blue), CSIM channel 2 (yellow), uncorrected CSIM channel 1 (red) and NRLSSI2 (purple) for 310-500nm. The blue line here indicates the divide in the recovery and post-recovery regions. Note percentage scale is on the right axis while absolute values are on the left axis. Here initial trends are compensated for slightly. However, this appears to trend negative, implying that this region was slightly under-corrected. Post-recovery data matches TSIS and model data extremely well. The second plot contains the same data as the first, but omits the uncorrected spectra to give a sense of scale of the efficacy of the corrections.

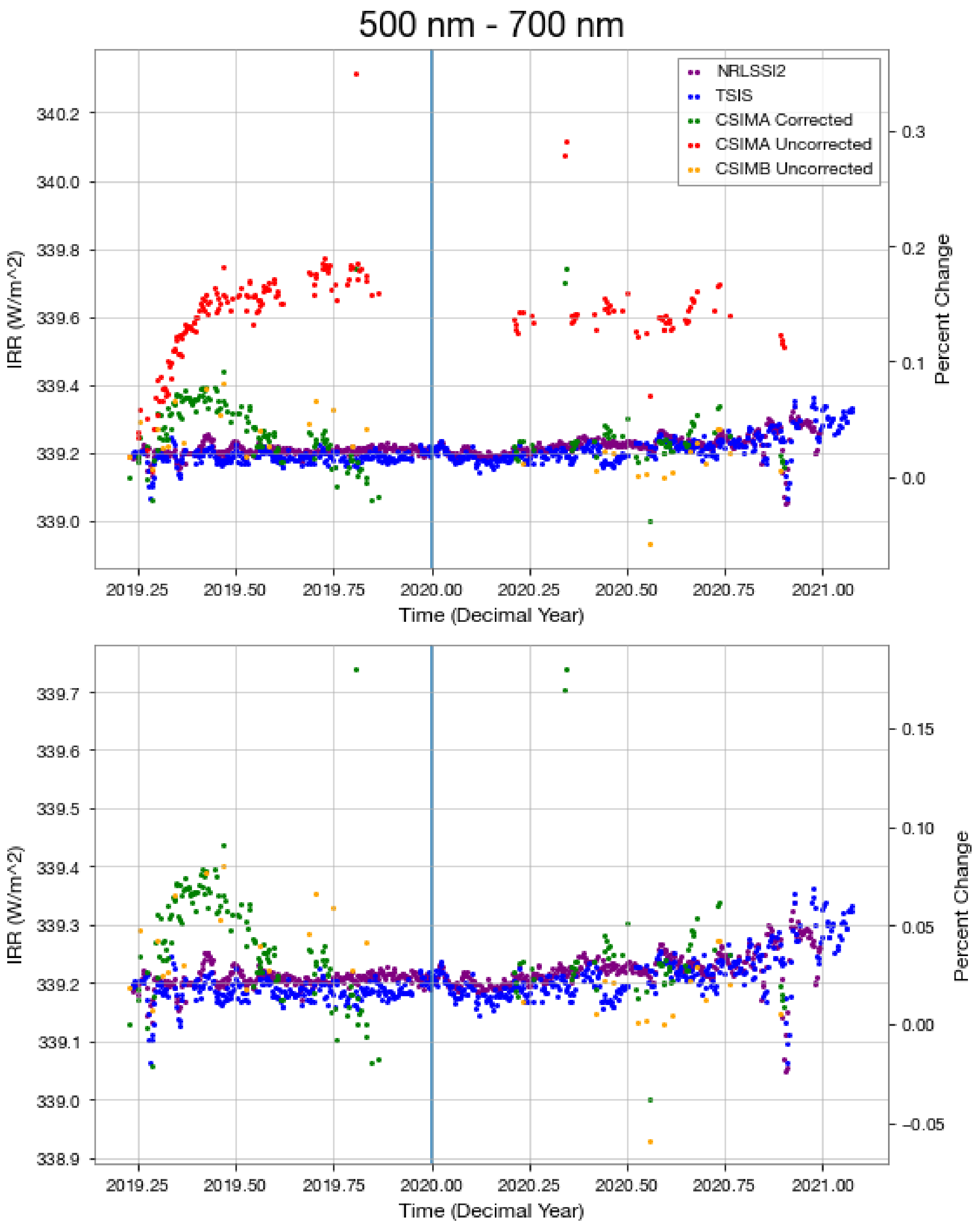
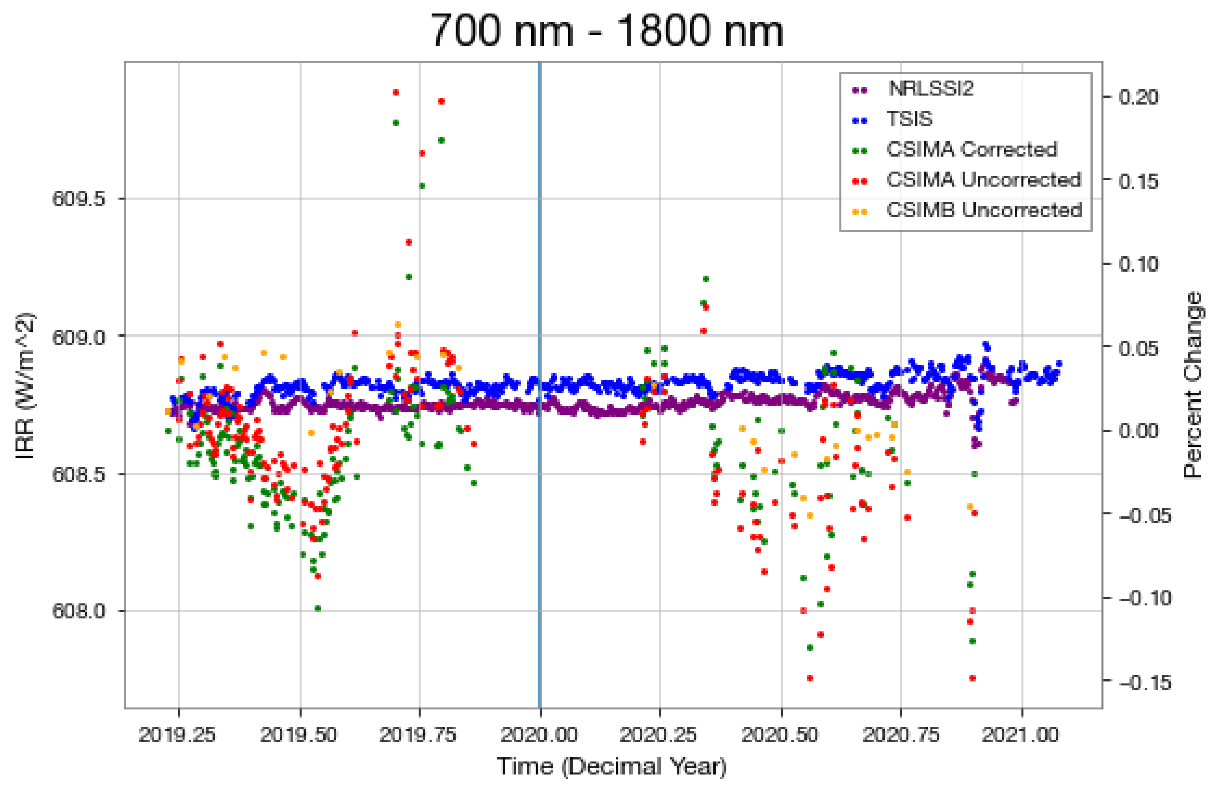


Figure 4.11: Comparison of two-stage corrected CSIM channel 1 (green) to TSIS (blue), CSIM channel 2 (yellow), uncorrected CSIM channel 1 (red) and NRLSSI2 (purple) for 500-700nm. The blue line here indicates the divide in the recovery and post-recovery regions. Note percentage scale is on the right axis while absolute values are on the left axis.Here the recovery region while somewhat brought down does not match TSIS or model trending. The post-recovery region matches TSIS and model data almost exactly. The second plot contains the same data as the first, but omits the uncorrected spectra to give a sense of scale of the efficacy of the corrections.



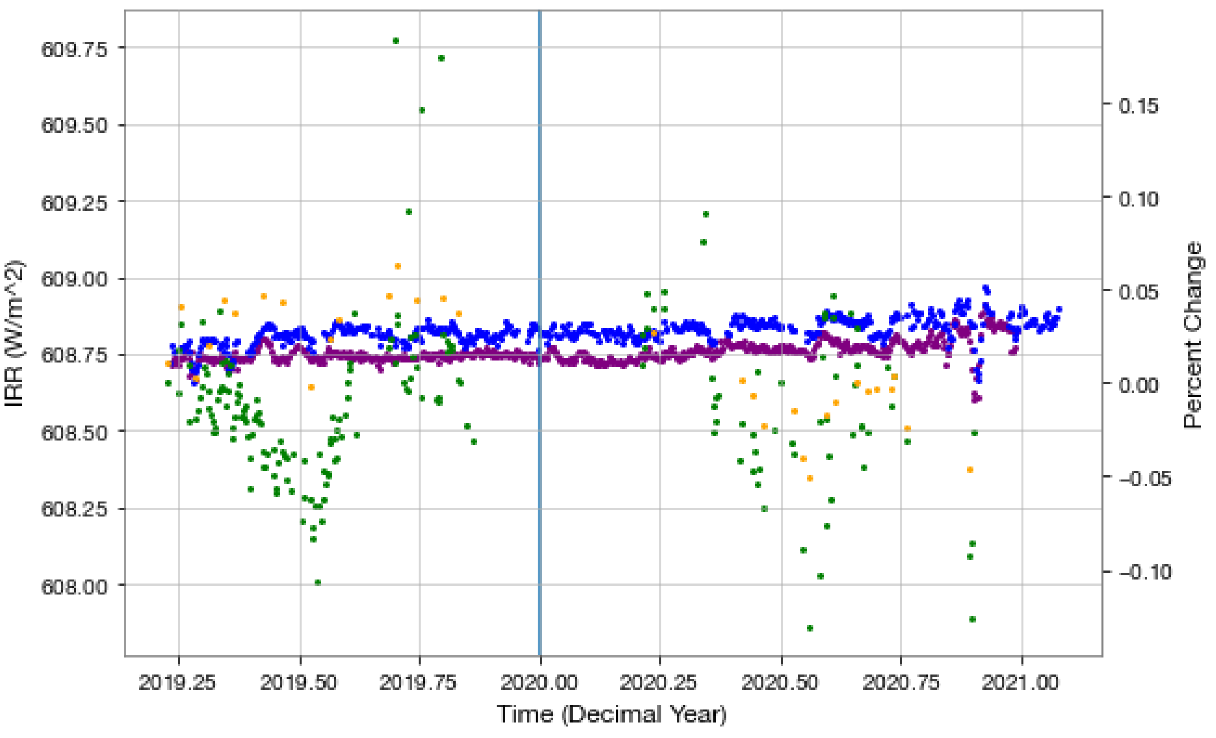
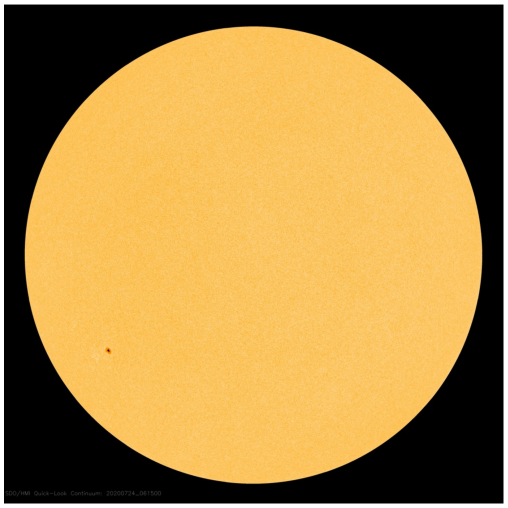


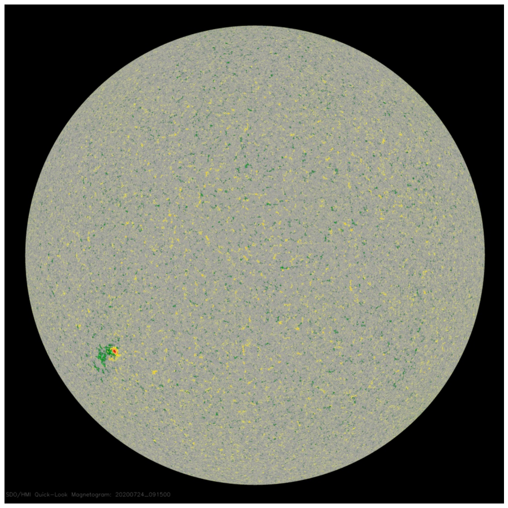
Figure 4.12: Comparison of two-stage kappa corrected CSIM channel 1 (green) to TSIS (blue), CSIM channel 2 (yellow), uncorrected CSIM channel 1 (red) and NRLSSI2 (purple) for 700-1800nm. Note percentage scale is on the right axis while absolute values are on the left axis.Here data is noisy, and appears to have a periodic trend in the beginning of the mission. The cause for this is as of yet unknown. Corrections here don't appear to have much effect, but this is likely due to the fact that degradation in these regions is close to zero. The second plot contains the same data as the first, but omits the uncorrected spectra to give a sense of scale of the efficacy of the corrections.

4.3 Short-term Solar Spectral Variability Comparison

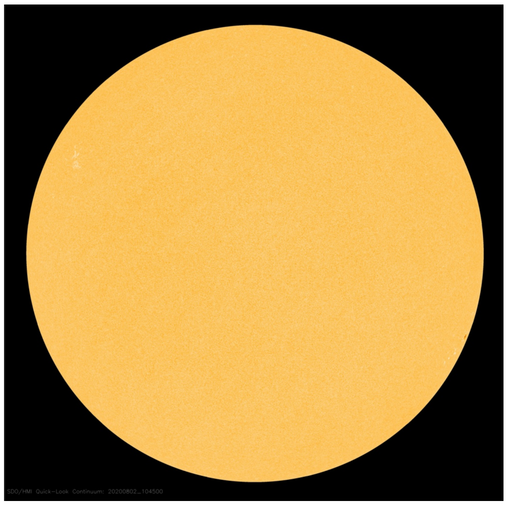
A final test to determine the efficacy of the CSIM instrument is to compare measurements of short-term solar variability. While ideally we would like to take a view on solar variability over many years, without accurate corrections it is only reasonable to look at variability over the time-span of a maximum of one month (assuming that this is in a period in which the instrument's degradation rate has slowed significantly). By comparing solar variability measurements to TSIS, one can make another case for CSIM: It is able to track this solar variability very close to TSIS, which is impressive considering the demonstration nature of the mission. In figure 4.13 a day with some solar activity (a visible sunspot on July 23rd 2020) and a nearby date with more diffuse, no visible sunspot solar activity (August 2 2020), are compared by taking a ratio of the spectra. Over this < 2 week duration the change in CSIM optical degradation magnitude is small, meaning uncorrected data can be used. This allows for a visualization of solar variability measurements dependent on wavelength. This is demonstrated in the figure 4.14. While there is significant noise in the 200-250 regions, The more frequent data occurs on the upper bound and aligns well with TSIS. From 300-400nm, CSIM captures solar variability to the same level as TSIS. After 500nm solar variability is extremely low, which is why these further regions are omitted here. In these regions CSIM and TSIS data is fairly flat (aside from some noise spikes around 500nm for CSIM). Both instruments have trouble capturing true solar variability after 1500nm (where models predict relative changes on the order of $<0.01\%$).



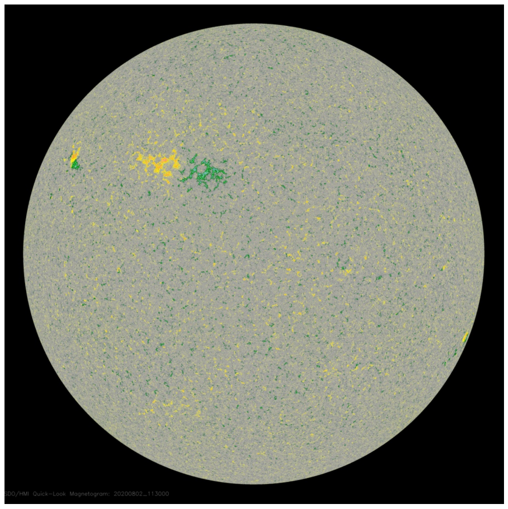
(a) Orange Flat Intensitygram of the sun on July 24th 2020



(b) Colorized Magnetogram of the sun on July 24th 2020



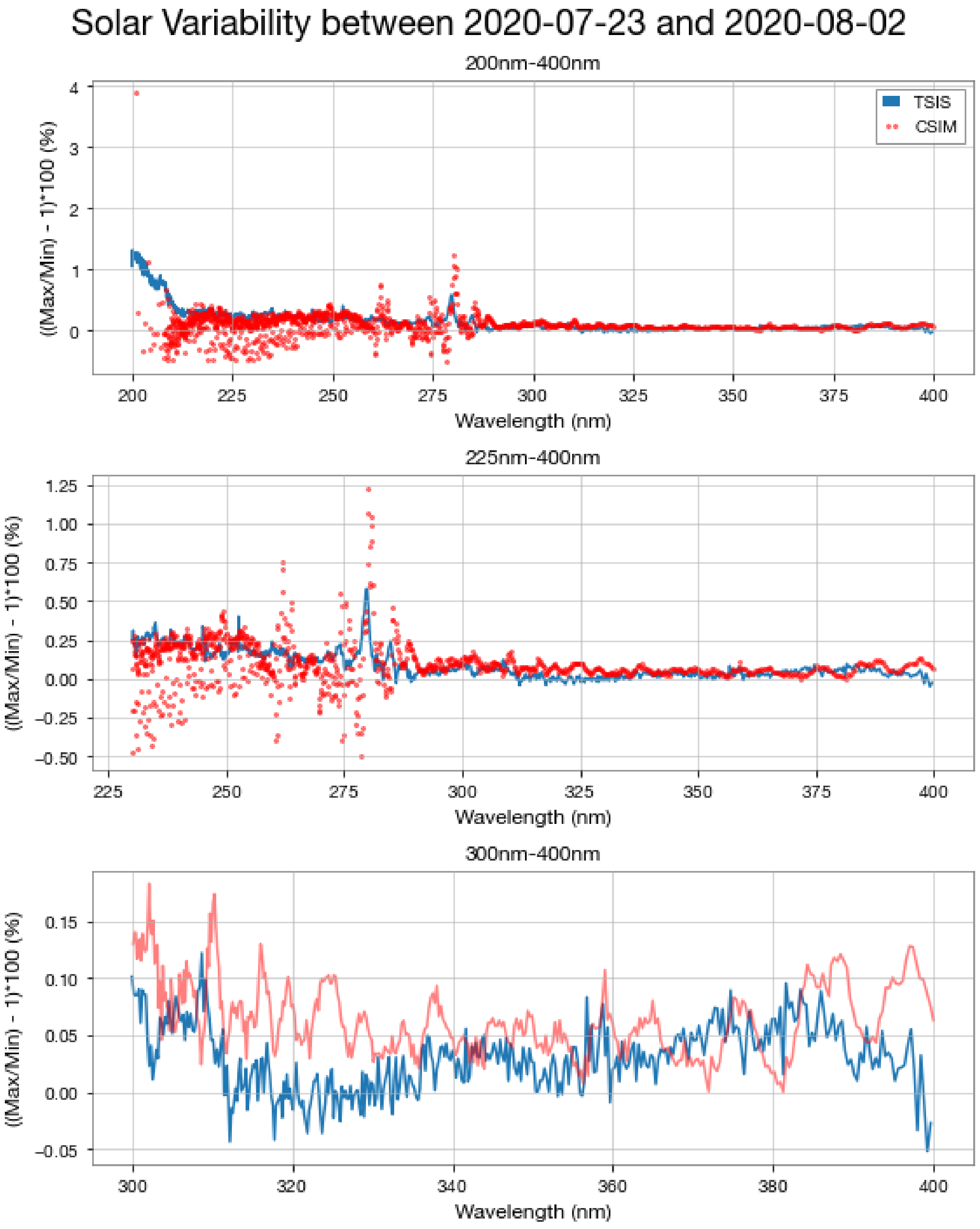
(c) Orange Flat Intensitygram of the sun on August 2 2020



(d) Colorized Magnetogram of the sun on August 2nd 2020

Figure 4.13: July 24th shows a day where sunspot activity dominates. August 2nd shows a day where faculae dominate. Images are from the NASA Solar Dynamics Observatory (SDO) satellite showing the Helioseismic and Magnetic Imager (HMI) visible (orange-flat) “intensitygram” (Left) and the corresponding HMI Colorized Magnetogram (Right) (SDO website: https://sdo.gsfc.nasa.gov/)

Figure 4.14: These plots demonstrate CSIM's capability when comparing solar variability. Here different wavelength regions are shown to give perspective on where CSIM is weak and where it performs well. Note that while noisy in the higher energy regions, upper bounds from 225nm to 275nm match TSIS measurements.



Chapter 5

Discussion

CSIM was launched almost exactly 1 year after TSIS, with the goal of quantifying CSIM capabilities by comparing it to TSIS, the primary science instrument for taking SSI data. However, this comparison was complicated by the fact that CSIM exhibited different degradation trends than have been previously seen. The primary culprit is of course the optical recovery effect. There are a few ideas for why this recovery is observed in the data. One possibility deals with the mirror that is used to reflect light onto the prism. This additional optical element likely propagates further errors, and pointing is thought to have contributed to this. For example, if CSIM points in a certain direction in the beginning of the mission, thus accumulating a degradation "layer" (or area) on the mirror at a specific location, and then CSIM has large pointing issues farther in the mission, this could result in parts of the mirror reflecting light that previously were untouched by solar radiation. This new spot then ”burns” similar to the original ”burn” spot, but because it is relatively fresh, transmissivity briefly increases, thus resulting in the recovery pattern shown. This naturally would be more apparent in regions that don't degrade as much, because the degradation in high energy regions is so extreme that it overcomes this second optical effect. Of course this is a speculation, because without fully understanding the sources of degradation (an open research question even on TSIS) it is difficult to fully account for this effect. Such is the challenge of space based solar viewing instrumentation designed to measure extremely small changes in solar radiation.

While an overall kappa method performed poorly, the two stage kappa method proved successful, ignoring the recovery period. Kappa wavelength dependency is similar to TSIS SIM and

SORCE SIM [24] [26]. Lacking the third channel present on TSIS, CSIM SSI data is not able to be corrected to the same level as TSIS, nor are the uncertainties as clear. Quantifying uncertainties on CSIM remains a non-trivial problem. If kappa comparison plots had more data points and could thus demonstrate higher levels of correlation, a standard deviation method applied to the degradation could potentially give error bars to corrected spectral data. While not accounting for time dependence this would at least give a flat estimate of potential uncertainties associated with corrections. However, attempting such a method with the current set of data results in error bars that are far too large to be useful. This is complicated by CSIM's many instrument problems, including its pointing stability, spacecraft resets, and complex optical degradation trends. TSIS uncertainties were quantifiable due to its third instrument channel [26], but without such a channel on CSIM, an analysis of uncertainties remains unclear. For example, one key assumption made in the analysis of CSIM that was proved false on TSIS was that the kappa function is the same regardless of channel. On TSIS, channel 1 and channel 2 do not degrade at the same rate (the fundamental assumption of the 2-channel, single kappa method). Considering these issues, CSIM in the post-recovery region showed excellent agreement with TSIS and model data. While these results itself cannot validate CSIM data, the lack of systematic difference between CSIM, TSIS SIM and the model is a good sign for both CSIM data as well as TSIS data.

Chapter 6

Conclusions

CSIM was launched as a flight demonstration mission during a period of solar minimum. To holistically evaluate this performance it's worth considering a few factors. First, this mission was not built to nearly the same cleanliness standards as TSIS, because of the cost barrier associated with trying to build ultraclean instrumentation. Second, many of the spacecraft issues were found to be with manufacturing, something easily corrected when designing a second iteration. Third, even when taking into account unexpected pointing effects, recovery effects, the SD card lockup, CSIM in the post-recovery region with an appropriate correction method takes data to close to TSIS standards, an impressive feat considering the significant reductions in cost, size, and development time of CSIM.

The degradation correction for this instrument is based on a simple measurement-based methodology used to make SORCE SIM and TSIS SIM corrections, but with modifications to account for the recovery exhibited in lower-energy regions. A piecewise correction demonstrates excellent agreement with TSIS SIM as well as the NRLSSI2 model (after January 2020), whereas the measurement period prior to January 2020 has been difficult to correct adequately. This prior region is also a time period of instability in CSIM's overall performance, so it is perhaps not so surprising that these factors have been difficult to quantify. On a positive note, these factors can potentially be mitigated in future instrument design. The goal of this instrument was to demonstrate new and innovative technologies and that long term SSI datasets could potentially be continued via CubeSats. The results of CSIM are more than encouraging in this regard.

Bibliography

[1] 2019. LASP: 2019 SORCE SIM Release Notes for Version 25, Level 3 data product.

[2] A. BenMoussa, S. Gissot, U. Schühle, G. Del Zanna, F. Auchère, S. Mekaoui, A.R. Jones, D. Walton, and C.J. Eyles. On-orbit degradation of solar instruments. Solar Phys, 288,389, 2013.

[3] S. Béland, J. Harder, and T. Woods. 10 years of degradation trends of the sorce sim instrument. SPIE, 8862:88620, 2013.

[4] S. Béland, J. Harder, and T. Woods. Eleven years of tracking the sorce sim instrument degradation caused by space radiation and solar exposure. Space Telescopes and Instrumentation, 2014. Optical, Infrared, and Millimeter Wave. 91434W.

[5] O. Coddington, J.L. Lean, P. Pilewskie, M. Snow, and D. Lindholm. A solar irradiance climate data record. Bull. Am. Meteorol. Soc, page 00265 1, 2015.

[6] S. Dewitte, D. Crommelynck, and A. Joukoff. Total solar irradiance observations from diarad/virgo. J. Geophys. Res. Space Phys, 109, 2004.

[7] I. Ermolli, K. Matthes, T. Dudok De Wit, N.A. Krivova, K. Tourpali, M. Weber, Y.C. Unruh, L. Gray, and U. Langematz. Recent variability of the solar spectral irradiance and its impact on climate modelling. Atmos. Chem. Phys, 13,3945:3945–2013, 2013.

[8] M. Fligge, S.K. Solanki, and Unruh. Y.c.: 2000 modelling irradiance variations from the surface distribution of the solar magnetic field. Astron. Astrophys, 353,380, 2000.

[9] L.J. Gray, J. Beer, M. Geller, J.D. Haigh, M. Lockwood, K. Matthes, U. Cubasch, D. Fleitmann, and G. Harrison. Solar influences on climate. Rev. Geophys, 48,RG4001, 2010.

[10] M. Haberreiter, M. Schöll, T. Wit, M. Kretzschmar, S. Misios, K. Tourpali, and W. Schmutz. A new observational solar irradiance composite. J. Geophys. Res. Space Phys, 122,5910, 2017.

[11] J.D. Haigh. The role of stratospheric ozone in modulating the solar radiative forcing of climate. Nature, 370,544, 1994.

[12] J.D. Haigh, A.R. Winning, R. Toumi, J.W. Harder, J. Harder, J.M. Fontenla, P. Pilewskie, E.C. Richard, and Woods. An influence of solar spectral variations on radiative forcing of climate. Nature, 467,696, 2010.

[13] J. Harder, G. Lawrence, J. Fontenla, G. Rottman, and T. Woods. The spectral irradiance monitor: Scientific requirements, instrument design, and operation modes. In G. Rottman, T. Woods, and V. George, editors, The Solar Radiation and Climate Experiment (SORCE. Springer, New York, 2005. NY, 141.

[14] G. Kopp. Magnitudes and timescales of total solar irradiance variability. J. Space Weather Space Clim, 6,A30, 2016.

[15] G. Kopp and G. Lawrence. The total irradiance monitor (tim): instrument design. In G. Rottman, T. Woods, and V. George, editors, The Solar Radiation and Climate Experiment (SORCE. Springer, New York, NY, 2005.

[16] N.A. Krivova, S.K. Solanki, M. Fligge, and Y.C. Unruh. Reconstruction of solar irradiance variations in cycle 23: Is solar surface magnetism the cause? astron. Astrophys, 399,L1, 2003.

[17] N.A. Krivova, S.K. Solanki, and L. Floyd. Reconstruction of solar uv irradiance in cycle 23. Astron. Astrophys, 452,631, 2006.

[18] J. Lean. Evolution of the sun's spectral irradiance since the maunder minimum. Geophys. Res. Lett, 27,2425, 2000.

[19] J. Lean, G. Rottman, J. Harder, and G. Kopp. Sorce contributions to new understanding of global change and solar variability. Solar Phys, 230,27:1527–2, 2005.

[20] J.L. Lean and DeLand. M.t.: 2012 how does the sun's spectrum vary? J. Clim, 25,2555, 2012.

[21] J.L. Lean and T.N. Woods. Solar spectral irradiance: Measurements and models. In C.J. Schrijver, Siscoe L, and George L, editors, Heliophysics: Evolving Solar Activity and the Climates of Space and Earth. 2010. Cambridge University Press, Cambridge, 269.

[22] S.V. Marchenko and M.T. Deland. Solar spectral irradiance changes during cycle 24. Astrophys. J, 789,117, 2014.

[23] S.V. Marchenko, M.T. DeLand, and Lean. J.l.: 2016 solar spectral irradiance variability in cycle 24: Observations and models. Space Weather Space Clim, 40,1, 2016.

[24] S. Mauceri, P. Pilewskie, E. Richard, O. Coddington, J. Harder, and T. Woods. Revision of the sun's spectral irradiance as measured by sorce sim. Solar Phys, 293,161. doi:11207–018–1379–1, 2018.

[25] S. Mauceri, P. Pilewskie, T. Woods, S. Beland, and Richard. E.c.: 2020 inter-comparing solar spectral irradiance from sorce sim. Rev, 2020.

[26] S. Mauceri, E. Richard, and P. Pilewskie. Degradation correction of tsis sim. Sol Phys, 295:152, 2020.

[27] W.E. McClintock, G.J. Rottman, and Woods. T.n.: 2005 solar-stellar irradiance comparison experiment ii (solstice ii): instrument concept and design. In G. Rottman, T. Woods, and V. George, editors, The Solar Radiation and Climate Experiment (SORCE. Springer, New York, NY, 2005.

[28] P. Pilewskie, E. Richard, O. Coddington, and J. Harder. Solar Spectral Irradiance and Climate: Current Understanding and Future Observations from the Total and Spectral Solar Irradiance Sensor. Int Radiat Symp Univ Auckland, New Zeal, 2016.

[29] E. Richard, D. Harber, O. Coddington, G. Drake, J. Rutkowski, M. Triplett, P. Pilewskie, and T. Woods. Si-traceable spectral irradiance radiometric characterization and absolute calibration of the tsis-1 spectral irradiance monitor (sim). remote sens, 2020.

[30] E. Richard, D. Harber, G. Drake, J. Rutkowsi, Z. Castelman, M. Smith, J. Sprunck, W. Zheng, P. Smith, M. Fisher, A. Sims, B. Cervelli, M. Fowle, M. Miller, M. Chambliss, T. Woods, P. Pilewskie, C. Yung, Stephens M., N. Tomlin, M. White, and J. Lehman. Compact spectral irradiance monitor flight demonstration mission". In Proc. SPIE 11131, CubeSats and SmallSats for Remote Sensing III, volume 1113105, 2019-08-30.

[31] T. Wit, G. Kopp, C. Fröhlich, and M. Schöll. Methodology to create a new total solar irradiance record: Making a composite out of multiple data records. Geophys. Res. Lett, 2017.

[32] T.N. Woods, F.G. Eparvier, J. Harder, and M. Snow. Decoupling solar variability and instrument trends using the multiple same-irradiance-level (musil) analysis technique. Solar Phys, 2018.

[33] K.L. Yeo, N.A. Krivova, S.K. Solanki, and K.H. Glassmeier. Reconstruction of total and spectral solar irradiance from 1974 to 2013 based on kpvt, soho/mdi, and sdo/hmi observations. Astron. Astrophys, 570,A85, 2014.

Appendix A

Python Scripts Used To Analyze CSIM Data

A.1 Declarations and Packages

```
#Author: Ansh Desai

#guidance from Michael Chambliss


# Import all Packages

import pandas as pandas

import numpy as np

import time

import datetime

from datetime import date

from datetime import timedelta

from astropy.time import Time

from gwpy.time import tconvert

from scipy import integrate

import matplotlib.pyplot as plt

#import gwpy

from collections import Counter

from scipy import interpolate
from scipy import stats

from numpy import array

from sklearn.linear_model import LinearRegression

import matplotlib.dates as mdates

from matplotlib.dates import DateFormatter

import math
```

A.2 load_wavelength_grid

```
def load_wavelength_grid(scan_times,full_scan):

    """

    Creates a standard wavelength grid by picking the longest wavelength range

    by going through every scan


    Parameters

    ----------

    scan_times : array

        This is the array of scan times to index by

    spacing: int

        This indicates the cutoff spacing for scans

    full scan: data frame

        Contains the original scan data



    Returns

    -------

    standard wavelength grid in Series form
    """
    max_range = 0
    ids = full_scan['OBS_ID'].values
    id = ids[0]
    for i in range(len(scan_times) - 1):
        indices = np.where(full_scan['SCAN_TIME'].values == scan_times[i])
        scan = full_scan.iloc[indices]

        scan = scan.drop_duplicates(subset=['WAVELENGTH'])

        if len(scan['WAVELENGTH'].values) > max_range:
            max_range = len(scan['WAVELENGTH'].values)
            std_wls = scan['WAVELENGTH']
    return std_wls
```

A.3 interpolate_scans

```
def interpolate_scans(std_wls,wl,irr,scan_time,id):
    """
    Interpolates irradiance of scan to standard wavelength grid

    Parameters
    ----------
    std_wls : series
        This is the wavelength grid to format to
    wl : Series
        Wavelengths of the scan indexed by times
    irr: Series
        The solar irradiance that needs to be interpolated
    obs_id: int
        The observation id associated with the scan
    n: int
        this is the number of the scan

    Returns
    -------
    DataFrame with Interpolated Scan
        Each DataFrame contains irradiance values indexed by scan time [in gps]
           and
        wavelength. Columns with scan number, exposure time and id are also added.
    """

    scan_std_wls = std_wls[std_wls.between(wl.min(), wl.max())]
    interpolator = interpolate.interp1d(wl.values, irr, kind='cubic')
    scan_interp_irr = interpolator(scan_std_wls.values)
    time_interpolator = interpolate.interp1d(wl.values, wl.index)
    int_times = time_interpolator(scan_std_wls.values)
    scantime = np.repeat(scan_time,len(int_times))
    obs_id = np.repeat(id,len(int_times))

    if id == 1:
        max_wl = 286
        min_wl = 200
```

```
if id == 2:
    min_wl = 286
    max_wl = 387
if id == 3:
    min_wl = 387
    max_wl = 900
if id == 4:
    min_wl = 900
    max_wl = 1500
if id == 5:
    min_wl = 1500
    max_wl = 2300



scan_std_wls = scan_std_wls.values




wavelength_filter = np.where((scan_std_wls >= min_wl) & (scan_std_wls <=
    max_wl))

int_times = int_times[wavelength_filter]
scan_interp_irr = scan_interp_irr[wavelength_filter]
scan_std_wls = scan_std_wls[wavelength_filter]
```

```
    scantime = scantime[wavelength_filter]
    obs_id = obs_id[wavelength_filter]




    d = {'Times': int_times, 'SIRR': scan_interp_irr, 'Wavelengths': scan_std_wls,
         'scan': scantime,'obs_id': obs_id}
    data = pandas.DataFrame(data=d)
    return data
```

A.4 time_convert

```
def time_convert(times,in_format,out_format):
    """
    Uses astropy's Time object to convert times
    possibilities for input and output format are the same as astropy
    https://docs.astropy.org/en/stable/time/

    Parameters
    ----------
    times : array,list
        This is the array of times to convert
    in_format: string
        The format of the input times
    out_format: string
        The requested format for output

    Returns
    -------
    An array that contains all the times converted to the requested format

    """
    epoch = 630719975
    if in_format == 'gps':
        times = times + epoch
    t = Time(times,format=in_format)
    t.format = out_format
    converted_times = t.value
    return converted_times
```

A.5 plot_scan

```

def plot_scan(scan,title):
    """
    Plots a scan with aa title

    Parameters
    ----------
    scan : data frame
        Contains a data frame with a scan (must have wavelength and sirr values)
    title: string
        This sets the title for the plot

    Returns
    -------
    Outputs a plot to the screen and as a file. Returns nothing.
    """

    %matplotlib inline
    wavelengths = scan['Wavelengths'].values
    sirr = scan['SIRR'].values
    fig,ax = plt.subplots()
    ax.plot(wavelengths,sirr)
    ax.set_xlabel('Wavelength')
    ax.set_ylabel("Solar␣Irradiance␣per␣Wavelength␣(W/m^2␣/␣nm)")
    ax.set_title(title)
    fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/'
    fname += str(title) + '.png'
    plt.show()
    fig.savefig(fname)
```

A.6 find_nearest

```
def find_nearest(array, value):
    """
    A simple function to find the nearest value in an array to a given value
    Parameters

    ----------

    array : array

        This is the array of values to search through

    value: int

        This is the value to search the array for





    Returns

    -------

    The element in an array that is closest to the given value

    """

    array = np.asarray(array)

    idx = (np.abs(array - value)).argmin()

    return array[idx]
```

A.7 plot_time_series

```
def plot_time_series(wavelength,data):

    """

    A function that plots the time series of a wavelength given the dataframe



    Parameters

    ----------

    wavelength : int

        This is the requested wavelength for a time series
```

```
data: dataframe

    This is the dataframae containing the given scan information




Returns

-------

Outputs a plot to the screen, returns nothing

"""

%matplotlib inline



wavelengths = data['Wavelengths'].values

closest_wl = find_nearest(wavelengths,wavelength)

indices = np.where(data['Wavelengths'].values == closest_wl)

filtered_data = data.iloc[indices]

times = filtered_data['Day'].values

times = time_convert(times,'jd','decimalyear')

sirr = filtered_data['SIRR'].values


filter_bad = np.where(sirr > 0.01)

times = times[filter_bad]

sirr = sirr[filter_bad]



plt.plot(times,sirr,'bo')

plt.xlabel('Time␣(Decimal␣Year)')
```

```
    plt.ylabel('SIRR␣(W/nm*m^2)')

    title = 'Time␣Series␣for␣Wavlength:␣' + str(closest_wl)

    plt.title(title)

    plt.show()

    return
```

A.8 get_scan

```
def get_scan(scan,day):

    """

    A function to retrieve a scan given a day



    Parameters

    ----------

    scan : data frame

        This contains the full scan

    day: iso string

        This is a string in ISO format for which date the scan is requested.






    Returns

    -------

    returns the longest scan on the requested day

    """



    day = time_convert(day,'iso','jd')
    day = int(day + 0.5)

    day_indices = np.where(scan['Day'].values == day)

    day_indices = day_indices[0]

    if len(day_indices) > 1:

        day_scan = scan.iloc[day_indices]

        day_scan = day_scan.sort_values(by='Wavelengths')

        day_scan = day_scan.drop_duplicates(subset=['Wavelengths'])


    else:

        day_scan = [-1]

    return day_scan
```

A.9 get_longest_scan

```
def get_longest_scan(scans):
    """

    A function to retrieve a scan given a day


    Parameters

    ----------

    scans : data frame

        This contains a scan range to search
    Returns
    -------
    rreturns the scan number of the longest scan in a set of scans
    """
    scan_times = scans['scan'].values


    x = Counter(scan_times)
    longest_scan = x.most_common(1)[0][0]
    return longest_scan
```

A.10 load_tsis_data

```
def load_tsis_data():
    """
    A function to load TSIS SIM data into a dataframe


    Parameters
    ----------
    None









    Returns
    -------
    Data frame containing the full set of scans
    """
    raw_scans = pandas.read_csv('tsis_ssi_24hr.csv')

    wavelengths = raw_scans['wavelength␣(nm)'].values

    sirr = raw_scans['irradiance␣(W/m^2/nm)'].values

    days = raw_scans['time␣(Julian␣Date)'].values

    days = days.astype(np.int64)

    d = {'Wavelengths': wavelengths, 'SIRR': sirr,'Day':days}

    df = pandas.DataFrame(data=d)

    return df
```

A.11 load_tsis_tsi

```
def load_tsis_tsi():

    """

    A function to load TSIS TIM data into a dataframe



    Parameters

    ----------

    None









    Returns

    -------

    Data frame containing the full set of scans

    """

    raw_scans = pandas.read_csv('tsis_tsi_24hr.csv')
    sirr = raw_scans['tsi_1au␣(W/m^2)'].values

    days = raw_scans['time␣(Julian␣Date)'].values

    days = days.astype(np.int64)

    d = {'TSI': sirr,'Day':days}

    df = pandas.DataFrame(data=d)

    return df
```

A.12 load_model_data

```
def load_model_data():
    """
    A function to load NRLSSI2 data into a dataframe


    Parameters
    ----------
    None





    Returns
    -------
    Data frame containing the full set of model data
    """
    raw_scans = pandas.read_csv('nrl2_ssi_P1D.csv')
    wavelengths = raw_scans['wavelength␣(nm)'].values
    sirr = raw_scans['irradiance␣(W/m^2/nm)'].values
    days = raw_scans['time␣(days␣since␣1610-01-01)'].values
    days = days.astype(np.int64)
    year = 1610
    month = 1
    day = 1
    epoch = (1461 * (year + 4800 + (month - 14)//12))//4 +(367 * (month - 2 - 12 *
        ((month - 14)//12)))//12 - (3 *((year + 4900 + (month - 14)//12)//100))
       //4 + day - 32075
    epoch = epoch - 3
    days = days + epoch
    d = {'Wavelengths': wavelengths, 'SIRR': sirr,'Day':days}
    df = pandas.DataFrame(data=d)
    return df
```

A.13 remove_bad_scans

```
def remove_bad_scans(full_scan,id,channel):
    """
    Removes incomplete scans from a set of scans in order to keep data clean

    Parameters
    ----------
    full_scan : dataframe
        Contains the full scan of data to filter
    id : int or string
        IDs 1,2,3,4,5
    channel : int
        Channel 1 or 2


    Returns
    -------
    DataFrame with All Filtered Scans
        The Final Data Frame contains all scans that fulfill length and
            completeness requirements.
    """
    small_dfs = []
    scan_times = np.unique(full_scan['scan'].values)



    min_len = 1
    if channel == 1:
        if id == 1:
            min_len = 1300

        if id == 2:
            min_len = 438
        if id == 3:
            min_len = 414
        if id == 4:
            min_len = 145
        if id == 5:
            min_len = 212
```

```
elif channel == 2:
    if id == 1:
        min_len = 1300

    if id == 2:
        min_len = 438
    if id == 3:
        min_len = 414
    if id == 4:
        min_len = 145
    if id == 5:
        min_len = 2

for i in scan_times:
    scan_indices = np.where(full_scan['scan'].values == i)
    scan = full_scan.iloc[scan_indices]
    wl = scan['Wavelengths'].values

    if len(wl) > min_len:
        small_dfs.append(scan)
filtered_scans = pandas.concat(small_dfs,ignore_index=True)
return filtered_scans
```

A.14 load_scan

```
def load_scan(channel,id):
    """
    Reads in data from CSIM .csv files, filters it by channel and ID. Then it
        reads through and
    standardizes all data to a single wavelength grid, and filters out incomplete
        scans. It returns
    a data frame with scan time, wavelength, solar irradiance, exposure time, and
        scan day

    Parameters
    ----------
    channel : int
        Channel 1 or 2
    id : int or string
        IDs 1,2,3,4,5

    Returns
    -------
    DataFrame with All Filtered Scans by ID
        The Final Data Frame contains irradiance values indexed by scan time [in
            gps] and
        wavelength. Columns with scan number, observation ID, julian day, exposure
             time are also added.
    """
    pd_scans = pandas.read_csv('pd_scans.csv')
    exp = pandas.read_csv('csim_exposure.csv')
```

```
pd_scans = pd_scans.sort_values(by=['TIME'],ignore_index=True)
#acceptable_ids = [1,2,3,4,5]
#pd_scans = pd_scans[pd_scans.OBS_ID.isin(acceptable_ids)]
pd_scans = pd_scans.loc[(pd_scans['SIRR'] >= 0) & (pd_scans['SIRR'] <= 2.5)]
#spacing was 14 and 50

id_scan = pd_scans.loc[pd_scans['OBS_ID'] == id]






#
full_scan = id_scan.loc[id_scan['CHANNEL'] == channel]
expCH1 = exp.loc[exp['CHANNELID'] == 1]
expCH2 = exp.loc[exp['CHANNELID'] == 2]

expCH1_time = expCH1['TIME']
expCH1_time_max = expCH1_time.max()
expCH1_time_min = expCH1_time.min()
expCH2_time = expCH2['TIME']
expCH2_time_max = expCH2_time.max()
expCH2_time_min = expCH2_time.min()
#default
max_time = expCH1_time_max
min_time = expCH1_time_min
if max_time > expCH2_time_max:
        max_time = expCH2_time_max

    if min_time < expCH2_time_min:
        min_time = expCH2_time_min



    full_scan = full_scan.loc[(full_scan['TIME'] <= max_time) & (full_scan['TIME']
         >= min_time)]
    if channel == 1:
        expCH1 = expCH1.loc[(expCH1['TIME'] <= max_time) & (expCH1['TIME'] >=
            min_time)]
        exptimesCH1 = expCH1.TIME
        exposureCH1 = expCH1.CUMULATIVE_EXPOSURE_SEC
        exp_interpol = interpolate.interp1d(exptimesCH1,exposureCH1)
    elif channel == 2:
        expCH2 = expCH2.loc[(expCH2['TIME'] <= max_time) & (expCH2['TIME'] >=
            min_time)]
        exptimesCH2 = expCH2.TIME
        exposureCH2 = expCH2.CUMULATIVE_EXPOSURE_SEC
        exp_interpol = interpolate.interp1d(exptimesCH2,exposureCH2)

    scan_times = np.unique(full_scan['SCAN_TIME'].values)
    print('number␣of␣initial␣scans',len(scan_times))

    #Load in wavlength Grid
    std_wls = load_wavelength_grid(scan_times,full_scan)
    small_dfs = []
```

```
for i in range(len(scan_times) - 1):
    indices = np.where(full_scan['SCAN_TIME'].values == scan_times[i])
    scan = full_scan.iloc[indices]
    if len(scan['WAVELENGTH'].values) > 100:#this is a good scan probably
        scan = scan.drop_duplicates(subset=['WAVELENGTH'])

        scan = scan.set_index('TIME')
        wl = scan.WAVELENGTH
        irr = scan.SIRR
        scan_time = scan['SCAN_TIME'].values[0]

        data = interpolate_scans(std_wls,wl,irr,scan_time,id)
        #print('max after interpolation',data.Wavelengths.max())
        small_dfs.append(data)




filtered_scans = pandas.concat(small_dfs,ignore_index=True)




filtered_scans = remove_bad_scans(filtered_scans,id,channel)
```

```
    print('channel:␣',channel,'id␣',id,'number␣of␣scans␣',len(np.unique(
        filtered_scans['scan'].values)) - 1)
    filtered_scan_times = filtered_scans['Times'].values
    days = time_convert(filtered_scan_times,'gps','jd')
    days = days.astype(int)
    exposure = exp_interpol(filtered_scan_times)
    #exposure = exposure / 3600
    filtered_scans['Day'] = days
    filtered_scans['Exposure'] = exposure



    return filtered_scans
```

A.15 remove_extra_scans

```
def remove_extra_scans(full_scan):
    """
    Read in the full set of scans and removes extra scans on a given observation
        day

    Parameters
    ----------
    full_scan : data frame
        Contains the data frame of all scanss


    Returns
    -------
    DataFrame with all extra scans removed


    """
    small_dfs = []
    uniq_days = np.unique(full_scan['Day'].values)
    for i in uniq_days:
        scan_indices = np.where(full_scan['Day'].values == i)
        scan = full_scan.iloc[scan_indices]
        scan_numbers = np.unique(scan['scan'].values)
        if len(scan_numbers) > 1: #multiple scans in one day
            scan_number = get_longest_scan(scan)
            scan = scan.iloc[np.where(scan['scan'].values == scan_number)]
        small_dfs.append(scan)
    filtered_scans = pandas.concat(small_dfs,ignore_index=True)
    return(filtered_scans)
```

A.16 load_all_scans

```
def load_all_scans(channel):
    """
    Creates a full set of scans covering all wavelength ranges given channel


    Parameters
    ----------
    channel : int
    Channel 1 or 2



Returns
-------
DataFrame with All Filtered Scans
    The Final Data Frame contains irradiance values indexed by scan time [in
        gps] and
    wavelength. Columns with scan number, observation ID, julian day, exposure
         time are also added.
"""


scan1 = load_scan(channel,1)
scan2 = load_scan(channel,2)
scan3 = load_scan(channel,3)
scan4 = load_scan(channel,4)
scan5 = load_scan(channel,5)

#need to fix problem of multiple scans per day


scan1 = remove_extra_scans(scan1)
scan2 = remove_extra_scans(scan2)
scan3 = remove_extra_scans(scan3)
scan4 = remove_extra_scans(scan4)
scan5 = remove_extra_scans(scan5)
```

```
days1 = scan1['Day'].values

days2 = scan2['Day'].values

days3 = scan3['Day'].values

days4 = scan4['Day'].values

days5 = scan5['Day'].values

intersection1 = np.intersect1d(days1,days2)

intersection2 = np.intersect1d(intersection1,days3)

final_intersection = np.intersect1d(intersection2,days4)

final_intersection = np.intersect1d(final_intersection,days5)


scan1_indices = scan1.Day.isin(final_intersection)

scan2_indices = scan2.Day.isin(final_intersection)

scan3_indices = scan3.Day.isin(final_intersection)

scan4_indices = scan4.Day.isin(final_intersection)

scan5_indices = scan5.Day.isin(final_intersection)


scan1 = scan1.loc[scan1_indices]

scan2 = scan2.loc[scan2_indices]

scan3 = scan3.loc[scan3_indices]

scan4 = scan4.loc[scan4_indices]

scan5 = scan5.loc[scan5_indices]






all_scans = [scan1,scan2,scan3,scan4,scan5]
```

```
    full_scan = pandas.concat(all_scans,ignore_index=True)
    full_scan.sort_values(by=['scan'])
    #full_scan = full_scan.rename({'scan': 'scan_time'},axis='columns')

    return full_scan
```

A.17 show_dates

```
def show_dates(scan):
    """
    Given a scan or set of scans, returns the dates of said scan(s) in iso format

    Parameters
    ----------
    scan: dataframe
        This data frame can contain any number of scans

    Returns
    -------
    Times in ISO format
    """
    times = scan['Day'].values
    times = np.unique(times)
    t = Time(times,format = 'jd', out_subfmt = 'date')
    times = t.iso
    #print(times)
    
    return times
```

A.18 show_overlap_dates

```
def show_overlap_dates():
    """
    The purpose of this function is to print dates where both CSIM and TSIS have a
        full set of scans.
    This is intended to be a reference function for solar variability comparisons
    
    Parameters
    ----------
    None
    
    Returns
    -------
    intersection
        This intersection contains all dates in iso format where CSIM and TSIS
            both had a full scan.
    """
    tsis = load_tsis_data()
    csim = load_all_scans(1)
    csim_days = show_dates(csim)
    tsis_days = show_dates(tsis)
    intersection = np.intersect1d(csim_days,tsis_days)
    return intersection
```

A.19 z_filter

```
def z_filter(array):

    """
    Given an array, returns a filter for said array based on statistical z score.
    Assuming relative consistency of data, data is filtered based on a z-score
        threshold of 3.
    Primarily used to filter integrated set of spectra.

    Parameters
    ----------
    array : array
        array containing mostly consistent dataset

    Returns
    -------
    filter
         this returns a numpy array with indices set to filter the given array

    """

    z = np.abs(stats.zscore(array))
    threshold = 3
    filter = np.where(z < threshold)
    return(filter)
```

A.20 plot_kappas

```
def plot_kappas(id):
    """
    This function takes several integrated regions and finds a kappa calculation
        for each of them
    It also plots the time series spectra,the integrated spectra for channel 1 and
         channel 2 based on exposure
    time, a comparison spectra plotted by calendar time, as well as the kappa line
         on top of the natural log vs
    delta exposure time plot.

    Parameters
    ----------

    id : int
        IDs 1,2,3,4,5

    Returns
    -------
    A list with two elements. The first contains all the kappa values, the second
        contains the average wavelength.
    """
```

```
%matplotlib inline
ch1_data = load_scan(1,id)
ch2_data = load_scan(2,id)

uniq_daysCH2 = ch2_data['Day'].values
uniq_daysCH1 = ch1_data['Day'].values

uniq_days = np.intersect1d(uniq_daysCH1,uniq_daysCH2)



kappas = []
avg_wl = []
if id == 1:
    abs_max = 286
    abs_min = 200
    divisor = 40
    increment = (abs_max - abs_min) / divisor
    max_sirrCH1 = 0.3
    min_sirrCH1 = 0.0001
    max_sirrCH2 = 0.3
    min_sirrCH2 = 0.005
elif id == 2:
    divisor = 10
    abs_max = 387
    abs_min = 286
    increment = (abs_max - abs_min) / divisor
    max_sirrCH1 = 100
    min_sirrCH1 = 0.05
    max_sirrCH2 = 100
    min_sirrCH2 = 0.05
elif id == 3:
    abs_max = 900
    abs_min = 387
    divisor = 10
    increment = (abs_max - abs_min) / divisor
    max_sirrCH1 = 2.15
    min_sirrCH1 = 0.7
    max_sirrCH2 = 2.15
    min_sirrCH2 = 0.7
elif id == 4:
    divisor = 10
    abs_max = 1500
    abs_min = 900
    increment = (abs_max - abs_min) / divisor
    max_sirrCH1 = 100
    min_sirrCH1 = 0.1
    max_sirrCH2 = 100
    min_sirrCH2 = 0.1
elif id == 5:
    divisor = 10
    abs_max = 2300
    abs_min = 1500
    increment = (abs_max - abs_min) / divisor

    max_sirrCH1 = 100

    min_sirrCH1 = 0.06

    max_sirrCH2 = 100

    min_sirrCH2 = 0.06

increment = int(increment)

j = abs_min

for d in range(divisor):



    print(j)

    int_sirrCH1 = []

    int_sirrCH2 = []

    scan_timesCH1 = []

    scan_timesCH2 = []

    exp_CH1 = []

    exp_CH2 = []

    fig, ax = plt.subplots(2,2)


    for i in range(len(uniq_days)):

        CH1indices = np.where(ch1_data['Day'].values == uniq_days[i])

        scanCH1 = ch1_data.iloc[CH1indices]

        scan_number = get_longest_scan(scanCH1)

        CH1indices = np.where(scanCH1['scan'].values == scan_number)

        scanCH1 = scanCH1.iloc[CH1indices]

        scanCH1 = scanCH1.sort_values(by=['Wavelengths'],ignore_index=True)
        scanCH1 = scanCH1.loc[(scanCH1['SIRR'] >= min_sirrCH1) & (scanCH1['
            SIRR'] <= max_sirrCH1)]
        scanCH1 = scanCH1.loc[(scanCH1['Wavelengths'] >= j) & (scanCH1['
            Wavelengths'] <= j + increment)]
        wavelengthsCH1 = scanCH1['Wavelengths'].values
        sirrCH1 = scanCH1['SIRR'].values

        if sirrCH1.size > 0:
            temp_int = integrate.trapz(sirrCH1,wavelengthsCH1)
            int_sirrCH1.append(temp_int)
            scan_timesCH1.append(scan_number)
            exp_CH1.append(np.average(scanCH1['Exposure'].values))
            ax[0,0].plot(wavelengthsCH1,sirrCH1)



    ax[0,0].set_xlabel('Wavelength␣(nm)')
    ax[0,0].set_ylabel('IRR␣((W/m^2)/nm)')
    ax[0,0].set_title('Solar␣Irradiance␣Channel␣1')



    for i in range(len(uniq_days)):
        CH2indices = np.where(ch2_data['Day'].values == uniq_days[i])
        scanCH2 = ch2_data.iloc[CH2indices]
        scan_number = get_longest_scan(scanCH2)
        CH2indices = np.where(scanCH2['scan'].values == scan_number)
        scanCH2 = scanCH2.iloc[CH2indices]
        scanCH2 = scanCH2.sort_values(by=['Wavelengths'],ignore_index=True)
```

```
        scanCH2 = scanCH2.loc[(scanCH2['SIRR'] >= min_sirrCH2) & (scanCH2['
            SIRR'] <= max_sirrCH2)]
        scanCH2 = scanCH2.loc[(scanCH2['Wavelengths'] >= j) & (scanCH2['
            Wavelengths'] <= j + increment)]
        wavelengthsCH2 = scanCH2['Wavelengths'].values
        sirrCH2 = scanCH2['SIRR'].values
        if sirrCH2.size > 0:
            temp_int = integrate.trapz(sirrCH2,wavelengthsCH2)
            int_sirrCH2.append(temp_int)
            scan_timesCH2.append(scan_number)
            exp_CH2.append(np.average(scanCH2['Exposure'].values))
            ax[1,0].plot(wavelengthsCH2,sirrCH2)




ax[1,0].set_xlabel('Wavelength␣(nm)')
ax[1,0].set_ylabel('IRR␣((W/m^2)/nm')
ax[1,0].set_title('Solar␣Irradiance␣Channel␣2')

scan_timesCH1 = np.array(scan_timesCH1)
scan_timesCH2 = np.array(scan_timesCH2)



int_sirrCH1 = np.array(int_sirrCH1)
int_sirrCH2 = np.array(int_sirrCH2)

csim_expCH1 = np.array(exp_CH1)
```

```
csim_expCH2 = np.array(exp_CH2)



nan_filterCH1 = ~np.isnan(int_sirrCH1)

nan_filterCH2 = ~np.isnan(int_sirrCH2)

int_sirrCH1 = int_sirrCH1[nan_filterCH1]

int_sirrCH2 = int_sirrCH2[nan_filterCH2]

scan_timesCH1 = scan_timesCH1[nan_filterCH1]

scan_timesCH2 = scan_timesCH2[nan_filterCH2]

csim_expCH1 = csim_expCH1[nan_filterCH1]

csim_expCH2 = csim_expCH2[nan_filterCH2]






zero_filterCH1 = np.where(int_sirrCH1 > 0)

zero_filterCH2 = np.where(int_sirrCH2 > 0)


int_sirrCH1 = int_sirrCH1[zero_filterCH1]

int_sirrCH2 = int_sirrCH2[zero_filterCH2]

scan_timesCH1 = scan_timesCH1[zero_filterCH1]

scan_timesCH2 = scan_timesCH2[zero_filterCH2]

csim_expCH1 = csim_expCH1[zero_filterCH1]

csim_expCH2 = csim_expCH2[zero_filterCH2]
```

```
int_filterCH1 = z_filter(int_sirrCH1)
int_filterCH2 = z_filter(int_sirrCH2)

int_sirrCH1 = int_sirrCH1[int_filterCH1]
int_sirrCH2 = int_sirrCH2[int_filterCH2]
scan_timesCH1 = scan_timesCH1[int_filterCH1]
scan_timesCH2 = scan_timesCH2[int_filterCH2]
csim_expCH1 = csim_expCH1[int_filterCH1]
csim_expCH2 = csim_expCH2[int_filterCH2]




if int_sirrCH1[0] > int_sirrCH2[0]:
    diff = int_sirrCH1[0] - int_sirrCH2[0]
    int_sirrCH2 = int_sirrCH2 + diff
elif int_sirrCH2[0] > int_sirrCH1[0]:
    diff = int_sirrCH2[0] - int_sirrCH1[0]
    int_sirrCH1 = int_sirrCH1 + diff


intersection = np.intersect1d(scan_timesCH1,scan_timesCH2)
maskCH1 = np.isin(scan_timesCH1,intersection)
maskCH2 = np.isin(scan_timesCH2, intersection)
mask_int_sirrCH1 = int_sirrCH1[maskCH1]
mask_csim_expCH1 = csim_expCH1[maskCH1]
mask_int_sirrCH2 = int_sirrCH2[maskCH2]
```

```
mask_csim_expCH2 = csim_expCH2[maskCH2]

mask_scan_timesCH1 = scan_timesCH1[maskCH1]

mask_scan_timesCH2 = scan_timesCH2[maskCH2]







y = mask_int_sirrCH1

y0 = y[~np.isnan(y)][0]

def to_percent(y):

    return (y - y0) / y0 * 100

def from_percent(y):

    return y



wavelength_title = '␣over␣band:␣'+ str(j) + "␣-␣" + str(j+increment)





times = time_convert(mask_scan_timesCH1,'gps','decimalyear')

times2 = time_convert(mask_scan_timesCH1,'gps','decimalyear')

ax[0,1].plot(times,mask_int_sirrCH1,'bo')

ax[0,1].plot(times,mask_int_sirrCH2,'ro')

ax[0,1].set_xlabel('Times␣(decimal␣year)')

ax[0,1].set_ylabel('Integrated␣Solar␣Irradiance␣(W/m^2)')
```

```
title = 'Integrated␣Time␣Comparison' + wavelength_title
ax[0,1].set_title(title)
y = mask_int_sirrCH1
y0 = y[~np.isnan(y)][0]
sec_ax = ax[0,1].secondary_yaxis('right', functions=(to_percent,
    from_percent))
sec_ax.set_ylabel('Percent␣Change')
















delta = mask_csim_expCH1 - mask_csim_expCH2
ratio = mask_int_sirrCH1 / mask_int_sirrCH2
ratio = np.log(ratio)

delta = delta.reshape(-1,1)
model = LinearRegression().fit(delta,ratio)

ax[1,1].plot(delta,ratio,'go')
x_vals = np.array(ax[1,1].get_xlim())
```

```
m = model.coef_[0]

b = model.intercept_

y_vals = m * x_vals + b





ax[1,1].plot(x_vals,y_vals)

ax[1,1].set_xlabel('Delta␣Exposure␣Time␣(s)')

ax[1,1].set_ylabel('ln(int_CH1/int_CH2)')

title = 'Kappa' + wavelength_title

r_squared = model.score(delta,ratio)

r_string = 'r^2␣value␣=␣' + str(round(r_squared,3))

ax[1,1].annotate(r_string, (np.average(delta), np.average(ratio)))

ax[1,1].set_title(title)

fig.tight_layout()

fig.set_figwidth(15)

fig.set_figheight(10)

fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/overall_kappa/kappa_'
    + str(j) + '.png'

fig.savefig(fname)

plt.show()



fig, ax = plt.subplots(2)
```

```
ax[0].plot(csim_expCH1,int_sirrCH1,'bo')


ax[0].set_xlabel('Exposure␣Time␣(s)')
ax[0].set_ylabel('Integrated␣Solar␣Irradiance␣Channel␣1␣((W/m^2))')
title = 'Integrated␣Solar␣Irradiance␣Channel␣1' + wavelength_title
ax[0].set_title(title)
y = int_sirrCH1
y0 = y[~np.isnan(y)][0]
sec_ax = ax[0].secondary_yaxis('right', functions=(to_percent,
   from_percent))
sec_ax.set_ylabel('Percent␣Change')







ax[1].plot(csim_expCH2,int_sirrCH2,'ro')


ax[1].set_xlabel('Exposure␣Time␣(s)')
ax[1].set_ylabel('Integrated␣Solar␣Irradiance␣Channel␣2␣((W/m^2))')
title = 'Integrated␣Solar␣Irradiance␣Channel␣2' + wavelength_title
ax[1].set_title(title)
y = int_sirrCH2
y0 = y[~np.isnan(y)][0]
sec_ax = ax[1].secondary_yaxis('right', functions=(to_percent,
   from_percent))
sec_ax.set_ylabel('Percent␣Change')
```

```
        fig.tight_layout()

        plt.show()




        kappas.append(m)

        avg = (j + (j+increment)) / 2

        avg_wl.append(avg)



        j = j + increment



    print(kappas)


    print('kappas␣returned',kappas)

    print(avg_wl)

    result = []

    result.append(kappas)

    result.append(avg_wl)

    return result
```

A.21 get_kappas

```
def get_kappas(id):

    """

    This performs the same analysis as plot_kappas, but without outputting plots.
```

```
Parameters
----------

id : int
    IDs 1,2,3,4,5

Returns
-------
A list with two elements. The first contains all the kappa values, the second
    contains the average wavelength.
"""



ch1_data = load_scan(1,id)
ch2_data = load_scan(2,id)

uniq_daysCH2 = ch2_data['Day'].values
uniq_daysCH1 = ch1_data['Day'].values

uniq_days = np.intersect1d(uniq_daysCH1,uniq_daysCH2)




kappas = []
avg_wl = []
if id == 1:
        abs_max = 286

        abs_min = 200

        divisor = 40

        increment = (abs_max - abs_min) / divisor

        max_sirrCH1 = 0.3

        min_sirrCH1 = 0.0001

        max_sirrCH2 = 0.3

        min_sirrCH2 = 0.005

    elif id == 2:

        divisor = 10

        abs_max = 387

        abs_min = 286

        increment = (abs_max - abs_min) / divisor

        max_sirrCH1 = 100

        min_sirrCH1 = 0.05

        max_sirrCH2 = 100

        min_sirrCH2 = 0.05

    elif id == 3:

        abs_max = 900

        abs_min = 387

        divisor = 10

        increment = (abs_max - abs_min) / divisor

        max_sirrCH1 = 2.15

        min_sirrCH1 = 0.7

        max_sirrCH2 = 2.15

        min_sirrCH2 = 0.7

    elif id == 4:
        divisor = 10
        abs_max = 1500
        abs_min = 900
        increment = (abs_max - abs_min) / divisor
        max_sirrCH1 = 100
        min_sirrCH1 = 0.1
        max_sirrCH2 = 100
        min_sirrCH2 = 0.1
    elif id == 5:
        divisor = 10
        abs_max = 2300
        abs_min = 1500
        increment = (abs_max - abs_min) / divisor
        max_sirrCH1 = 100
        min_sirrCH1 = 0.06
        max_sirrCH2 = 100
        min_sirrCH2 = 0.06
    increment = int(increment)
    j = abs_min
    for d in range(divisor):


        int_sirrCH1 = []
        int_sirrCH2 = []
        scan_timesCH1 = []
        scan_timesCH2 = []
        exp_CH1 = []
```

```
exp_CH2 = []
plt.figure(0)

for i in range(len(uniq_days)):
    CH1indices = np.where(ch1_data['Day'].values == uniq_days[i])
    scanCH1 = ch1_data.iloc[CH1indices]
    scan_number = get_longest_scan(scanCH1)
    CH1indices = np.where(scanCH1['scan'].values == scan_number)
    scanCH1 = scanCH1.iloc[CH1indices]
    scanCH1 = scanCH1.sort_values(by=['Wavelengths'],ignore_index=True)
    scanCH1 = scanCH1.loc[(scanCH1['SIRR'] >= min_sirrCH1) & (scanCH1['
        SIRR'] <= max_sirrCH1)]
    scanCH1 = scanCH1.loc[(scanCH1['Wavelengths'] >= j) & (scanCH1['
        Wavelengths'] <= j + increment)]
    wavelengthsCH1 = scanCH1['Wavelengths'].values
    sirrCH1 = scanCH1['SIRR'].values

    if sirrCH1.size > 0:
        temp_int = integrate.trapz(sirrCH1,wavelengthsCH1)
        int_sirrCH1.append(temp_int)
        scan_timesCH1.append(scan_number)
        exp_CH1.append(np.average(scanCH1['Exposure'].values))



for i in range(len(uniq_days)):
    CH2indices = np.where(ch2_data['Day'].values == uniq_days[i])
```

```
        scanCH2 = ch2_data.iloc[CH2indices]
        scan_number = get_longest_scan(scanCH2)
        CH2indices = np.where(scanCH2['scan'].values == scan_number)
        scanCH2 = scanCH2.iloc[CH2indices]
        scanCH2 = scanCH2.sort_values(by=['Wavelengths'],ignore_index=True)
        scanCH2 = scanCH2.loc[(scanCH2['SIRR'] >= min_sirrCH2) & (scanCH2['
            SIRR'] <= max_sirrCH2)]
        scanCH2 = scanCH2.loc[(scanCH2['Wavelengths'] >= j) & (scanCH2['
            Wavelengths'] <= j + increment)]
        wavelengthsCH2 = scanCH2['Wavelengths'].values
        sirrCH2 = scanCH2['SIRR'].values
        if sirrCH2.size > 0:
            temp_int = integrate.trapz(sirrCH2,wavelengthsCH2)
            int_sirrCH2.append(temp_int)
            scan_timesCH2.append(scan_number)
            exp_CH2.append(np.average(scanCH2['Exposure'].values))



    scan_timesCH1 = np.array(scan_timesCH1)
    scan_timesCH2 = np.array(scan_timesCH2)



    int_sirrCH1 = np.array(int_sirrCH1)
    int_sirrCH2 = np.array(int_sirrCH2)

    csim_expCH1 = np.array(exp_CH1)
    csim_expCH2 = np.array(exp_CH2)
```

```
nan_filterCH1 = ~np.isnan(int_sirrCH1)

nan_filterCH2 = ~np.isnan(int_sirrCH2)

int_sirrCH1 = int_sirrCH1[nan_filterCH1]

int_sirrCH2 = int_sirrCH2[nan_filterCH2]

scan_timesCH1 = scan_timesCH1[nan_filterCH1]

scan_timesCH2 = scan_timesCH2[nan_filterCH2]

csim_expCH1 = csim_expCH1[nan_filterCH1]

csim_expCH2 = csim_expCH2[nan_filterCH2]




zero_filterCH1 = np.where(int_sirrCH1 > 0)

zero_filterCH2 = np.where(int_sirrCH2 > 0)


int_sirrCH1 = int_sirrCH1[zero_filterCH1]

int_sirrCH2 = int_sirrCH2[zero_filterCH2]

scan_timesCH1 = scan_timesCH1[zero_filterCH1]

scan_timesCH2 = scan_timesCH2[zero_filterCH2]

csim_expCH1 = csim_expCH1[zero_filterCH1]

csim_expCH2 = csim_expCH2[zero_filterCH2]



int_filterCH1 = z_filter(int_sirrCH1)
```

```
int_filterCH2 = z_filter(int_sirrCH2)


int_sirrCH1 = int_sirrCH1[int_filterCH1]
int_sirrCH2 = int_sirrCH2[int_filterCH2]
scan_timesCH1 = scan_timesCH1[int_filterCH1]
scan_timesCH2 = scan_timesCH2[int_filterCH2]
csim_expCH1 = csim_expCH1[int_filterCH1]
csim_expCH2 = csim_expCH2[int_filterCH2]





if int_sirrCH1[0] > int_sirrCH2[0]:
    diff = int_sirrCH1[0] - int_sirrCH2[0]
    int_sirrCH2 = int_sirrCH2 + diff
elif int_sirrCH2[0] > int_sirrCH1[0]:
    diff = int_sirrCH2[0] - int_sirrCH1[0]
    int_sirrCH1 = int_sirrCH1 + diff


intersection = np.intersect1d(scan_timesCH1,scan_timesCH2)
maskCH1 = np.isin(scan_timesCH1,intersection)
maskCH2 = np.isin(scan_timesCH2, intersection)
mask_int_sirrCH1 = int_sirrCH1[maskCH1]
mask_csim_expCH1 = csim_expCH1[maskCH1]
mask_int_sirrCH2 = int_sirrCH2[maskCH2]
mask_csim_expCH2 = csim_expCH2[maskCH2]
```

```
mask_scan_timesCH1 = scan_timesCH1[maskCH1]

mask_scan_timesCH2 = scan_timesCH2[maskCH2]

delta = mask_csim_expCH1 - mask_csim_expCH2

ratio = mask_int_sirrCH1 / mask_int_sirrCH2

ratio = np.log(ratio)


delta = delta.reshape(-1,1)

model = LinearRegression().fit(delta,ratio)


#using y = mx+B

m = model.coef_[0]
```

```
        kappas.append(m)

        avg = (j + (j+increment)) / 2

        avg_wl.append(avg)



        j = j + increment



    result = []

    result.append(kappas)

    result.append(avg_wl)

    return result
```

A.22 solar_var

```
def solar_var(date1,date2,channel):

    """
```

```
The purpose of this function is to plot solar variability of TSIS and CSIM
    given two dates and the
channel to analyze.


Parameters
----------
date1,date2 : string
    two dates in iso format
channel: int
    The instrument channel to look at. Either 1 or 2


Returns
-------
Outputs two spectra plots of CSIM (one per date), two spectra plots of TSIS (
    one per date)
the solar variability for both spacecraft, and three comparison plots. Returns
     nothing

"""


%matplotlib inline
test = 0
full_scan = load_all_scans(channel)
scan1 = get_scan(full_scan,date1)
if len(scan1) == 1:
    print('No␣scan␣found␣on␣' + date1)
    test = 1
```

```
scan2 = get_scan(full_scan,date2)
if len(scan2) == 1:
    print('No␣scan␣found␣on␣' + date2)
    test = 1



if test == 1:
    print('could␣not␣complete␣ratio')
    return
else:



    plot_title1 = 'Solar␣Irradiance␣on␣' + date1
    plot_title2 = 'Solar␣Irradiance␣on␣' + date2

    plot_scan(scan1,plot_title1)
    plot_scan(scan2,plot_title2)





    #default assume day 1 has min and max

    min_wl = scan1['Wavelengths'].min()
    max_wl = scan1['Wavelengths'].max()

    if min_wl < scan2['Wavelengths'].min():
        min_wl = scan2['Wavelengths'].min()



    if max_wl > scan2['Wavelengths'].max():
        max_wl = scan2['Wavelengths'].max()



    scan1 = scan1.loc[scan1.Wavelengths.between(min_wl,max_wl)]

    scan2 = scan2.loc[scan2.Wavelengths.between(min_wl,max_wl)]



    sirr1 = scan1['SIRR'].values
    sirr2 = scan2['SIRR'].values
    wavelengths1 = scan1['Wavelengths'].values
    wavelengths2 = scan2['Wavelengths'].values
    print(len(sirr1))
    print(len(sirr2))



    if len(sirr1) != len(sirr2):
        wavelength_grid = np.intersect1d(wavelengths1,wavelengths2)
        scan1_indices = scan1.Wavelengths.isin(wavelength_grid)
        scan2_indices = scan2.Wavelengths.isin(wavelength_grid)

        scan1 = scan1.loc[scan1_indices]
        scan2 = scan2.loc[scan2_indices]
        sirr1 = scan1['SIRR'].values

        sirr2 = scan2['SIRR'].values




    else:

        wavelength_grid = wavelengths1







    ratio = ((sirr2 / sirr1) - 1)*100

    if np.average(sirr1) > np.average(sirr2):

        ratio = ((sirr1 / sirr2) - 1)*100



    ratio_filter = np.where((ratio < 10) & (ratio > -0.5))


    ratio = ratio[ratio_filter]

    wavelength_grid = wavelength_grid[ratio_filter]




    fig, ax = plt.subplots()


    line = ax.plot(wavelength_grid,ratio)
```

```
ax.set_xlabel('Wavelength␣(nm)')
ax.set_ylabel('((Max/Min)␣-␣1)*100␣(%)')

plot_title = 'CSIM␣Solar␣Variability␣between␣' + date1 + '␣and␣' + date2
ax.set_title(plot_title)

plt.show()


fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/csim_solarvar_'
fname += str(date1) + '-' + str(date2) + '.png'

fig.savefig(fname)




shorten_indices = np.where(wavelength_grid < 400)
shortened_wavelengths = wavelength_grid[shorten_indices]
shortened_ratio = ratio[shorten_indices]
```

```
fig, ax = plt.subplots()

line = ax.plot(shortened_wavelengths,shortened_ratio)



ax.set_xlabel('Wavelength␣(nm)')
ax.set_ylabel('((Max/Min)␣-␣1)*100␣(%)')

plot_title = 'CSIM␣Solar␣Variability␣between␣' + date1 + '␣and␣' + date2
ax.set_title(plot_title)

plt.show()


fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/short_csim_solarvar_'
fname += str(date1) + '-' + str(date2) + '.png'

fig.savefig(fname)




#plot TSIS ones now
```

```
test = 0

tsis_scan = load_tsis_data()

tsis_scan1 = get_scan(tsis_scan,date1)

if len(tsis_scan1) == 1:

    print('No␣scan␣found␣on␣' + date1)

    test = 1

tsis_scan2 = get_scan(tsis_scan,date2)

if len(tsis_scan2) == 1:

    print('No␣scan␣found␣on␣' + date2)

    test = 1



if test == 1:

    print('could␣not␣complete␣ratio␣for␣TSIS')

    return

else:



    plot_title1 = 'TSIS␣Solar␣Irradiance␣on␣' + date1

    plot_title2 = 'TSIS␣Solar␣Irradiance␣on␣' + date2


    plot_scan(tsis_scan1,plot_title1)

    plot_scan(tsis_scan2,plot_title2)




    tsis_sirr1 = tsis_scan1['SIRR'].values
```

```
tsis_sirr2 = tsis_scan2['SIRR'].values



tsis_ratio = ((tsis_sirr2 / tsis_sirr1) - 1)*100

if np.average(sirr1) > np.average(sirr2):

    tsis_ratio = ((tsis_sirr1 / tsis_sirr2) - 1)*100


tsis_wavelengths = tsis_scan1['Wavelengths'].values




tsis_ratio_filter = np.where((tsis_ratio < 10) & (tsis_ratio > -10))


tsis_ratio = tsis_ratio[tsis_ratio_filter]

tsis_wavelengths = tsis_wavelengths[tsis_ratio_filter]



fig, ax = plt.subplots()


line = ax.plot(tsis_wavelengths,tsis_ratio)




ax.set_xlabel('Wavelength␣(nm)')

ax.set_ylabel('((Max/Min)␣-␣1)*100␣(%)')


plot_title = 'TSIS␣Solar␣Variability␣between␣' + date1 + '␣and␣' + date2
```

```
ax.set_title(plot_title)

plt.show()



fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/tsis_solarvar_'
fname += str(date1) + '-' + str(date2) + '.png'

fig.savefig(fname)




#show shorter version
#plt.figure(1)
shorten_indices = np.where(tsis_wavelengths < 400)
shortened_tsis_wavelengths = tsis_wavelengths[shorten_indices]
shortened_tsis_ratio = tsis_ratio[shorten_indices]



fig, ax = plt.subplots()

line = ax.plot(shortened_tsis_wavelengths,shortened_tsis_ratio)




ax.set_xlabel('Wavelength␣(nm)')
ax.set_ylabel('((Max/Min)␣-␣1)*100␣(%)')
```

```
plot_title = 'TSIS␣Solar␣Variability␣between␣' + date1 + '␣and␣' + date2
ax.set_title(plot_title)

plt.show()



fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/short_tsis_solarvar_'
fname += str(date1) + '-' + str(date2) + '.png'

fig.savefig(fname)






#show a combined plot of tsis and csim solar variability




fig, ax = plt.subplots(3)

ax[0].plot(shortened_tsis_wavelengths,shortened_tsis_ratio,label='TSIS')
ax[0].plot(shortened_wavelengths,shortened_ratio,'ro',alpha=0.5,label='
    CSIM',markersize=2)
```

```
ax[0].set_xlabel('Wavelength␣(nm)')

ax[0].set_ylabel('((Max/Min)␣-␣1)*100␣(%)')

ax[0].set_title("200nm-400nm")

plot_title = 'Solar␣Variability␣between␣' + date1 + '␣and␣' + date2

fig.suptitle(plot_title,size=20)









#shorten


shorten_indices = np.where(shortened_tsis_wavelengths > 230)

shortened_tsis_wavelengths = shortened_tsis_wavelengths[shorten_indices]

shortened_tsis_ratio = shortened_tsis_ratio[shorten_indices]

shorten_indices = np.where(shortened_wavelengths > 230)

shortened_wavelengths = shortened_wavelengths[shorten_indices]

shortened_ratio = shortened_ratio[shorten_indices]




ax[1].plot(shortened_tsis_wavelengths,shortened_tsis_ratio,label='TSIS')

ax[1].plot(shortened_wavelengths,shortened_ratio,'ro',alpha=0.5,label='
    CSIM',markersize=2)
```

```
ax[1].set_xlabel('Wavelength␣(nm)')

ax[1].set_ylabel('((Max/Min)␣-␣1)*100␣(%)')

ax[1].set_title("225nm-400nm")




#shorten to show different range

shorten_indices = np.where(shortened_tsis_wavelengths > 300)

shortened_tsis_wavelengths = shortened_tsis_wavelengths[shorten_indices]

shortened_tsis_ratio = shortened_tsis_ratio[shorten_indices]

shorten_indices = np.where(shortened_wavelengths > 300)

shortened_wavelengths = shortened_wavelengths[shorten_indices]

shortened_ratio = shortened_ratio[shorten_indices]




ax[2].plot(shortened_tsis_wavelengths,shortened_tsis_ratio,label='TSIS')
```

```
ax[2].plot(shortened_wavelengths,shortened_ratio,'r',alpha=0.5,label='CSIM
    ')



ax[2].set_xlabel('Wavelength␣(nm)')
ax[2].set_ylabel('((Max/Min)␣-␣1)*100␣(%)')
ax[2].set_title("300nm-400nm")

ax[0].legend()







fig.set_figwidth(8)
fig.set_figheight(10)
fig.tight_layout()
#fig.subplots_adjust(top=0.95)

plt.show()
fname= '/Users/asm0945/Documents/IDL/CSIM/esr/plots/
    csim_tsis_solarvar_comparison.png'
fig.savefig(fname)
```

```
    return
```

A.23 two_stage_kappa

```
def two_stage_kappa(scanCH1,scanCH2,id,key):
    """
    This function takes several integrated regions and finds a kappa calculation
       for each of them
    It also plots the time series spectra,the integrated spectra for channel 1 and
        channel 2 based on exposure
    time, a comparison spectra plotted by calendar time, as well as the kappa line
        on top of the natural log vs
    delta exposure time plot. This function does this by splitting the spectra at
       the global variable exposure split

    Parameters
    ----------

    scanCH1: dataframe
        This data frame contains the entire data frame of channel 1 at a given ID

    scanCH2: dataframe
        This data frame contains the entire data frame of channel 2 at a given ID

    id : int
        IDs 1,2,3,4,5
```

```
Returns
-------
A list with two elements. The first contains all the kappa values, the second
    contains the average wavelength.
"""

plot = 0

%matplotlib inline
ch1_data = scanCH1
ch2_data = scanCH2

uniq_daysCH2 = ch2_data['Day'].values
uniq_daysCH1 = ch1_data['Day'].values

uniq_days = np.intersect1d(uniq_daysCH1,uniq_daysCH2)




kappas = []
avg_wl = []
if id == 1:
    abs_max = 286
    abs_min = 200
    divisor = 40
    increment = (abs_max - abs_min) / divisor
    max_sirrCH1 = 0.3
    min_sirrCH1 = 0.0001
    max_sirrCH2 = 0.3
    min_sirrCH2 = 0.005
elif id == 2:
    divisor = 10
    abs_max = 387
    abs_min = 286
    increment = (abs_max - abs_min) / divisor
    max_sirrCH1 = 100
    min_sirrCH1 = 0.05
    max_sirrCH2 = 100
    min_sirrCH2 = 0.05
elif id == 3:
    abs_max = 900
    abs_min = 387
    divisor = 10
    increment = (abs_max - abs_min) / divisor
    max_sirrCH1 = 2.15
    min_sirrCH1 = 0.7
    max_sirrCH2 = 2.15
    min_sirrCH2 = 0.7
elif id == 4:
    divisor = 10
    abs_max = 1500
    abs_min = 900
    increment = (abs_max - abs_min) / divisor
    max_sirrCH1 = 100
    min_sirrCH1 = 0.1
    max_sirrCH2 = 100
    min_sirrCH2 = 0.1
elif id == 5:
    divisor = 10
    abs_max = 2300
    abs_min = 1500
    increment = (abs_max - abs_min) / divisor
    max_sirrCH1 = 100
    min_sirrCH1 = 0.06
    max_sirrCH2 = 100
    min_sirrCH2 = 0.06
increment = int(increment)
j = abs_min
for d in range(divisor):

    #unacceptable_wavs =
        [959,1108,1257,1514,1602,811]#[248,252,254,260,262,266,270,288,256,264,272,292,296,3

    #if j in unacceptable_wavs or (j == 810 and id == 5):
     # continue
    print(j)
    int_sirrCH1 = []
    int_sirrCH2 = []
    scan_timesCH1 = []
```

```
scan_timesCH2 = []

exp_CH1 = []

exp_CH2 = []

if plot == 0:

    fig, ax = plt.subplots(2,2)


for i in range(len(uniq_days)):

    CH1indices = np.where(ch1_data['Day'].values == uniq_days[i])

    scanCH1 = ch1_data.iloc[CH1indices]

    scan_number = get_longest_scan(scanCH1)

    CH1indices = np.where(scanCH1['scan'].values == scan_number)

    scanCH1 = scanCH1.iloc[CH1indices]

    scanCH1 = scanCH1.sort_values(by=['Wavelengths'],ignore_index=True)

    scanCH1 = scanCH1.loc[(scanCH1['SIRR'] >= min_sirrCH1) & (scanCH1['
        SIRR'] <= max_sirrCH1)]

    scanCH1 = scanCH1.loc[(scanCH1['Wavelengths'] >= j) & (scanCH1['
        Wavelengths'] <= j + increment)]

    wavelengthsCH1 = scanCH1['Wavelengths'].values

    sirrCH1 = scanCH1['SIRR'].values


    if sirrCH1.size > 0:

        temp_int = integrate.trapz(sirrCH1,wavelengthsCH1)



        int_sirrCH1.append(temp_int)

        scan_timesCH1.append(scan_number)

        exp_CH1.append(np.average(scanCH1['Exposure'].values))
```

```
            if plot == 0:
                ax[0,0].plot(wavelengthsCH1,sirrCH1)

    if plot == 0:
        ax[0,0].set_xlabel('Wavelength␣(nm)')
        ax[0,0].set_ylabel('IRR␣((W/m^2)/␣nm)')
        ax[0,0].set_title('Solar␣Irradiance␣Channel␣1')



    for i in range(len(uniq_days)):
        CH2indices = np.where(ch2_data['Day'].values == uniq_days[i])
        scanCH2 = ch2_data.iloc[CH2indices]
        scan_number = get_longest_scan(scanCH2)
        CH2indices = np.where(scanCH2['scan'].values == scan_number)
        scanCH2 = scanCH2.iloc[CH2indices]
        scanCH2 = scanCH2.sort_values(by=['Wavelengths'],ignore_index=True)
        scanCH2 = scanCH2.loc[(scanCH2['SIRR'] >= min_sirrCH2) & (scanCH2['
            SIRR'] <= max_sirrCH2)]
        scanCH2 = scanCH2.loc[(scanCH2['Wavelengths'] >= j) & (scanCH2['
            Wavelengths'] <= j + increment)]
        wavelengthsCH2 = scanCH2['Wavelengths'].values
        sirrCH2 = scanCH2['SIRR'].values
        if sirrCH2.size > 0:
            temp_int = integrate.trapz(sirrCH2,wavelengthsCH2)
            int_sirrCH2.append(temp_int)

            scan_timesCH2.append(scan_number)

            exp_CH2.append(np.average(scanCH2['Exposure'].values))

            if plot == 0:

                ax[1,0].plot(wavelengthsCH2,sirrCH2)



    if plot == 0:

        ax[1,0].set_xlabel('Wavelength␣(nm)')

        ax[1,0].set_ylabel('IRR␣((W/m^2)/nm')

        ax[1,0].set_title('Solar␣Irradiance␣Channel␣2')


    if len(int_sirrCH2) > 2 and len(int_sirrCH1) > 2:


        scan_timesCH1 = np.array(scan_timesCH1)

        scan_timesCH2 = np.array(scan_timesCH2)



        int_sirrCH1 = np.array(int_sirrCH1)

        int_sirrCH2 = np.array(int_sirrCH2)


        csim_expCH1 = np.array(exp_CH1)

        csim_expCH2 = np.array(exp_CH2)



        nan_filterCH1 = ~np.isnan(int_sirrCH1)

        nan_filterCH2 = ~np.isnan(int_sirrCH2)
```

```
int_sirrCH1 = int_sirrCH1[nan_filterCH1]

int_sirrCH2 = int_sirrCH2[nan_filterCH2]

scan_timesCH1 = scan_timesCH1[nan_filterCH1]

scan_timesCH2 = scan_timesCH2[nan_filterCH2]

csim_expCH1 = csim_expCH1[nan_filterCH1]

csim_expCH2 = csim_expCH2[nan_filterCH2]





zero_filterCH1 = np.where(int_sirrCH1 > 0)

zero_filterCH2 = np.where(int_sirrCH2 > 0)


int_sirrCH1 = int_sirrCH1[zero_filterCH1]

int_sirrCH2 = int_sirrCH2[zero_filterCH2]

scan_timesCH1 = scan_timesCH1[zero_filterCH1]

scan_timesCH2 = scan_timesCH2[zero_filterCH2]

csim_expCH1 = csim_expCH1[zero_filterCH1]

csim_expCH2 = csim_expCH2[zero_filterCH2]




int_filterCH1 = z_filter(int_sirrCH1)

int_filterCH2 = z_filter(int_sirrCH2)


int_sirrCH1 = int_sirrCH1[int_filterCH1]

int_sirrCH2 = int_sirrCH2[int_filterCH2]
```

```
scan_timesCH1 = scan_timesCH1[int_filterCH1]

scan_timesCH2 = scan_timesCH2[int_filterCH2]

csim_expCH1 = csim_expCH1[int_filterCH1]

csim_expCH2 = csim_expCH2[int_filterCH2]







if int_sirrCH1[0] > int_sirrCH2[0]:

   diff = int_sirrCH1[0] - int_sirrCH2[0]

   int_sirrCH2 = int_sirrCH2 + diff

elif int_sirrCH2[0] > int_sirrCH1[0]:

   diff = int_sirrCH2[0] - int_sirrCH1[0]

   int_sirrCH1 = int_sirrCH1 + diff



intersection = np.intersect1d(scan_timesCH1,scan_timesCH2)

maskCH1 = np.isin(scan_timesCH1,intersection)

maskCH2 = np.isin(scan_timesCH2, intersection)

mask_int_sirrCH1 = int_sirrCH1[maskCH1]

mask_csim_expCH1 = csim_expCH1[maskCH1]

mask_int_sirrCH2 = int_sirrCH2[maskCH2]

mask_csim_expCH2 = csim_expCH2[maskCH2]

mask_scan_timesCH1 = scan_timesCH1[maskCH1]

mask_scan_timesCH2 = scan_timesCH2[maskCH2]
```

```
wavelength_title = '␣over␣band:␣'+ str(j) + "␣-␣" + str(j+increment)

base = mask_int_sirrCH1[0]

if plot == 0:

    times = time_convert(mask_scan_timesCH1,'gps','decimalyear')
    #times = mask_scan_timesCH1
    ax[0,1].plot(times,mask_int_sirrCH1,'bo')
    ax[0,1].plot(times,mask_int_sirrCH2,'ro')
    ax[0,1].set_xlabel('Times␣(decimal␣year)')
    ax[0,1].set_ylabel('Integrated␣Solar␣Irradiance␣(W/m^2)')
    title = 'Integrated␣Spectra␣Time␣Comparison' + wavelength_title
    ax[0,0].set_title(title)
    y = mask_int_sirrCH1
    y0 = y[~np.isnan(y)][0]
    def to_percent(y):
        return (y - y0) / y0 * 100
    def from_percent(y):
        return y
    sec_ax = ax[0,1].secondary_yaxis('right', functions=(to_percent,
        from_percent))
    sec_ax.set_ylabel('Percent␣Change')













delta = mask_csim_expCH1 - mask_csim_expCH2

ratio = mask_int_sirrCH1 / mask_int_sirrCH2

ratio = np.log(ratio)


delta = delta.reshape(-1,1)

model = LinearRegression().fit(delta,ratio)

m = model.coef_[0]

b = model.intercept_

if plot == 0:


    ax[1,1].plot(delta,ratio,'go')

    x_vals = np.array(ax[1,1].get_xlim())

    #using y = mx+B


    y_vals = m * x_vals + b
```

```
ax[1,1].plot(x_vals,y_vals)

ax[1,1].set_xlabel('Delta␣Exposure␣Time␣(s)')

ax[1,1].set_ylabel('ln(int_CH1/int_CH2)')

title = 'Kappa' + wavelength_title

ax[1,1].set_title(title)

r_squared = model.score(delta,ratio)

r_string = 'r^2␣value␣=␣' + str(round(r_squared,3))

ax[1,1].annotate(r_string, (np.average(delta), np.average(ratio)))


fig.tight_layout()

fig.set_figwidth(15)

fig.set_figheight(10)

fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/

   two_stage_kappa/two_stage_kappa_' + str(j) + str(key) + '.png'

fig.savefig(fname)

plt.show()
```

```
    if plot == 0:
        fig, ax = plt.subplots(2)


        ax[0].plot(csim_expCH1,int_sirrCH1,'bo')


        ax[0].set_xlabel('Exposure␣Time␣(s)')
        ax[0].set_ylabel('IRR((W/m^2))')
        title = 'Integrated␣Solar␣Irradiance␣Channel␣1' + wavelength_title
        ax[0].set_title(title)
        y = int_sirrCH1
        y0 = y[~np.isnan(y)][0]
        sec_ax = ax[0].secondary_yaxis('right', functions=(to_percent,
            from_percent))
        sec_ax.set_ylabel('Percent␣Change')




        ax[1].plot(csim_expCH2,int_sirrCH2,'ro')


        ax[1].set_xlabel('Exposure␣Time␣(s)')
        ax[1].set_ylabel('IRR␣((W/m^2))')
        title = 'Integrated␣Solar␣Irradiance␣Channel␣2' + wavelength_title
        ax[1].set_title(title)
        y = int_sirrCH2
        y0 = y[~np.isnan(y)][0]
        sec_ax = ax[1].secondary_yaxis('right', functions=(to_percent,
            from_percent))
```

```
                sec_ax.set_ylabel('Percent␣Change')

                plt.show()



            kappas.append(m)
            avg = (j + (j+increment)) / 2
            avg_wl.append(avg)



            j = j + increment
        else:
            print('no␣scans␣in␣this␣region')
            j = j + increment



    print(kappas)
    print('kappas␣returned',kappas)
    print(avg_wl)
    result = []
    result.append(kappas)
    result.append(avg_wl)
    return result
```

A.24 two_stage_correction

```
def two_stage_correction(channel):
    """
    This function runs the two stage kappa analysis for every wavelength,
        generates a kappa curve
    and then applies corrections to the entire set of spectra.

    Parameters
    ----------

    channel : int
        1 or 2

    Returns
    -------
    A dataframe with corrected data
    """
    corr_dfs = []
    ids = [1,2,3,4,5]
    k_bef = []
    k_aft = []
    wl = []
    for i in ids:
        scanCH1 = load_scan(1,i)
        scanCH2 = load_scan(2,i)
        first_halfCH1 = scanCH1.iloc[np.where(scanCH1['Exposure'].values <=
            exposure_split)]
        second_halfCH1 = scanCH1.iloc[np.where(scanCH1['Exposure'].values >
            exposure_split)]
        result_bef = two_stage_kappa(first_halfCH1,scanCH2,i,'region_one')
        result_aft = two_stage_kappa(second_halfCH1,scanCH2,i,'region_two')
        k_bef += result_bef[0]
        k_aft += result_aft[0]
        wl += result_bef[1]



    wl += [2300,2400]
    k_aft += [0,0]
    k_bef += [0,0]
    print('first␣region␣kappas',k_bef)
    print('second␣region␣kappas',k_aft)
    print('wavelengths',wl)
    k_bef_function = interpolate.interp1d(wl,k_bef,kind='cubic')
    k_aft_function = interpolate.interp1d(wl,k_aft,kind='cubic')

    wl_grid = np.linspace(201,2400,3000)

    fig, ax = plt.subplots(3,2)

    ax[0,0].plot(wl_grid,-k_bef_function(wl_grid),'r')
    ax[0,0].set_xlabel('Wavelength␣(nm)')
    ax[0,0].set_ylabel('Kappa')
    title = 'First␣Region␣Overall␣Kappa'
    ax[0,0].set_title(title)
```

```
ax[0,1].plot(wl_grid,-k_aft_function(wl_grid),'b')
ax[0,1].set_xlabel('Wavelength␣(nm)')
ax[0,1].set_ylabel('Kappa')
title = 'Second␣Region␣Overall␣Kappa'
ax[0,0].set_title(title)


wl_grid_low = wl_grid[np.where((wl_grid > 210) &(wl_grid < 280))]
wl_grid_high = wl_grid[np.where((wl_grid > 280) &(wl_grid < 700))]



ax[1,0].plot(wl_grid_low,-k_bef_function(wl_grid_low),'r')
ax[1,0].set_xlabel('Wavelength␣(nm)')
ax[1,0].set_ylabel('Kappa')
title = 'First␣Region␣Kappa␣210-280'
ax[1,0].set_title(title)


ax[1,1].plot(wl_grid_low,-k_aft_function(wl_grid_low),'b')
ax[1,1].set_xlabel('Wavelength␣(nm)')
ax[1,1].set_ylabel('Kappa')
title = 'Second␣Region␣Kappa␣210-280'
ax[1,1].set_title(title)
```

```
ax[2,0].plot(wl_grid_high,-k_bef_function(wl_grid_high),'r')
ax[2,0].plot()
ax[2,0].set_xlabel('Wavelength␣(nm)')
ax[2,0].set_ylabel('Kappa')
title = 'First␣Region␣Kappa␣280-700'
ax[2,0].set_title(title)


ax[2,1].plot(wl_grid_high,-k_aft_function(wl_grid_high),'b')
ax[2,1].set_xlabel('Wavelength␣(nm)')
ax[2,1].set_ylabel('Kappa')
title = 'Second␣Region␣Kappa␣280-700'
ax[2,1].set_title(title)
fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/two_stage_kappas.png'
fig.tight_layout()
fig.set_figwidth(10)
fig.set_figheight(15)
fig.savefig(fname)
plt.show()
```

```
all_scans = load_all_scans(channel)
uniq_days = np.unique(all_scans['Day'].values)
for i in uniq_days:
    scan = all_scans.iloc[np.where(all_scans['Day'].values == i)]
    scan = scan.loc[(scan['Wavelengths'] >= 201)]
    wavelengths = scan['Wavelengths'].values
    exp_time = scan['Exposure'].values

    if max(exp_time) > exposure_split:
        kappas = k_aft_function(wavelengths)
    else:
        kappas = k_bef_function(wavelengths)

    scans = scan['scan'].values
    obs_id = scan['obs_id'].values
    sirr = scan['SIRR'].values
    denom = np.exp(kappas*exp_time)
    #print('denominator',denom)
    corr_sirr = sirr / denom
    day = np.repeat(i,len(corr_sirr))
    d = {'SIRR': corr_sirr, 'Wavelengths': wavelengths, 'Day': day, 'Exposure'
        : exp_time, 'scan': scans, 'obs_id': obs_id}
    data = pandas.DataFrame(data=d)
    corr_dfs.append(data)
corrected_scans = pandas.concat(corr_dfs,ignore_index=True)
```

```
    return corrected_scans
```

A.25 kappa_correction

```
def kappa_correction(channel):
    """
    This function runs the overall mission kappa analysis for every wavelength,
        generates a kappa curve
    and then applies corrections to the entire set of spectra.

    Parameters
    ----------

    channel : int
        1 or 2

    Returns
    -------
    A dataframe with corrected data
    """

    plot = 0
```

```
corr_dfs = []
kaps = []
wavs = []
if plot == 0:
    result1 = plot_kappas(1)
    kaps += result1[0]
    wavs += result1[1]
    result2 = plot_kappas(2)
    kaps += result2[0]
    wavs += result2[1]
    result3 = plot_kappas(3)
    kaps += result3[0]
    wavs += result3[1]
    result4 = plot_kappas(4)
    kaps += result4[0]
    wavs += result4[1]
    result5 = plot_kappas(5)
    kaps += result5[0]
    wavs += result5[1]
if plot == 1:
    result1 = get_kappas(1)
    kaps += result1[0]
    wavs += result1[1]
    result2 = get_kappas(2)
    kaps += result2[0]
    wavs += result2[1]
    result3 = get_kappas(3)
    kaps += result3[0]

    wavs += result3[1]

    result4 = get_kappas(4)

    kaps += result4[0]

    wavs += result4[1]

    result5 = get_kappas(5)

    kaps += result5[0]

    wavs += result5[1]



wavs += [2300,2400]

kaps += [0,0]

print('kappas',kaps)

print('wavelengths',wavs)

kappa_function = interpolate.interp1d(wavs,kaps,kind='cubic')


#plot kappa functions

wl_grid = np.linspace(201,2400,3000)


fig, ax = plt.subplots(3)


ax[0].plot(wl_grid,-kappa_function(wl_grid),'r')

ax[0].set_xlabel('Wavelength␣(nm)')

ax[0].set_ylabel('Kappa')

title = 'Overall␣Mission␣Kappa'

ax[0].set_title(title)
```

```
wl_grid_low = wl_grid[np.where((wl_grid > 210) &(wl_grid < 280))]

wl_grid_high = wl_grid[np.where((wl_grid > 280) &(wl_grid < 700))]




ax[1].plot(wl_grid_low,-kappa_function(wl_grid_low),'r')

ax[1].plot()

ax[1].set_xlabel('Wavelength␣(nm)')

ax[1].set_ylabel('Kappa')

title = 'Overall␣Kappa␣210-280'

ax[1].set_title(title)






ax[2].plot(wl_grid_high,-kappa_function(wl_grid_high),'r')

ax[2].plot()

ax[2].set_xlabel('Wavelength␣(nm)')

ax[2].set_ylabel('Kappa')

title = 'Overall␣Kappa␣280-700'

ax[2].set_title(title)

fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/overall_kappa.png'

fig.tight_layout()

fig.set_figwidth(10)

fig.set_figheight(15)

fig.savefig(fname)
```

```
plt.show()



all_scans = load_all_scans(channel)


uniq_days = np.unique(all_scans['Day'].values)
for i in uniq_days:
    scan = all_scans.iloc[np.where(all_scans['Day'].values == i)]
    scan = scan.loc[(scan['Wavelengths'] >= 201)]
    wavelengths = scan['Wavelengths'].values
    kappas = kappa_function(wavelengths)
    exp_time = scan['Exposure'].values
    scans = scan['scan'].values
    obs_id = scan['obs_id'].values
    sirr = scan['SIRR'].values
    denom = np.exp(kappas*exp_time)
    #print('denominator',denom)
    corr_sirr = sirr / denom
    day = np.repeat(i,len(corr_sirr))
    d = {'SIRR': corr_sirr, 'Wavelengths': wavelengths, 'Day': day, 'Exposure'
        : exp_time, 'scan': scans, 'obs_id': obs_id}
    data = pandas.DataFrame(data=d)
    corr_dfs.append(data)
corrected_scans = pandas.concat(corr_dfs,ignore_index=True)
```

```
    return corrected_scans
```

A.26 plot_total_integrated_spectra

```
def plot_total_integrated_spectra(full_scan,key):
    """
    This function takes in a full set of scans, a key for filtering purposes
    and integrates over the entire wavelength region to view a time series

    Parameters
    ----------

    full_scan : data frame
        This data frame contains an entire set of scans

    Returns
    -------
    A list of two elements, each element being a list. The first is the time in
        julian days and the second is the integrated spectra corresponding to each
         day
    """
    int_spectra = []
    uniq_days = np.unique(full_scan['Day'].values)
    for i in uniq_days:
        indices = np.where(full_scan['Day'].values == i)
        scan = full_scan.iloc[indices]
        sirr = scan['SIRR'].values
        wavelengths = scan['Wavelengths'].values
        if len(wavelengths) > 1000:


            temp = integrate.trapz(sirr,wavelengths)
            #if temp > 1500:
             # print(scan['Wavelengths'].values)
            int_spectra.append(temp)


    int_spectra = np.array(int_spectra)


    if key == 'TSIS':
        filter_indices = np.where(int_spectra > 1308)
        int_spectra = int_spectra[filter_indices]
        uniq_days = uniq_days[filter_indices]
    elif key == 'CH2':
        filter_indices = np.where((int_spectra > 1280) & (int_spectra < 1360))
        int_spectra = int_spectra[filter_indices]
        uniq_days = uniq_days[filter_indices]


    y = int_spectra
    y0 = y[~np.isnan(y)][0]
    def to_percent(y):
        return (y - y0) / y0 * 100
    def from_percent(y):
        return y
    fig, ax = plt.subplots()



    uniq_days = time_convert(uniq_days,'jd','decimalyear')
    ax.plot(uniq_days,int_spectra,'bo')

    ax.set_xlabel('Time␣(Decimal␣Year)')
    ax.set_ylabel('Solar␣Irradiance␣((W/m^2))')
    title = 'Total␣Integrated␣Solar␣Irradiance'
    ax.set_title(title)
    sec_ax = ax.secondary_yaxis('right', functions=(to_percent, from_percent))
    sec_ax.set_ylabel('Percent␣Change')



    plt.show()
    result = [uniq_days,int_spectra]
    return result
```

A.27 plot_all_time_series

```
def plot_all_time_series(wavelength,corr_scans,all_scans,all_scansCH2,tsis_scans,
    model):
    """
    This function takes in all data sets and plots the time series of all given a
        particular wavelength

```

```
Parameters
----------

wavelength : int
    The requested wavelength to plot
corr_scans: dataframe
    This is the dataframe containing all corrected scans (using an overall
        mission kappa)
all_scans: dataframe
    This is the dataframe containing all uncorrected CSIM channel 1 scans
all_scansCH2: dataframe
    This is a dataframe containing all uncorrected CSIM channel 2 scans
tsis_scans: dataframe
    This is a dataframe containing all corrected TSIS Channel 1 Data
model: dataframe
    This is a dataframe containing all model data


Returns
-------
Nothing. Outputs plots to a screen and a folder.
"""

%matplotlib inline
#plots time series given a wavelength
```

```
#First TSIS data


tsis_wavelengths = tsis_scans['Wavelengths'].values

closest_wl = find_nearest(tsis_wavelengths,wavelength)

indices = np.where(tsis_scans['Wavelengths'].values == closest_wl)

filtered_tsis_data = tsis_scans.iloc[indices]

tsis_times = filtered_tsis_data['Day'].values

tsis_times = time_convert(tsis_times,'jd','decimalyear')

tsis_sirr = filtered_tsis_data['SIRR'].values


initial_tsis_filter = np.where(tsis_sirr > 0.01)


tsis_times = tsis_times[initial_tsis_filter]

tsis_sirr = tsis_sirr[initial_tsis_filter]




#model data


model_wavelengths = model['Wavelengths'].values

closest_wl = find_nearest(model_wavelengths,wavelength)

indices = np.where(model['Wavelengths'].values == closest_wl)

filtered_model_data = model.iloc[indices]

model_times = filtered_model_data['Day'].values

model_times = time_convert(model_times,'jd','decimalyear')

model_sirr = filtered_model_data['SIRR'].values
```

```
#Then Channel B Uncorrected Data
csimCH2_uncorr_wavelengths = all_scansCH2['Wavelengths'].values
closest_wl = find_nearest(csimCH2_uncorr_wavelengths,wavelength)
indices = np.where(all_scansCH2['Wavelengths'].values == closest_wl)
filtered_uncorrCH2_data = all_scansCH2.iloc[indices]
csimCH2_uncorr_times = filtered_uncorrCH2_data['Day'].values
csimCH2_uncorr_sirr = filtered_uncorrCH2_data['SIRR'].values
csimCH2_uncorr_times = time_convert(csimCH2_uncorr_times,'jd','decimalyear')

initial_ch2_filter = np.where(csimCH2_uncorr_sirr > 0.01)

csimCH2_uncorr_times = csimCH2_uncorr_times[initial_ch2_filter]
csimCH2_uncorr_sirr = csimCH2_uncorr_sirr[initial_ch2_filter]



#Then Uncorrected Data

csim_uncorr_wavelengths = all_scans['Wavelengths'].values
closest_wl = find_nearest(csim_uncorr_wavelengths,wavelength)
indices = np.where(all_scans['Wavelengths'].values == closest_wl)
filtered_uncorr_data = all_scans.iloc[indices]
csim_uncorr_times = filtered_uncorr_data['Day'].values
csim_uncorr_sirr = filtered_uncorr_data['SIRR'].values
csim_uncorr_times = time_convert(csim_uncorr_times,'jd','decimalyear')
```

```
#Then Corrected Data


csim_corr_wavelengths = corr_scans['Wavelengths'].values

closest_wl = find_nearest(csim_corr_wavelengths,wavelength)

indices = np.where(corr_scans['Wavelengths'].values == closest_wl)

filtered_corr_data = corr_scans.iloc[indices]

csim_corr_times = filtered_corr_data['Day'].values

csim_corr_times = time_convert(csim_corr_times,'jd','decimalyear')

csim_corr_sirr = filtered_corr_data['SIRR'].values






#filter TSIS Times

intersection = np.intersect1d(csim_uncorr_times,tsis_times)


mask = np.isin(tsis_times,intersection)


temp = tsis_sirr[mask]


intersection2 = np.intersect1d(csim_corr_times,tsis_times)

mask2 = np.isin(tsis_times,intersection2)

temp2 = tsis_sirr[mask2]



temp_times = tsis_times[mask]
```

```
indices = np.where(tsis_times > temp_times[0])
tsis_sirr = tsis_sirr[indices]
tsis_times = tsis_times[indices]



diff = np.average(temp[:5]) - np.average(csim_uncorr_sirr[:5])
csim_uncorr_sirr = csim_uncorr_sirr + diff
diff2 = np.average(temp[:5]) - np.average(csimCH2_uncorr_sirr[:5])
csimCH2_uncorr_sirr = csimCH2_uncorr_sirr + diff2

intersection2 = np.intersect1d(csim_corr_times,tsis_times)
mask2 = np.isin(tsis_times,intersection2)



diff3 = np.average(temp2[:5]) - np.average(csim_corr_sirr[:5])
csim_corr_sirr = csim_corr_sirr + diff3




#make sure model starts on same day

intersection = np.intersect1d(model_times,tsis_times)
mask = np.isin(model_times,intersection)
model_times = model_times[mask]
model_sirr = model_sirr[mask]
#offset to match TSIS
```

```
model_diff = tsis_sirr[0] - model_sirr[0]
model_sirr = model_sirr + model_diff


print(’Offset␣of␣’ + str(model_diff) + ’␣applied␣to␣model␣data␣to␣match␣TSIS’)
print(’Offset␣of␣’ + str(diff) + ’␣applied␣to␣uncorrected␣channel␣1␣data␣to␣
    match␣TSIS’)
print(’Offset␣of␣’ + str(diff2) + ’␣applied␣to␣uncorrected␣channel␣2␣data␣to␣
    match␣TSIS’)
print(’Offset␣of␣’ + str(diff3) + ’␣applied␣to␣corrected␣channel␣1␣␣data␣to␣
    match␣TSIS’)








# then filter all for outliers


#Filters Based on Statistical Z Score


tsis_z = np.abs(stats.zscore(tsis_sirr))
csim_uncorr_z = np.abs(stats.zscore(csim_uncorr_sirr))
csim_corr_z = np.abs(stats.zscore(csim_corr_sirr))
csimCH2_uncorr_z = np.abs(stats.zscore(csimCH2_uncorr_sirr))


temp = np.argsort(tsis_z)
temp = tsis_z[temp]
n = 5
```

```
largest = temp[-n:]
#print("{} largest values:".format(n), largest)
temp = np.argsort(csim_uncorr_z)
temp = csim_uncorr_z[temp]
n = 5
largest = temp[-n:]
#print("{} largest values:".format(n), largest)
temp = np.argsort(csim_corr_z)
temp = csim_corr_z[temp]
n = 5
largest = temp[-n:]
#print("{} largest values:".format(n), largest)

threshold = 5
tsis_filter = np.where(tsis_z < threshold)
csim_uncorr_filter = np.where(csim_uncorr_z < threshold)
csim_corr_filter = np.where(csim_corr_z < threshold)
csimCH2_filter = np.where(csimCH2_uncorr_z < threshold)



csim_corr_sirr = csim_corr_sirr[csim_corr_filter]
csim_corr_times = csim_corr_times[csim_corr_filter]

csim_uncorr_sirr = csim_uncorr_sirr[csim_uncorr_filter]
csim_uncorr_times = csim_uncorr_times[csim_uncorr_filter]

tsis_sirr = tsis_sirr[tsis_filter]
```

```
tsis_times = tsis_times[tsis_filter]



csimCH2_uncorr_sirr = csimCH2_uncorr_sirr[csimCH2_filter]

csimCH2_uncorr_times = csimCH2_uncorr_times[csimCH2_filter]




y = csim_uncorr_sirr

y0 = y[~np.isnan(y)][0]

def to_percent(y):

    return (y - y0) / y0 * 100

def from_percent(y):

    return y





fig, ax = plt.subplots()


line5 = ax.plot(model_times,model_sirr,'o',color='purple',label='NRL2␣Model',
    markersize=2)


line = ax.plot(tsis_times,tsis_sirr,'bo',label='TSIS␣Corrected',markersize=2)


line2 = ax.plot(csim_uncorr_times,csim_uncorr_sirr,'ro',label='CSIMA␣
    Uncorrected',markersize=2)
```

```
line3 = ax.plot(csim_corr_times,csim_corr_sirr,'go',label='CSIMA␣Corrected',
    markersize=2)
line4 = ax.plot(csimCH2_uncorr_times,csimCH2_uncorr_sirr,'o',color='orange',
    label='CSIMB␣Uncorrected',markersize=2)




ax.set_xlabel('Time␣(Decimal␣Year)')
ax.set_ylabel('SIRR␣(W/nm*m^2)')

title = 'Time␣Series␣of␣Irradiance␣at␣' + str(wavelength) + 'nm'
ax.set_title(title)

sec_ax = ax.secondary_yaxis('right', functions=(to_percent, from_percent))
sec_ax.set_ylabel('Percent␣Change')
ax.legend()
plt.show()



fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/csimtsis_timeseries'
fname += str(wavelength) + 'nm.png'

fig.savefig(fname)



fig, ax = plt.subplots()
```

```
y = csim_corr_sirr
y0 = y[~np.isnan(y)][0]

ax.plot(model_times,model_sirr,'o',color='purple',label='NRL2␣Model',
    markersize=2)
ax.plot(tsis_times,tsis_sirr,'bo',label='TSIS',markersize=2)
ax.plot(csim_corr_times,csim_corr_sirr,'go',label='CSIMA␣Corrected',markersize
    =2)
ax.plot(csimCH2_uncorr_times,csimCH2_uncorr_sirr,'o',color='orange',label='
    CSIMB␣Uncorrected',markersize=2)



ax.set_xlabel('Time␣(Decimal␣Year)')
ax.set_ylabel('SIRR␣(W/nm*m^2)')

title = 'Time␣Series␣of␣Irradiance␣at␣' + str(wavelength) + 'nm'
ax.set_title(title)
ax.legend()
sec_ax = ax.secondary_yaxis('right', functions=(to_percent, from_percent))
sec_ax.set_ylabel('Percent␣Change')





plt.show()
fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/corr_csimtsis_timeseries'
fname += str(wavelength) + 'nm.png'
```

```
    fig.savefig(fname)

    return
```

A.28 plot_all_time_series_two_stage

```
def plot_all_time_series_two_stage(wavelength,corr_scans,all_scans,all_scansCH2,
    tsis_scans,model):
    """
    This function takes in all data sets and plots the time series of all given a
        particular wavelength
    This particular version is designed for two stage kappa, as offsets have to be
         applied differently

    Parameters
    ----------

    wavelength : int
        The requested wavelength to plot
    corr_scans: dataframe
        This is the dataframe containing all corrected scans (using an two stage
            mission kappa)
    all_scans: dataframe
        This is the dataframe containing all uncorrected CSIM channel 1 scans
    all_scansCH2: dataframe
        This is a dataframe containing all uncorrected CSIM channel 2 scans
```

```
tsis_scans: dataframe

    This is a dataframe containing all corrected TSIS Channel 1 Data

model: dataframe

    This is a dataframe containing all model data



Returns

-------

Nothing. Outputs plots to a screen and a folder.

"""


%matplotlib inline

#plots time series given a wavelength



#First TSIS data


tsis_wavelengths = tsis_scans['Wavelengths'].values

closest_wl = find_nearest(tsis_wavelengths,wavelength)

indices = np.where(tsis_scans['Wavelengths'].values == closest_wl)

filtered_tsis_data = tsis_scans.iloc[indices]

tsis_times = filtered_tsis_data['Day'].values

tsis_times = time_convert(tsis_times,'jd','decimalyear')

tsis_sirr = filtered_tsis_data['SIRR'].values


initial_tsis_filter = np.where(tsis_sirr > 0.01)
```

```
tsis_times = tsis_times[initial_tsis_filter]

tsis_sirr = tsis_sirr[initial_tsis_filter]



#model data


model_wavelengths = model['Wavelengths'].values

closest_wl = find_nearest(model_wavelengths,wavelength)

indices = np.where(model['Wavelengths'].values == closest_wl)

filtered_model_data = model.iloc[indices]

model_times = filtered_model_data['Day'].values

model_times = time_convert(model_times,'jd','decimalyear')

model_sirr = filtered_model_data['SIRR'].values





#Then Channel B Uncorrected Data

csimCH2_uncorr_wavelengths = all_scansCH2['Wavelengths'].values

closest_wl = find_nearest(csimCH2_uncorr_wavelengths,wavelength)

indices = np.where(all_scansCH2['Wavelengths'].values == closest_wl)

filtered_uncorrCH2_data = all_scansCH2.iloc[indices]

csimCH2_uncorr_times = filtered_uncorrCH2_data['Day'].values

csimCH2_uncorr_sirr = filtered_uncorrCH2_data['SIRR'].values

csimCH2_uncorr_times = time_convert(csimCH2_uncorr_times,'jd','decimalyear')


initial_ch2_filter = np.where(csimCH2_uncorr_sirr > 0.01)
```

```
csimCH2_uncorr_times = csimCH2_uncorr_times[initial_ch2_filter]

csimCH2_uncorr_sirr = csimCH2_uncorr_sirr[initial_ch2_filter]



#Then Uncorrected Data


csim_uncorr_wavelengths = all_scans['Wavelengths'].values

closest_wl = find_nearest(csim_uncorr_wavelengths,wavelength)

indices = np.where(all_scans['Wavelengths'].values == closest_wl)

filtered_uncorr_data = all_scans.iloc[indices]

csim_uncorr_times = filtered_uncorr_data['Day'].values

csim_uncorr_sirr = filtered_uncorr_data['SIRR'].values

csim_uncorr_times = time_convert(csim_uncorr_times,'jd','decimalyear')




#Then Corrected Data


csim_corr_wavelengths = corr_scans['Wavelengths'].values

closest_wl = find_nearest(csim_corr_wavelengths,wavelength)

indices = np.where(corr_scans['Wavelengths'].values == closest_wl)

filtered_corr_data = corr_scans.iloc[indices]

csim_corr_times = filtered_corr_data['Day'].values

csim_corr_times = time_convert(csim_corr_times,'jd','decimalyear')

csim_corr_sirr = filtered_corr_data['SIRR'].values

csim_corr_exp = filtered_corr_data['Exposure'].values
```

```
#filter TSIS Times
intersection = np.intersect1d(csim_uncorr_times,tsis_times)


mask = np.isin(tsis_times,intersection)


temp = tsis_sirr[mask]


diff2 = np.average(temp[:5]) - np.average(csimCH2_uncorr_sirr[:5])
csimCH2_uncorr_sirr = csimCH2_uncorr_sirr + diff2


intersection2 = np.intersect1d(csim_corr_times,tsis_times)
mask2 = np.isin(tsis_times,intersection2)
temp2 = tsis_sirr[mask2]



temp_times = tsis_times[mask]
indices = np.where(tsis_times > temp_times[0])
tsis_sirr = tsis_sirr[indices]
tsis_times = tsis_times[indices]



exp_break = np.where(csim_corr_exp > exposure_split)
csim_corr_region2 = csim_corr_sirr[exp_break]
csim_corr_times2 = csim_corr_times[exp_break]
```

```
intersect3 = np.intersect1d(csim_corr_times2,tsis_times)

mask3 = np.isin(tsis_times,intersect3)

temp3 = tsis_sirr[mask3]



diff4 = np.average(temp3[:5]) - np.average(csim_corr_region2[:5])




diff = np.average(temp[:5]) - np.average(csim_uncorr_sirr[:5])

csim_uncorr_sirr = csim_uncorr_sirr + diff



intersection2 = np.intersect1d(csim_corr_times,tsis_times)

mask2 = np.isin(tsis_times,intersection2)



diff3 = np.average(temp2[:5]) - np.average(csim_corr_sirr[:5])




for i in range(len(csim_corr_exp)):

    if csim_corr_exp[i] < exposure_split:

        csim_corr_sirr[i] = csim_corr_sirr[i] + diff3

    else:

        csim_corr_sirr[i] = csim_corr_sirr[i] + diff4
```

```
#make sure model starts on same day


intersection = np.intersect1d(model_times,tsis_times)

mask = np.isin(model_times,intersection)

model_times = model_times[mask]

model_sirr = model_sirr[mask]








#offset to match TSIS


model_diff = tsis_sirr[0] - model_sirr[0]

model_sirr = model_sirr + model_diff


print('Offset␣of␣' + str(model_diff) + '␣applied␣to␣model␣data␣to␣match␣TSIS')


print('Offset␣of␣' + str(diff) + '␣applied␣to␣uncorrected␣channel␣1␣data␣to␣
    match␣TSIS')
print('Offset␣of␣' + str(diff2) + '␣applied␣to␣uncorrected␣channel␣2␣data␣to␣
    match␣TSIS')
print('Offset␣of␣' + str(diff3) + '␣applied␣to␣corrected␣channel␣1␣(region␣1)␣
    data␣to␣match␣TSIS')
print('Offset␣of␣' + str(diff4) + '␣applied␣to␣corrected␣channel␣1␣(region␣2)␣
    data␣to␣match␣TSIS')
```

```
# then filter all for outliers

#Filters Based on Statistical Z Score

tsis_z = np.abs(stats.zscore(tsis_sirr))
csim_uncorr_z = np.abs(stats.zscore(csim_uncorr_sirr))
csim_corr_z = np.abs(stats.zscore(csim_corr_sirr))
csimCH2_uncorr_z = np.abs(stats.zscore(csimCH2_uncorr_sirr))

temp = np.argsort(tsis_z)
temp = tsis_z[temp]
n = 5
largest = temp[-n:]
#print("{} largest values:".format(n), largest)
temp = np.argsort(csim_uncorr_z)
temp = csim_uncorr_z[temp]
n = 5
largest = temp[-n:]
#print("{} largest values:".format(n), largest)
temp = np.argsort(csim_corr_z)
temp = csim_corr_z[temp]
n = 5
```

```
largest = temp[-n:]
#print("{} largest values:".format(n), largest)


threshold = 5
tsis_filter = np.where(tsis_z < threshold)
csim_uncorr_filter = np.where(csim_uncorr_z < threshold)
csim_corr_filter = np.where(csim_corr_z < threshold)
csimCH2_filter = np.where(csimCH2_uncorr_z < threshold)



csim_corr_sirr = csim_corr_sirr[csim_corr_filter]
csim_corr_times = csim_corr_times[csim_corr_filter]


csim_uncorr_sirr = csim_uncorr_sirr[csim_uncorr_filter]
csim_uncorr_times = csim_uncorr_times[csim_uncorr_filter]


tsis_sirr = tsis_sirr[tsis_filter]
tsis_times = tsis_times[tsis_filter]



csimCH2_uncorr_sirr = csimCH2_uncorr_sirr[csimCH2_filter]
csimCH2_uncorr_times = csimCH2_uncorr_times[csimCH2_filter]
```

```
fig, ax = plt.subplots()


y = csim_uncorr_sirr

y0 = y[~np.isnan(y)][0]

def to_percent(y):

    return (y - y0) / y0 * 100

def from_percent(y):

    return y





line5 = ax.plot(model_times,model_sirr,'o',color='purple',label='NRLSSI2',

    markersize=2)

line = ax.plot(tsis_times,tsis_sirr,'bo',label='TSIS␣Corrected',markersize=2)


line2 = ax.plot(csim_uncorr_times,csim_uncorr_sirr,'ro',label='CSIMA␣

    Uncorrected',markersize=2)

line3 = ax.plot(csim_corr_times,csim_corr_sirr,'go',label='CSIMA␣Corrected',

    markersize=2)

line4 = ax.plot(csimCH2_uncorr_times,csimCH2_uncorr_sirr,'o',color='orange',

    label='CSIMB␣Uncorrected',markersize=2)
```

```
ax.set_xlabel('Time␣(Decimal␣Year)')
ax.set_ylabel('SIRR␣(W/nm*m^2)')

title = 'Time␣Series␣of␣Irradiance␣at␣' + str(wavelength) + 'nm'
ax.set_title(title)

sec_ax = ax.secondary_yaxis('right', functions=(to_percent, from_percent))
sec_ax.set_ylabel('Percent␣Change')
ax.legend()
plt.show()



fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/
   csimtsis_timeseries_twostage_'
fname += str(wavelength) + 'nm.png'

fig.savefig(fname)



fig, ax = plt.subplots()





y = csim_corr_sirr
y0 = y[~np.isnan(y)][0]
```

```
ax.plot(model_times,model_sirr,'o',color='purple',label='NRLSSI2',markersize
    =2)
ax.plot(tsis_times,tsis_sirr,'bo',label='TSIS',markersize=2)
ax.plot(csim_corr_times,csim_corr_sirr,'go',label='CSIMA␣Corrected',markersize
    =2)
ax.plot(csimCH2_uncorr_times,csimCH2_uncorr_sirr,'o',color='orange',label='
    CSIMB␣Uncorrected',markersize=2)


ax.set_xlabel('Time␣(Decimal␣Year)')
ax.set_ylabel('SIRR␣(W/nm*m^2)')

title = 'Time␣Series␣of␣Irradiance␣at␣' + str(wavelength) + 'nm'
ax.set_title(title)
ax.legend()
sec_ax = ax.secondary_yaxis('right', functions=(to_percent, from_percent))
sec_ax.set_ylabel('Percent␣Change')




plt.show()
```

```
    fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/
        corr_csimtsis_timeseries_twostage_'
    fname += str(wavelength) + 'nm.png'

    fig.savefig(fname)

    return
```

A.29 integrate_region

```
def integrate_region(start_wl,end_wl,spectra):
    """
    This function integrates a spectra over a particular wavelength range

    Parameters
    ----------

    start_wl : int
        The beginning of the wavelength region
    end_wl: int
        The end of the wavelength region
    spectra: dataframe
        This is the dataframe containing all spectral data for integrating


    Returns
```

```
-------
Returns a list containing two numpy arrays, one with decimalyear time data and
     one with corresponding integrated spectra
"""
int_spectra = []
uniq_days = np.unique(spectra['Day'].values)
for i in uniq_days:
    indices = np.where(spectra['Day'].values == i)
    scan = spectra.iloc[indices]
    wl_indices = np.where((scan['Wavelengths'].values > start_wl) & (scan['
        Wavelengths'].values < end_wl))
    scan = scan.iloc[wl_indices]
    sirr = scan['SIRR'].values
    wavelengths = scan['Wavelengths'].values
    if len(wavelengths) > 2:
        temp = integrate.trapz(sirr,wavelengths)
        int_spectra.append(temp)

int_spectra = np.array(int_spectra)
uniq_days = time_convert(uniq_days,'jd','decimalyear')
result = [uniq_days,int_spectra]
return result
```

A.30 int_regions_comparison_two_stage

```
def int_regions_comparison_two_stage(start_wl,end_wl,corr_scans,all_scans,
    all_scansCH2,tsis_scans,model):
    """
    This function takes in all data sets and plots the time series of an
        integrated region of a given start and end wavelength
    This particular version is designed for two stage kappa, as offsets have to be
         applied differently


    Parameters
    ----------


    start_wl : int
        The requested start of the wavlength range to compare
    end_wl: int
        The requested end of the wavelength range to compare
    corr_scans: dataframe
        This is the dataframe containing all corrected scans (using a two stage
            mission kappa)
    all_scans: dataframe
        This is the dataframe containing all uncorrected CSIM channel 1 scans
    all_scansCH2: dataframe
        This is a dataframe containing all uncorrected CSIM channel 2 scans
    tsis_scans: dataframe
        This is a dataframe containing all corrected TSIS Channel 1 Data
    model: dataframe
        This is a dataframe containing all model data
```

```
Returns
-------
Nothing. Outputs plots to a screen and a folder.
"""
tsis_integrated = integrate_region(start_wl,end_wl,tsis_scans)
model_integrated = integrate_region(start_wl,end_wl,model)
ch1_corr_integrated = integrate_region(start_wl,end_wl,corr_scans)
ch1_uncorr_integrated = integrate_region(start_wl,end_wl,all_scans)
ch2_integrated = integrate_region(start_wl,end_wl,all_scansCH2)

tsis_days = tsis_integrated[0]
int_tsis_spectra = tsis_integrated[1]

ch1_corr_days = ch1_corr_integrated[0]
int_ch1_corr_spectra = ch1_corr_integrated[1]

ch1_uncorr_days = ch1_uncorr_integrated[0]
int_ch1_uncorr_spectra = ch1_uncorr_integrated[1]

ch2_days = ch2_integrated[0]
int_ch2_spectra = ch2_integrated[1]

model_days = model_integrated[0]
int_model_spectra = model_integrated[1]
```

```
intersection = np.intersect1d(ch2_days,model_days)
mask = np.isin(model_days,intersection)

temp_times = model_days[mask]
indices = np.where(model_days > temp_times[0])

int_model_spectra = int_model_spectra[indices]
model_days = model_days[indices]



#offset all to match TSIS TSI

intersection = np.intersect1d(tsis_days,ch2_days)

mask = np.isin(tsis_days,intersection)

temp_times = tsis_days[mask]
indices = np.where(tsis_days > temp_times[0])
int_tsis_spectra = int_tsis_spectra[indices]
tsis_days = tsis_days[indices]
```

```
diff = int_model_spectra[0] - int_tsis_spectra[0]

int_tsis_spectra = int_tsis_spectra + diff



day_break = np.where(ch1_corr_days > day_split)


int_ch1_corr_spectra_region2 = int_ch1_corr_spectra[day_break]

ch1_corr_days2 = ch1_corr_days[day_break]


intersect3 = np.intersect1d(ch1_corr_days2,tsis_days)

mask3 = np.isin(tsis_days,intersect3)

temp3 = int_tsis_spectra[mask3]

diff5 = temp3[0] - np.average(int_ch1_corr_spectra_region2[:5])



diff2 = int_model_spectra[0] - np.average(int_ch1_corr_spectra[:5])
```

```
diff3 = int_model_spectra[0] - int_ch1_uncorr_spectra[0]

int_ch1_uncorr_spectra = int_ch1_uncorr_spectra +diff3



diff4 = int_model_spectra[0] - int_ch2_spectra[0]

int_ch2_spectra = int_ch2_spectra + diff4







for i in range(len(ch1_corr_days)):

    if ch1_corr_days[i] < day_split:

        int_ch1_corr_spectra[i] = int_ch1_corr_spectra[i] + diff2

    else:

        int_ch1_corr_spectra[i] = int_ch1_corr_spectra[i] + diff5





print('Offset␣of␣' + str(diff) + '␣applied␣to␣tsis␣data␣to␣match␣NRLSSI2')
print('Offset␣of␣' + str(diff2) + '␣applied␣to␣corrected␣channel␣1␣data␣region
    ␣one␣to␣match␣NRLSSI2')
print('Offset␣of␣' + str(diff5) + '␣applied␣to␣corrected␣channel␣1␣data␣region
    ␣two␣to␣match␣TSIS␣SIM')
print('Offset␣of␣' + str(diff3) + '␣applied␣to␣uncorrected␣channel␣2␣data␣to␣
    match␣NRLSSI2')
print('Offset␣of␣' + str(diff4) + '␣applied␣to␣uncorrected␣channel␣1␣␣data␣to␣
    match␣NRLSSI2')



#filter tsis data because it is bad
if start_wl == 200:
    tsis_min = 1.25

elif start_wl == 240:
    tsis_min = 18

elif start_wl == 310:
    tsis_min = 275.5

elif start_wl == 500:
    tsis_min = 338.5

elif start_wl == 700:
    tsis_min = 608
else:
    tsis_min = 0
```

```
tsis_filter = np.where(int_tsis_spectra > tsis_min)

tsis_days = tsis_days[tsis_filter]

int_tsis_spectra = int_tsis_spectra[tsis_filter]





#filter channel 2 data because it is bad

if start_wl == 200:

    ch2_min = 0


elif start_wl == 240:

    ch2_min = 0


elif start_wl == 310:

    ch2_min = 275.5


elif start_wl == 500:

    ch2_min = 338.5


elif start_wl == 700:

    ch2_min = 608

else:

    ch2_min = 0
```

```
ch2_filter = np.where(int_ch2_spectra > ch2_min)
ch2_days = ch2_days[ch2_filter]
int_ch2_spectra = int_ch2_spectra[ch2_filter]


y = int_ch1_corr_spectra
y0 = y[~np.isnan(y)][0]
def to_percent(y):
    return (y - y0) / y0 * 100
def from_percent(y):
    return y


fig, ax = plt.subplots(2)
ax[0].plot(model_days,int_model_spectra,'o',color='purple',label='NRLSSI2',
    markersize=2)
ax[0].plot(tsis_days,int_tsis_spectra,'bo',label='TSIS',markersize=2)
ax[0].plot(ch1_corr_days,int_ch1_corr_spectra,'go',label='CSIMA␣Corrected',
    markersize=2)
ax[0].plot(ch1_uncorr_days,int_ch1_uncorr_spectra,'ro',label='CSIMA␣
    Uncorrected',markersize=2)
ax[0].plot(ch2_days,int_ch2_spectra,'o',color='orange',label='CSIMB␣
    Uncorrected',markersize=2)


ax[0].set_xlabel('Time␣(Decimal␣Year)')
```

```
ax[0].set_ylabel('IRR␣(W/m^2)')


title = str(start_wl) + '␣nm␣-␣' + str(end_wl) + '␣nm'
fig.suptitle(title)
ax[0].legend()
ax[0].axvline(x=day_split)
sec_ax = ax[0].secondary_yaxis('right', functions=(to_percent, from_percent))
sec_ax.set_ylabel('Percent␣Change')


y = int_ch1_corr_spectra
y0 = y[~np.isnan(y)][0]



ax[1].plot(model_days,int_model_spectra,'o',color='purple',label='NRLSSI2',
    markersize=2)
ax[1].plot(tsis_days,int_tsis_spectra,'o',color='blue',label='TSIS',markersize
    =2)
ax[1].plot(ch1_corr_days,int_ch1_corr_spectra,'o',color='green',label='CSIMA␣
    Corrected',markersize=2)
ax[1].plot(ch2_days,int_ch2_spectra,'o',color='orange',label='CSIMB␣
    Uncorrected',markersize=2)



ax[1].set_xlabel('Time␣(Decimal␣Year)')
ax[1].set_ylabel('IRR␣(W/m^2)')


sec_ax = ax[1].secondary_yaxis('right', functions=(to_percent, from_percent))
```

```
    sec_ax.set_ylabel('Percent␣Change')
    fig.set_figwidth(8)
    fig.set_figheight(10)
    fig.tight_layout()
    fig.subplots_adjust(top=0.95)
    ax[1].axvline(x=day_split)
    plt.show()
    fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/region_' + str(start_wl)
        + '_' + str(end_wl) + '_two_stage.png'
    fig.savefig(fname)
```

A.31 int_regions_comparison

```
def int_regions_comparison(start_wl,end_wl,corr_scans,all_scans,all_scansCH2,
    tsis_scans,model):
    """
    This function takes in all data sets and plots the time series of an
        integrated region of a given start and end wavelength
    This particular version is designed for the overall mission kappa

    Parameters
    ----------

    start_wl : int
        The requested start of the wavlength range to compare
    end_wl: int
        The requested end of the wavelength range to compare
    corr_scans: dataframe
        This is the dataframe containing all corrected scans (using an overall
           mission kappa)
    all_scans: dataframe
        This is the dataframe containing all uncorrected CSIM channel 1 scans
    all_scansCH2: dataframe
        This is a dataframe containing all uncorrected CSIM channel 2 scans
    tsis_scans: dataframe
        This is a dataframe containing all corrected TSIS Channel 1 Data
    model: dataframe
        This is a dataframe containing all model data



    Returns
    -------
    Nothing. Outputs plots to a screen and a folder.
    """
    tsis_integrated = integrate_region(start_wl,end_wl,tsis_scans)
    model_integrated = integrate_region(start_wl,end_wl,model)
    ch1_corr_integrated = integrate_region(start_wl,end_wl,corr_scans)
    ch1_uncorr_integrated = integrate_region(start_wl,end_wl,all_scans)
    ch2_integrated = integrate_region(start_wl,end_wl,all_scansCH2)

    tsis_days = tsis_integrated[0]
    int_tsis_spectra = tsis_integrated[1]
```

```
ch1_corr_days = ch1_corr_integrated[0]

int_ch1_corr_spectra = ch1_corr_integrated[1]


ch1_uncorr_days = ch1_uncorr_integrated[0]

int_ch1_uncorr_spectra = ch1_uncorr_integrated[1]


ch2_days = ch2_integrated[0]

int_ch2_spectra = ch2_integrated[1]


model_days = model_integrated[0]

int_model_spectra = model_integrated[1]









intersection = np.intersect1d(ch2_days,model_days)

mask = np.isin(model_days,intersection)


temp_times = model_days[mask]

indices = np.where(model_days > temp_times[0])


int_model_spectra = int_model_spectra[indices]

model_days = model_days[indices]
```

```
#offset all to match TSIS TSI

intersection = np.intersect1d(tsis_days,ch2_days)

mask = np.isin(tsis_days,intersection)

temp_times = tsis_days[mask]
indices = np.where(tsis_days > temp_times[0])
int_tsis_spectra = int_tsis_spectra[indices]
tsis_days = tsis_days[indices]





diff = int_model_spectra[0] - int_tsis_spectra[0]
int_tsis_spectra = int_tsis_spectra + diff

diff2 = int_model_spectra[0] - int_ch1_corr_spectra[0]
int_ch1_corr_spectra = int_ch1_corr_spectra + diff2


diff3 = int_model_spectra[0] - int_ch1_uncorr_spectra[0]
```

```
int_ch1_uncorr_spectra = int_ch1_uncorr_spectra +diff3



diff4 = int_model_spectra[0] - int_ch2_spectra[0]

int_ch2_spectra = int_ch2_spectra + diff4






print('Offset␣of␣' + str(diff) + '␣applied␣to␣tsis␣data␣to␣match␣NRLSSI2')

print('Offset␣of␣' + str(diff2) + '␣applied␣to␣corrected␣channel␣1␣data␣to␣
   match␣NRLSSI2')

print('Offset␣of␣' + str(diff3) + '␣applied␣to␣uncorrected␣channel␣2␣data␣to␣
   match␣NRLSSI2')

print('Offset␣of␣' + str(diff4) + '␣applied␣to␣uncorrected␣channel␣1␣␣data␣to␣
   match␣NRLSSI2')



#filter tsis data because it is bad

if start_wl == 200:

    tsis_min = 1.25


elif start_wl == 240:

    tsis_min = 18


elif start_wl == 310:

    tsis_min = 275.5
```

```
elif start_wl == 500:
    tsis_min = 338.5

elif start_wl == 700:
    tsis_min = 608
else:
    tsis_min = 0



tsis_filter = np.where(int_tsis_spectra > tsis_min)
tsis_days = tsis_days[tsis_filter]
int_tsis_spectra = int_tsis_spectra[tsis_filter]





#filter channel 2 data because it is bad
if start_wl == 200:
    ch2_min = 0

elif start_wl == 240:
    ch2_min = 0

elif start_wl == 310:
    ch2_min = 275.5
```

```
elif start_wl == 500:
    ch2_min = 338.5


elif start_wl == 700:
    ch2_min = 608
else:
    ch2_min = 0



ch2_filter = np.where(int_ch2_spectra > ch2_min)
ch2_days = ch2_days[ch2_filter]
int_ch2_spectra = int_ch2_spectra[ch2_filter]



y = int_ch1_corr_spectra
y0 = y[~np.isnan(y)][0]
def to_percent(y):
    return (y - y0) / y0 * 100
def from_percent(y):
    return y



fig, ax = plt.subplots(2)
ax[0].plot(model_days,int_model_spectra,'o',color='purple',label='NRLSSI2',
    markersize=2)
ax[0].plot(tsis_days,int_tsis_spectra,'bo',label='TSIS',markersize=2)
```

```
ax[0].plot(ch1_corr_days,int_ch1_corr_spectra,'go',label='CSIMA␣Corrected',
   markersize=2)
ax[0].plot(ch1_uncorr_days,int_ch1_uncorr_spectra,'ro',label='CSIMA␣
   Uncorrected',markersize=2)
ax[0].plot(ch2_days,int_ch2_spectra,'o',color='orange',label='CSIMB␣
   Uncorrected',markersize=2)

ax[0].set_xlabel('Time␣(Decimal␣Year)')
ax[0].set_ylabel('IRR␣(W/m^2)')

title = str(start_wl) + '␣nm␣-␣' + str(end_wl) + '␣nm'
fig.suptitle(title,size=20)
ax[0].legend()
sec_ax = ax[0].secondary_yaxis('right', functions=(to_percent, from_percent))
sec_ax.set_ylabel('Percent␣Change')



y = int_ch1_corr_spectra
y0 = y[~np.isnan(y)][0]



ax[1].plot(model_days,int_model_spectra,'o',color='purple',label='NRLSSI2',
   markersize=2)
ax[1].plot(tsis_days,int_tsis_spectra,'o',color='blue',label='TSIS',markersize
   =2)
ax[1].plot(ch1_corr_days,int_ch1_corr_spectra,'o',color='green',label='CSIMA␣
   Corrected',markersize=2)
```

```
    ax[1].plot(ch2_days,int_ch2_spectra,'o',color='orange',label='CSIMB␣
        Uncorrected',markersize=2)

    ax[1].set_xlabel('Time␣(Decimal␣Year)')
    ax[1].set_ylabel('IRR␣(W/m^2)')

    sec_ax = ax[1].secondary_yaxis('right', functions=(to_percent, from_percent))
    sec_ax.set_ylabel('Percent␣Change')
    fig.set_figwidth(8)
    fig.set_figheight(10)
    fig.tight_layout()
    fig.subplots_adjust(top=0.95)
    plt.show()

    fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/region_' + str(start_wl)
        + '_' + str(end_wl) + '.png'
    fig.savefig(fname)
```

A.32 plot_all_integrated_spectra

```
def plot_all_integrated_spectra(corr_scans,all_scans,all_scansCH2,tsis_scans,
    tsis_TSI,key):
    """
    This function takes in all data sets and plots the total integrated spectra in
        order to make
    total solar irradiance comparisons
```

```
Parameters
----------



corr_scans: dataframe
    This is the dataframe containing all corrected scans (using an overall
       mission kappa)
all_scans: dataframe
    This is the dataframe containing all uncorrected CSIM channel 1 scans
all_scansCH2: dataframe
    This is a dataframe containing all uncorrected CSIM channel 2 scans
tsis_scans: dataframe
    This is a dataframe containing all corrected TSIS Channel 1 Data
tsis_TSI: dataframe
    This is a dataframe containing all TSIS TIM data
key: string
    Intended to change figure output names depending on which corrected data
       set is being plotted.

Returns
-------
Nothing. Outputs plots to a screen and a folder.
"""
tsis_result = plot_total_integrated_spectra(tsis_scans,'TSIS')
ch1_uncorr_result = plot_total_integrated_spectra(all_scans,'')
ch1_corr_result = plot_total_integrated_spectra(corr_scans,'')
```

```
ch2_result = plot_total_integrated_spectra(all_scansCH2,'CH2')


tsis_days = tsis_result[0]

int_tsis_spectra = tsis_result[1]


ch1_corr_days = ch1_corr_result[0]

int_ch1_corr_spectra = ch1_corr_result[1]


ch1_uncorr_days = ch1_uncorr_result[0]

int_ch1_uncorr_spectra = ch1_uncorr_result[1]


ch2_days = ch2_result[0]

int_ch2_spectra = ch2_result[1]


tsi_days = tsis_TSI['Day'].values

tsi_days = time_convert(tsi_days,'jd','decimalyear')


tsi_irr = tsis_TSI['TSI'].values



intersection = np.intersect1d(ch2_days,tsi_days)

mask = np.isin(tsi_days,intersection)





temp_times = tsi_days[mask]
```

```
indices = np.where(tsi_days > temp_times[0])
tsi_irr = tsi_irr[indices]
tsi_days = tsi_days[indices]




zero_filter = np.where(tsi_irr > 0)
tsi_days = tsi_days[zero_filter]
tsi_irr = tsi_irr[zero_filter]




#offset all to match TSIS TSI

intersection = np.intersect1d(tsis_days,ch2_days)

mask = np.isin(tsis_days,intersection)

temp_times = tsis_days[mask]
indices = np.where(tsis_days > temp_times[0])
int_tsis_spectra = int_tsis_spectra[indices]
tsis_days = tsis_days[indices]
```

```
diff = tsi_irr[0] - int_tsis_spectra[0]
int_tsis_spectra = int_tsis_spectra + diff

diff2 = tsi_irr[0] - np.average(int_ch1_corr_spectra[:5])
int_ch1_corr_spectra = int_ch1_corr_spectra + diff2



diff3 = tsi_irr[0] - np.average(int_ch1_uncorr_spectra[:5])
int_ch1_uncorr_spectra = int_ch1_uncorr_spectra +diff3



diff4 = tsi_irr[0] - np.average(int_ch2_spectra[:5])
int_ch2_spectra = int_ch2_spectra + diff4





print('Offset␣of␣' + str(diff) + '␣applied␣to␣tsis␣data␣to␣match␣TSIS␣TSI')
print('Offset␣of␣' + str(diff2) + '␣applied␣to␣corrected␣channel␣1␣data␣to␣
    match␣TSIS␣TSI')
print('Offset␣of␣' + str(diff3) + '␣applied␣to␣uncorrected␣channel␣2␣data␣to␣
    match␣TSIS␣TSI')
print('Offset␣of␣' + str(diff4) + '␣applied␣to␣uncorrected␣channel␣1␣␣data␣to␣
    match␣TSIS␣TSI')
```

```
y = int_ch1_uncorr_spectra
y0 = y[~np.isnan(y)][0]
def to_percent(y):
    return (y - y0) / y0 * 100
def from_percent(y):
    return y




fig, ax = plt.subplots(2)
title = 'Total␣Solar␣Irradiance␣Time␣Series'
fig.suptitle(title,size=20)
ax[0].plot(tsi_days,tsi_irr,'o',color='purple',label='TSIS␣TSI',markersize=2)
ax[0].plot(tsis_days,int_tsis_spectra,'bo',label='TSIS',markersize=2)
ax[0].plot(ch1_corr_days,int_ch1_corr_spectra,'go',label='CSIMA␣Corrected',
    markersize=2)
ax[0].plot(ch1_uncorr_days,int_ch1_uncorr_spectra,'ro',label='CSIMA␣
    Uncorrected',markersize=2)
ax[0].plot(ch2_days,int_ch2_spectra,'o',color='orange',label='CSIMB␣
    Uncorrected',markersize=2)


ax[0].set_xlabel('Time␣(Decimal␣Year)')
```

```
ax[0].set_ylabel('IRR␣(W/m^2)')


ax[0].legend()


sec_ax = ax[0].secondary_yaxis('right', functions=(to_percent, from_percent))
sec_ax.set_ylabel('Percent␣Change')




y = int_ch1_corr_spectra
y0 = y[~np.isnan(y)][0]


ax[1].plot(tsi_days,tsi_irr,'o',color='purple',label='TSIS␣TSI',markersize=2)
ax[1].plot(tsis_days,int_tsis_spectra,'o',color='blue',label='TSIS',markersize
    =2)
ax[1].plot(ch1_corr_days,int_ch1_corr_spectra,'o',color='green',label='CSIMA␣
    Corrected',markersize=2)
ax[1].plot(ch2_days,int_ch2_spectra,'o',color='orange',label='CSIMB␣
    Uncorrected',markersize=2)



ax[1].set_xlabel('Time␣(Decimal␣Year)')
ax[1].set_ylabel('IRR␣(W/m^2)')


sec_ax = ax[1].secondary_yaxis('right', functions=(to_percent, from_percent))
sec_ax.set_ylabel('Percent␣Change')
```

```
    fig.set_figwidth(8)

    fig.set_figheight(10)

    fig.tight_layout()

    fig.subplots_adjust(top=0.95)

    plt.show()

    fname = '/Users/asm0945/Documents/IDL/CSIM/esr/plots/tsi_time_series_' + key +
        '.png'

    fig.savefig(fname)
```

A.33 Analysis Code

```
#The following is the code I ran to generate the plots and perform a comparison
    analysis


date1 = '2020-07-24'

date2 = '2020-08-02'

solar_var(date1,date2,1)


tsis_scans = load_tsis_data()

model = load_model_data()

tsis_TSI = load_tsis_tsi()

all_scans = load_all_scans(1)

all_scansCH2 = load_all_scans(2)


#global variable to define where split happens

day_split = 2020.0
```

```
times = time_convert(all_scans['Day'].values,'jd','decimalyear')
exposure_split = all_scans.iloc[np.where(times > day_split)].Exposure.min()


corr_scans = kappa_correction(1)
two_stage_corr_scans = two_stage_correction(1)
plot_all_integrated_spectra(corr_scans,all_scans,all_scansCH2,tsis_scans,tsis_TSI
    ,'overall')
plot_all_integrated_spectra(two_stage_corr_scans,all_scans,all_scansCH2,
    tsis_scans,tsis_TSI,'two-stage')


int_regions_comparison(200,240,corr_scans,all_scans,all_scansCH2,tsis_scans,model
    )
int_regions_comparison(240,310,corr_scans,all_scans,all_scansCH2,tsis_scans,model
    )
int_regions_comparison(310,500,corr_scans,all_scans,all_scansCH2,tsis_scans,model
    )
int_regions_comparison(500,700,corr_scans,all_scans,all_scansCH2,tsis_scans,model
    )
int_regions_comparison(700,1800,corr_scans,all_scans,all_scansCH2,tsis_scans,
    model)



int_regions_comparison_two_stage(200,240,two_stage_corr_scans,all_scans,
    all_scansCH2,tsis_scans,model)
int_regions_comparison_two_stage(240,310,two_stage_corr_scans,all_scans,
    all_scansCH2,tsis_scans,model)
```

```
int_regions_comparison_two_stage(310,500,two_stage_corr_scans,all_scans,
    all_scansCH2,tsis_scans,model)
int_regions_comparison_two_stage(500,700,two_stage_corr_scans,all_scans,
    all_scansCH2,tsis_scans,model)
int_regions_comparison_two_stage(700,1800,two_stage_corr_scans,all_scans,
    all_scansCH2,tsis_scans,model)
```