Solar Flare Frequency Analysis and Categorisation

Daniel Vagg

BSc (Hons) 2011

Solar Flare Frequency Analysis and Categorisation

Daniel Vagg

School of Science
Waterford Institute of Technology

Supervisors: Dr Kieran Murphy
Joe Zender (ESA)

Submitted in partial fulfilment of the requirements
for the degree of BSc (Hons) in Physics with Computing
at Waterford Institute of Technology.

April 2011

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of BSc (Hons) in Physics with Computing is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: _____ ID No: 20029140

Date: 18 April, 2011

Acknowledgements

LYRA is a project of the Centre Spatial de Liege, the Physikalisch- Meteorologisches Observatorium Davos and the Royal Observatory of Belgium funded by the Belgian Federal Science Policy Office (BELSPO) and by the Swiss Bundesamt fr Bildung und Wissenschaft.

SWAP is a project of the Centre Spatial de Liege and the Royal Observatory of Belgium funded by the Belgian Federal Science Policy Office (BELSPO).

Contents

List of Figures

List of Tables

Glossary

BELSPO	Belgian Federal Science Policy Office	4, 9
CDS	Coronal Diagnostic Spectrometer	25
CME	Coronal Mass Ejection	11
COI	Cone of Influence	17
CWT	Continuous Wavelet Transform	17
DWT	Discrete Wavelet Transform	17
E-M	electromagnetic	3
ECG	Electrocardiography	13
ESA	European Space Agency	1
EUV	Extreme Ultraviolet	10
FFT	Fast Fourier Transform	13
FITS	Flexible Image Transport System	11
GOES	Geostationary Operational Environmental Satellite	4, 17
IAU	International Astronomical Union	11
LAR	Large Angle Rotation	19
LYRA	Large Yield Radiometer	1, 5
PROBA	Project for On-board Autonomy	5
ROB	Royal Observatory of Belgium, Brussels	13
SAA	South Atlantic Anomaly	20
SIDC	Solar Influences Data Analysis Center	13
SOHO	Solar and Heliospheric Observatory	25
SUMER	Solar Ultraviolet Measurement of Emitted Radiation	25
SWAP	Sun Watcher using Active pixel system detectors and image Processing telescope	5

SXI	Solar X-Ray Imager	17
TRACE	Transition Region and Coronal Explorer	25

Chapter 1

Introduction

The aim of this project is to investigate the categorising of solar flares based on frequency analysis of solar data obtained primarily from the Large Yield Radiometer (LYRA) instrument of the PROBA-2 satellite. The project will be performed in collaboration with the European Space Agency (ESA).

Solar flares are large explosions in the Sun's atmosphere which involve the heating of plasma to tens of millions of degrees and the acceleration of electrons, protons and heavier ions close to the speed of light. Most flares occur in regions near sunspots and are powered by a sudden release of magnetic energy stored in the sun. Solar flares are of significant interest as they have the potential to disrupt communications on Earth and pose radiation hazards to spacecraft and astronauts.

Recent analysis has indicated that there are distinct frequencies embedded within some solar flares. For example, a solar flare which occurred on June 13^{th}, 2010, contained a frequency band between 0.013 Hz and 0.015 Hz [6]. The goal of this project is to examine recorded solar flare data in order to characterise solar flares based on the presence of these embedded frequencies.

Chapter 2

Background Theory

2.1 Introduction

In order to analyse emissions from the Sun, several factors must first be considered; for example, the type of emissions to be analysed, their origin, and how they are measured. The aim of this chapter is to provide an overview of the project, and a brief explanation of each component involved, followed by a more in-depth description of the individual components.

2.1.1 Project Overview

In order to clarify the role of each component of the project, a brief overview of data flow and data analysis is given.

- The PROBA-2 satellite was launched by ESA using the Rockot launch system on November 2nd, 2009. This satellite orbits Earth at an altitude of approximately 750 km, completing one orbit every 100 minutes.
- The LYRA instrument on the PROBA-2 satellite analyses the Sun's emissions in various bands of wavelengths ranging from 1 – 220 nm.
- LYRA data is sent to the Royal Observatory of Belgium when the satellite is in a suitable position to transmit data.
- After the data is received, it is stored and made publicly available in the form of FITS files.
- This data is downloaded and stored locally for analysis.
- Frequency analysis is performed.
- The results of the analysis are stored and later used to categorise flares.

2.2 Components of the Sun

The Sun is composed of several layers; the inner core, the radiative zone, the convective zone, the photosphere, the chromosphere, the transition region, and the corona, as illustrated in figure 2.1. The main source of the Sun's energy is generated in the inner core through fusion

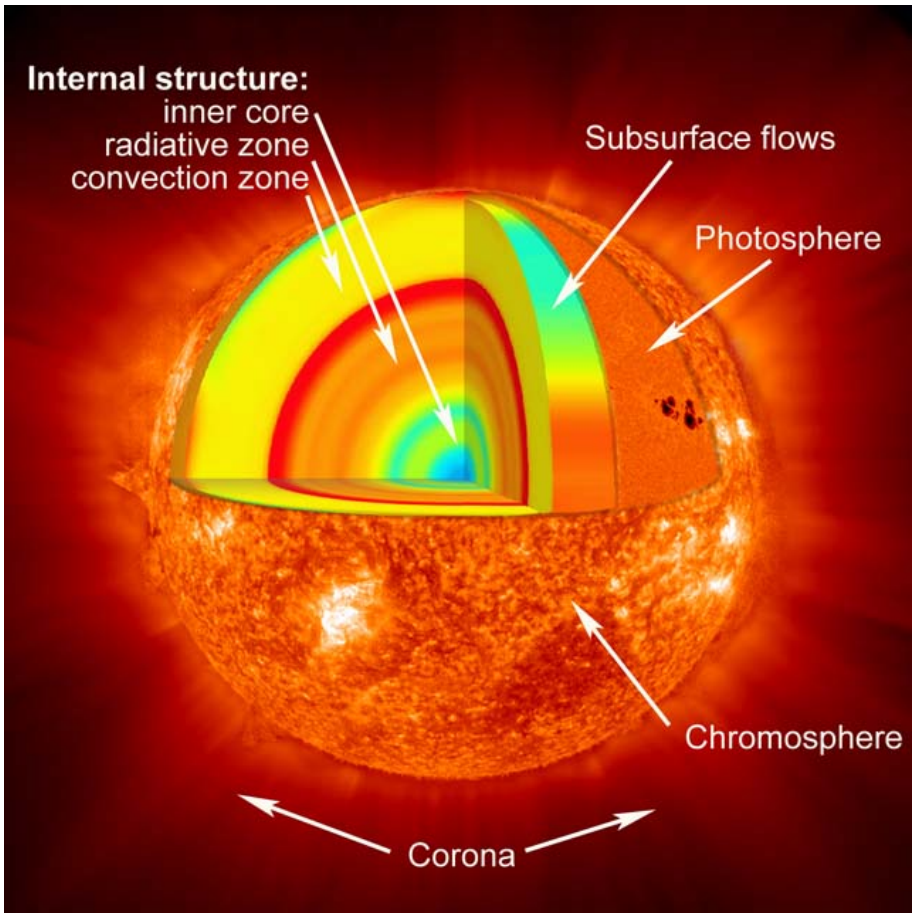


Figure 2.1: A cutaway view of the Sun (source [1]).

reactions where hydrogen nuclei are fused to produce helium. The energy released is emitted in the form of electromagnetic (E-M) radiation and neutrinos.

When fusion takes place in core of the Sun, the resulting E-M radiation travels through a layer of dense hydrogen/helium plasma known as the radiative zone. Atoms within the radiative zone can absorb this radiation, causing them to become excited and emit slightly weaker E-M radiation in a random direction. As this absorption and re-emission is not instantaneous and propagates randomly, it can take E-M radiation the order of hundreds of thousands of years to reach the convective zone after being emitted in the core.

Once the temperature of the interior decreases sufficiently, convection currents are created. The layer of the Sun where convection is the primary method of energy transfer is known as the convective zone. Within this region, convection transmits energy much faster than absorption and re-emission, with columns of gas being recorded as having a velocity of the order of kilometres per second. In the convective zone the temperature decreases from approximately 2×10^6 K to approximately 6×10^3 K.

The photosphere is a relatively thin layer with a thickness of the order of hundreds of kilometres in comparison to the Suns radius of approximately 700×10^3 km; below this layer, the Sun becomes opaque to visible light. At the surface of this layer, 99.5% of light in the visible spectrum emitted from the sun is visible, however, only 4% is visible at a depth of 400 km [9];

the change in opacity is due to a decreasing amount of hydrogen ions in the photosphere, which can absorb visible light easily. The absorption spectra of sunlight indicates that the temperature of this region is approximately 6×10^3 K.

All layers above the photosphere are known collectively as the solar atmosphere [10]. The first layer (i.e. the layer closest to the photosphere) is known as the chromosphere, meaning 'colour sphere'. It is so-called due to its reddish colour caused by the H-alpha spectral line of hydrogen. In this region, the temperature increases from about 6×10^3 K – 20×10^3 K.

The transition region of the Sun lies between the chromosphere and the corona. Similarly to the chromosphere, the temperature in this region increases outward from the core of the Sun; current research indicates that this is caused by the complete ionization of helium. The temperature of the transition region ranges from 20×10^3 K to approximately 1×10^6 K. The solar corona is the final layer of the Sun. This region can extend millions of kilometres into space and has a temperature range of 1×10^6 K – 3×10^6 K. The emissions from this region correspond to highly ionized atoms of iron.

2.2.1 Solar Flares

Solar flares are large explosions in the Sun's atmosphere that can release up to 60 YJ of energy. Research has shown that these eruptions are caused by magnetic reconnection, and affect all layers of the solar atmosphere. Magnetic reconnection occurs when two opposing magnetic fields reconnect and release the energy stored in their respective fields. Magnetic fields of the Sun can connect once they become sufficiently warped due to the differential rotation of the Sun; the sudden release of energy from magnetic reconnection causes solar flares. These magnetic fields can often extend through the convective zone and past the photosphere before they reconnect [11]. If there is a concentrated magnetic flux through the convective zone, the convection will become disrupted leading to a decrease in temperature from approximately 6000 K to 3000 K, producing sunspots. When a solar flare occurs, emissions are produced in multiple regions of the electromagnetic spectrum, including X-rays, gamma rays and radio waves.

Solar flares are classified using their peak flux in a wavelength range of 0.1 nm – 0.8 nm, measured in watts per square meter (W m^{-2}). These measurements are provided by the Geostationary Operational Environmental Satellite (GOES) spacecraft. There are 5 different classes of flare with each class having a peak flux 10 times greater than the last. Within each class there is a linear scale from 1 to 9. The five classes are A, B, C, M, and X. For example, an M2 flare is twice as powerful as an M1 flare, four times more powerful than an C5 flare, and a hundred times greater than a B2 flare.

Figure 2.2: An artist's rendering of rear view of PROBA-2 satellite (source [2]).

2.3 The PROBA-2 Satellite

PROBA-2 is the second satellite in the European Space Agency's series of Project for On-board Autonomy (PROBA) satellites; an artists representation of PROBA-2 is shown in figure 2.2. It is flown in a low earth orbit at about 750 km above the Earth's surface. The satellite has several prototype instruments on-board in their final stage of testing, as well as four scientific instruments [12]. Of the four scientific instruments, two will be used during this project. These two instruments are the Sun Watcher using Active pixel system detectors and image Processing telescope (SWAP) and the LYRA, illustrated in figure 2.3. The primary source of data analysed in this project is gathered from the LYRA instrument. The images from the SWAP instrument can also be used during this project to determine the position of flares.

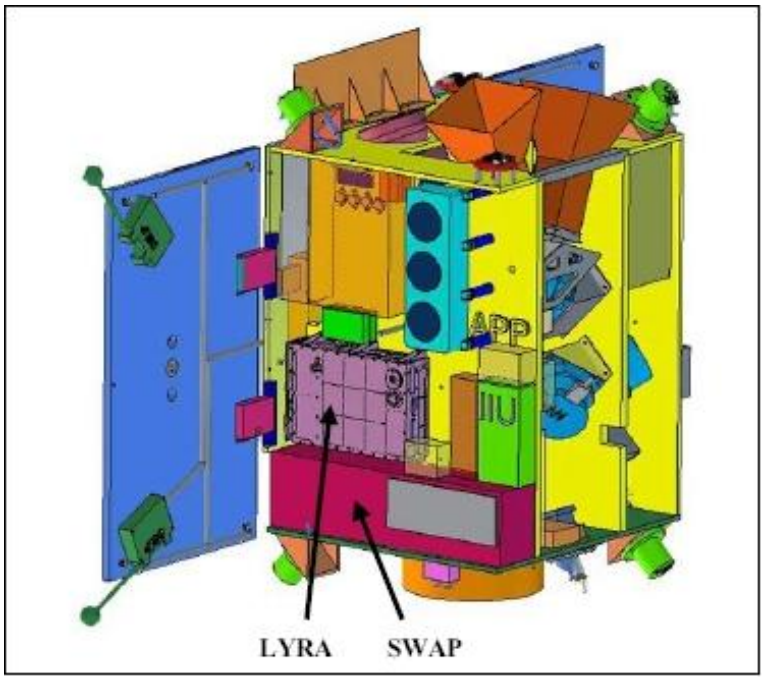


Figure 2.3: Open view of the PROBA-2 payload showing the locations of the SWAP and LYRA instruments (source [2]).

2.4 The Large Yield Radiometer (LYRA)

LYRA is a solar radiometer located on the PROBA-2 satellite as shown in figure 2.3. The instrument is designed to measure emissions of the Sun across 4 different passbands, each corresponding to different features of the Sun [13, 14]. The instrument is composed of 3 units, each of them consisting of the same four channels as illustrated in figure 2.4.

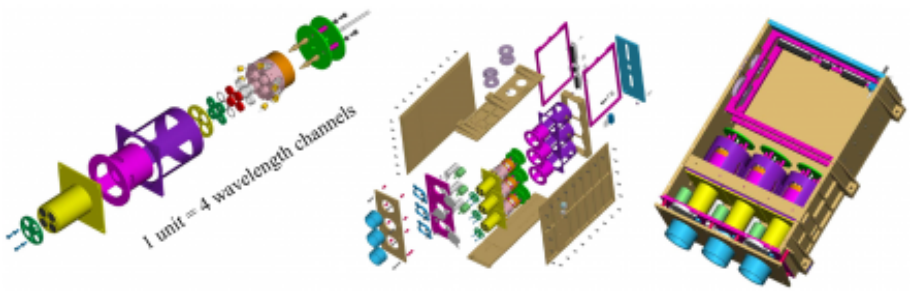


Figure 2.4: An exploded view of the LYRA instrument (source [3]).

The units differ in the technology used for each detector. The nominal unit (Unit 2) uses diamond detectors, the other two units (1 and 3) contain a mix of diamond detectors and silicon detectors. Units 1 and 3 are used during calibrations to measure and analyse the performance of the detectors over time. The LYRA instrument can acquire data simultaneously with one or two units at a configurable frequency between 0.1 Hz and 100 Hz. Each detector contains

Channel Number	Channel Name	Wavelength range (nm)	Solar feature
1	Lyman-Alpha	120 – 123	Light emitted from hydrogen as it falls from the $n = 2$ energy level to the $n = 1$ energy level
2	Herzberg	200 – 220	Upper solar photosphere emissions
3	Aluminium	17 – 80 1 – 8 (during flares)	Extreme UV radiation (and X-rays)
4	Zirconium	6 – 20 1 – 6 (during flares)	Extreme UV radiation (and X-rays)

Table 2.1: Details of channels within each of the three units in the LYRA instrument [4].

two LEDs, of wavelength 375 nm and 470 nm, inserted between the filter and detector. These are used to assist in detector calibration [4]. Calibration of the data is discussed further in the LYRA analysis manual [15].

The different passbands that can be measured in each channel and their corresponding solar features, are summarised in table 2.1 [4]. The spectral transmittance for each channel is shown in subsection 2.4.1.

2.4.1 Spectral Transmittance

Figures 2.5 – 2.8 show the spectral transmittance of each channel in units 1 to 3. Each plot shows the same channel from all three units; for example, figure 2.5 shows the Lyman-alpha channel in units 1 to 3, labelled as 'Channel 1-1', 'Channel 2-1', 'Channel 3-1' respectively [4].

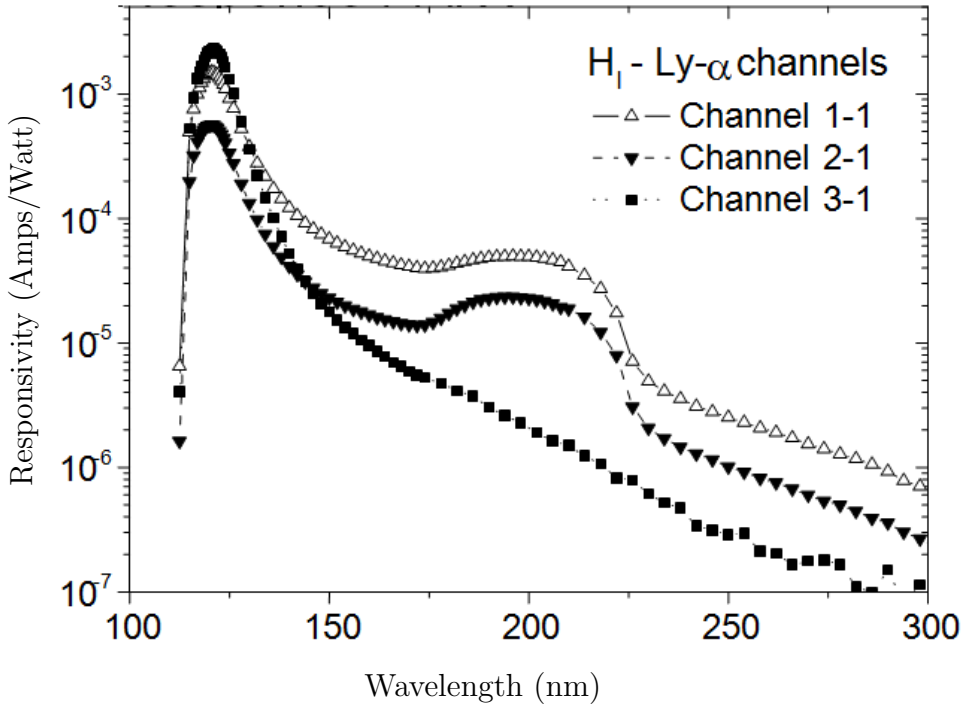


Figure 2.5: Absolute responsivity of the Lyman-Alpha channels of LYRA for wavelengths between 100 nm and 300 nm (source [4]).

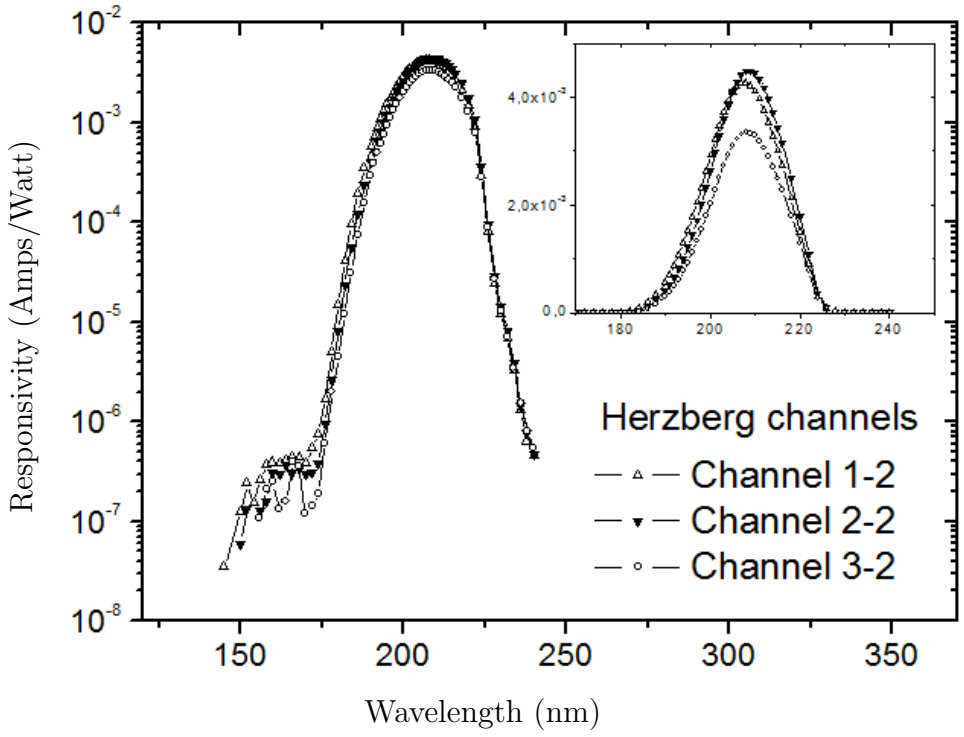


Figure 2.6: Absolute responsivity of the Herzberg channels of LYRA for wavelengths between 150 nm and 250 nm (source [4]).

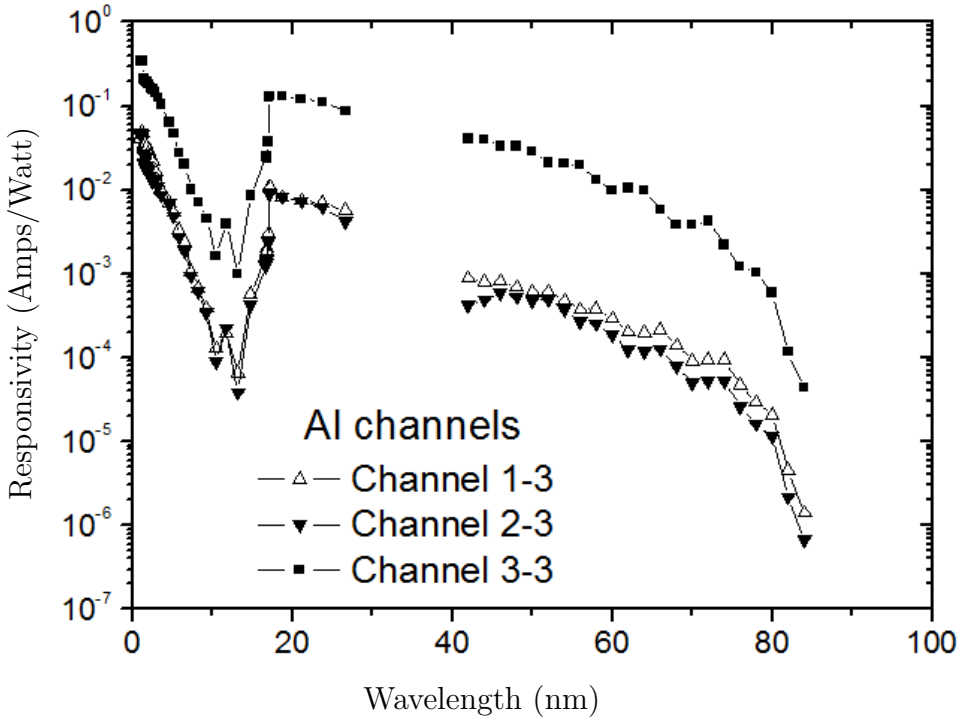


Figure 2.7: Absolute responsivity of the Aluminium channels of LYRA for wavelengths between 1 nm and 84 nm. Data omitted between 30 – 40 nm as the data was compromised by a measurement problem (source [4]).

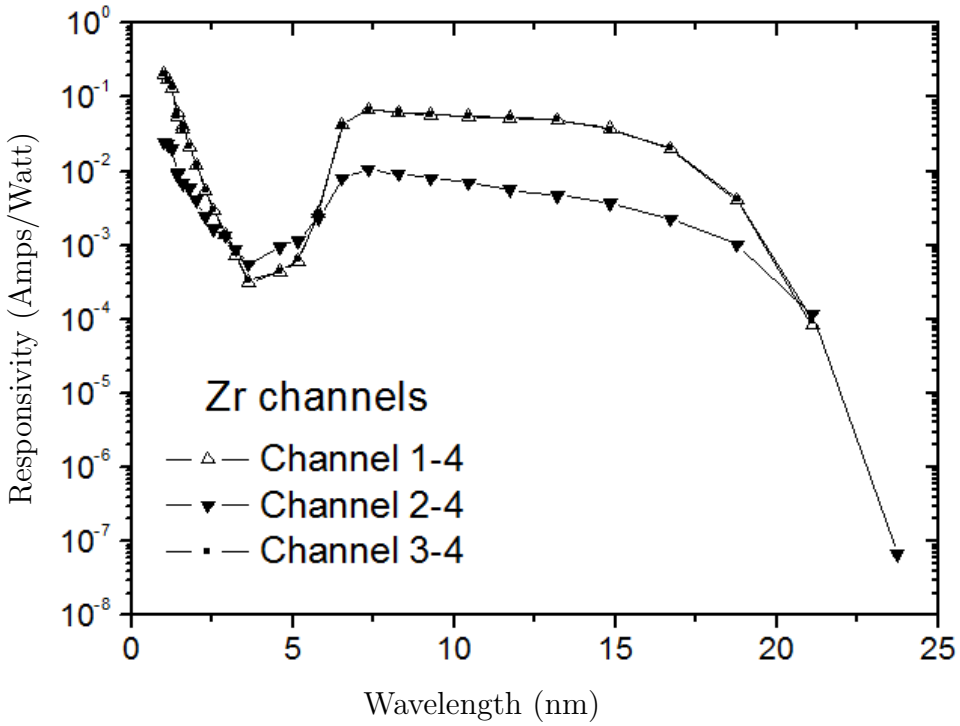


Figure 2.8: Absolute responsivity of the Zirconium channels of LYRA for wavelengths between 1 nm and 25 nm (source [4]).

LYRA is a project of the Centre Spatial de Liege, the Physikalisch- Meteorologisches Observatorium Davos and the Royal Observatory of Belgium funded by the BELSPO and by the

Swiss Bundesamt fr Bildung und Wissenschaft.

2.5 The SWAP Telescope

The second solar instrument on PROBA-2, the SWAP telescope, is a solar telescope in the Extreme Ultraviolet (EUV) range. SWAP provides images of the sun at a frequency of up to 1 minute with a band-pass filter centred on 17.5 nm [16, 17].

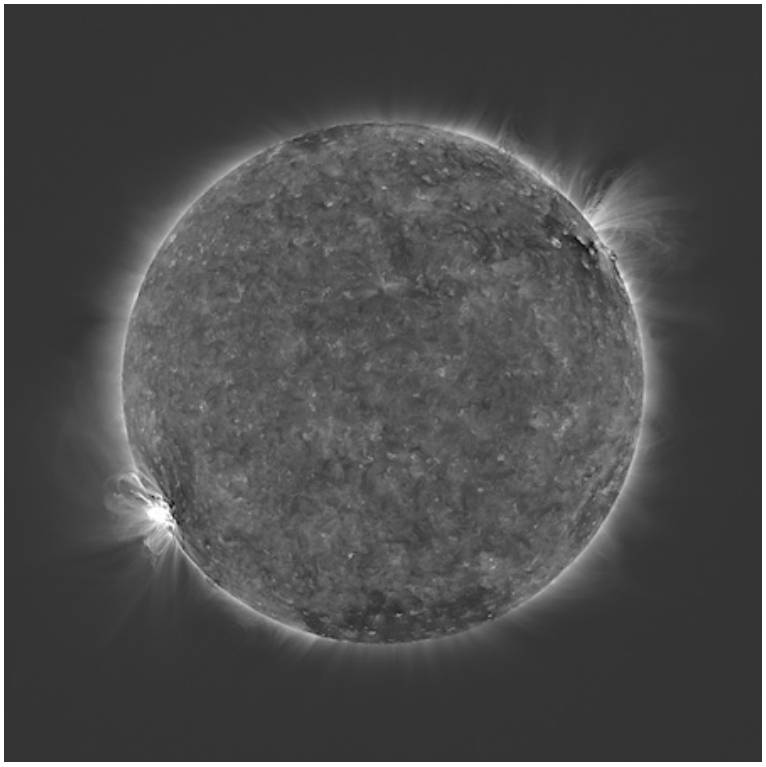


Figure 2.9: A sample image from the SWAP telescope (source [5]).

Images provided by the SWAP instrument can be used during analysis to provide information not available in LYRA data, such as the locations of flares. Similar image archives of the Sun are available online from several other satellites, but images showing the view of the Sun from the exact perspective of the LYRA instrument may prove useful. A sample image from the SWAP instrument is shown in figure 2.9.

The bandpass filter used by SWAP, centred at 17.5 nm, detects some of the wavelengths emitted by iron ions in the solar corona. These ions correspond to temperatures at approximately 1×10^6 K. Table 2.2 shows the emissions that are detected by SWAP and their estimated contributions to solar features. In table 2.2, 'Quiet Sun' refers to regions of the Sun with minimal solar activity, 'Active Region' refers to a region of the Sun where magnetic fields emerge through the photosphere, 'Coronal Hole' refers to a low temperature region of the Corona, and 'Solar Flare' refers to emissions from solar flares.

Ion	$\ln(T_{max})$	Quiet Sun (%)	Active Region (%)	Coronal Hole (%)	Solar Flare (%)
O_{VI}	5.5	0.7	0.5	4.7	2.6
Fe_{IIX}	5.8	0.5	0.3	1.6	0.5
Fe_{IX}	5.9	38.2	31.6	57.5	26.0
Fe_{X}	6.0	51.4	54.6	31.8	35.1
Fe_{XI}	6.1	8.9	11.5	2.3	7.0
Fe_{XII}	6.2	0.3	0.4	0.0	0.3
Fe_{XX}	7.0	0.0	0.0	0.0	14.3

Table 2.2: Preliminary estimation of the relative contributions of various spectral lines to solar features based on CHIANTI simulations (source [8]).

The SWAP instrument is also capable of a degree of autonomy. One of the autonomous features of SWAP is Coronal Mass Ejection (CME) tracking where the PROBA-2 satellite is off-pointed to allow the SWAP instrument to track CMEs [8]. This off-pointing affects the measurements from the LYRA instrument, causing difficulties in data analysis. This is described further in subsection 2.11.2.

SWAP is a project of the Centre Spatial de Liege and the Royal Observatory of Belgium funded by the Belgian Federal Science Policy Office (BELSPO).

2.6 Flexible Image Transport System (FITS) Format

The Flexible Image Transport System (FITS) format is a digital file format used to store, transmit, and manipulate scientific data and images. The format was first standardised in 1981 [18] for the transfer of images and consisted of a binary array preceded by an ASCII text header describing the organisation of the array. It is endorsed by NASA and the International Astronomical Union (IAU) (since 1982) [19]. Since its initial development, it has been revised and extended regularly to accommodate advances in technology, while maintaining full backwards compatibility to preserve the integrity of existing data archives. The FITS format is now the most commonly used digital file format in astronomy, and is currently capable of storing multi-dimensional scientific data sets (1-D spectra, 2-D images, or 3-D data cubes) while including many provisions for describing calibration information in addition to metadata. The FITS format is used by both the LYRA and SWAP instruments to store data.

2.6.1 Structure of FITS files generated by SWAP and LYRA

On a single day of normal operation, the LYRA instrument can produce a 'standard' FITS file of ~ 70 Mb. There may also be several smaller FITS files produced each day corresponding to calibrations and metadata. These are downloaded to the Royal Observatory of Belgium (ROB)

when the satellite is in a suitable position to maintain a downlink. Once downloaded, they are then processed at ROB [18] and made publicly available.

There are different sets of LYRA data available corresponding to different levels of processing. Level 1 FITS files consist of daily metadata and uncalibrated LYRA data. Level 2 FITS files consist of daily calibrated LYRA data. Level 3 FITS files consist of daily calibrated LYRA data averaged over 1 minute. The different types of FITS files that are created in each level are briefly explained below along with their filename patterns:

- Meta Data:
  This data contains metadata from the instrument.
  - Meta Data: "lyra_yyyymmdd-hhmmss_levX_**met.fits**"
- Primary Data:
  This data will be taken from one unit in particular (Unit 2).
  - Standard Data: "lyra_yyyymmdd-hhmmss_levX_**std.fits**"
    Contains recorded data from the instrument taken during normal operations.
  - Calibration Data: "lyra_yyyymmdd-hhmmss_levX_**cal.fits**"
    Contains recorded data from the instrument during calibration campaigns.
  - Rejected Data: "lyra_yyyymmdd-hhmmss_levX_**rej.fits**"
    Contains erroneous data.
- Secondary Data:
  This data will be taken from either of the other two units (Unit 1 or 3).
  - Standard Data: "lyra_yyyymmdd-hhmmss_levX_**bst.fits**"
    Contains recorded data from the instrument taken during normal operations.
  - Calibration Data: "lyra_yyyymmdd-hhmmss_levX_**bca.fits**"
    Contains recorded data from the instrument during calibration campaigns.
  - Rejected Data: "lyra_yyyymmdd-hhmmss_levX_**bre.fits**"
    Contains erroneous data.

For example, a sample FITS file corresponding to uncalibrated, standard data, observed from the beginning of May 1st, 2010 would have a filename "lyra_20100501-000000_lev1_std.fits". As LYRA can sample from multiple channels at the same time, there may be both primary and secondary FITS files produced during a single day of data.

During nominal operations, the SWAP instrument images the Sun at a cadence of up to 1 minute. Each image is stored as a FITS file, with a header describing the image. Similarly to LYRA data, there are different sets of SWAP data corresponding to different levels of processing. Level 0 SWAP data contains all information that is required to process any particular image. Level 1 SWAP data contains processed images corrected for all known instrumental defects, and is the most used set of scientific data from SWAP [20].

2.7 The Royal Observatory of Belgium

The Solar Influences Data Analysis Center (SIDC) within the Royal Observatory of Belgium, Brussels (ROB) processes and stores the data collected from the PROBA-2 satellite. Within ROB, there is a team dedicated to the processing and analysis of LYRA and SWAP data. The principle investigators of both teams are M.Dominique and D.Berghmans, respectively [21, 22].

2.8 Frequency Analysis

Frequency analysis is used in many branches of physics to help distinguish the frequency components of signals and has a huge range of applications. Such applications of frequency analysis include Electrocardiography (ECG) which is used to monitor electrical signals controlling the heart. Frequency analysis typically involves transforming data from the time-domain to the frequency-domain, where the frequency components of the signal can be distinguished more clearly. Two methods of frequency analysis are described in more detail in subsections 2.8.1 and 2.8.2, but an introduction is given here to explain the motivation behind frequency analysis of flares.

When solar flares occur, there is a large increase in amplitude of certain emissions. In the LYRA instrument, the Aluminium and Zirconium channels show a distinct increase in amplitude due to the increase of these emissions [4]. Previous frequency analysis has suggested that these two channels detect distinct oscillatory components within some flares. These components are difficult to detect due to the large amplitude of solar flares in comparison to the amplitude of these components.

Frequency analysis will be performed throughout the data to detect where specified frequencies exist — these frequencies and the motivation behind choosing them in particular will be discussed further in chapter 3. The frequencies will be analysed using windowed Fast Fourier Transforms (FFT) and wavelet transforms.

2.8.1 Windowed FFT

Signals are usually shown in the time-domain but it is often useful to view the signal in the frequency domain. The Fourier transform is used to convert a signal from the time-domain to the frequency-domain as demonstrated in figure 2.10.

The Fourier transform allows signals to be transferred back and forth between time and frequency domains. However, the signal can only be viewed in one domain at a time. In other words, the signal can not contain information about where frequency components occur in time while being viewed in the frequency domain. This means, that if a signal had multiple frequency components which existed one after the other, rather than at the same time, the corresponding signal in the frequency domain would not show this information.

There are numerous issues associated with using FFT to detect frequencies in a digital

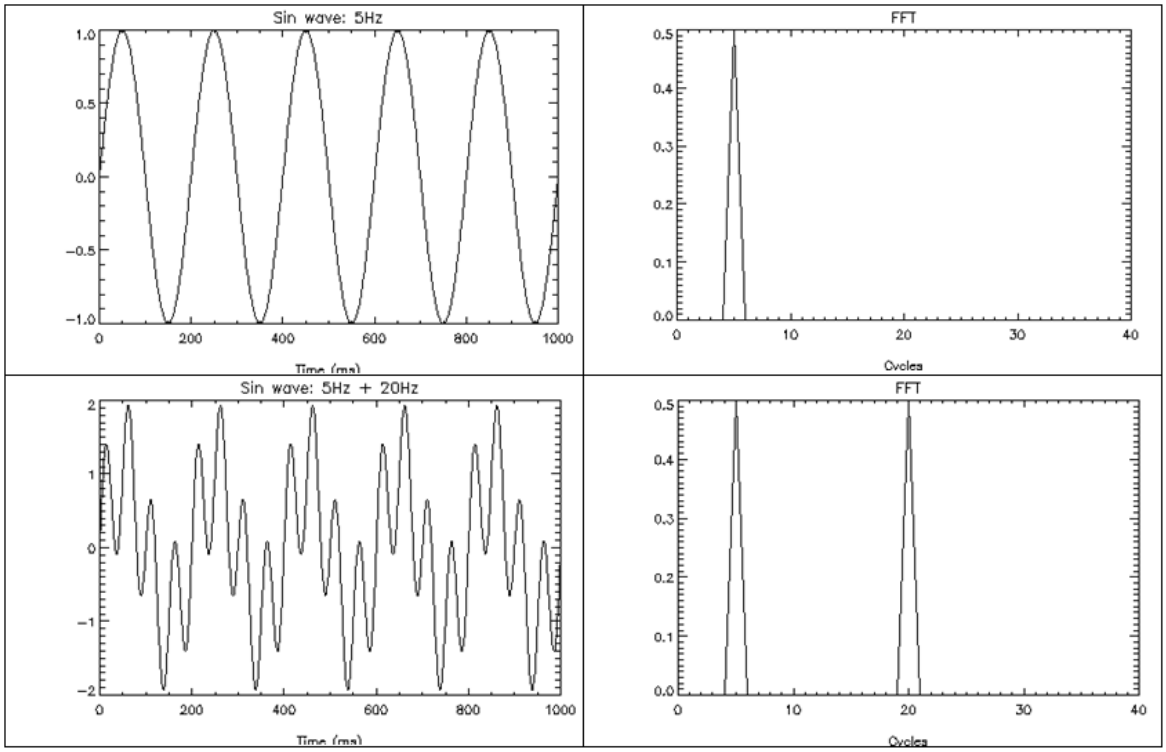


Figure 2.10: Example of applying the Fourier transform to a signal composed of a sine wave at 5 Hz and a signal composed of two sine waves at 5 Hz and 20 Hz. Signals are shown in the time domain and frequency domain, in the left and right panels, respectively.

signal. These include the assumption that the signal is repetitive and the the loss of temporal information. To add an element of temporal information to the results of FFT analysis, smaller samples of the signal can be analysed individually to give the frequency information of that particular subsection. To minimise the affects that are caused by analysing non-repeating data, windowing functions can be used. These windows are zero valued outside of a specified interval. An example of one such window function is the Hann window (often referred to as Hanning window). The Hann window has the advantage of very low aliasing (where the results of FFT analysis become distorted), with the trade-off of having decreased frequency resolution [23]. The decrease in frequency resolution will cause artefacts in signal reconstruction, but this issue is not of relevance here as signal reconstruction is not intended for this research.

2.8.2 Wavelet Analysis

The process of a wavelet transform can be illustrated simply as a time-domain signal which is analysed using a sliding window along the entirety of the signal. The signal is analysed several times with different sized sliding windows. In each window, the signal is analysed using a wavelet (a short-lived wave, beginning and ending with 0), and the transform is computed individually for each segment of the time-domain signal. The original wavelet which is scaled and translated during this process is known as the mother-wavelet.

A signal analysed using wavelet transforms is shown in figure 2.11 – 2.14 to demonstrate the benefits of this type of transform.

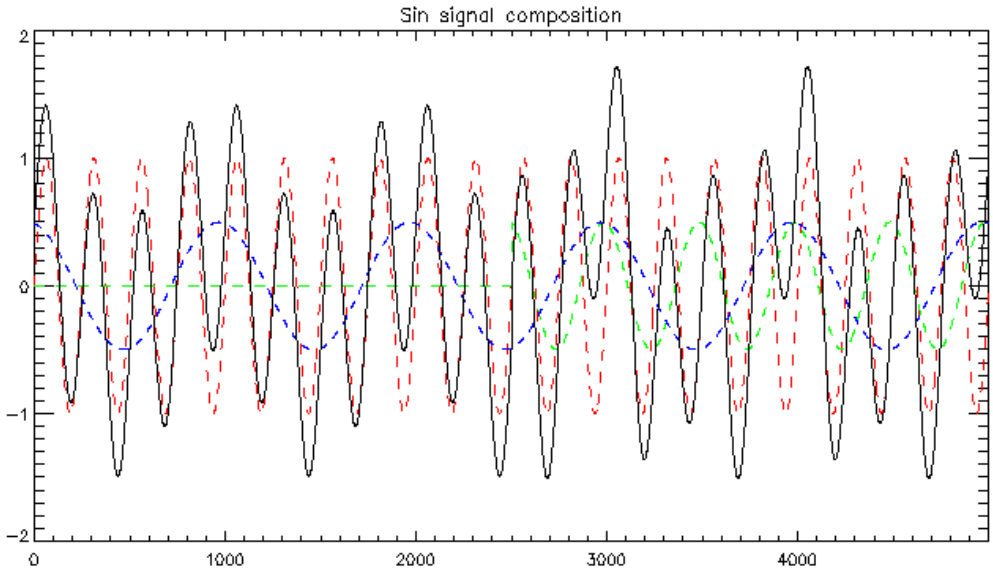


Figure 2.11: Generation of a signal using the sum of 3 different sine waves. The sine wave shown in green is set to 0 for the first half of the signal duration to demonstrate the ability of the wavelet transform to retain temporal information. The final signal is shown in black.

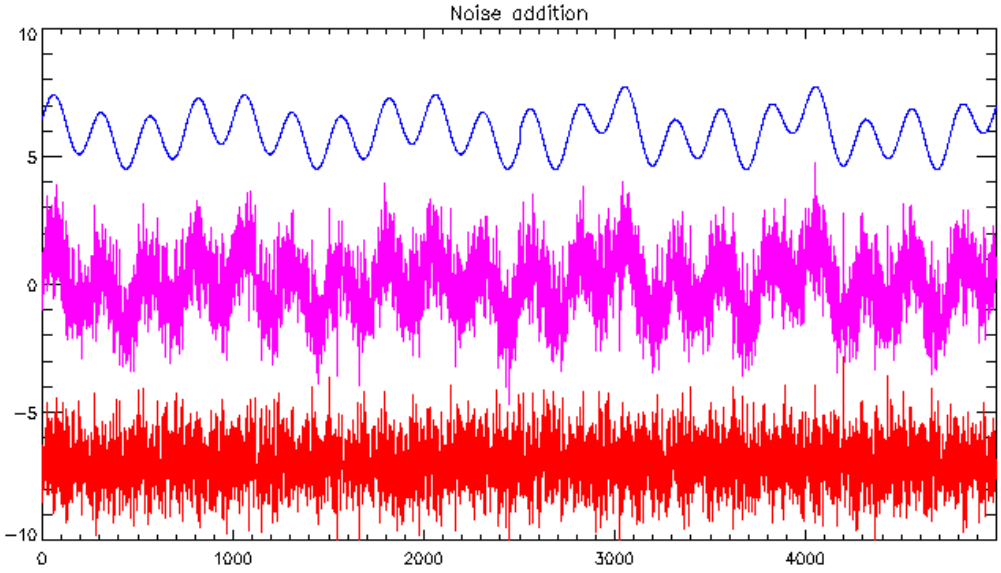


Figure 2.12: Addition of noise (red) to the generated signal (blue) to form the signal that will be analysed (purple).

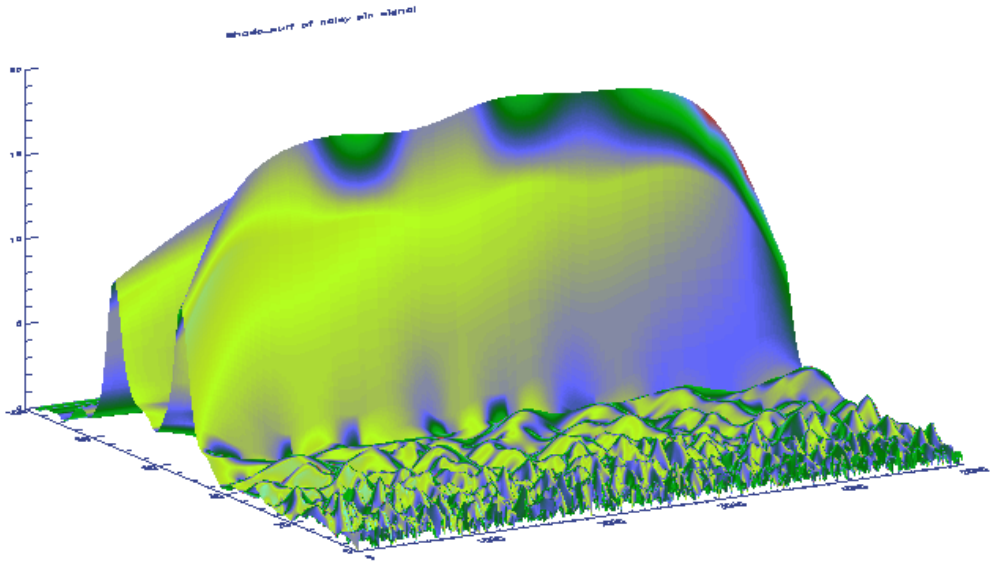


Figure 2.13: 3D surface plot of the wavelet transform of the signal. The result of the wavelet transform is 3-dimensional as it represents the time, wavelet scale (related to frequency), and amplitude.

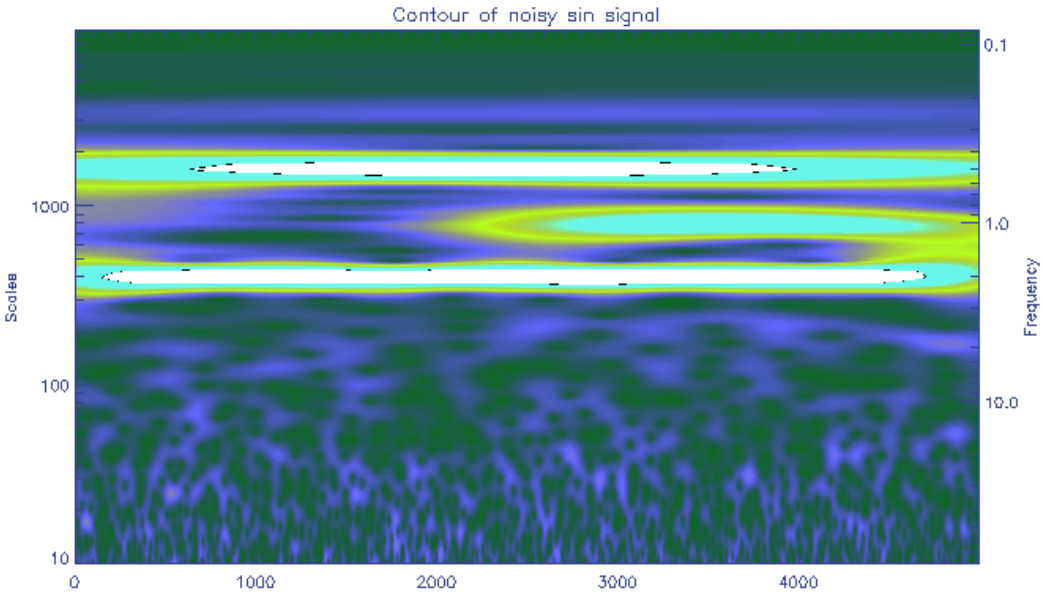


Figure 2.14: A contour plot of the wavelet transform of the signal showing two continuous ridges representing the two continuous sine signals, and a ridge existing for the second half of the duration representing the sine wave that was set to zero for the first half of the duration.

The ridges in the wavelet transform represent a frequency component that exists for the duration of that ridge. If the signal contained a frequency that lasted for half of the full signal time, a ridge would exist only for half of the transform width. In figure 2.14, the 3 ridges correspond to the 3 frequencies in the signal, with the middle ridge existing for only half the

duration of the signal. It can be seen that the amplitudes of the ridges drop off at the edge of the transform. This is caused by the sliding window used in the wavelet transform being partially outside the scope of the signal [24]; this is often referred to as the 'edge effect'.

Wavelet analysis is the primary technique used in this research project to detect frequency components in LYRA data. Wavelet analysis is more useful than the windowed FFT techniques described in subsection 2.8.1 due to the transient nature of the frequencies that are intended to be analysed; however, the resolution of wavelet analysis varies at different multiples, or scales, of the mother-wavelet. The type of wavelet analysis that will be used is the Continuous Wavelet Transform (CWT). Wavelet analysis, as described earlier, uses a mother-wavelet to detect frequency components in signals. As the intended transform is a CWT, the Morlet wavelet can be used as the mother-wavelet. On the other hand, if a Discrete Wavelet Transform (DWT) was used, the Morlet wavelet would not be ideal as it is not orthogonal, and does not satisfy the admissibility condition [25].

The DWT is sometimes more desirable than the CWT as it stores less information in comparison to the CWT while retaining sufficient information to reconstruct the signal. However, the results of the CWT are easier to interpret than the results of the DWT, therefore the CWT is used for analysis in this research project.

2.8.2.1 Cone of Influence (COI)

In CWT analysis, the Cone of Influence (COI) is a region of the wavelet spectrum where edge effects become important. The COI arises from analysing non-cyclic data and is defined as the e-folding time for the autocorrelation of wavelet power at each scale. The e-folding time differs depending on the mother-wavelet used. For example, the e-folding time of the Morlet wavelet is given as $\sqrt{2}s$, where s represents scale [26].

2.9 Flare Detection

As part of analysing LYRA data, flare detection is performed. Flare detection is used to allow frequency analysis to be performed selectively as there is no motivation for performing frequency analysis on quiet data. Two types of flare detection are used in this project; the methods use the SolarSoft archive, and LYRA data, respectively.

The SolarSoft archive, provided by the Lockheed Martin Solar and Astrophysics Laboratory (`http://www.lmsal.com/solarsoft/`), contains a record of solar flares consisting of start/end/peak times and strength. These flares are automatically detected using the GOES satellites containing the Solar X-Ray Imager (SXI) instrument. This automatic detection is detailed in a paper published by S.M. Hill *et al* which discusses the GOES-12 SXI instrument [27]. As the flare detection routines used with the GOES satellites use wavelet transforms of 2-dimensional images of the sun, they shall not be considered further. It is worth noting at this point, that the passband of both the Zirconium and Aluminium channels in the LYRA

instrument can detect most of the wavelengths used in the SXI instrument during a flare as shown in table 2.1.

Flare detection algorithms using LYRA data will be based on exploiting the lag in power response between the Zirconium and Aluminium channels as shown in figure 2.15. Flare detection in the project will be performed using both the SolarSoft archives and the LYRA instrument as the Aluminium and Zirconium channels use a bandpass filter with a range larger than the bandpass used by GOES (1–20 nm compared to 0.6–6 nm). Although emissions of the wavelengths detectable by GOES are expected during flares, it can not be guaranteed that the oscillations intended to be detected in this research project exist exclusively in the bandpass used by the GOES satellites. The details of flares recorded in the SolarSoft archive will also be used to validate the results of flare detection performed using the LYRA instrument.

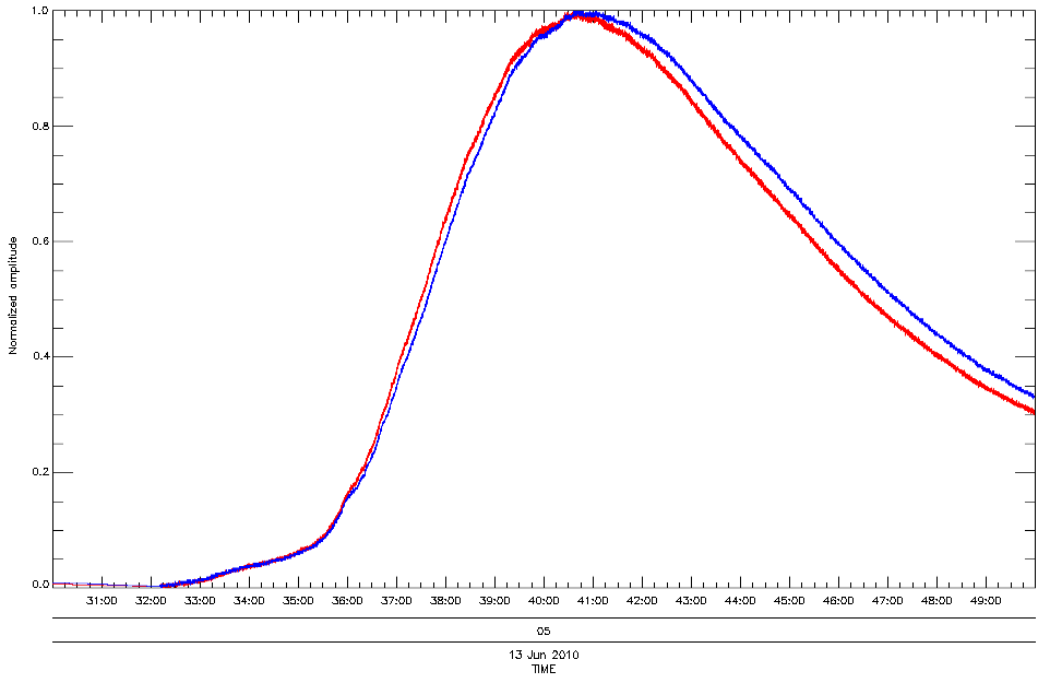


Figure 2.15: The Zirconium (red) and Aluminium (blue) channels of the LYRA instrument during an M-class flare occurring between 05:30 and 05:50 UT on June 13th, 2010. Both channels have been normalised between 0 and 1 to demonstrate the lag between them. The linear section of both channels from 05:30 to 05:32 is due to interpolation over a Large Angle Rotation.

2.10 Flare Categorisation

Flare categorisation will be used to determine the correlation between flare events and frequency components found in LYRA data. To determine this correlation, the LYRA data must first be investigated to ensure that instrumental peculiarities have not affected the detection of frequency components. Issues that may cause such peculiarities are described in section 2.11.

2.11 Anticipated Obstacles

There are a number of issues in analysing LYRA data that are caused by satellite operations, orbits, and instrumental operations. These need to be addressed in order to facilitate analysis. Some of these issues are outlined in the following sections.

2.11.1 Orbit Effects

The PROBA-2 satellite is in helio-synchronous orbit at an average of 757 km above Earth. It takes 100 minutes for the satellite to complete one rotation of the Earth. The orbit of the satellite causes a sinusoidal component to LYRA data with one cycle every 100 minutes. This component is most apparent in the Lyman-Alpha channel (channel 1) as the amplitude of this signal is often larger than any solar effect in this channel. As a result, the data will often have an underlying sinusoidal component due to this effect. For example, in figure 2.16, the data obtained from the Lyman-Alpha channel (channel 1) of unit 2 from approximately 11:30 to 16:30 during May 25th, 2010, demonstrates the sinusoidal component of the signal due to orbit effects. The spikes are caused by Large Angle Rotations (LARs).

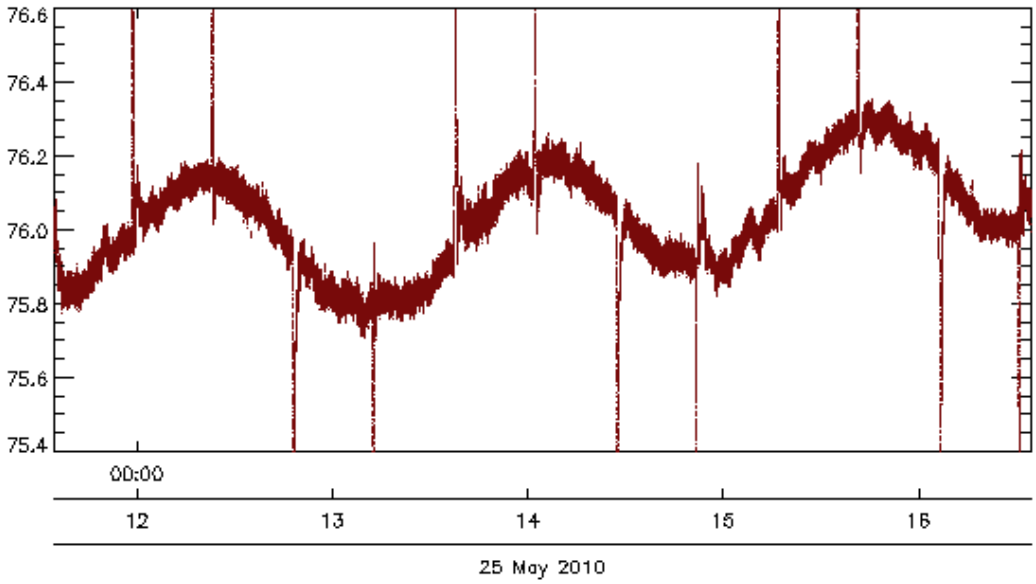


Figure 2.16: The Lyman-Alpha channel (channel 1) of unit 2 in the LYRA instrument demonstrating orbit effects during 300 minutes of operation.

2.11.2 Large Angle Rotation Effects

Large Angle Rotations (LARs) take place every 25 minutes during the satellites orbit. LARs are required to prevent blinding of the star-trackers on PROBA-2 from the reflection of light from Earth. LARs have a significant effect on the data from the LYRA instrument as the

PROBA-2 satellite is pointing at the sun off-axis, which means that a rotation of PROBA-2 requires a combined rotation in all 3 axis, causing the LYRA instrument to point away from the sun by up to 1° for a brief period of time. This event affects all channels of LYRA; the Herzberg channel being most affected followed by the Lyman-Alpha, Zirconium and Aluminium channels. LARs usually cause a sudden decaying oscillation in LYRA data but the behaviour can deviate unpredictably, adding difficulty to analysing data. For example, a plot of the Lyman-Alpha channel from 11:55 to 12:05 during May 25th, 2010 is shown in figure 2.17, demonstrating the effect of a Large Angle Rotation.

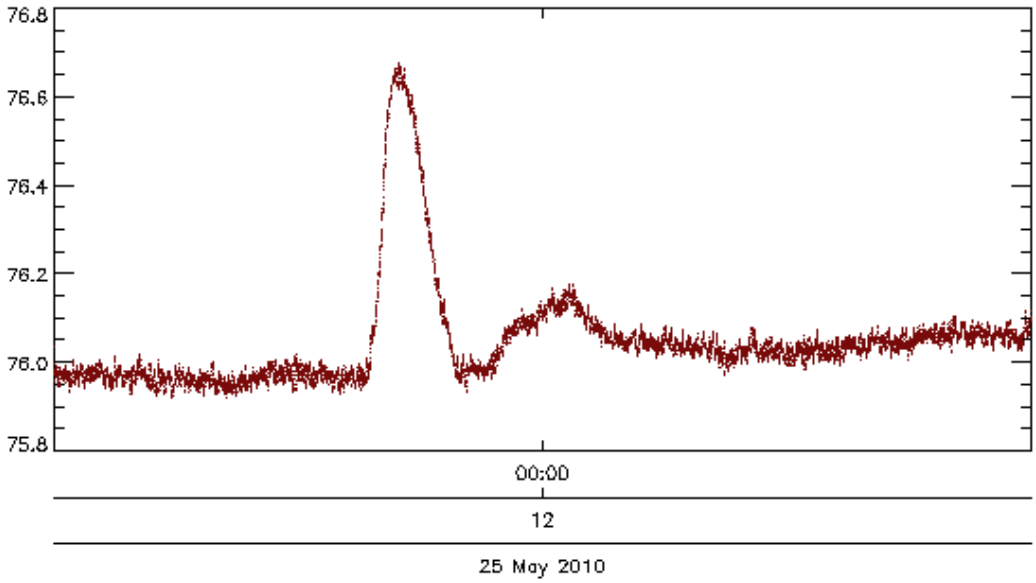


Figure 2.17: The Lyman-Alpha channel (channel 1) of unit 2 of the LYRA instrument during a Large Angle Rotation during 10 minutes of operation.

2.11.3 South Atlantic Anomaly Effects

The South Atlantic Anomaly (SAA) is basically an area over the South Atlantic containing high-energy radiation [30, 31]. The effect of the South Atlantic Anomaly is an increase of noise in LYRA data due to a large concentration of high energy particles in the region. The amplitude of this effect on different channels in LYRA data depends on the channel and the satellites current position. The Lyman-Alpha and Zirconium channels are more severely affected by the SAA than the Herzberg and Aluminium channels. Figure 2.18 shows the four channels of unit 2 in the LYRA instrument scaled between 0 and 1 as the PROBA-2 satellite passes through the SAA. The Herzberg and Aluminium channels (shown in green and blue, respectively) are seen as intermittent lines due to discretisation of the detectors. The brown and pink lines represent data from the Lyman-Alpha and Zirconium channels, respectively.

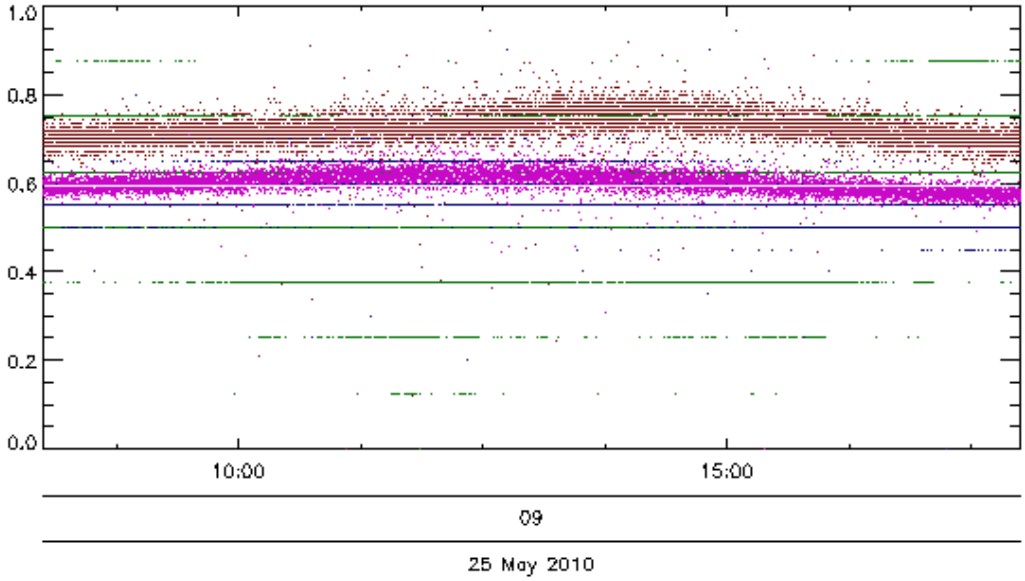


Figure 2.18: Channels 1 to 4 of unit 2 in the LYRA instrument while the PROBA-2 satellite passes through the South Atlantic Anomaly during 10 minutes of operation. Each channel has been normalised between 0 and 1.

2.11.4 Off-Pointing Effects

Off-pointing is where the orientation of the satellite is altered due to CME tracking of the SWAP instrument [13]. This has a dramatic affect on all channels of the LYRA instrument, most notably the Lyman-alpha and Herzberg channels (channels 1 and 2, respectively). Figure 2.19 shows the Herzberg channel (channel 2) from 03:10 to 04:30 on June 12$^{\text{th}}$, 2010. The dramatic changes in amplitude demonstrate the effect of off-pointing on the Herzberg channel.

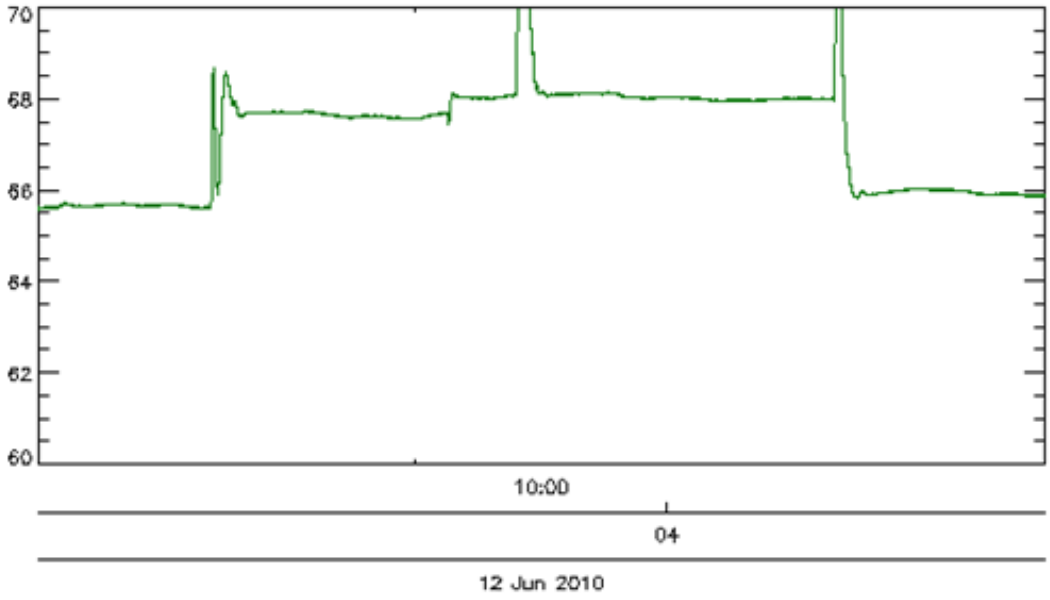


Figure 2.19: The Herzberg channel (channel 2) of unit 2 in the LYRA instrument during off-pointing during 80 minutes of operation.

2.12 Interactive Data Language (IDL)

In order to analyse LYRA data, a suite of software has been developed using the Interactive Data Language (IDL). IDL is used in many fields of science, including astronomy and medicine, due to its ease of use and simplicity. IDL is a dynamically typed language meaning that variables can be changed at run-time which makes development and debugging of software easier [28].

Due to the extensive use of IDL within the astronomy community, a large repository of IDL routines have been developed and maintained by this community. This repository is known as 'The IDL Astronomy User's Library' and contains procedures which are used in this project to access the FITS data generated by the LYRA and SWAP instruments [29].

There are two types of routines in IDL, procedures and functions. Functions return a value when called, whereas procedures do not. Both types of file will be referred to in the implementation chapters of this project.

Chapter 3

Literature Review

3.1 Introduction

In this literature review chapter, research publications relevant to this project are discussed. There are several papers focussed on frequency analysis of solar data, but the emission wavelengths and data sampling rates can differ considerably from the emission wavelengths and sampling rates of LYRA data, reducing their relevance to this project. One paper in particular, by J.J. Zender *et al* [6], will be discussed in more detail as it arose from work carried out during my student placement in ESA (2010), which forms the basis of this project.

3.2 Discussion of the Concept Paper, by J.J. Zender et al

3.2.1 Introduction

This project stemmed from a paper currently in development, 'Temporal and Frequency Variations of Flares Observed by LYRA Onboard Proba2', by J.J. Zender *et al* [6]. This paper focusses on temporal and frequency analysis of an M-class flare that occurred on June 13^{th}, 2010. In the paper, temporal and frequency analysis of the flare is performed using data from the short wavelength channels of the LYRA instrument (Aluminium and Zirconium channels). This analysis will be discussed briefly and compared to the analysis performed in this research project.

3.2.2 Temporal Analysis

J.J. Zender *et al* discuss temporal analysis of the M-class flare occurring during June 13^{th}, 2010, using the SWAP and LYRA instruments on-board PROBA-2 and two sets of data from the GOES satellite. The flare is analysed during a 50 minute window, as a result, some of the data under consideration contains artefacts from LARs. In contrast, the strategy adopted in this research project avoided the effects arising from LARs by only considering windows of data between LARs. It should also be noted that the work reported in this project will be the first iteration over a large data set in order to quantify the correlation between oscillations as described in J.J. Zender *et al* [6] and flare events; as a result, temporal analysis undertaken here will be less complex than that undertaken in the work reported in J.J. Zender *et al* as shown in figure 3.1.

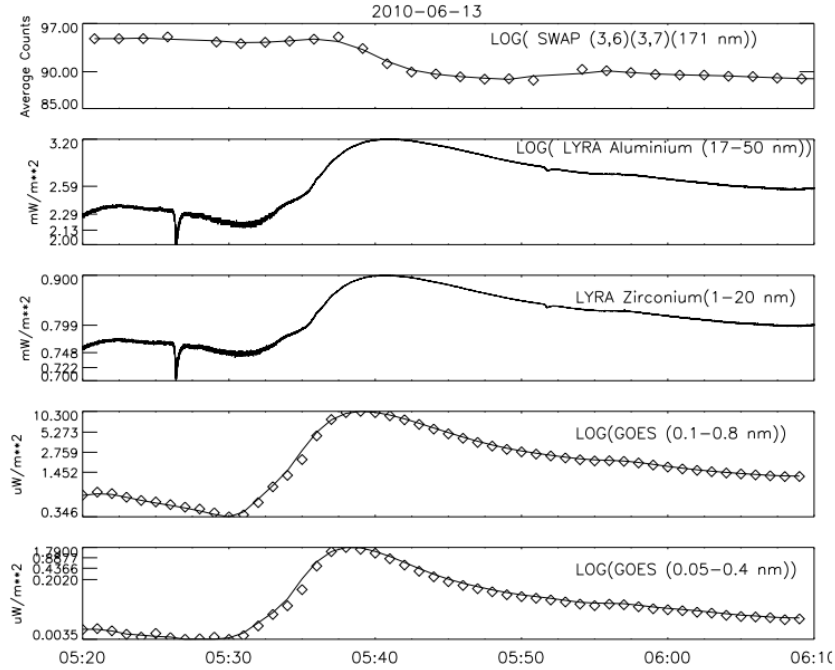


Figure 3.1: Temporal analysis using SWAP, LYRA, and GOES of the M-class flare occurring between 05:20 and 06:10 UT on the 13^{th} of June, 2010 (source [6]).

3.2.3 Frequency Analysis

Wavelet analysis performed by J.J. Zender *et al* uses a continuous wavelet transform. Results from this analysis reveals several frequency components. The most prominent frequency band detected has a frequency between 0.013 – 0.015 Hz, corresponding to a periodic time between 66 – 77 seconds. The mother-wavelet used in the wavelet analysis is a 6^{th} order Morlet wavelet.

3.3 Consideration of Related Papers

3.3.1 Introduction

The concept of oscillatory components in solar features has been known for over 40 years [32]. Since the discovery of these oscillatory components, instruments capable of high resolution sampling of solar data have been deployed, allowing for the discovery and identification of new solar features. As a result, much work relating to solar data analysis using wavelet techniques has been published; the three papers most relevant to this project will be discussed below.

3.3.2 Data Investigated

The data investigated in these papers corresponds to emissions from a variety of wavelengths including hard X-ray emissions (wavelengths of 0.1 to 0.01 nm) in work reported by M.J. Aschwanden *et al* [33], emissions from the solar corona (wavelengths of 170 nm and 33.5 nm)

in work reported by E. O'Shea *et al* [7], and emissions at 17.1 and 154.81 nm in work reported by J. Rendtel *et al* [34]. Only some of the wavelengths analysed in these papers are detected by the Zirconium channel of the LYRA instrument, because the Zirconium channel has a passband of 1 to 20 nm. The Aluminium channel has a passband which covers wavelengths between 17 and 80 nm but this channel is not intended to be analysed.

The sampling rate of the data to be analysed in each paper also differs. M.J. Aschwanden *et al* [33], analyses flares that are observed using the Compton Gamma Ray Observatory, allowing a time resolution of 64 ms over 4 minutes in each flare [33]. In E. O'Shea *et al* [7], the data that was analysed was sourced from the Coronal Diagnostic Spectrometer (CDS) on the Solar and Heliospheric Observatory (SOHO), and the Transition Region and Coronal Explorer (TRACE) satellite. These instruments allowed a cadence of 20 and 22 s, respectively [7]. Similarly, in J. Rendtel *et al*, the data obtained from the Solar Ultraviolet Measurement of Emitted Radiation (SUMER) spectrograph had a cadence of approximately 25 s [35]. As a result, it is probable that the data analysed in E. O'Shea *et al* and J. Rendtel *et al* may not have had a high enough sampling frequency to detect the oscillations found by J.J. Zender *et al* [6].

3.3.3 Analysis Techniques Used

M.J. Aschwanden *et al*, E. O'Shea *et al*, and J. Rendtel *et al* all use wavelet analysis to analyse solar data. Although the techniques used in each paper are not identical to the intended method of wavelet analysis for this research project (Continuous Wavelet Transform), the analysis shall be discussed and compared below.

Work completed by M.J. Aschwanden *et al* consisted of using triangle-shaped wavelets with a discrete wavelet transform to decompose the signals of 647 flares. The results of this analysis were then converted into a standard distribution function of physical time-scales using an algorithm developed in the paper [33]. Discrete wavelet transforms and conversions are not intended to be used during this research project; however, analysing the duration of the oscillatory components discussed in J.J. Zender *et al* using discrete wavelet transforms could be used in future research to develop a more thorough analysis.

Analysis performed by E. O'Shea *et al* is more closely related to the work reported in this project as the same mother-wavelet (Morlet) is used [7, 36]. Work reported in E. O'Shea *et al* also includes the removal of low frequency background oscillations from the signal. An example of the results of analysis performed by E. O'Shea *et al* is shown in figure 3.2. The hatching seen in the middle-left image in figure 3.2 represents a region of unreliable results due to the Cone of Influence (COI), as discussed in subsection 2.8.2.1. A similar COI is expected in analysis performed in this project.

Unfortunately, the details of the wavelet analysis used by J. Rendtel *et al* are unspecified and, therefore the analysis performed can not be discussed further.

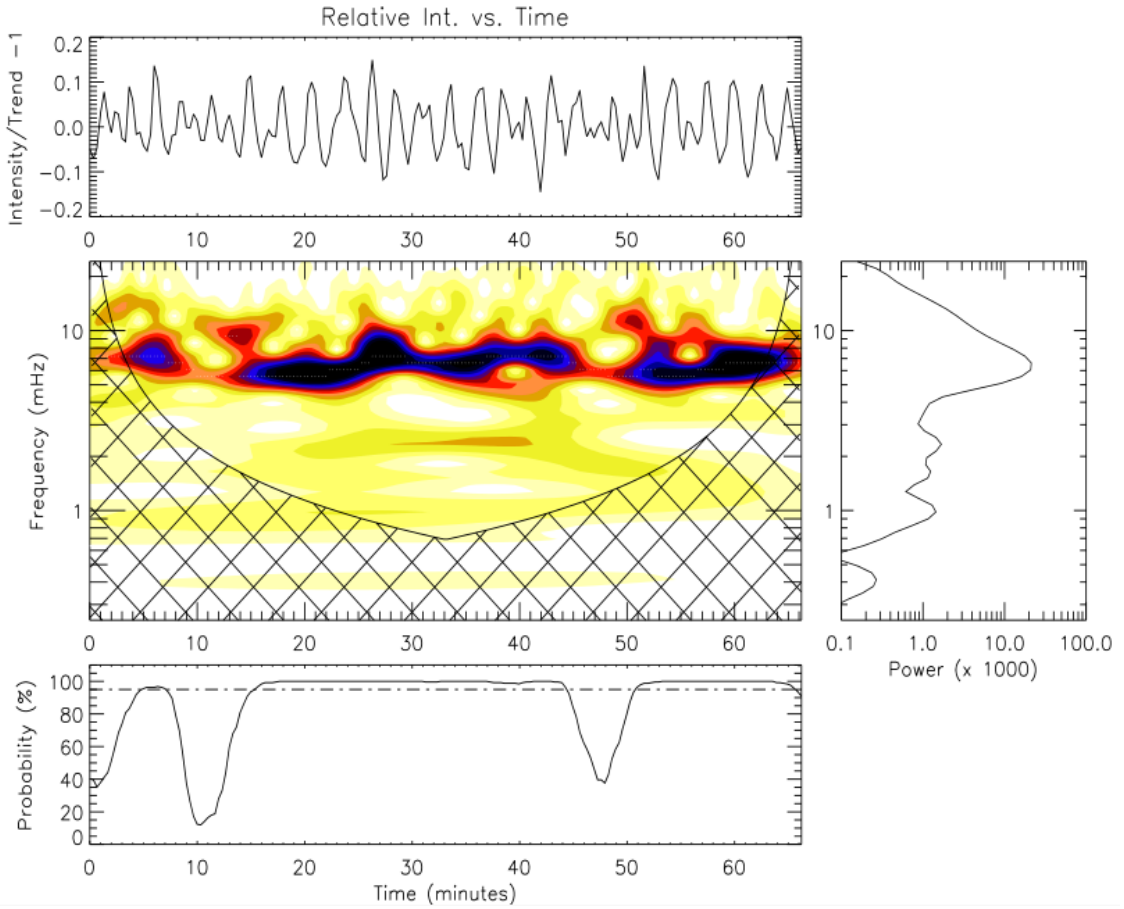


Figure 3.2: Wavelet analysis of data obtained from the CDS instrument (source [7]).

3.3.4 Findings in Related Papers

The findings in M.J. Aschwanden *et al* [33] reveal the duration of several time structures found in hard X-ray emissions using wavelet analysis. The range of time-scales for some structures are found to be of the order of 0.1 – 0.7 seconds [33]. However, as the goal of this research project is to detect oscillatory components in emissions with wavelengths not analysed by M.J. Aschwanden *et al*, the results shall not be discussed further.

E. O'Shea *et al* [7] show that wavelet analysis revealed oscillations present across several emissions. These emissions correspond to different temperatures with frequencies in the range 5.4 mHz to 8.9 mHz. There was found to be a time delay between the oscillations in the different emissions. Once this time delay was taken into account, there was found to be a good correlation between the frequencies measured at the different emission temperatures [7].

Work published by J. Rendtel *et al* shows that wavelet analysis revealed oscillations in both the chromosphere and transition region lines of the Sun. Oscillations of the order of 5 minutes were found to dominate in the chromosphere, while the transition region lines show oscillations at shorter periods of 2 to 3 minutes [34].

3.4 Conclusion

From the literature reviewed, it can be seen that wavelet has been proven to provide qualitative results and detect a range of oscillatory components in several solar features. It can also be seen that oscillations in solar features have been found at frequencies quite close to those observed by J.J. Zender *et al*, with periods of 2 – 3 minutes in comparison to 66 – 77 seconds. Both the detection of similar frequency components to those discussed in J.J. Zender *et al*, and the proven use of wavelet analysis, lends confidence to the detection of these frequencies across multiple solar events in LYRA data.

Chapter 4

Final Implementation

4.1 Introduction

To perform analysis of LYRA data, a suite of IDL procedures were written. These procedures underwent a significant number of revisions during this project. In order to simplify the description of this software, only the final implementation will be discussed in this chapter. The source code is given in appendix A.2.

4.1.0.1 Description

The final iteration of the software is designed to analyse a single day of LYRA data using a collection of IDL procedures. The majority of these procedures are designed to be used with a small selection of LYRA data at a time. Taking this approach has several benefits including reducing the time required for analysis and localising errors from unexpected events in data. All parameters that are designed to be configurable are accessible in the main procedure, known as 'lyraScript'.

An illustration of the sequence of steps used to analyse LYRA data is shown in section 4.2. Each step shown in the illustration is described further in the following sections.

4.2 Implementation Flowchart

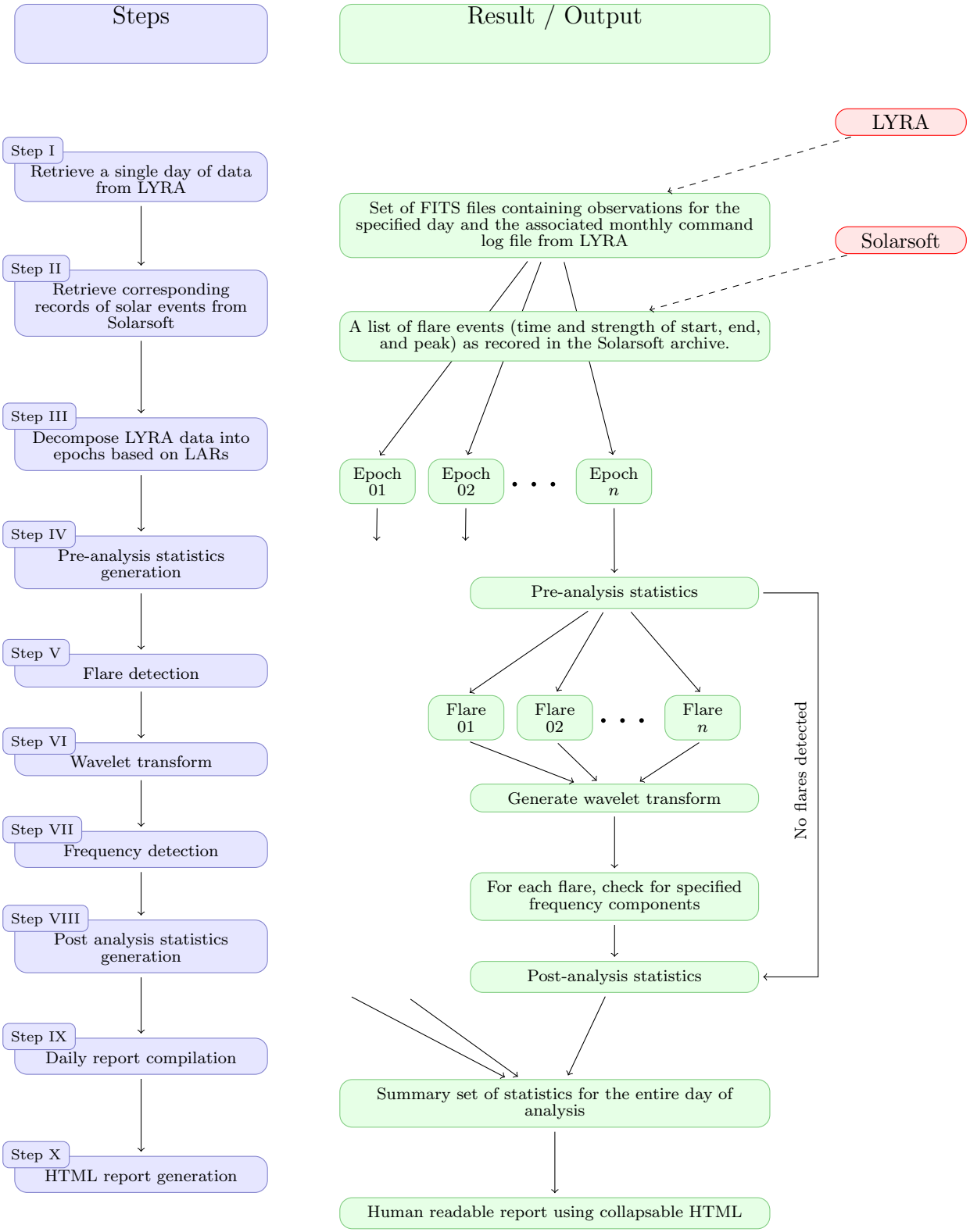


Figure 4.1: A flowchart illustrating an overview of the analysis software.

4.3 Step I — Downloading a single day of data

4.3.1 Outline

Before analysing LYRA data, the FITS files corresponding to each day of observations must be downloaded. This process is separated from analysis procedures so FITS data can be automatically downloaded regularly. Having local copies of the LYRA data speeds up subsequent analysis.

4.3.2 Implementation

This step was implemented using a BASH script [37]. The script downloads all available FITS files corresponding to a single day which is chosen by the user. The script also downloads command logs for the PROBA-2 satellite corresponding to the period of LYRA data that is being observed. These logs contain up to a month of instructions to be performed by the satellite, including instructions for Large Angle Rotations.

The date can be specified by the user by calling the script with 3 arguments corresponding to the day, month, and year of that date. If the script is called without any arguments, the script defaults to downloading data from the previous day.

4.3.3 Issues

There are no known issues with this script.

4.4 Step II — Retrieving SolarSoft flare records for a single day of LYRA operations.

4.4.1 Outline

In order to categorize solar flares, solar flare events must be reliably detected and recorded. One of the methods used to detect solar flares uses LYRA data, however, this is an experimental process and is not completely accurate. As a result, the SolarSoft flare archive is parsed to provide a secondary list of flare events in case the flare detection performed using LYRA data fails.

4.4.2 Implementation

To retrieve the list of solar flares provided by SolarSoft, the online archive is parsed using a procedure written in IDL. This procedure retrieves the HTML from the online SolarSoft archive, then uses pattern matching on the returned HTML to find the specific web-page that contains all solar flares occurring on a date specified by the user. A second request is then made to

retrieve the HTML from that page. Once received, the returned HTML is parsed to create a structure containing the start, end, and peak times of all flares that occurred on the date of interest as well as their respective strengths.

4.4.3 Issues

Due to the reliance of this procedure on pattern matching, there is a strong dependency on the formatting of the HTML returned from the SolarSoft page. If this formatting changes, the patterns used in the procedure must be changed.

There is scope for improving the retrieval of flare archives using the SolarSoft software package for IDL. But this was not known prior to the original implementation of this procedure and has not since been investigated due to time constraints.

4.5 Step III — Decomposition of a day of LYRA data

4.5.1 Outline

As the LYRA data is strongly affected by Large Angle Rotations, the program was designed to only analyse the LYRA data in the intervals between LARs. This reduces the complexity of analysis as effects arising from LARs are difficult to isolate from real data and can potentially lead to false positives in flare and frequency detection.

4.5.2 Implementation

In this step, the command logs that are downloaded in Step I are used to determine the times of each Large Angle Rotation manoeuvre performed during a day of analysis. An outline of the procedure is given below.

1. Each line of the commanding log is read into an array.
2. Any entries in the array that do not correspond to the current period of LYRA data being analysed are removed.
3. Entries corresponding to Large Angle Rotation are found.
4. LAR entries are parsed to determine the times of each event.
5. The LYRA data is then split into smaller windows of data between each LAR (approximately 20 minutes long).

The length of time omitted before and after a LAR can be adjusted using two parameters. Once the start and end times of data to be analysed are determined, their corresponding indexes in LYRA data are found. Afterwards, both the times and indexes for each section of data between Large Angle Rotations are stored in a structure. A structure is made for every section of data to

be analysed, resulting in an array of structures which are iterated over during analysis. There is a possibility of arbitrarily short windows if there is a small section of data available after a LAR. Such windows are discarded in later analysis.

4.5.3 Issues

This are no known issues with this feature but it does require command logs to be supplied. A future enhancement can be made to automatically detect Large Angle Rotations using other channels in the LYRA instrument if a command log is not available. This enhancement has not been implemented in the project due to the limited time and marginal benefit.

4.6 Step IV — Pre-analysis statistics generation

4.6.1 Outline

The aim of this step in analysis is to provide a method of generating statistics of a window of data prior to frequency analysis. These statistics are not currently used in analysis, but can be built upon in future enhancements to the software.

4.6.2 Implementation

The implementation of this step is relatively simple in comparison to others; the section of LYRA data specified by the window of data generated in step III is passed into the procedure and statistics are generated. Currently the statistics generated include the number of points being analysed and length of time corresponding to the data but as mentioned before, this step is a place-holder to future enhancements. Once analysis has been performed, a structure containing all results is generated and returned.

4.6.3 Issues

There are no known issues in this procedure.

4.7 Step V — Flare detection

4.7.1 Outline

In order to categorize solar flares using frequency analysis, flare events must be successfully detected. As well as using the SolarSoft archive, flare detection is also performed using LYRA data. The motivation behind using the LYRA data to detect a

4.7.2 Implementation

In this step, the Aluminium and Zirconium channels of the LYRA instrument are compared to determine any lag between their respective responses during a potential flare event. An example of this lag is shown in figure 4.2 (Put in figure here). The detection of flares uses the following equation to exploit this lag, based on power-response curves suggested by Bernard Foing in previous work:

$$\frac{\text{Zirconium} - \text{Aluminium}}{\text{Zirconium/Aluminium}} \tag{4.1}$$

One of the difficulties in performing this detection arises from the degradation of the detectors over time. To overcome this, the median value of equation 4.1 is determined for the entire day of LYRA data. This is only calculated once during analysis of a single day of LYRA data and then stored for future reference. When this procedure is called, the equation is evaluated at each point in the window to be analysed, and the maximum value is then compared against the median; if the difference is above a user-definable threshold, the window of LYRA data is flagged as containing a flare.

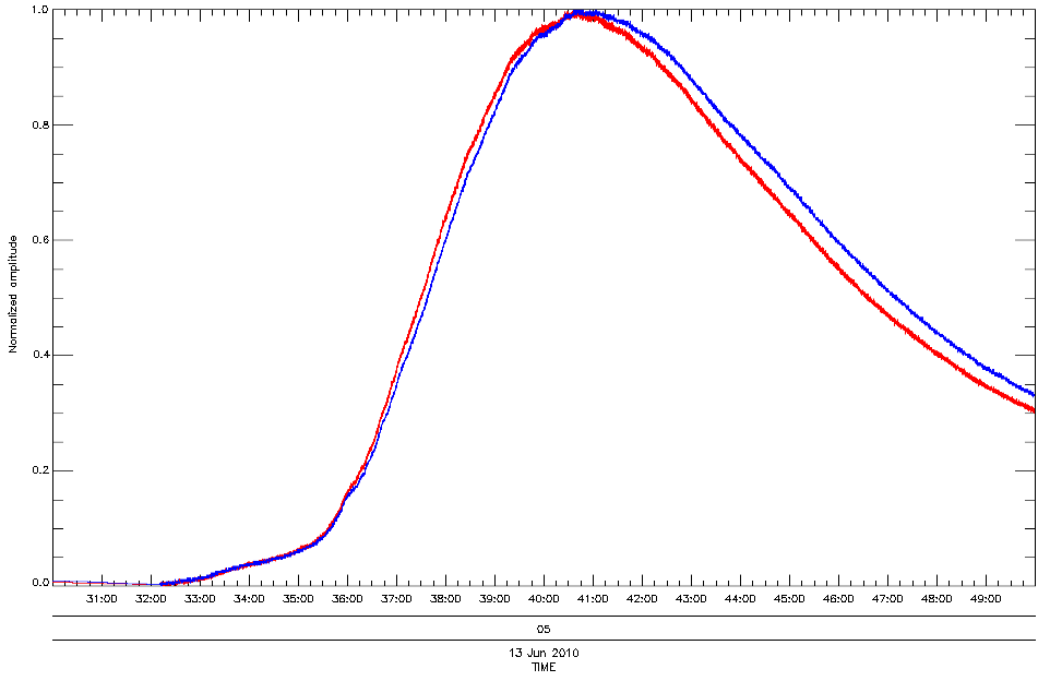


Figure 4.2: The Zirconium (red) and Aluminium (blue) channels of the LYRA instrument during an M-class flare occurring between 05:30 and 05:50 UT on the 13$^{\text{th}}$ of June, 2010. Both channels have been normalised between 0 and 1 to demonstrate the lag between them. The linear section of both channels from 05:30 to 05:32 is due to interpolation over a Large Angle Rotation.

4.7.3 Issues

There are known issues in this step related to false negatives and false positives. False negatives can occur in particularly weak flares (usually below a B class flare). False positives can occur for short lived events (the order of 2 minutes) and anomalous spikes. There is potential to improve the algorithms accuracy and enforce duration requirements for all events but this has not been investigated due to the limited time available.

4.8 Step VI — Wavelet transform

4.8.1 Outline

In this step of analysis, a continuous wavelet transform is performed on a window of data from the LYRA instrument. The motivation behind using a wavelet transform arises from the ability of wavelet analysis to detect short lived periodicities. Once this transform is performed, the result can then be analysed to detect if frequencies existed. Other methods such as Short Term Fourier Transforms could be used, but the duration of the frequency components detected can be quite short, sometimes lasting for up to only 4 periods, which would lead to poor detection in STFT.

4.8.2 Implementation

The wavelet transform is implemented using the wavelet procedures included with the IDL Astronomy User's Library [29]. The parameters used in the wavelet transform, such as the mother-wavelet family and order, can be specified by the user in the main procedure of the software. Before wavelet analysis is performed, the trend of the LYRA data is obtained using a moving average; the parameters for determining the moving average can also be specified by the user. The original signal is then divided by the trend to give the relative intensity of any events within the LYRA data. The default mother wavelet used in the analysis is a 10^{th} order Morlet wavelet. After the wavelet transform has been completed, the following are determined; the Cone of Influence (COI), the regions with a significance of 99% and above, and the total power of each scale outside of regions not affected by the COI.

The Cone of Influence is determined using the relationships described in C.Torrence and G.P.Compo *et al* [26]. The current implementation of this step requires a Morlet mother-wavelet to be used. The Cone of Influence will be returned a 2 dimensional array, the same size as the original wavelet transform, mapped with either a 1 or 0 to indicate a region in the Cone of Influence.

The regions of a significance equal to or above 99% are determined using an IDL procedure written by C.Torrence and G.P.Compo based on work discussed in C.Torrence and G.P.Compo *et al* [26]. The implementation of this IDL procedure requires the parameters used for wavelet analysis as the significance regions are dependant of the type of wavelet used.

The final process in this step requires the summing of the power in each scale of the wavelet transform. The power is only summed outside of the Cone of Influence to prevent edge effects affecting the power and potentially causing false positives in frequency detection.

4.8.3 Issues

The current implementation of this step relies on the Morlet mother-wavelet being used. Further enhancements can be made to this step to allow the use of other families of mother-wavelets. Further analysis could be performed on LYRA data with the trend subtracted from the original rather than analysis the relative intensity. This would be beneficial in quantifying the amplitude of periodicities which can not be performed in the current implementation of the software.

4.9 Step VII — Frequency detection

4.9.1 Outline

This step analyses the wavelet transform to detect if a frequency component was present using a set of criteria that can be adjusted by the user.

4.9.2 Implementation

The implementation of this step is designed to detect all frequencies found between a user-defined range. The wavelet transform is first analysed to find all scales occurring within the specified frequency range. The wavelet transform at these scales are then combined using the maximum value between all scales at each point, resulting in an array of the same dimensions as a single scale of the transform. When the maximum is being determined, data within the Cone of Influence is ignored to prevent edge-effects leading to erroneous data. The resulting array containing the maximum between all scales is then measured to detect any regions of the array that exceed a threshold specified by the user. The durations of these regions are then measured to ensure they occur for longer than the average size of the Cone of Influence for the range of scales that were used.

4.9.3 Issues

This step avoids measuring scales inside the COI, which can cause the duration requirement of periodicities to trigger a false negative. This issue is made worse by analysing relatively short subsections of data, leading to the Cone of Influence affecting a larger region of the wavelet transform. This effect could be reduced by using a mother-wavelet with a smaller associated Cone of Influence or measuring the duration of the signal including the portion with the COI, provided the maximum strength of the signal occurs outside the COI.

4.10 Step VIII — Post analysis statistics generation

4.10.1 Outline

The aim of post-analysis statistics generation is to generate a uniform layout of statistics based on the results of previous steps in analysis. The motivation behind this is to simplify parsing performed in later steps.

4.10.2 Implementation

The implementation of this step takes all statistics generated from previous analysis as input and then summarises them to form a single uniform structure. Statistics that are considered include the results from flare detection, the results from wavelet analysis, and pre-analysis statistics. For example, the results of flare detection indicate whether a flare was detected using the SolarSoft archive or LYRA data; these results can be simplified to simply indicate if a flare was detected or not. More complex tasks could also be performed based on signal durations, strengths of frequencies and solar events, to generate a more comprehensive set of analysis from the results of previous steps, but this level of analysis is not intended for this project. Summarising these statistics also simplifies the implementation of later steps in analysis, and enforces a structure to be used in storing analysis.

4.10.3 Issues

There are no known issues with this procedure.

4.11 Step IX — Daily report compilation

4.11.1 Outline

After post-analysis statistics have been generated, this procedure reads each set of statistics for each window and generates a summarised set of statistics for the entire day of analysis.

4.11.2 Implementation

This procedure is designed to take an arbitrarily sized array of statistics generated from analysis of each window and generate a summarised report which can be enhanced if required in future enhancements to the software. The results of this step are used to generate the report in Step X. One of the outputs from this procedure is a correlation table to show the relation between different results of analysis such as the detection of frequencies and detection of flare events using either SolarSoft archives or LYRA data. This table is dynamically generated using the statistics generated in the previous step. Further statistics can be added as required to the output of this step.

4.11.3 Issues

There are no known issues with this procedure.

4.12 Step X — HTML report generation

4.12.1 Outline

The aim of this final step is to generate an easily readable report of the analysis performed on a day of LYRA data.

4.12.2 Implementation

The implementation of this step has two core components. The first is a template HTML file which is included with the source code. The second is a recursive function which is used to iterate over a single structure of analysis. The analysis structure can be configured by the user. The HTML file provides the template design of the report as well as containing a unique tag which indicates where the results of analysis should be included. The recursive function allows an arbitrary structure containing any data type in IDL, including arrays and structures, to be processed and added to the report.

When the procedure is called, the template HTML file is read line by line into an output file, until it reaches a unique tag as specified in the procedure. When it reaches this tag, the recursive function is called to parse the analysis structure and generate HTML output on the fly for each element of analysis. When the structure has been fully read, the remaining lines of the HTML template are added to the report. The final result of this procedure is a HTML report following the design of the template file containing all of the analysis in the analysis structure. The format of the analysis in the report reflects the original structure (e.g. an array of data in the structure will cause a table to be generated).

4.12.3 Issues

The implementation is limited to outputting all of the analysis in a single position of the report. In the future, the software could be improved to only output specific sections of the summarised analysis at different locations in the template document. This step could also be enhanced to generate a report in any format based on instructions within a template document; for example, a LaTeX report. These enhancements have not been attempted due to their relatively low priority.

Chapter 5

Results

5.1 Introduction

In this chapter, the software tool outlined in chapter 4 is used to examine LYRA data for a number of days. While this tool can be applied to the complete set of available LYRA data (from January 1th – present), this strategy would be of little benefit because of the large volume of data and the need to manually examine all results due to the large number of factors that can invalidate results. In other words, the intended use of the software tool is to aid and speed up manual analysis of particular intervals rather than complete automation of analysis.

In order to demonstrate the type of analysis that is possible using the software tool, three sets of analysis are demonstrated in sections 5.2 to 5.4. These three sections demonstrate analysis of an ideal day (with relatively clean data and one large solar flare), a typical day (with multiple flares and some unknown events), and a higher level view of analysis over 20 days of data. Also, in the first section, which discusses an ideal day of data, the output of the software tool is explained in parallel to interpretation of the results.

As LYRA is an ongoing scientific experiment, the data is subject to unforeseen events which can have instrumental, atmospheric, or solar origins; therefore discussion of some pathological intervals of data is presented in section 5.5.

Finally, a brief discussion of some epochs of interest found in analysis of March, 2011, is presented in section 5.6.

The conclusions of all results are discussed in section 6.2.

5.1.1 Analysis Parameters

The CWT was performed using a 10th Morlet mother-wavelet. The trend of the signal was determined using a 30 second moving average. The Frequency detection algorithm were configured to detect frequencies with a periodic time between 60 and 100 seconds. The LYRA-based flare detection algorithm was configured to ignore flares weaker than a B5.0 class flare; this configuration was chosen as flares below this level are usually short lived, and therefore have a very low probability of containing the particular frequency components that are of interest in this research project.

5.1.2 Terminology

Analysis of LYRA data is performed using intervals of data occurring between Large Angle Rotations and the start/end of a day of LYRA data, as discussed in section 4.5. In the following

discussion of analysis, each of these intervals will be referred to as an epoch.

5.2 Analysis of an Ideal Interval (07 August 2010)

5.2.1 Introduction

This section discusses analysis performed on LYRA data recorded during August 7th, 2010. This date was chosen as it contains an ideal set of data consisting of 3 minor solar events which are below the threshold for LYRA-based flare detection, as well as one M-class flare containing the oscillatory components desired in this analysis.

5.2.2 Overview

The list of flares that occurred during this interval of data are shown in table 5.1. A plot of the Zirconium channel during this interval is shown in figure 5.1. The red overlay of LYRA data in this figure represents the location of flares as reported by Solarsoft. The effect from Large Angle Rotations, as discussed in section 2.11.2, can be seen in figure 5.1 as the regularly spaced downward spikes occurring every 25 minutes. These spikes will not be evident in data due to analysis being performed on epochs of LYRA data (see section 4.5).

	Time (HH:MM)		
Flare Strength	Beginning	End	Peak
B3.0	02:30	02:56	02:44
B1.7	05:09	05:15	05:13
M1.0	17:55	18:24	18:24
B2.9	22:40	22:46	22:43

Table 5.1: Flares recorded by Solarsoft occurring on August 7th, 2010.

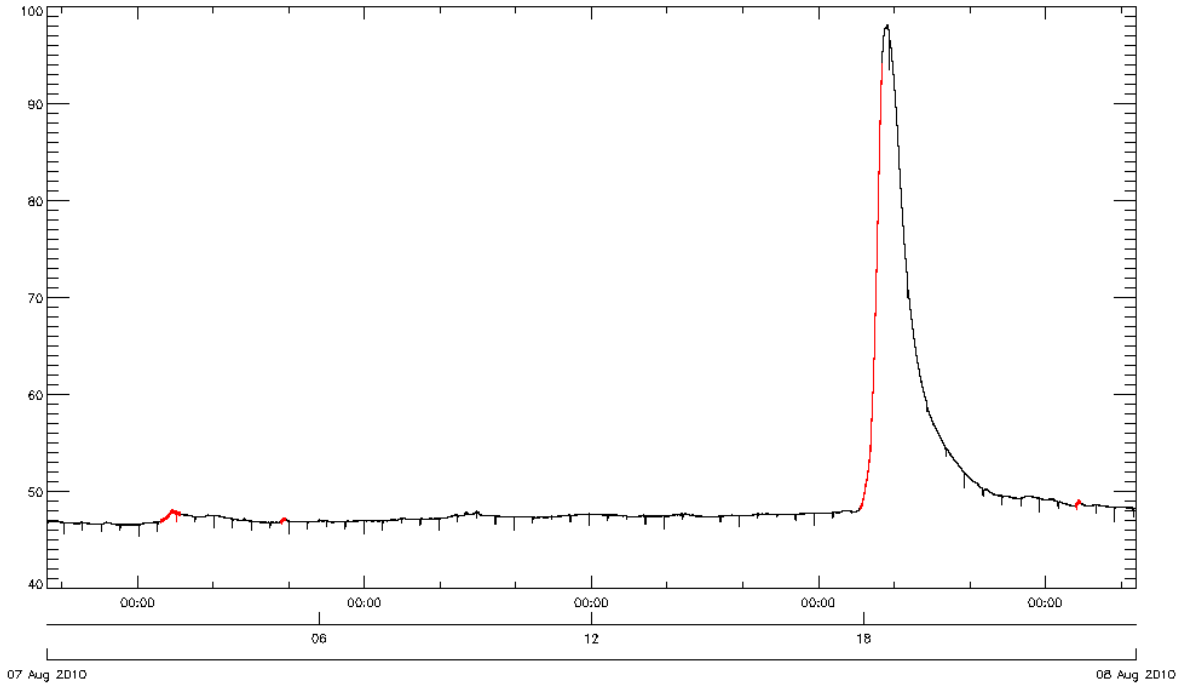


Figure 5.1: Plot of Zirconium channel during August 7th, 2010.

5.2.3 Description and Interpretation of Analysis

5.2.3.1 Correlation Tables

Analysis of LYRA data is performed on intervals of data specified by epochs; analysing a single day of LYRA data can involve up to 57 epochs. In order to simplify analysis and comparisons of large sets of data, a correlation table is generated. This correlation table highlights relations between epochs analysed within an interval of data (usually a single day).

The epoch correlation table for August 7th, 2010, as extracted from the analysis report, is shown in figure 5.2. A typeset version of this correlation table is included for clarity. The names of each field in the correlation table are explained in table 5.2.

Field	SSW_Y	SSW_N	LDW_Y	LDW_N	FCW_Y	FCW_N
SSW_Y	6	0	2	4	1	5
SSW_N	0	51	6	45	0	51
LDW_Y	2	6	8	0	1	7
LDW_N	4	45	0	49	0	49
FCW_Y	1	0	1	0	1	0
FCW_N	5	51	7	49	0	56

		Solarsoft		LYRA		FC	
		Yes	No	Yes	No	Yes	No
Solarsoft	Yes	6	.	2	4	1	5
	No	.	50	6	44	.	50
LYRA	Yes	2	6	8	.	1	7
	No	4	44	.	48	.	48
FC	Yes	1	.	1	.	1	.
	No	5	50	7	48	.	55

Figure 5.2: Epoch correlation table for August 7th, 2010.

Field	Abbreviation	Description
SolarSoft Detected	SSW	Epoch containing data within the beginning and end times of a flare as recorded by Solarsoft.
LYRA Detected	LDW	Epoch containing data which has been flagged as a flare using LYRA-based flare detection.
Frequency Detected	FCW	Epoch flagged as containing the particular oscillatory components under investigation using wavelet analysis.

Table 5.2: Description of fields used in epoch correlations tables.

In order to demonstrate interpretation of correlation tables, the correlation between flares detected using LYRA-based flare detection and flares recorded by Solarsoft will be investigated. From investigating the correlation table, table 5.2, the following can be observed:

- 8 epochs in total, were detected as containing flares using LYRA-based flare detection (LDW_Y–LDW_Y); of these 8 epochs, 2 were also reported by Solarsoft as containing flares (LDW_Y–SSW_Y).
- 6 epochs in total were reported as containing a flare by Solarsoft. Of these 6 epochs, 4 epochs were not detected by LYRA-based flare detection (SSW_Y–LDW_N).

In summary, it can be seen that 2 out of 6 epochs containing flares recorded by Solarsoft were detected using LYRA-based flare detection, and 6 epochs were detected using LYRA-based flare detection which were not recorded by Solarsoft.

It can be seen from the table that there are often multiple epochs corresponding to a single solar flare; 6 epochs are flagged as containing flares by Solarsoft, but there are only 4 events

recorded by Solarsoft during this day (see table 5.1). This can cause misinterpretation of results as the oscillations under investigation in this research project may only be present for a single epoch. In order to clarify the correlation of events in data, the epochs relating to each event are analysed collectively. An example of the results from this analysis is shown in table 5.3.

Event Type	Frequency
Solarsoft Events	4
Events Detected by LYRA	1
Events Containing Desired Oscillations	1

Table 5.3: Event correlation summary for August 7^{th}, 2010.

From table 5.3, it can be seen that 4 events were recorded by Solarsoft (Solarsoft Events), as expected. Of these 4 events, 1 was detected using LYRA-based flare detection. Using the information presented in both table 5.2 and 5.3, it can be determined that the two epochs detected as containing a flare using LYRA-based flare detection (SSW_N–LDW_Y of table 5.2), correspond to a single event ('Events Detected by LYRA' of table 5.2).

5.2.3.2 Flare Detection

One of the aims of this research project was to detect a particular range of oscillatory components within solar flares using data collected by the LYRA instrument. As LYRA data can record up to 1.7×10^6 points of data during a nominal day of operations, frequency analysis was restricted to epochs flagged as containing flares. Epochs are flagged as containing flares using both the LYRA-based flare detection algorithm and Solarsoft archives as discussed in section 4.7. As LYRA-based flare detection is experimental, the solar flare records provided by Solarsoft are used for validation. This section will discuss the results of LYRA-based flare detection while referring to Solarsoft records.

In each set analysis report, there are a list of images corresponding to each cell of the epoch correlation table. These images are used to manually verify the results of analysis; complete automation of analysis is impossible as LYRA is an ongoing scientific experiment subject to unexpected behaviours. The results of flare detection will be summarised in table 5.4, figure 5.3, and then discussed.

LDW_Y–SSW_Y		LDW_Y–SSW_N		LDW_N–SSW_Y	
Start Time	End Time	Start Time	End Time	Start Time	End Time
17:44	18:05	18:34	18:55	02:26	02:47
18:09	18:30	18:59	19:20	02:51	03:12
		19:23	19:44	04:55	05:16
		19:48	20:09	22:42	23:08
		20:13	20:34		
		20:38	20:59		

Table 5.4: List of epochs flagged as containing flares using LYRA-based flare detection and Solarsoft archives from August 7th, 2010. (LDW_Y–SSW_Y) contains epochs flagged by both LYRA-based flare detection and Solarsoft. (LDW_Y–SSW_N) contains epochs flagged by LYRA-based flare detection only. (LDW_N–SSW_Y) contains epochs flagged by Solarsoft only.

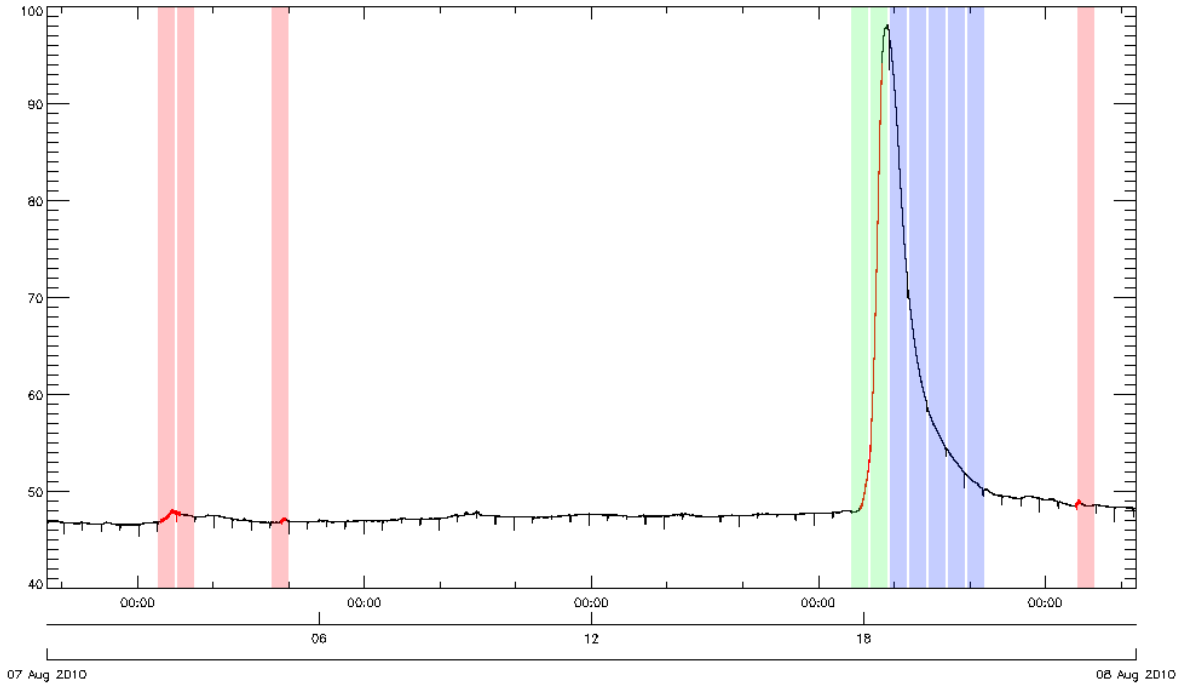


Figure 5.3: Plot of Zirconium channel during August 7th, 2010. The regions where the Zirconium data is dark red indicate flares recorded by Solarsoft. The green highlighted sections correspond to (LDW_Y–SSW_Y) epochs. Blue highlighted sections correspond to (LDW_Y–SSW_N) epochs. Red highlighted sections correspond to (LDW_N–SSW_Y) epochs.

From table 5.4, it can be seen that there are 2 epochs flagged using LYRA-based flare detection and Solarsoft. These epochs can be seen to occur during the duration of the M1.0 class flare as reported by Solarsoft in table 5.1. These epochs are shown in figure 5.3 as the regions highlighted in green.

The 6 epochs flagged as a flare using LYRA-based flare detection which do not correspond to any epochs flagged by Solarsoft are represented in figure 5.3 as the 6 regions highlighted in blue. There is a possibility that Solarsoft did not detect this flare correctly as the peak and end times of the flare are identical in the Solarsoft archive, and it can also be seen that he peak seen in LYRA data occurs after the end time reported by Solarsoft. It should be noted at this point that the LYRA-based flare detection algorithm correctly detected the peak and decay of the flare.

The 4 epochs that were not detected corresponded to B3.0, B1.7, and B2.9 events which are all below the minimum strength detectable using LYRA-based flare detection.

In conclusion, LYRA-based flare detection performed correctly during this interval of data, detecting the entirety of the M1.0 class flare.

5.2.3.3 Frequency Detection

Frequency detection is performed on all epochs flagged as containing a flare — the process is described in section 4.9. However, as the interval of data being discussed in this section is particularly quiet with a single large flare, there is only one epoch flagged as containing frequencies.

The data from the Zirconium channel corresponding to this epoch is shown in figure 5.4. The results of wavelet analysis performed on this data are shown in figure 5.5.

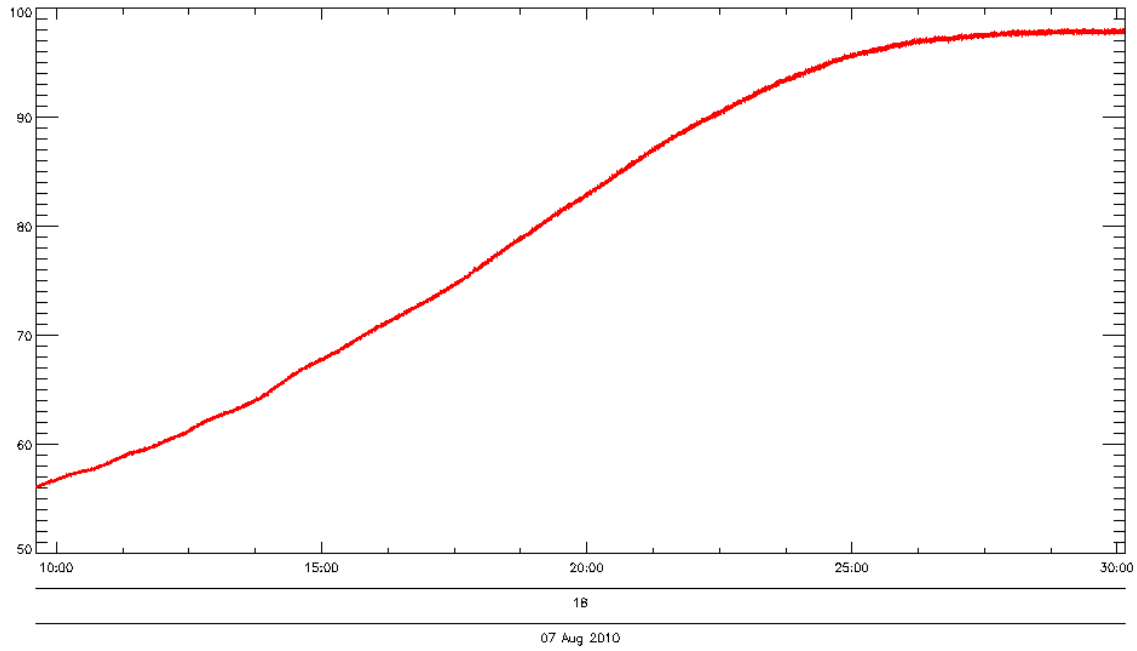


Figure 5.4: Plot of Zirconium channel data during August 7th, 2010, between 18:09 and 18:30.

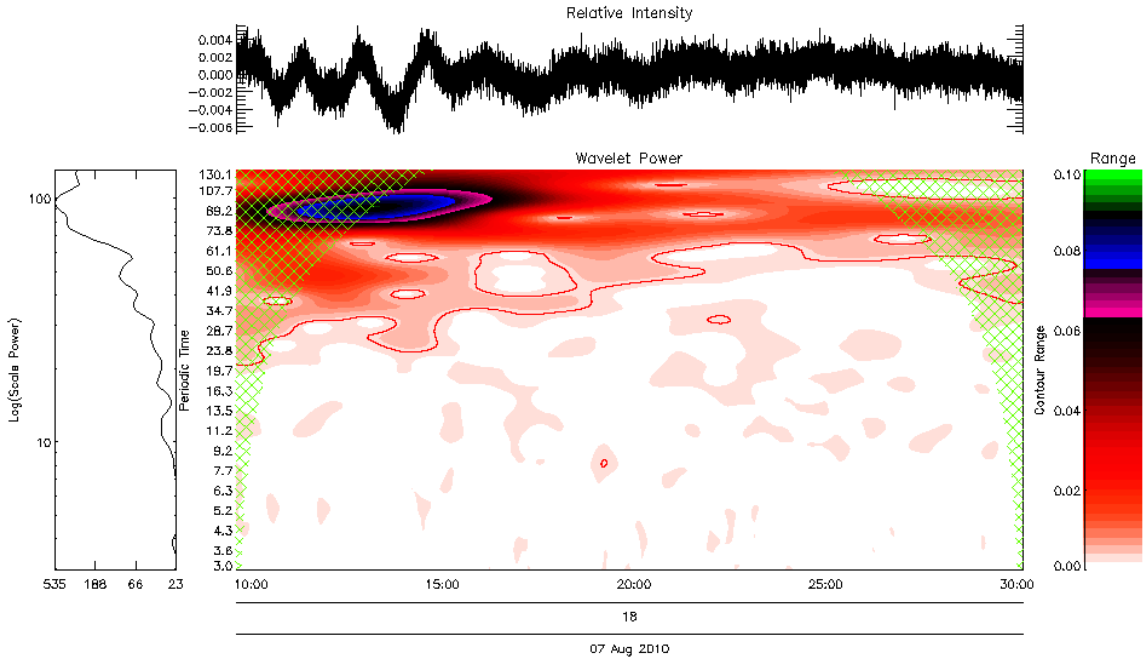


Figure 5.5: Wavelet analysis of Zirconium channel data during August 7th, 2010, between 18:09 and 18:30.

It can be seen from figure 5.5 that there are multiple components involved in the results of wavelet analysis. These components will be explained individually before explaining interpretation of the analysis.

Wavelet Transform The wavelet transform is the main component of wavelet analysis and is represented using the contour plot. At the left and right of the plot, there are two hatched green overlays. These overlays represent the region affected by the Cone Of Influence (COI) as discussed in section 2.8.2.1. Frequency detection only analyses the wavelet transform where it is not influenced by the COI, which can lead to false negatives if the majority of an oscillation occurs in this region; this will be discussed further in section 6.2.3.

Relative Intensity Positioned above the wavelet transform, a plot of the relative intensity of the signal is shown; this is the signal that is used to generate the wavelet transform. The method used to generate the relative intensity is explained in section 4.8.2. The relative intensity of the signal is included in the results of analysis to allow manual validation of results.

Contour Range The corresponding colours for each level of the contour plot are shown to the right of the wavelet transform. As frequency detection is performed using a minimum strength specified in analysis configuration, having a legend explaining the associated strengths of contours helps to identify false negatives in analysis.

Scale Power The total power for each scale in the wavelet transform is shown to the left of the wavelet transform in figure 5.5. This plot is used to help identify clearly the strongest oscillations of the wavelet transform.

Interpretation of Analysis By inspecting the wavelet transform, it can be seen that there is a particularly strong oscillation, shown within the blue and pink region, with a peak strength at approximately 18:13, which decays over time. Inspecting the relative intensity confirms that oscillations were present in the signal. It can then be seen from inspection of the scale power plot that the highest peak corresponds to an oscillation with a periodic time of between 100 and 95 s.

5.3 Analysis of a Typical Interval (13 June 2010)

5.3.1 Introduction

This section discusses analysis performed on LYRA data recorded during June 13^{th}, 2010. Analysis was performed on this interval of data in order to compare results against work reported in J.J. Zender *et al.* However, it should be noted that LYRA data from June, 2010, was used in development of the analysis software; therefore, this analysis is not used to indicate the performance of LYRA-based flare and frequency detection.

5.3.2 Overview

The list of flares that occurred during this interval of data are shown in table 5.5. An overview of data from the Zirconium channel of LYRA during this interval is shown in figure 5.6.

	Time (HH:MM)		
Flare Strength	Beginning	End	Peak
B8.8	00:06	00:26	00:14
B4.6	02:28	02:35	02:31
M1.0	05:30	05:44	05:39
C1.2	06:08	06:13	06:11
B5.3	06:55	07:00	06:58
C1.2	07:05	07:10	07:08
C1.2	07:31	07:38	07:34
C1.2	08:06	08:16	08:13
C1.7	09:41	09:48	09:46
C1.5	10:47	10:55	10:53
B4.3	12:24	12:29	12:27
B2.5	13:20	13:26	13:23
B2.4	14:24	14:30	14:27
B4.2	15:45	15:59	15:52
B1.8	18:10	18:17	18:13

Table 5.5: Flares recorded by Solarsoft occurring on June 13th, 2010.

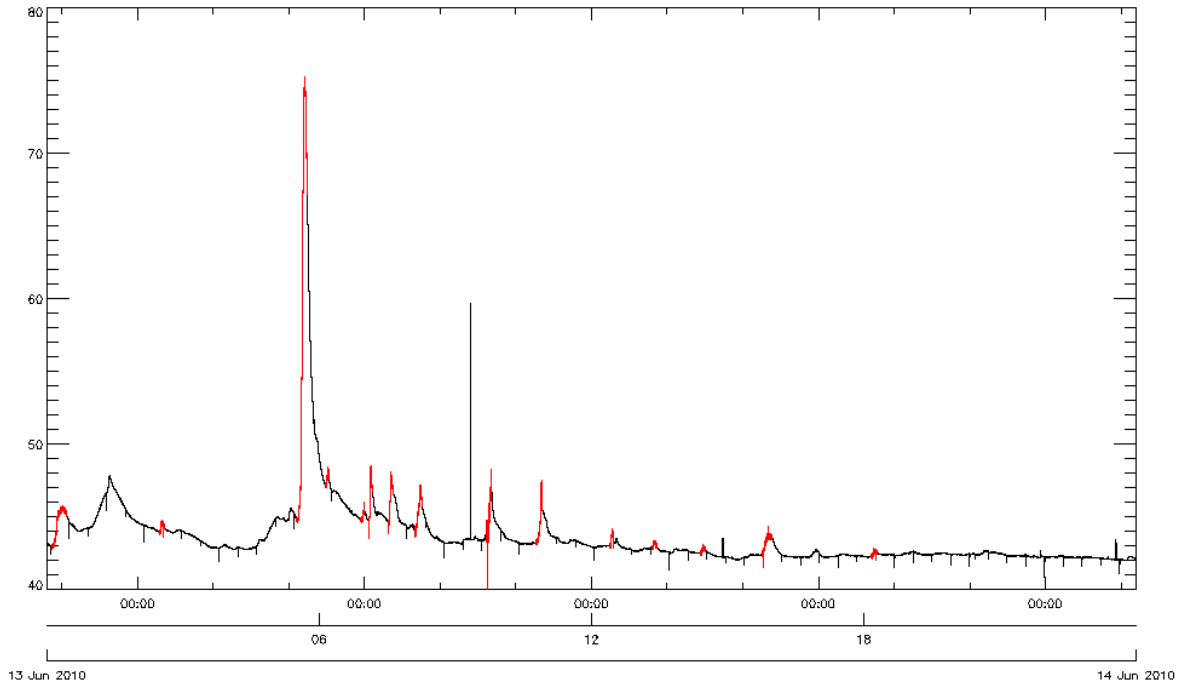


Figure 5.6: Plot of Zirconium channel during June 13th, 2010.

It can be seen from the overview of data that this interval is much more eventful than the interval discussed in section 5.2, with 15 events in comparison to 4. Depending on the activity

of the Sun during a day of analysis, this interval may represent a more realistic profile of LYRA data.

5.3.3 Interpretation of Analysis

5.3.3.1 Correlation Tables

The epoch correlation table for June 13th, 2010, as extracted from the analysis report is shown in figure 5.2. A typeset version of this correlation table is included for clarity.

Field	SSW_Y	SSW_N	LDW_Y	LDW_N	FCW_Y	FCW_N
SSW_Y	17	0	8	9	4	13
SSW_N	0	40	4	36	0	40
LDW_Y	8	4	12	0	4	8
LDW_N	9	36	0	45	0	45
FCW_Y	4	0	4	0	4	0
FCW_N	13	40	8	45	0	53

		Solarsoft		LYRA		FC	
		Yes	No	Yes	No	Yes	No
Solarsoft	Yes	17	.	8	9	4	13
	No	.	40	4	36	.	40
LYRA	Yes	8	4	12	.	4	8
	No	9	36	.	45	.	45
FC	Yes	4	.	4	.	4	.
	No	13	40	8	45	.	55

Figure 5.7: Epoch correlation table for June 13th, 2010.

From the epoch correlation table shown in figure 5.2 the following can be observed:

- 12 epochs were flagged as containing a flare using LYRA-based flare detection. Of these 12 epochs, 8 corresponded to epochs flagged by Solarsoft.
- In total, 4 epochs were detected as containing the particular oscillatory components under investigation. 4 epochs were flagged by Solarsoft and LYRA-based flare detection, therefore they are the same 4 epochs.
- LYRA-based flare detection did not detect all events with a strength larger than B5.0 class flare. However, the exact number of events that were not detected can not be determined at this point as the 15 events recorded by Solarsoft occurred over 17 epochs in total; so there is a possibility that LYRA-based flare detection detected more than one epoch during a single event.

The event detection summary for this interval of data is shown in table 5.6. From the event detection summary it can be seen that 8 different events were detected by LYRA-based flare detection which means that, of the 9 events with a strength greater than a B5.0, only 1 event was not detected. The event detection summary also shows that frequencies were detected in 4 different events.

Event Type	Frequency
Solarsoft Events	15
Events Detected by LYRA	8
Events Containing Desired Oscillations	4

Table 5.6: Event detection summary for June 13th, 2010.

5.3.3.2 Flare Detection

The results of flare detection are summarised in table 5.7 and figure 5.8. By comparing the list of epochs in table 5.7 with the list of events reported by Solarsoft in table 5.5, it can be seen that the event which should have been detected occurs between 00:06 and 00:26. Events which were not detected using LYRA-based flare detection, although they were reported as being sufficiently strong by Solarsoft, will be collectively discussed in section 6.2.2.

LDW_Y–SSW_Y		LDW_Y–SSW_N		LDW_N–SSW_Y	
Start Time	End Time	Start Time	End Time	Start Time	End Time
05:27	05:48	00:54	01:15	00:05	00:25
05:52	06:12	01:19	01:40	02:08	02:30
06:42	07:02	01:44	02:04	02:34	02:54
07:06	07:27	06:16	06:38	12:04	12:25
07:31	07:52			13:19	13:39
07:56	08:17			14:08	14:28
09:35	09:56			15:47	16:08
10:50	11:10			17:51	18:12
				18:16	18:37

Table 5.7: List of epochs flagged as containing flares using LYRA-based flare detection and Solarsoft archives from June 13th, 2010. (LDW_Y–SSW_Y) contains epochs flagged by both LYRA-based flare detection and Solarsoft. (LDW_Y–SSW_N) contains epochs flagged by LYRA-based flare detection only. (LDW_N–SSW_Y) contains epochs flagged by Solarsoft only.

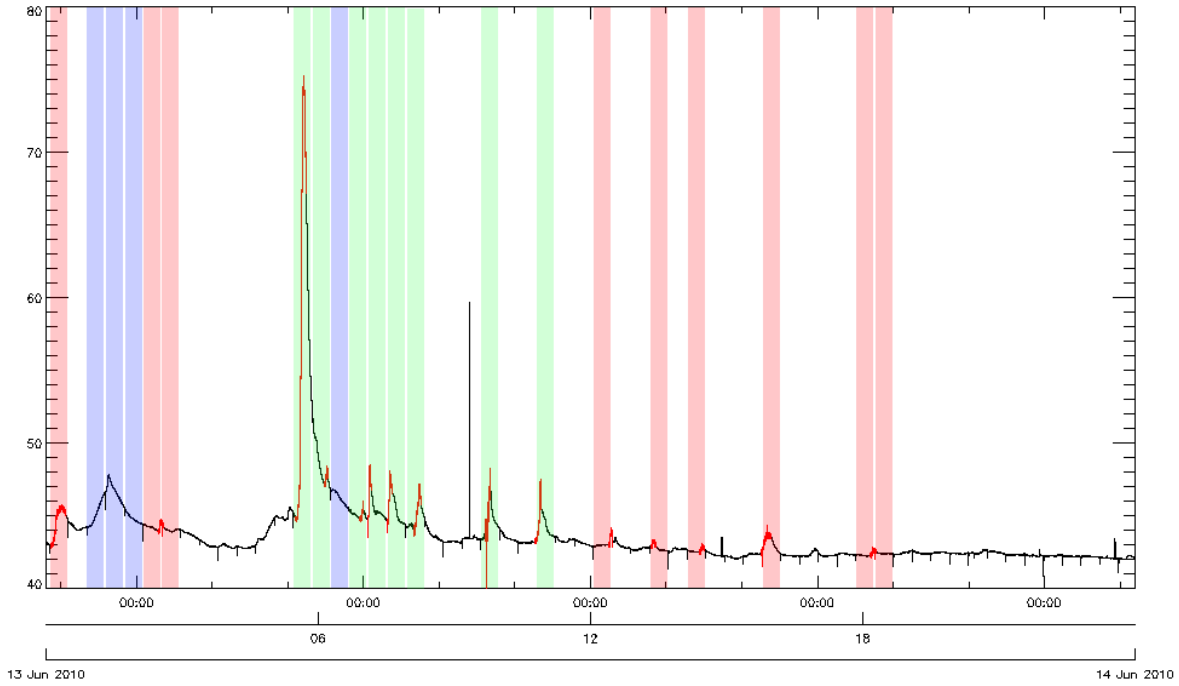


Figure 5.8: Plot of Zirconium channel during June 13th, 2010. The regions where the Zirconium data is dark red indicate flares recorded by Solarsoft. The green highlighted sections correspond to (LDW_Y–SSW_Y) epochs. Blue highlighted sections correspond to (LDW_Y–SSW_N) epochs. Red highlighted sections correspond to (LDW_N–SSW_Y) epochs.

From figure 5.8 and table 5.7 it can be seen that LYRA-based flare detection detected 4 epochs which were not recorded by Solarsoft. One of these epochs occurs during the decay of the M1.0 class flare; the remaining 3 epochs correspond to an event which was not recorded by Solarsoft. The epoch of LYRA data containing the peak of this event is shown in figure 5.9. Events which are detected using LYRA-based flare detection but do not match Solarsoft records will be collectively discussed in section 6.2.2.

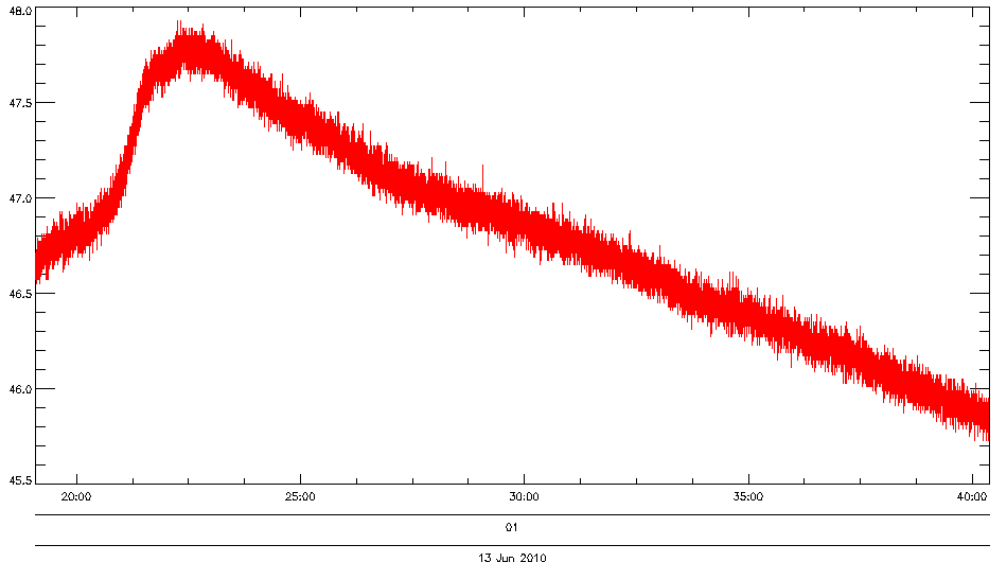


Figure 5.9: Plot of Zirconium channel between 01:19 and 01:40 on June 13th, 2010.

5.3.3.3 Frequency Detection

In figure 5.7 it can be seen that 4 epochs were detected as containing desired oscillatory components; the details of these 4 epochs are shown in table 5.8. As each of the epochs detected correspond to flares recorded by Solarsoft, the strength of the corresponding flares are also given in the table. The wavelet analysis corresponding to the M-class flare will be discussed separately from the 3 C-class flares due to the differences in strength and duration between these events.

Flare Strength	Start Time	Stop Time
M1.0	05:30	05:44
C1.2	08:06	08:16
C1.7	09:41	09:48
C1.5	10:47	10:55

Table 5.8: Flares flagged as containing desired oscillations from June 13th, 2010.

Wavelet Analysis of M-class flare The first epoch detected as containing oscillations spans between 05:27 and 05:48 and corresponds to an M1.0 class flare. A plot of the Zirconium channel during this epoch is shown in figure 5.10 with the corresponding wavelet analysis shown in figure 5.11.

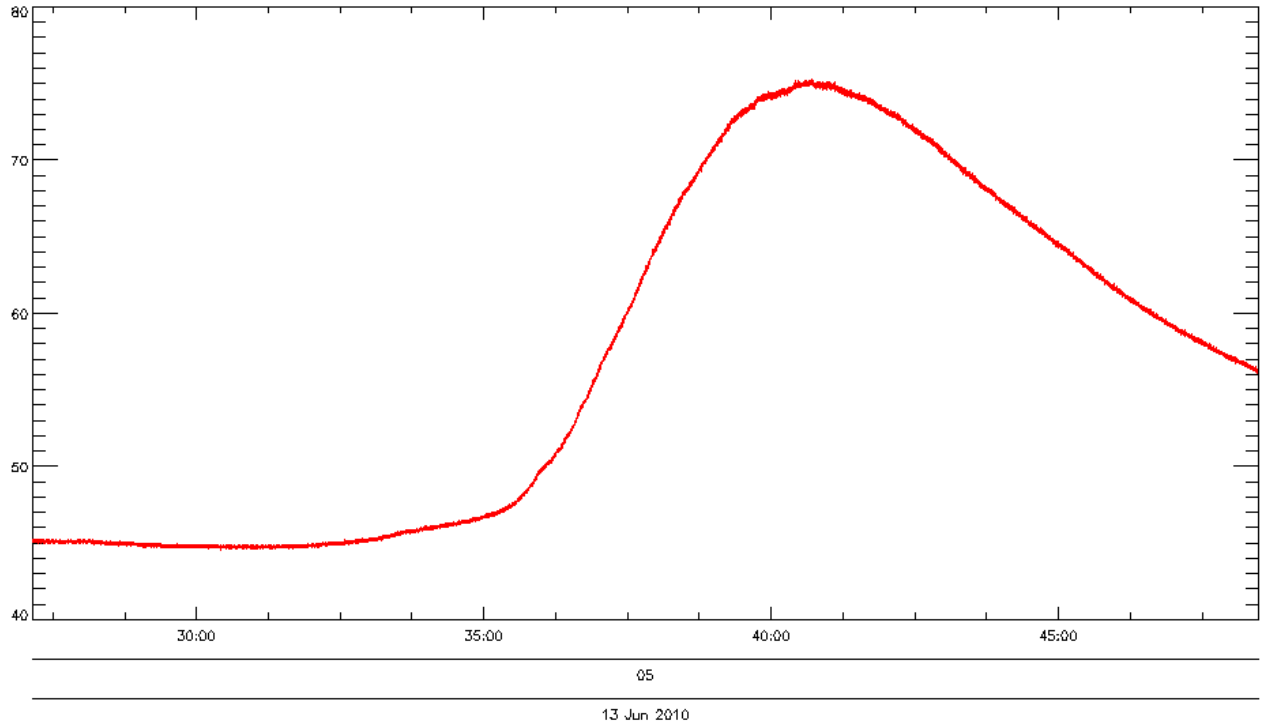


Figure 5.10: Original signal contained within an epoch spanning between 05:27 and 05:48 on June 13th, 2010, containing an M1.0 class flare occurring between 05:30 and 05:44.

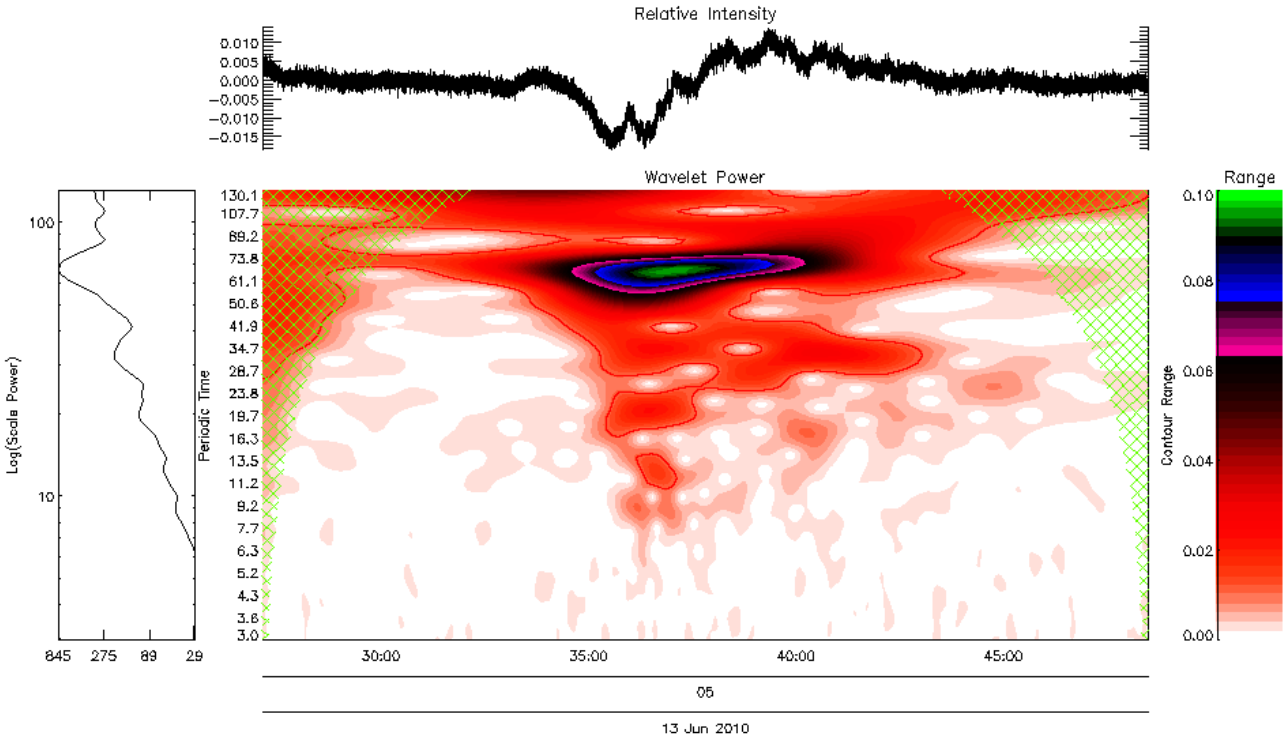


Figure 5.11: The wavelet transform of an epoch spanning between 05:27 and 05:48 on June 13th, 2010, containing an M1.0 class flare occurring between 05:27 and 05:48.

Inspection of wavelet analysis shows that oscillations are present in both the relative intensity of the signal and the wavelet transform, with a peak amplitude seen at approximately 05:37. The plot of scale power shows that the strongest oscillation present in the transform has a periodic time of between 65 and 70 s. It should be noted at this point that the oscillations found in this epoch have a shorter periodic time than the oscillations found in the M-class flare occurring on August 7th, 2010 (see section 5.2.3.3), which had a periodic time of approximately 100 s. The relationship between the periodic times of these oscillations and their corresponding flares will be investigated further in section 6.2.4.1.

Wavelet Analysis of 3 C-class flares The 3 remaining epochs detected as containing oscillations correspond to C1.2, C1.7, and C1.5 flares. As each of these flares last for a similar duration (under 10 minutes) and exhibit similar behaviour to each other, they will be collectively discussed using analysis of one of these flares; the C1.2 flare. The Zirconium channel during the epoch corresponding to the C1.2 flare is shown in figure 5.12 with corresponding wavelet analysis shown in figure 5.13.

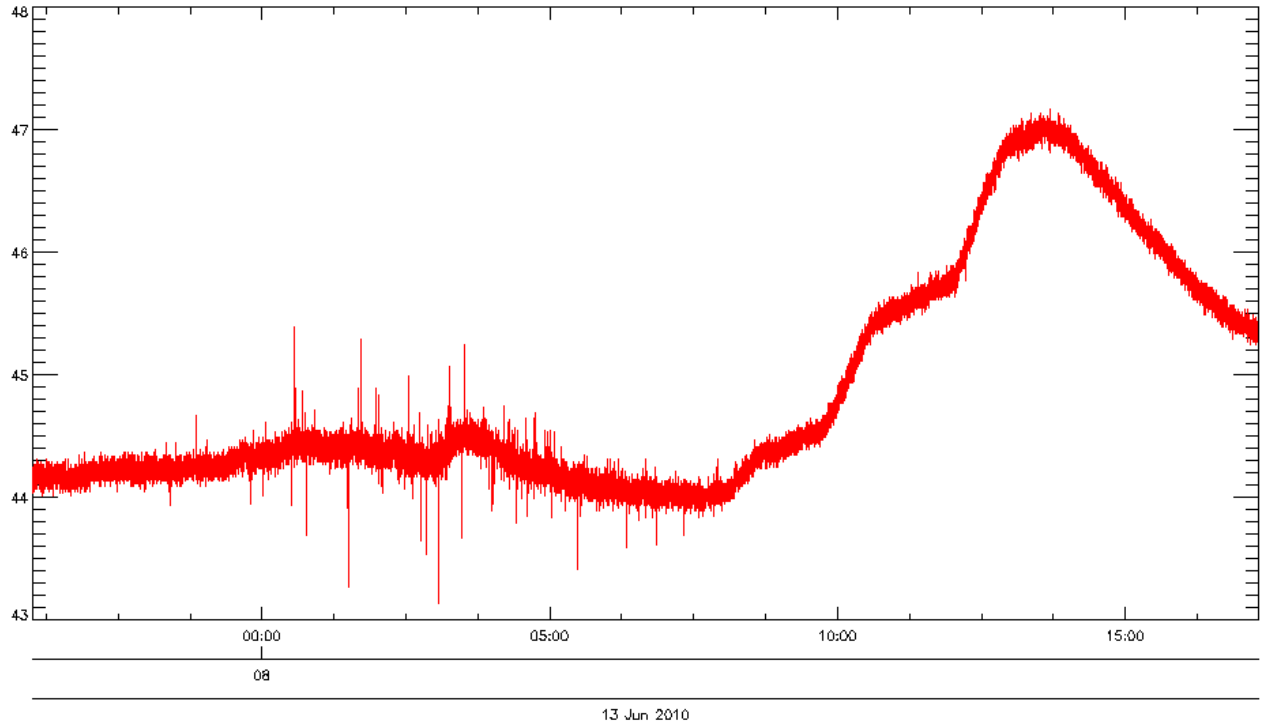


Figure 5.12: LYRA data corresponding to a C1.2 class flare occurring during an epoch between 07:56 and 08:19 on June 13$^{\text{th}}$, 2010.

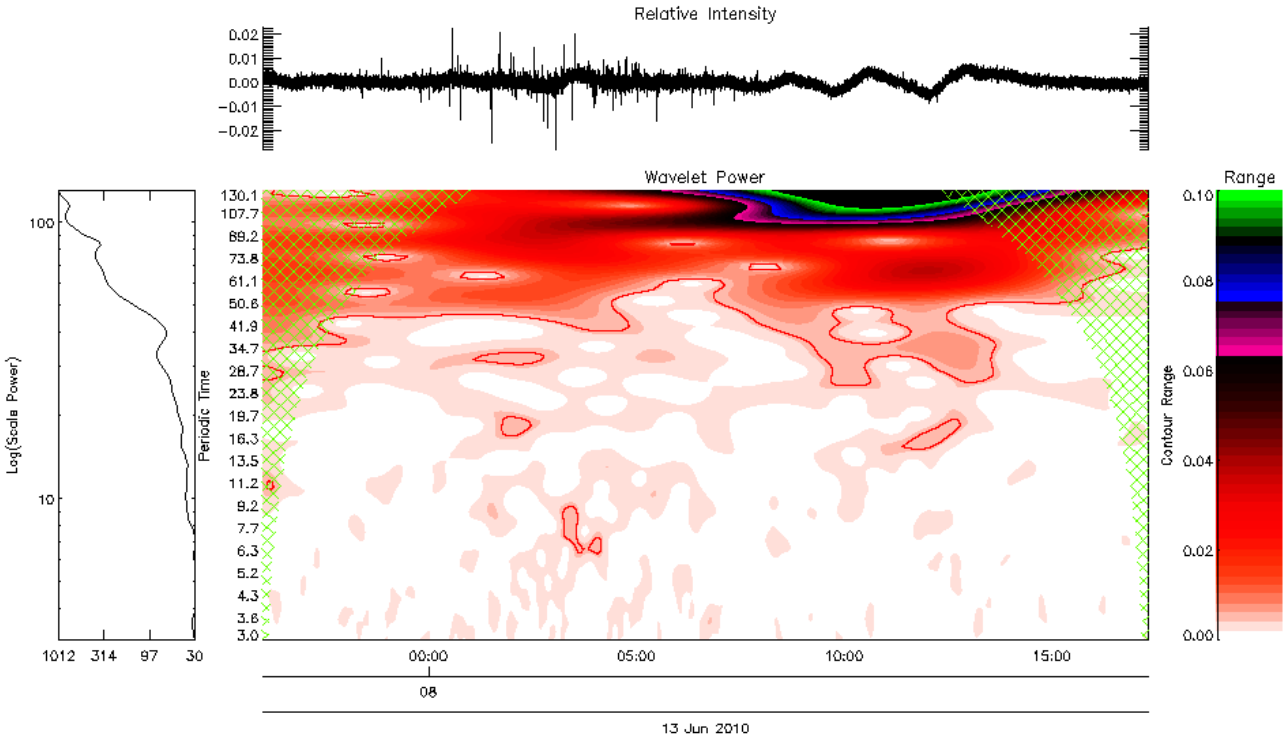


Figure 5.13: Wavelet analysis of a C1.2 class flare occurring during an epoch between 07:56 and 08:19 on June 13th, 2010.

From figure 5.12, a stepping behaviour in the C1.2 flare can be observed. The region of increased noise observed before the event, occurring between approximately 07:58 and 08:07, is caused by the South Atlantic Anomaly, as discussed in section 2.11.3. Frequency detection of this epoch was not classed as a positive detection due to the short duration of the oscillations. Frequency detection was classed as false for the remaining 2 events for the same reason. All epochs flagged as containing false frequency detections will be collectively discussed in further detail in section 6.2.3.

5.4 Analysis over a Longer Interval (01 August 2010 – 20 August 2010)

5.4.1 Introduction

This section discusses analysis performed on LYRA data recorded over 20 days between August 1st, and August 20th, 2010, inclusive. Analysis was performed on this interval of data to detect if any oscillations occurred within flares during this interval, and demonstrate the software tools ability to analyse several days of data. In this section, the results of analysis will be summarised for clarity.

5.4.2 Overview

All flares which occurred during this interval of data are summarised in table 5.5. A plot of the event count during this interval for each class off flare is shown in figure 5.14.

	Day of August, 2010																			
Flare Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
B	2	3	7	5	2	4	3	2	2	3	6	2	4	3	7	8	7	1	0	0
C	2	0	0	0	1	0	0	0	0	0	0	0	0	2	1	1	1	1	0	0
M	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5.9: Flares recorded by Solarsoft occurring between August 1st, and August 20th, 2010, inclusive.

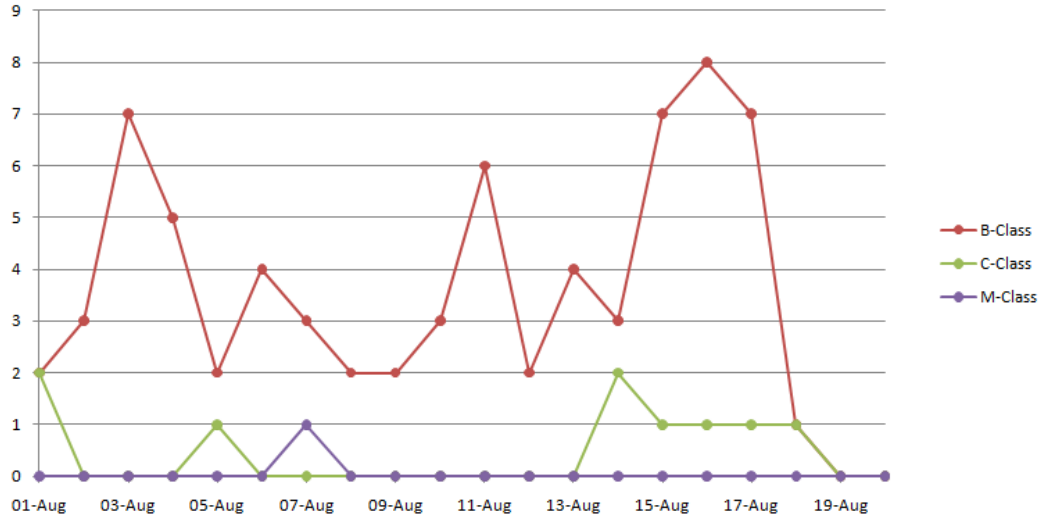


Figure 5.14: A plot of the number of B, C, and M class flares occurring between August 1st, and August 20th, 2010, inclusive.

5.4.3 Interpretation of Analysis

5.4.3.1 Correlation Tables

The epoch correlation table for all epochs in the interval between August 1st, and August 20th, 2010, is shown in table 5.2. This correlation table is not generated as part of analysing multiple days of data due to a current limitation of the analysis software, however a solution to this is discussed in section 6.3.

		Solarsoft		LYRA		FC	
		Yes	No	Yes	No	Yes	No
Solarsoft	Yes	108	.	22	83	8	100
	No	.	869	28	841	9	869
LYRA	Yes	22	28	50	.	10	40
	No	83	841	.	927	5	922
FC	Yes	8	9	10	5	14	.
	No	100	869	40	922	.	963

Table 5.10: Epoch correlation table for the interval of data occurring between August 1st, and August 20th, 2010.

From the epoch correlation table shown in table 5.10 the following can be observed:

- 50 epochs were flagged as containing a flare using LYRA-based flare detection. Of these 50 epochs, 22 corresponded to epochs flagged by Solarsoft.
- In total, 14 epochs were detected as containing the particular oscillatory components under investigation. 8 of these epochs corresponded to flares reported by Solarsoft, and 10 corresponded to flares detected using LYRA-based flare detection. This implies that 4 windows must have been detected by both Solarsoft and LYRA-based flare detection.

5.4.3.2 Flare Detection

The results of flare detection are summarised in figure 5.15. Although flares were recorded by Solarsoft on August 17th and August 18th, the LYRA data corresponding to these events was not available; all data corresponding to the 17th was missing, and the portion containing the flare reported by Solarsoft on the 18th was missing. As a result, the recorded Solarsoft flare count was set to 0 for this day to prevent misinterpretation.

Conclusions of flare detection will be discussed in section 6.2.2.

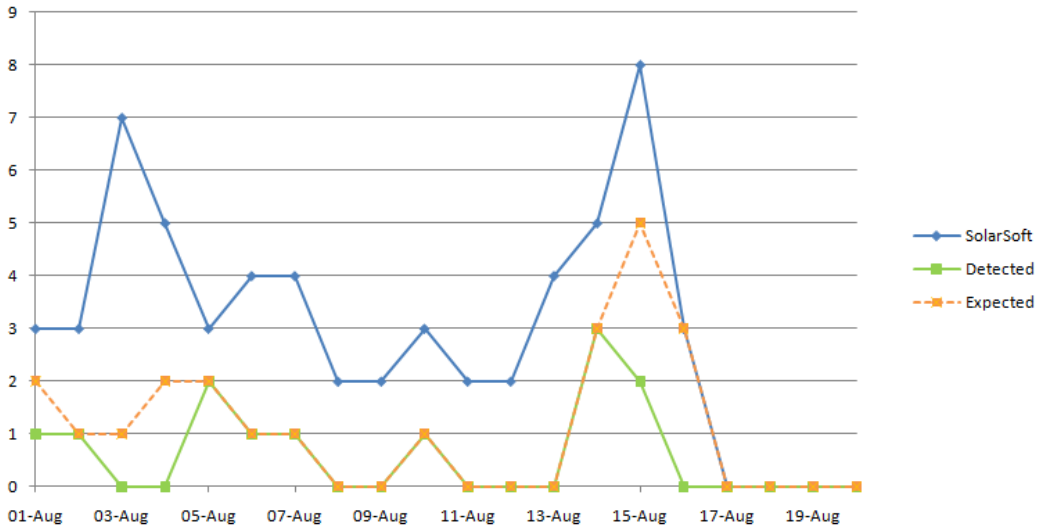


Figure 5.15: Plot of flares recorded by Solarsoft, flares expected to be detected, and detected flares occurring between August 1st, and August 20th, 2010, inclusive.

From figure 5.15 it can be seen that LYRA-based flare detection performed as expected between the 5th and 14th of August, 2010. However, it can be seen that LYRA-based flare detection failed to detect one event on August 1st and 3rd, two events on August 4th, and three events on August 15th and 16th. These events will be shortly discussed below in the following paragraphs — references to 'events' in the following paragraphs refer to solar flare events which were above the threshold required for LYRA-based flare detection.

False Negative (01 August 2010) The event which was not detected during this day was a C3.2 class flare occurring between 07:55 and 09:35. Interestingly, this event occurred within the duration of another C3.2 flare which was observed between 07:24 and 10:25. There is a possibility that the flare that was not detected was an erroneous detection by Solarsoft, or that the two events overlapped and were indistinguishable. The overview for this day is shown in figure 5.16.

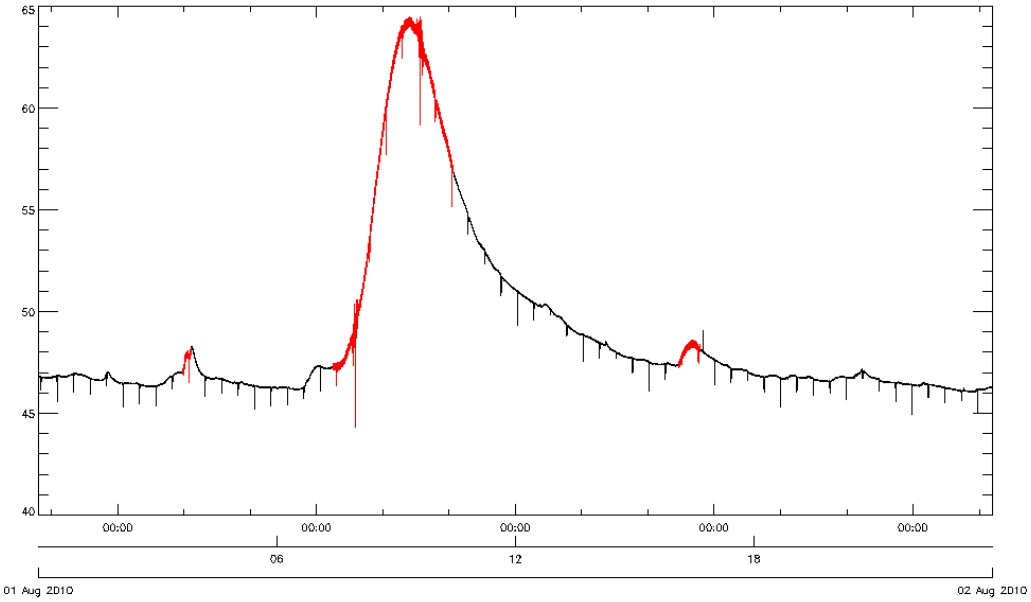


Figure 5.16: Plot of Zirconium channel during August 1st, 2010.

False Negative (03 August 2010) The event which was not detected during this day was a B5.2 flare, occurring between 11:40 and 11:55. There were no unusual events during the flares duration which would contribute to a false negative. This flare is just above the threshold of detection, so the event should have been detected using LYRA-based flare detection. The overview for this day is shown in figure 5.16.

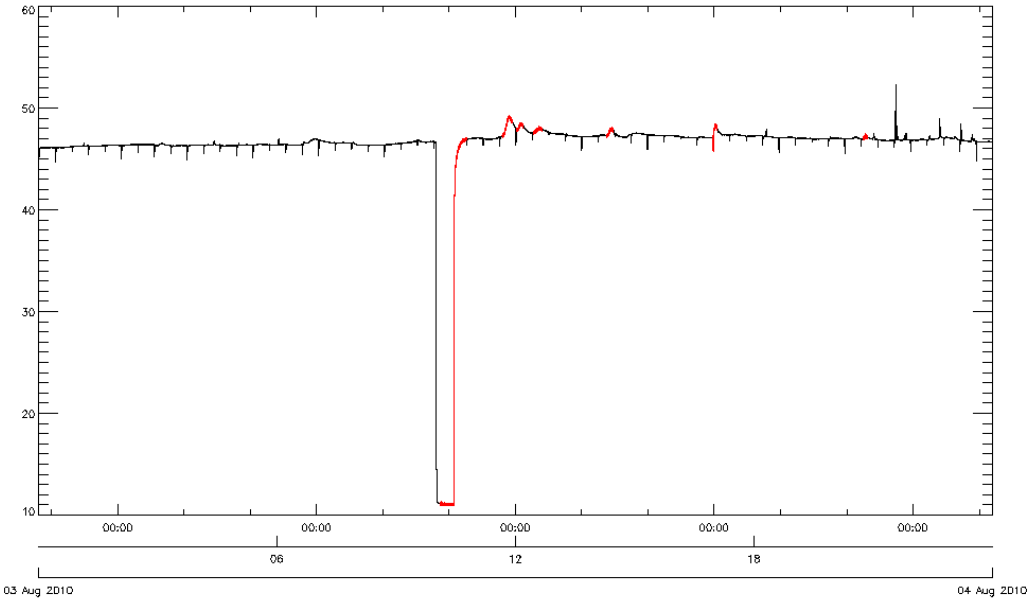


Figure 5.17: Plot of Zirconium channel during August 3rd, 2010.
The dip in signal strength is of instrumental origin.

False Negative (04 August 2010) The two events which were not detected during this day correspond to a B6.6 flare, occurring between 02:51 and 02:58, and a B6.5 flare, occurring

between 09:13 and 09:20. The rising phase of the B6.6 flare occurred outside of an epoch but the peak was contained within the following epoch. In contrast to the B6.6 flare, the rising phase and peak of the B6.5 flare were completely contained within the relevant epoch. So it is likely that the two failed detections should have been detected using LYRA-based flare detection. The overview for this day is shown in figure 5.16.

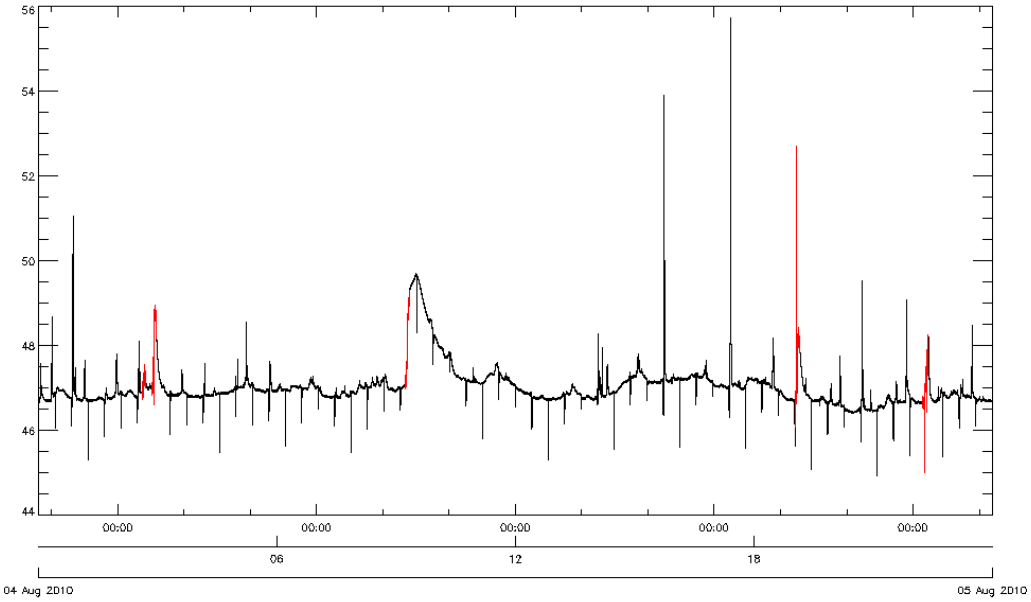


Figure 5.18: Plot of Zirconium channel during August 4$^{\text{th}}$, 2010.

It can be seen from this overview that the day contains a series of particularly large spikes in data that are not recorded by Solarsoft. Upon inspection of the corresponding epochs, these spikes seem to be a type of solar event; an example is shown in figure 5.19. These events will be referred to in further analysis as 'erratic events'; these events will be collectively discussed in further detail in section 6.2.7.

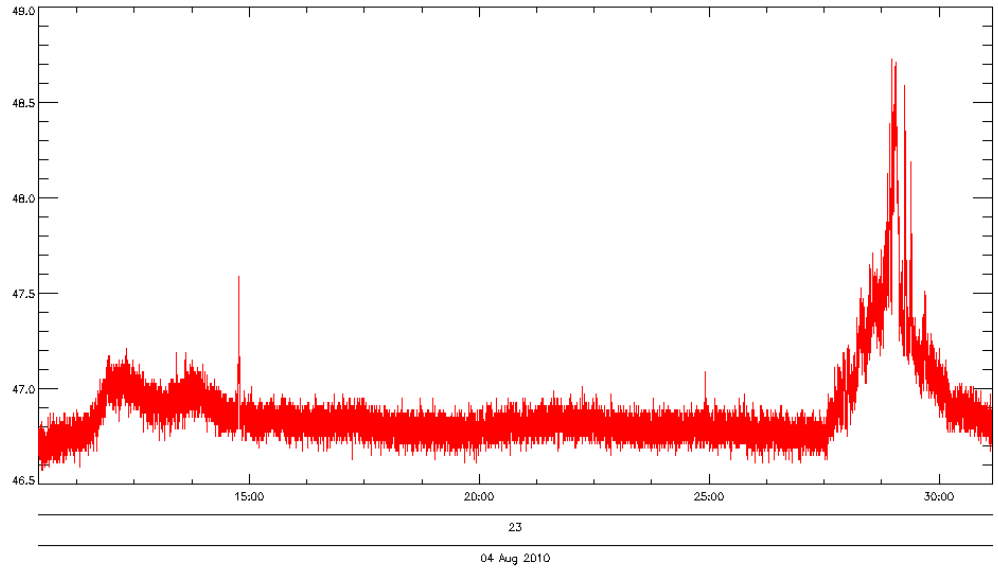


Figure 5.19: Plot of Zirconium channel between 23:10 and 23:31 of August 4th, 2010, demonstrating an erratic event.

False Negative (15 August 2010) There were three events which were not detected during this day. These correspond to a B7.8 flare, occurring between 12:43 and 12:55, a B5.3 flare, occurring between 16:04 and 16:17, and a B6.4 flare, occurring between 18:52 and 19:06. Despite a B7.8 flare not being detected during this day, a B6.7 flare was successfully detected. It is likely that the three failed detections should have been detected by LYRA-based flare detection. The overview for this day is shown in figure 5.16.

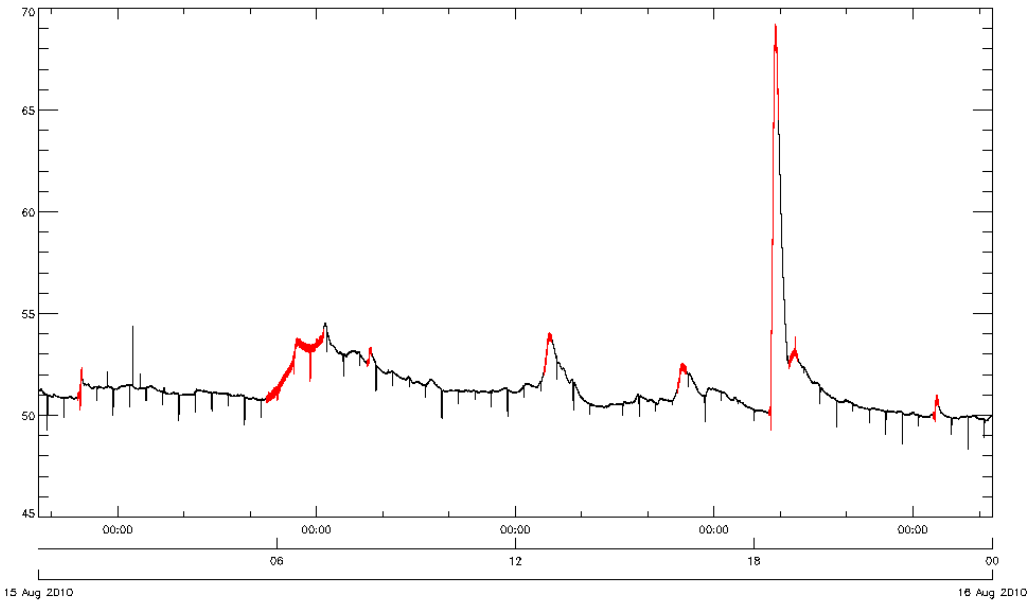


Figure 5.20: Plot of Zirconium channel during August 15th, 2010.

False Negative (16 August 2010) There were three events which were not detected during this day. These correspond to a B5.6 flare, occurring between 11:34 and 11:51, a B7.6 flare, occurring between 15:16 and 15:28, and a C1.4 flare, occurring between 16:34 and 16:51. These events were not detected as LYRA data only recorded data from 00:00 to 10:29 on this day, so the data corresponding to these events was not available.

5.4.3.3 Frequency Detection

The results of frequency detection are summarised in figure 5.21. The epochs containing desired frequency components that are detected by Solarsoft or LYRA respectively are shown in figure 5.22.

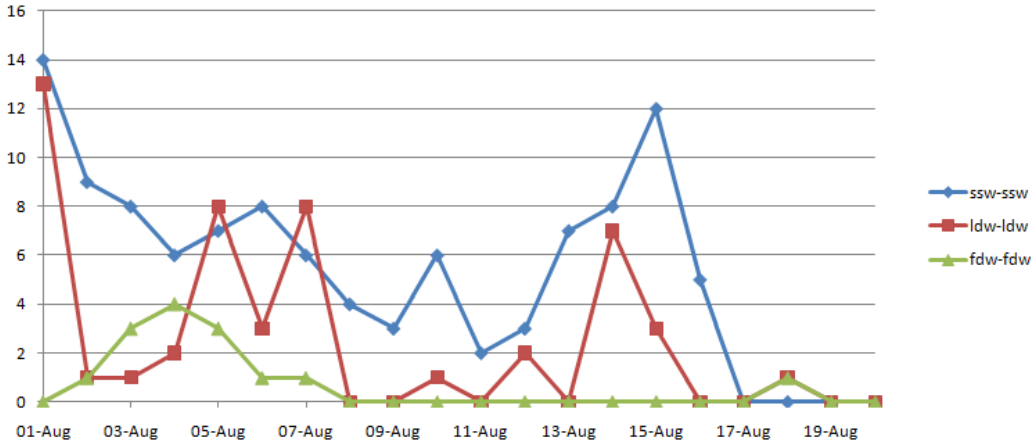


Figure 5.21: Plot of epochs detected as containing flares in analysis of the 20 day interval of August 1st, to August 20th, 2010, inclusive. The number of epochs recorded as containing desired oscillatory components per day is shown in green. The number of epochs containing flares recorded by Solarsoft per day is shown in blue. The number of epochs containing flares, as reported by LYRA-based flare detection for each day is shown in red.

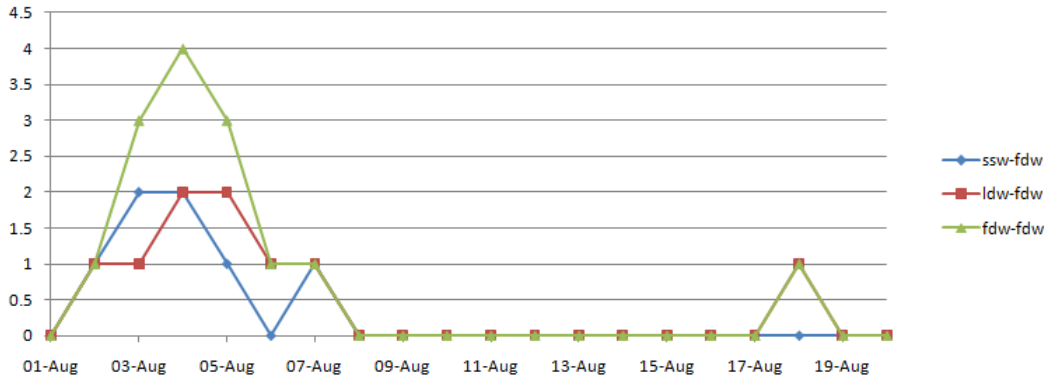


Figure 5.22: Plot of the number of epochs containing desired frequency components in the interval of August 1st, to August 20th, 2010. The total number of epochs detected as containing frequency components are shown in green. The number of epochs containing desired oscillatory components and events recorded by Solarsoft are shown in blue. The number of epochs containing desired oscillatory components and events reported by the LYRA-based flare detection are shown in red.

From figure 5.21 it seems that the number of epochs containing desired oscillatory components follows a trend. Due to the relatively low number of these epochs, they will each be discussed for each day of analysis contained within the interval. In the following paragraphs, frequency detection will refer to detection of the desired frequencies as per configuration of the software as discussed in section 5.1.1. The results of frequency detection will be concluded in section 6.2.3 using three different levels of confidence in the existence of desired oscillatory components; certain, uncertain, false. These different levels of confidence will be determined by the clarity of oscillations and the profile of data in each epoch.

Frequency Detection (2 August 2010) There was one epoch detected as containing frequencies during this day of data, occurring between 04:11 and 04:32, corresponding to a B8.9 flare. The original signal and corresponding wavelet analysis of this epoch are shown in figures 5.23 5.24.

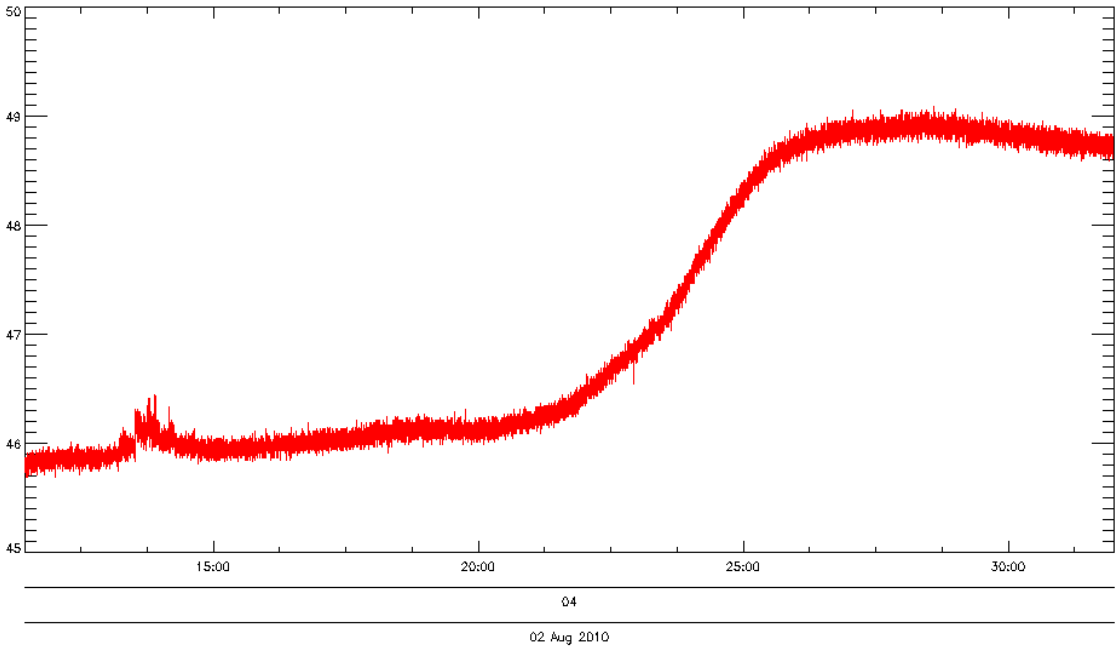


Figure 5.23: The original signal of the epoch between 04:11 and 04:32 on August 2nd, 2010.

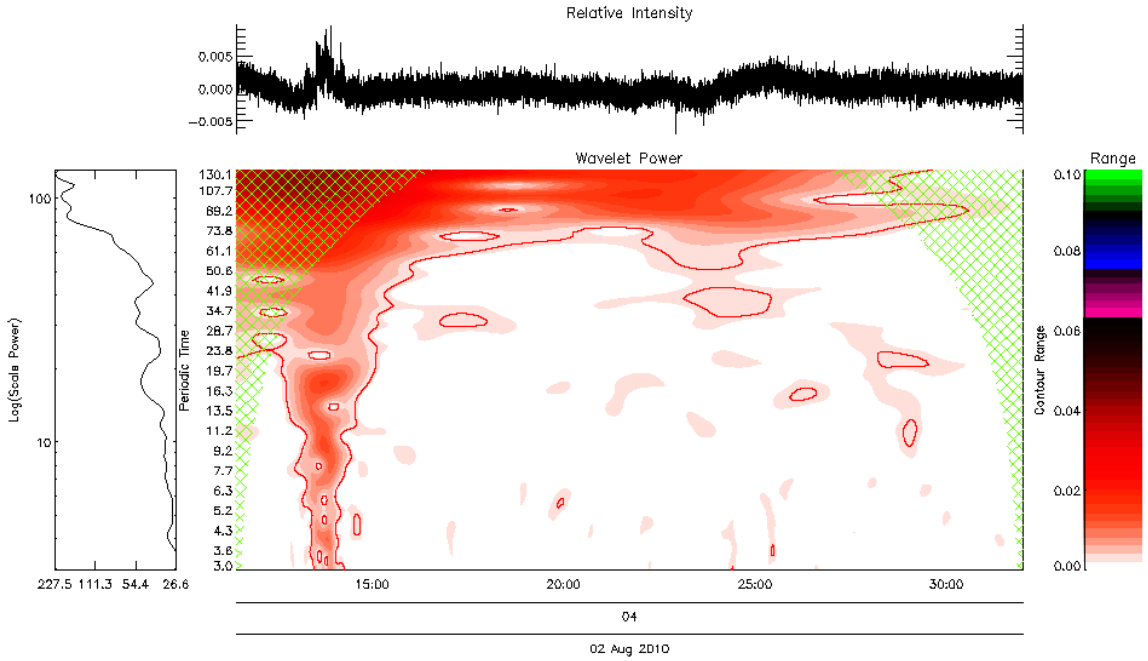


Figure 5.24: Wavelet analysis of a B8.9 class flare occurring in an epoch between 04:11 and 04:32 on August 2nd, 2010.

Analysis of this epoch shows that oscillations seem to be present in the flare, but the detection of frequencies may have been caused by artefacts of the erratic event occurring between 04:13 and 04:15. There is also a possibility that the oscillations seen between 04:20 and 04:30

are created as an artefact of generating the relative intensity of the flare, due to its short duration and the use of a 30 second moving average to determine the trend of the flare. The existence of desired oscillatory components in this epoch is classed as false due to the erratic event present in this signal as well as the short interval of time between the beginning and peak of the flare.

Frequency Detection (3 August 2010) There were three epochs detected as containing frequencies during this day of data, occurring between 09:57 and 10:18, 10:22 and 10:43, and 21:32 and 21:52. The two epochs beginning at 09:57 and 10:22 were caused by an instrumental event. As both events correspond to the beginning and end of this event with similar results in analysis, only the first will be analysed. The original signal, and corresponding wavelet analysis of the first epoch are shown in figures 5.25 and 5.26.

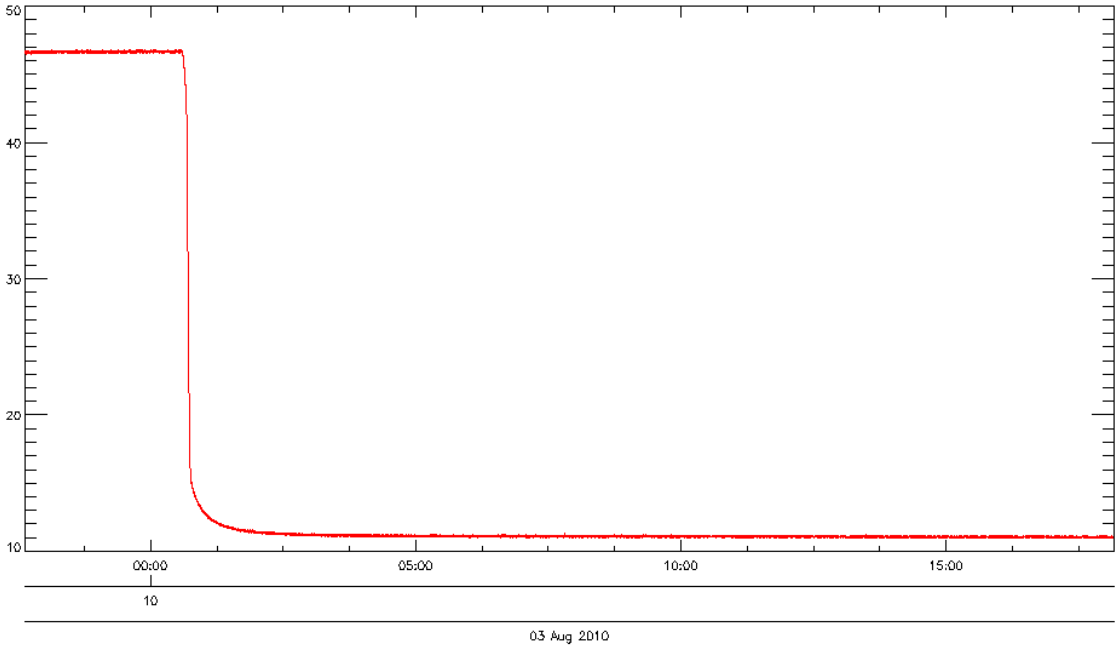


Figure 5.25: A plot of data from the Zirconium channel of an epoch occurring between 04:11 and 04:32, containing an instrumental event on August 3th, 2010.

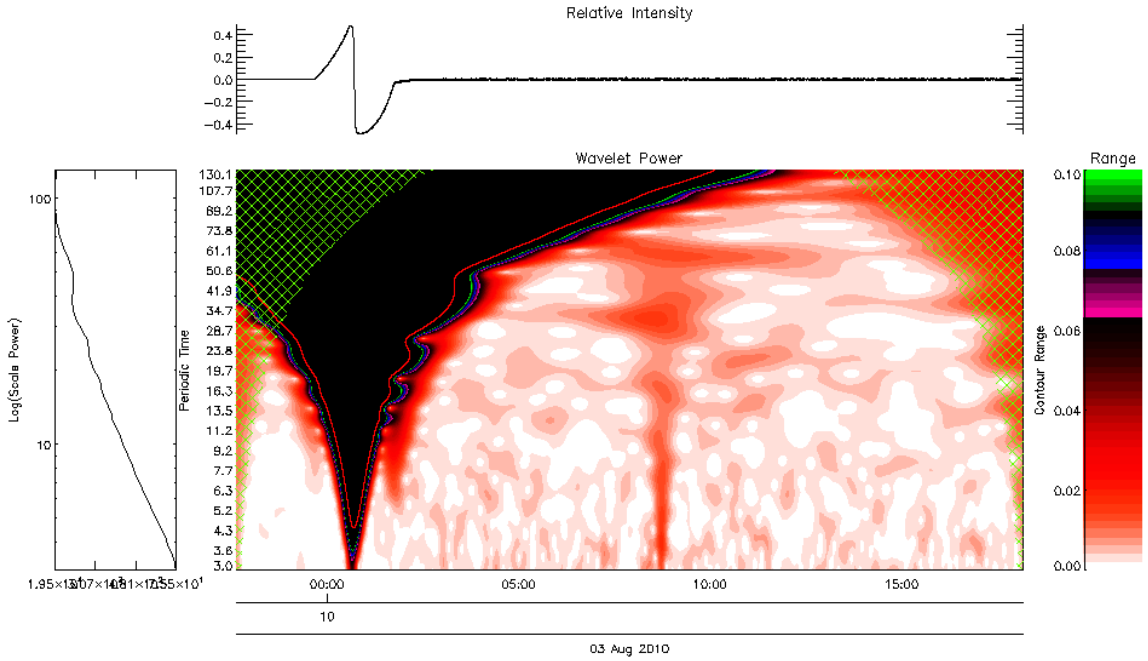


Figure 5.26: Wavelet analysis of an epoch occurring between 04:11 and 04:32, containing an instrumental event on August 3$^{\text{th}}$, 2010.

Analysis of this epoch demonstrates the instrumental origins of the event as a drop of X-rays this severe is impossible. Analysis also demonstrates the effect of this instrumental event on wavelet analysis.

The original signal and wavelet analysis of the third epoch detected as containing frequencies are shown in figures 5.27 and 5.28.

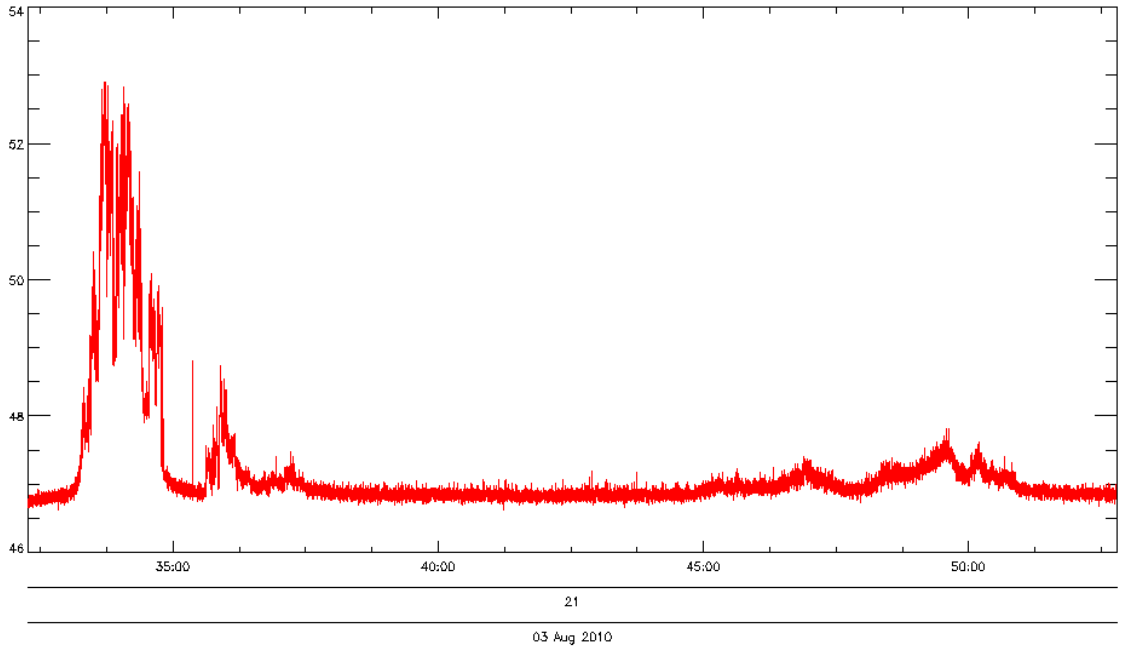


Figure 5.27: Original signal of the epoch occurring between 21:32 and 21:52 on August 3rd, 2010.

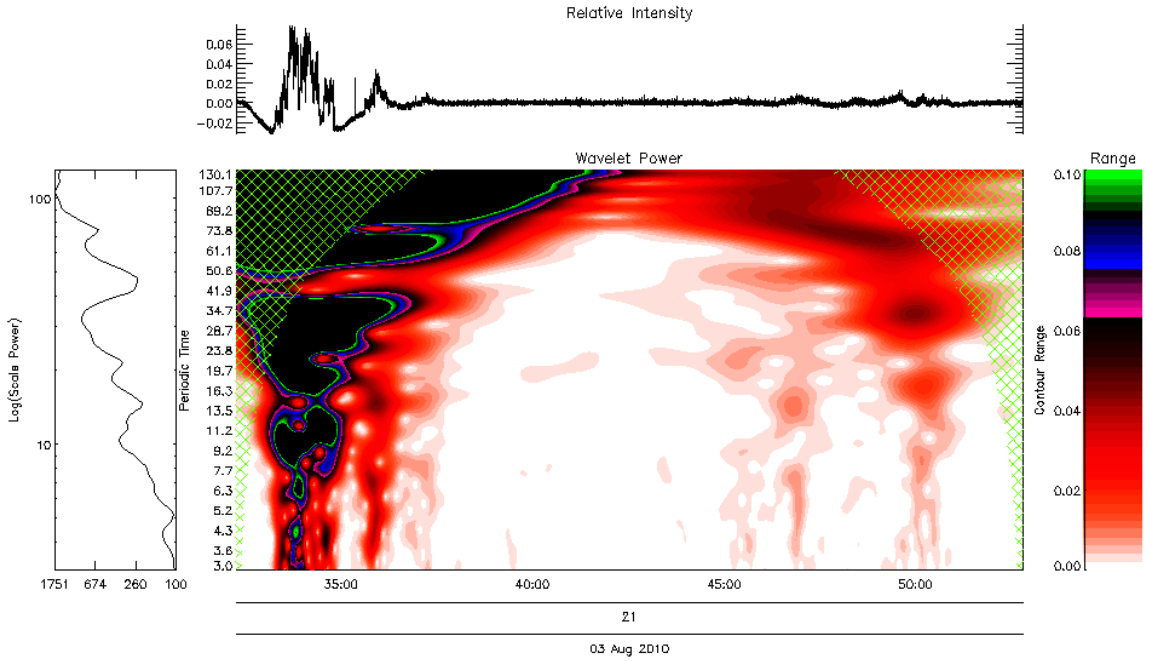


Figure 5.28: Wavelet analysis of the epoch occurring between 21:32 and 21:52 on August 3rd, 2010.

Inspection of the data presented in figures 5.27 and 5.28 show that the signal contains an erratic event with a much larger amplitude which occurs between 21:32 and 21:35, followed by a weaker, similar signal occurring between 21:35 and 21:37. However, oscillations seem to be

present in both raw data and wavelet analysis of data between 21:45 and 21:51 of this day. The existence of desired oscillatory components in the two epochs corresponding to the instrumental event are classed as false. The existence of desired oscillatory components in the third epoch is classed as uncertain due to the erratic event.

Frequency Detection (4 August 2010) There were four epochs detected as containing frequencies during this day of data, occurring between 09:07 and 09:27, 15:43 and 16:04, 17:23 and 17:43, and 22:20 and 22:41. The first epoch corresponds to a B6.5 flare. The original signal of this epoch and corresponding wavelet analysis are shown in figures 5.29 and 5.30.

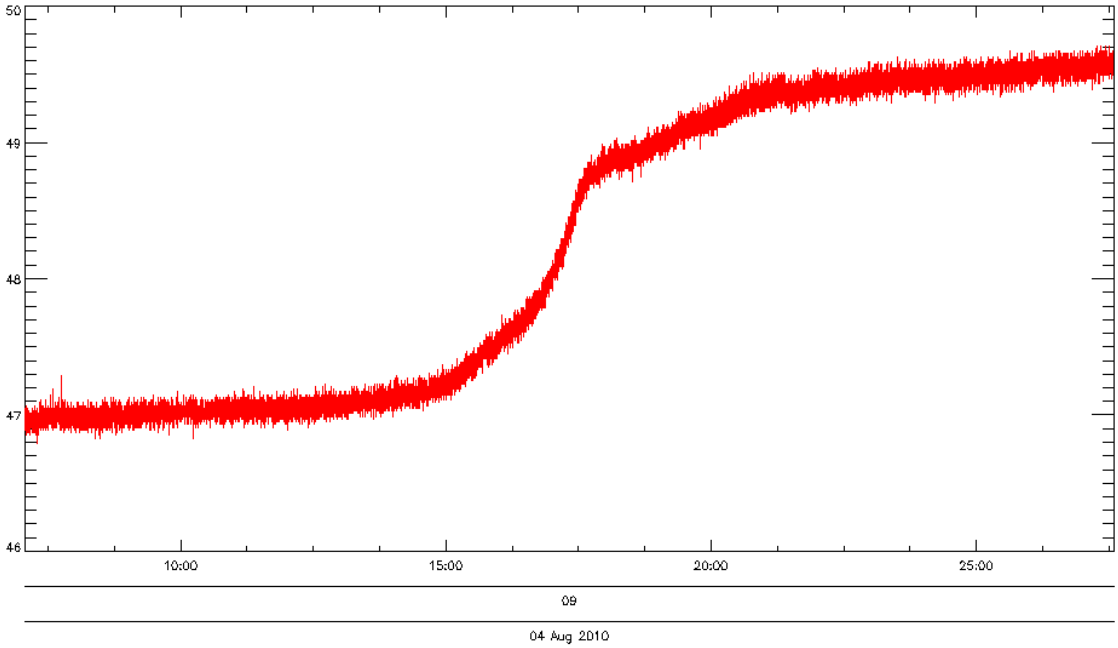


Figure 5.29: Original signal of the epoch occurring between 09:07 and 09:27 on August 4th, 2010.

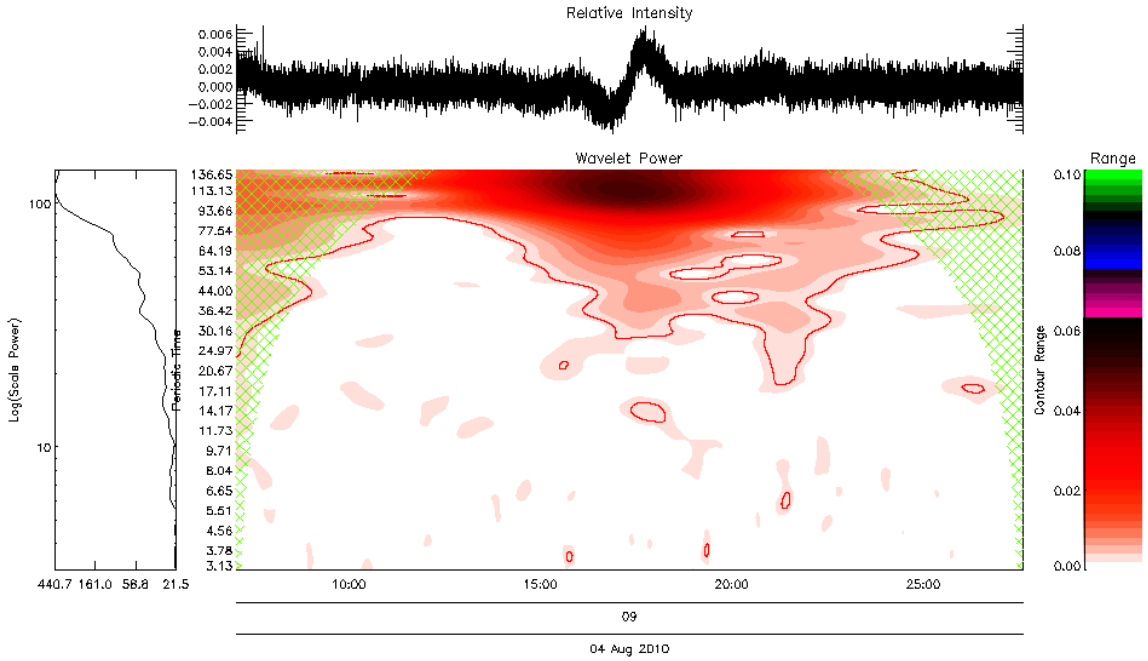


Figure 5.30: Wavelet analysis of the epoch occurring between 09:07 and 09:27 on August 4th, 2010.

Analysis of the relative intensity of the flare shown in figure 5.30 indicates that the oscillations may be caused by the short duration of the rising phase of the flare.

The remaining three epochs contain erratic events similar to the event seen in figure 5.27 as demonstrated in figure 5.31.

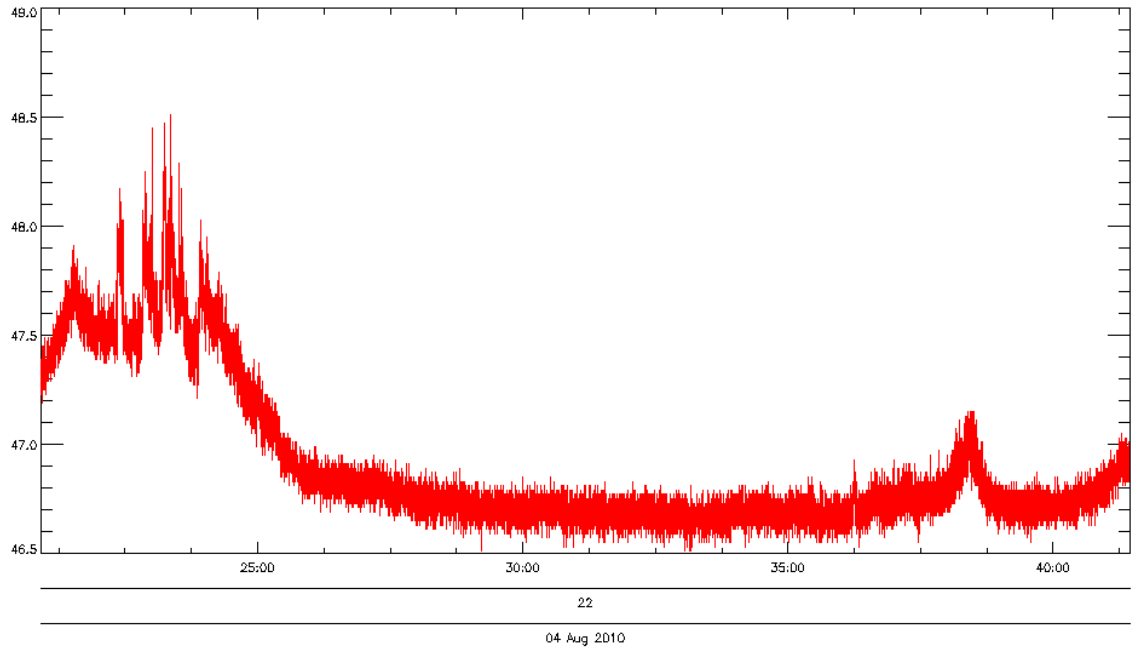


Figure 5.31: Original signal of the epoch occurring between 22:20 and 22:41 on August 4th, 2010.

The existence of desired oscillatory components in the first epoch is classed as false due to the short duration of the flare. The existence of frequency components in the remaining three epochs detected during this day are classed as uncertain.

Frequency Detection (5 August 2010) There were three epochs detected as containing frequencies during this day of data, occurring between 00:00 and 00:20, 00:49 and 01:10, and 01:39 and 01:59. Each of these epochs contained events similar to the event seen in figure 5.27. As these events will be collectively discussed later, analysis of this day will not be discussed further in this section. The existence of desired oscillatory components for the three epochs detected during this day are classed as uncertain.

Frequency Detection (6 August 2010) There was one epoch detected as containing frequencies during this day of data, occurring between 08:15 and 08:35. This event seems to be of instrumental origin. The Zirconium data during this epoch is shown in figure 5.32. The existence of desired oscillatory components for this epoch is classed as false due to the behaviour of the event.

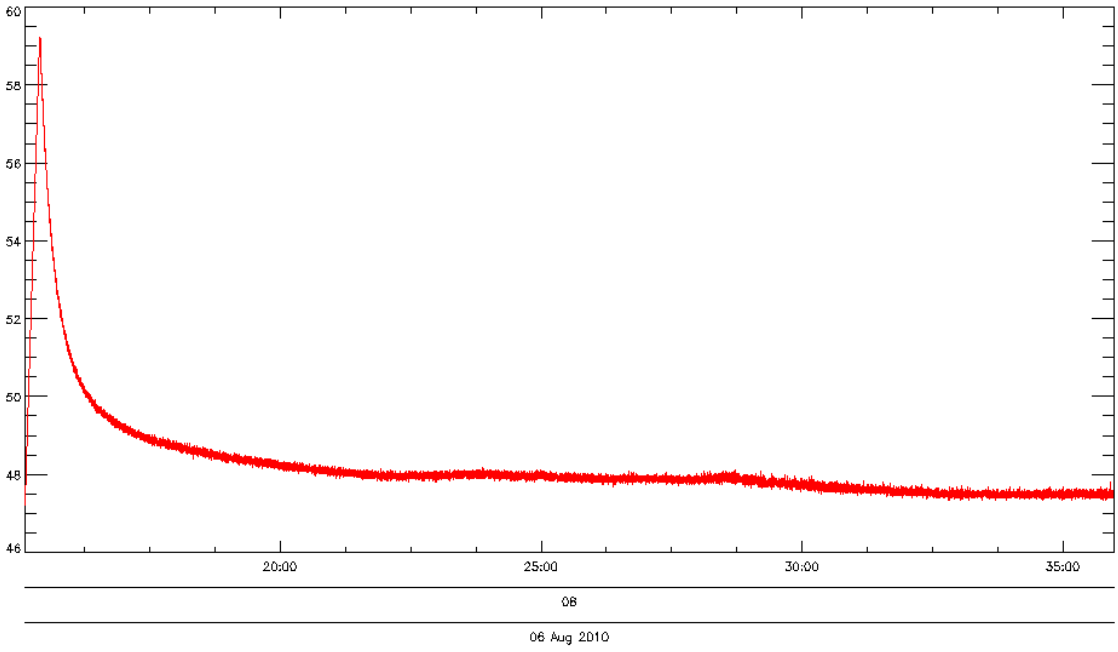


Figure 5.32: Original signal of the epoch occurring between 08:15 and 08:35 on August 6th, 2010.

Frequency Detection (7 August 2010) There was one epoch detected as containing frequencies during this day of data, occurring between 18:09 and 18:37. This event corresponds to an M1.0 flare. The original signal of this epoch and corresponding wavelet analysis are shown in figures 5.33 and 5.34.

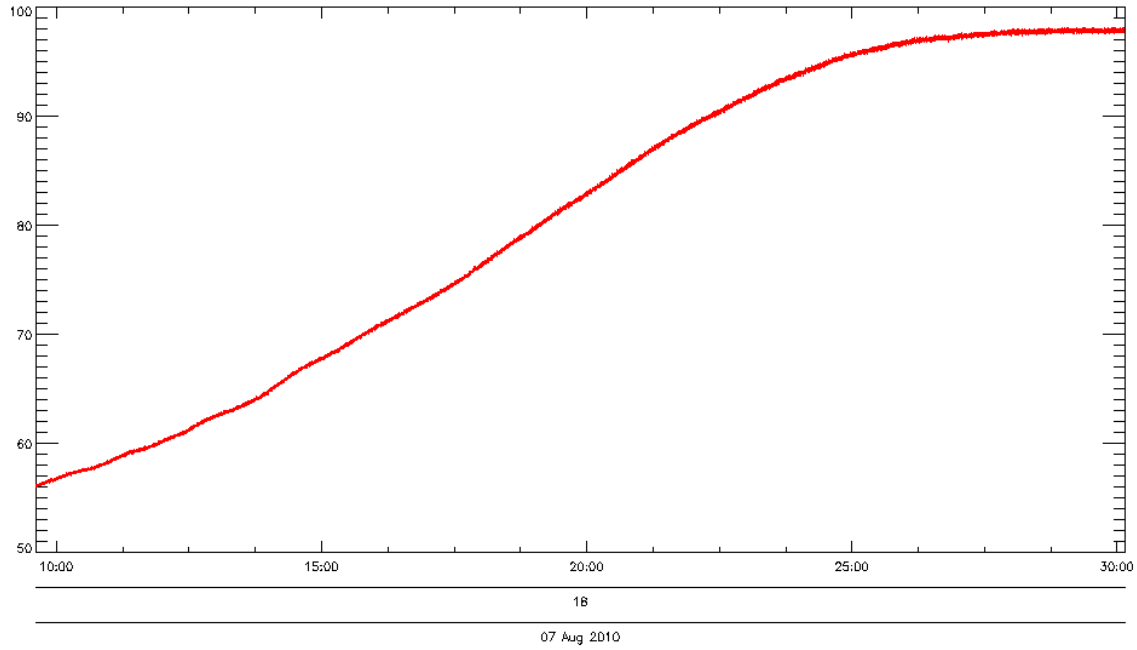


Figure 5.33: Original signal of the epoch occurring between 18:09 and 18:37 on August 7th, 2010.

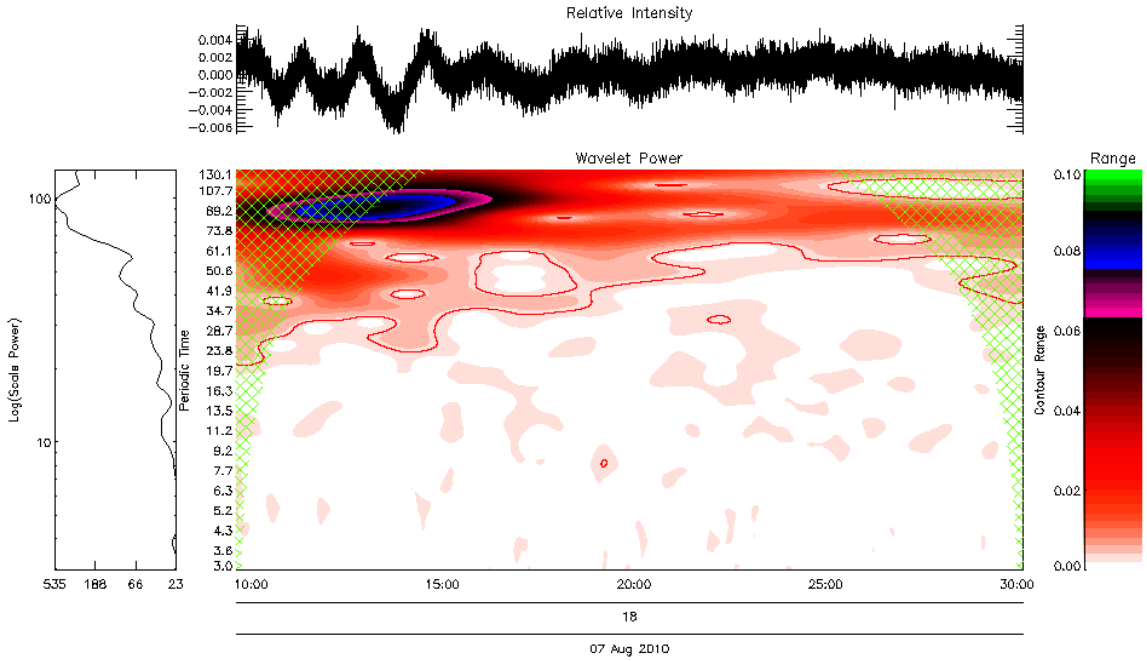


Figure 5.34: Wavelet analysis of the epoch occurring between 18:09 and 18:37 on August 7th, 2010.

The existence of desired oscillatory components for this epoch is classed as certain due to the clarity of the oscillations in wavelet analysis and the lack of events in the data that may cause artefacts in frequency detection.

Frequency Detection (18 August 2010) There was one epoch detected as containing frequencies during this day of data, occurring between 19:12 and 19:32. During this day, no events were recorded by Solarsoft. The original signal of this epoch and corresponding wavelet analysis are shown in figures 5.33 and 5.30.

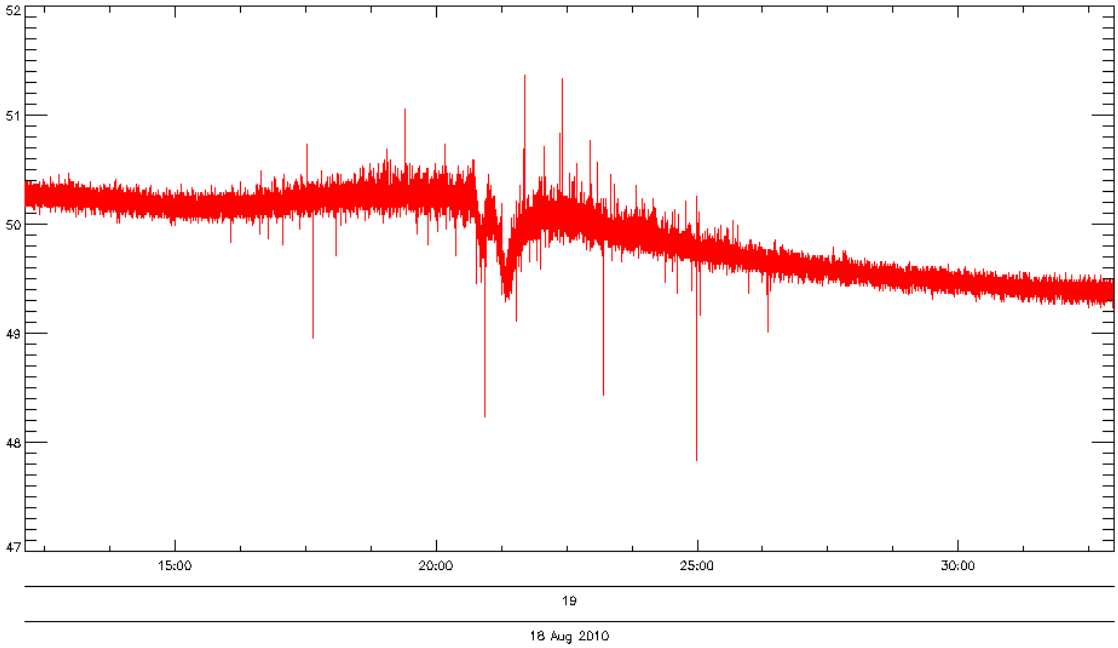


Figure 5.35: Original signal of the epoch occurring between 19:12 and 19:32 on August 18th, 2010.

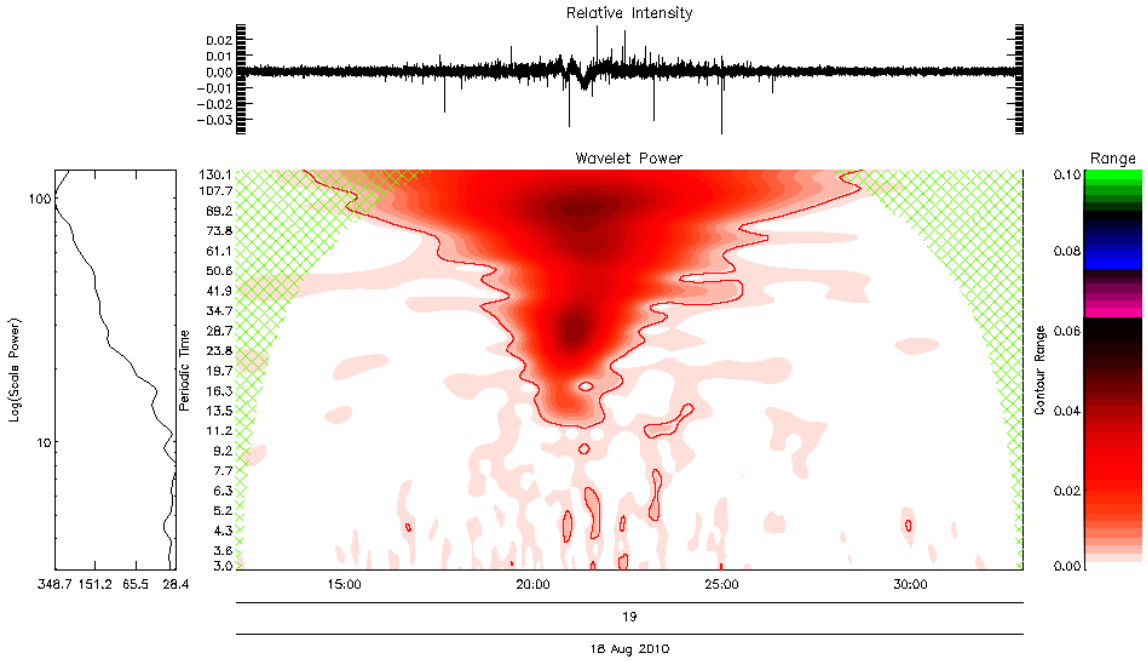


Figure 5.36: Wavelet analysis of the epoch occurring between 19:12 and 19:32 on August 18th, 2010.

Inspection of the original signal and wavelet analysis show that this event occurred while the LYRA instrument was being affected by the South Atlantic Anomaly. An oscillation can be seen to occur between 19:20 and 19:22 but this event seems to be instrumental, and has too short a duration to reliably contain desired oscillations. The existence of desired oscillatory components for this epoch is classed as false.

5.5 Survey of Some Pathological Intervals

5.5.1 Introduction

As LYRA is an ongoing scientific experiment, unexpected events can occur. A short list of some events which have been detected during this research project are discussed in this section to help illustrate the difficulty in analysis of this data.

5.5.2 Atmospheric Effects (05 October 2010)

During the month of November, LYRA data was affected by absorption of radiation detected by the Zirconium channel. In communications with J.J. Zender, it was suggested that the effects present in data were due to absorption of radiation by the Ionosphere of the Earth. The Ionosphere of the Earth begins at approximately 50 km and extends hundreds of kilometres above the Earth's surface. It contains many free electrons which can absorb short wavelength E-M radiation, which is typically the cause of the Aurora Borealis.

To demonstrate the effect observed in LYRA data, two epochs affected by this absorption are shown below in figures 5.37 and 5.39. The wavelet analysis corresponding to these epochs are shown in figures 5.38 and 5.40, respectively. Three SWAP images demonstrating the effect on the SWAP detector are shown in figure 5.41.

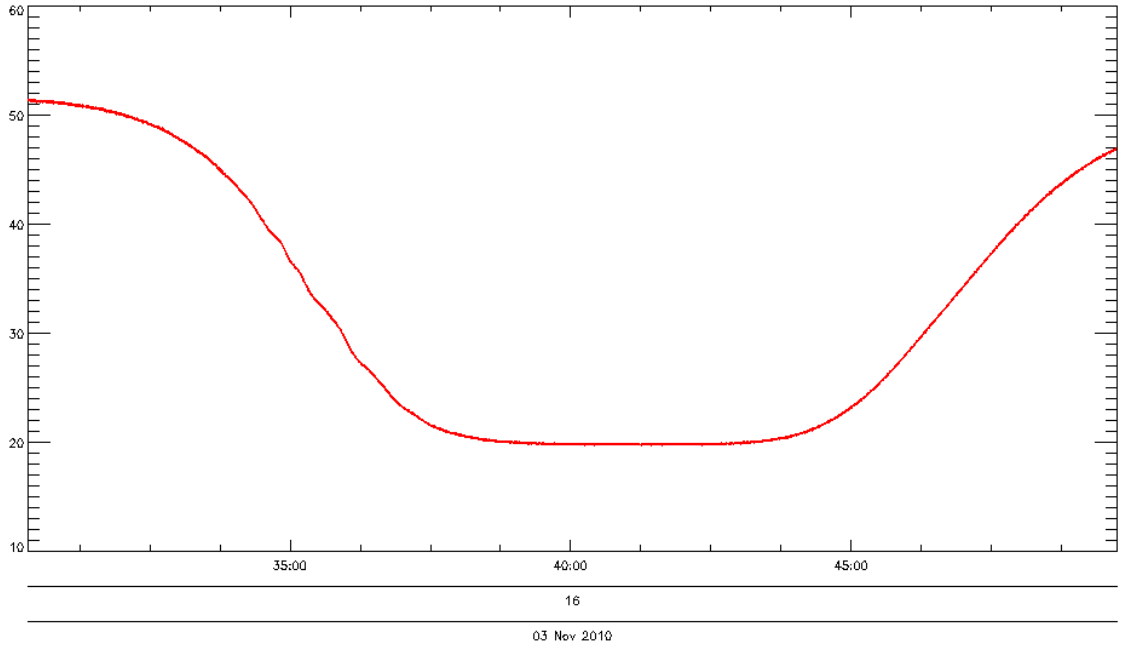


Figure 5.37: Original signal of the epoch occurring between 16:30 and 16:49 on November 3rd, 2010.

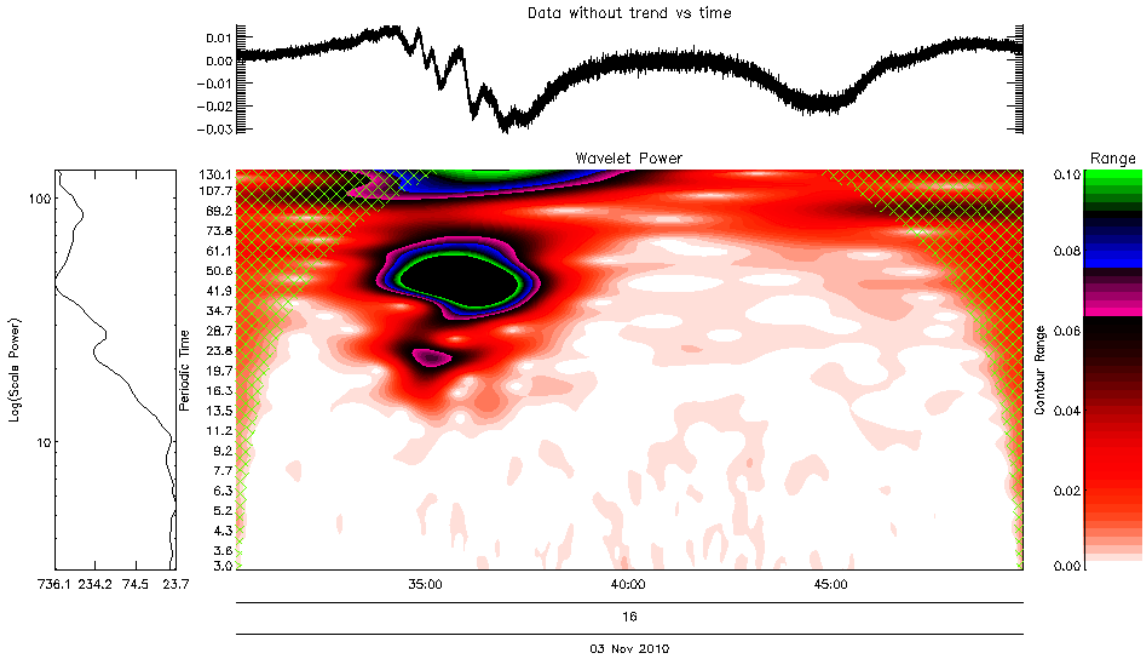


Figure 5.38: Original signal of the epoch occurring between 16:30 and 16:49 on November 3rd, 2010.

From figure 5.37, the decrease in signal amplitude can be clearly seen. Interestingly, during the decay of detector strength, oscillations can be seen in the signal. These oscillations can also be seen in the relative intensity of the signal shown in wavelet analysis; see figure 5.38. These oscillations have a peak strength with a periodic time of approximately 45 s.

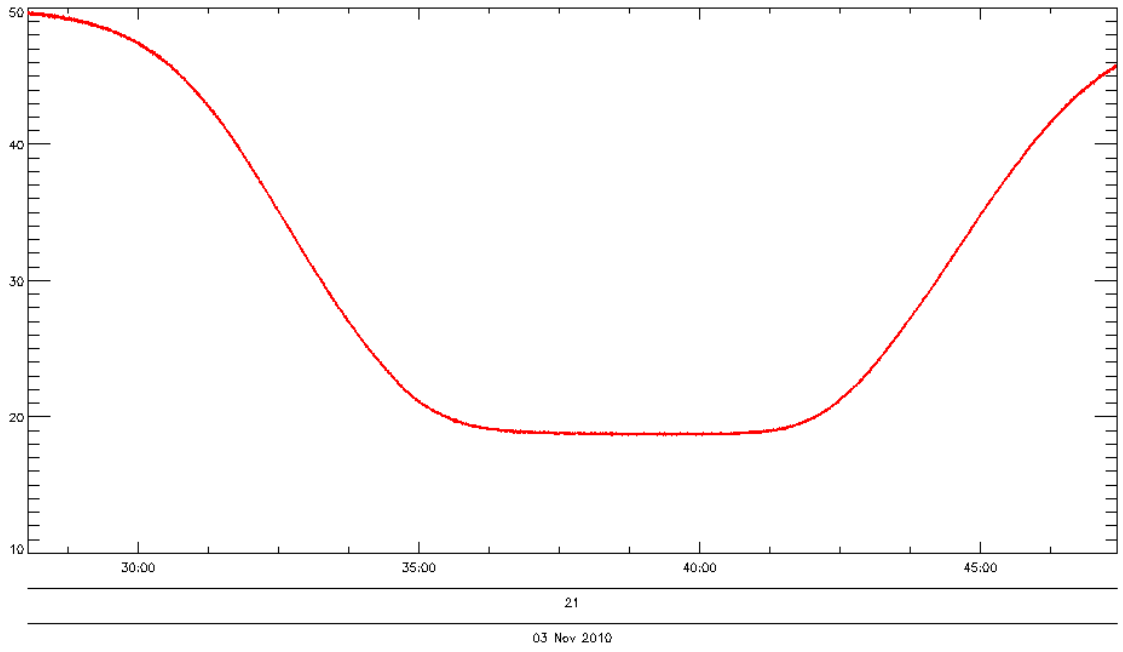


Figure 5.39: Original signal of the epoch occurring between 21:28 and 21:47 on November 3rd, 2010.

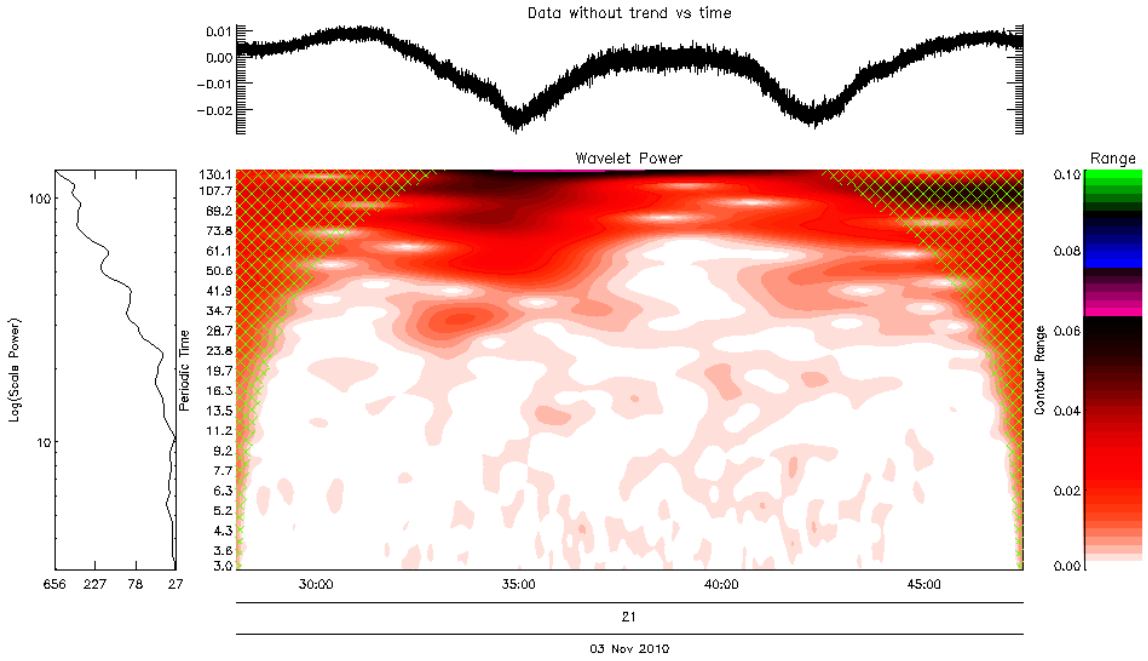


Figure 5.40: Original signal of the epoch occurring between 21:28 and 21:47 on November 3rd, 2010.

From figure 5.39, the decay of signal amplitude can be seen as in figure 5.37, however, oscillations can not be seen during the decay phase of this epoch in the raw signal or wavelet analysis. There is an oscillation due to the overall trend of the signal, but this is ignored.

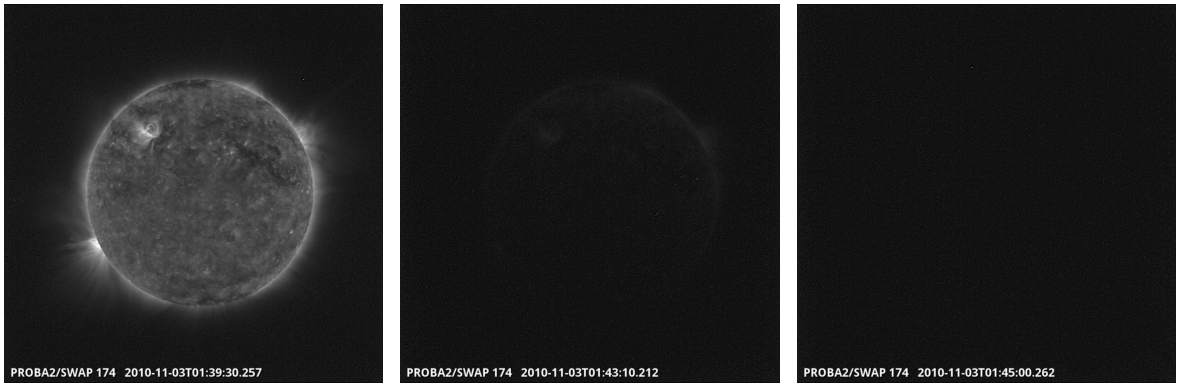


Figure 5.41: Demonstration of signal depredation seen in SWAP images taken at 01:39, 01:43, and 01:45 on November 3rd, 2010.

The reduction in signal intensity can also be seen in the SWAP images shown in figure 5.41. In the first image, a particularly bright plume can be seen at the lower-right region of the Sun. In the second image, this plume is more difficult to distinguish than plumes visible in the upper regions of the Sun, as seen in the first image, which suggests that this image was taken during the decay in detector strength. Unfortunately, due to the time constraints for analysis, these effects could not be further investigated.

5.5.3 Instrumental Events

During calibrations of the LYRA instrument or off-pointing of the satellite, LYRA data can sometimes be affected and include artefacts. An example of two artefacts are shown in figures 5.42 and 5.43. These two events will not be discussed further as the events did not trigger any false positive detections in analysis, but they demonstrate some of the unexpected behaviour that can exist within LYRA data which had to be taken into account during development of the analysis software.

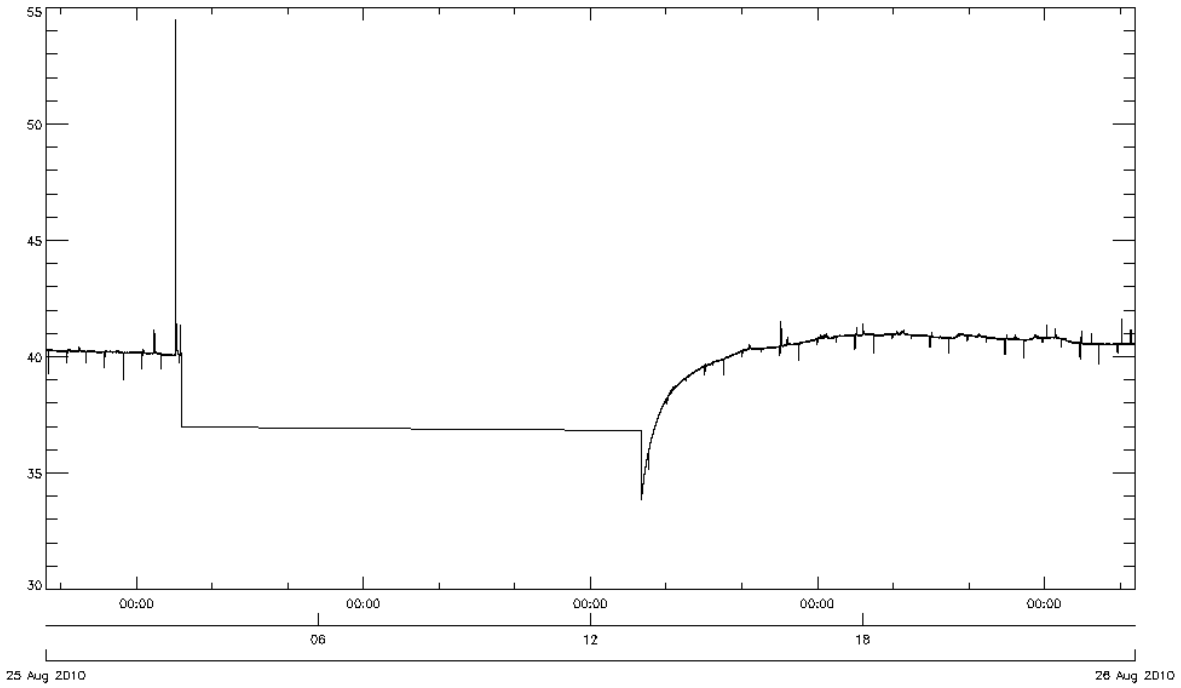


Figure 5.42: Plot of the Zirconium channel of the LYRA instrument during August 25$^{\text{th}}$, 2010.

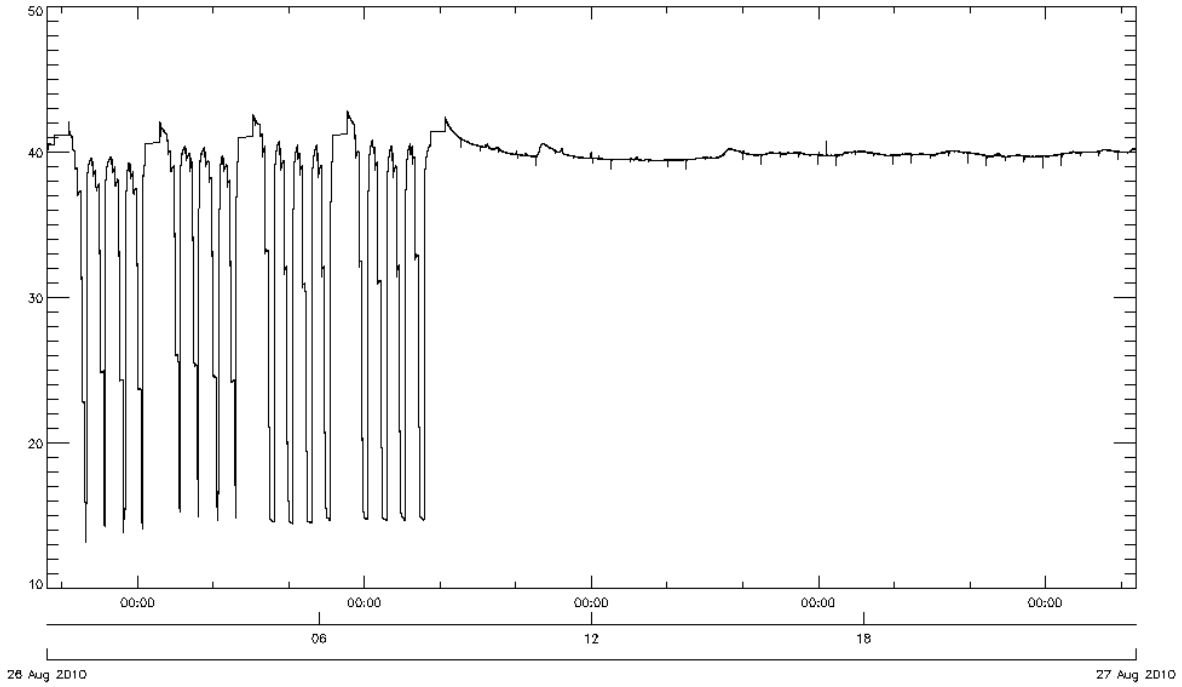


Figure 5.43: Plot of the Zirconium channel of the LYRA instrument during August 26$^{\text{th}}$, 2010.

5.6 Partial Analysis of March, 2011

5.6.1 Introduction

This section briefly demonstrates some analysis performed on LYRA data from March, 2011. The epochs discussed in this section were chosen to demonstrate other flares detected as containing desired oscillatory components. These epochs are demonstrated due to the limited number of epochs flagged as reliably containing desired oscillations in section 5.4. Flare detection can not be discussed for this interval of LYRA data (March, 2011) because, at the time of analysis, Solarsoft archives did not contain any records of events during this period. The unavailability of Solarsoft records means that all epochs analysed were detected as containing flares using LYRA-based flare detection; analysis of this data would not have been otherwise possible within a reasonable time-frame as the only alternative would be to perform frequency analysis on all epochs.

5.6.2 Overview

In this section, some epochs detected as containing frequency components during March, 2011, are presented. The epochs discussed in this section are selectively chosen to demonstrate some events flagged as containing oscillatory components. The results of frequency analysis will be labelled similarly to results discussed in section 5.4.3.3 as certain, and uncertain. Epochs

that were flagged as containing a false detection of frequency components are omitted as the intention of this section is to demonstrate positive detections.

5.6.3 Demonstration of Epochs Containing Desired Frequency Components

5.6.3.1 March 7th, 2011

The epoch shown in this section occurs between 19:43 and 20:04 of March 7th, 2011. The event shown in this epoch is particularly strong, but can not be accurately classified due to the absence of Solarsoft archives for this day. The original signal of this epoch and corresponding wavelet analysis are shown in figures 5.44 and 5.45.

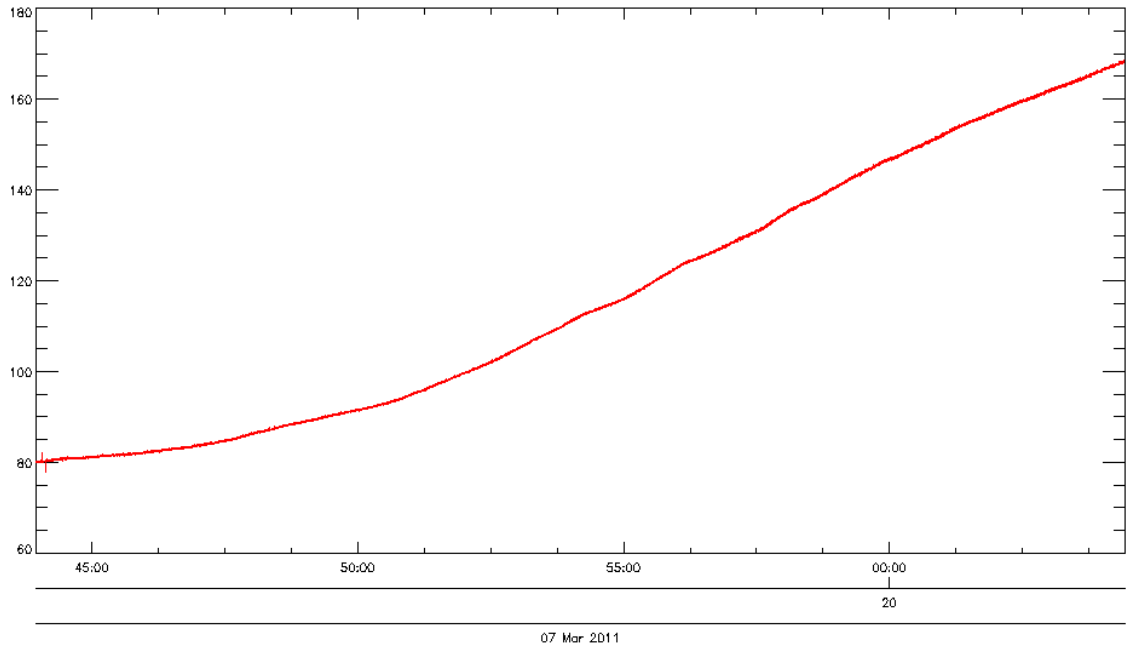


Figure 5.44: Original signal of the epoch occurring between 19:43 and 20:04 on March 7th, 2011.

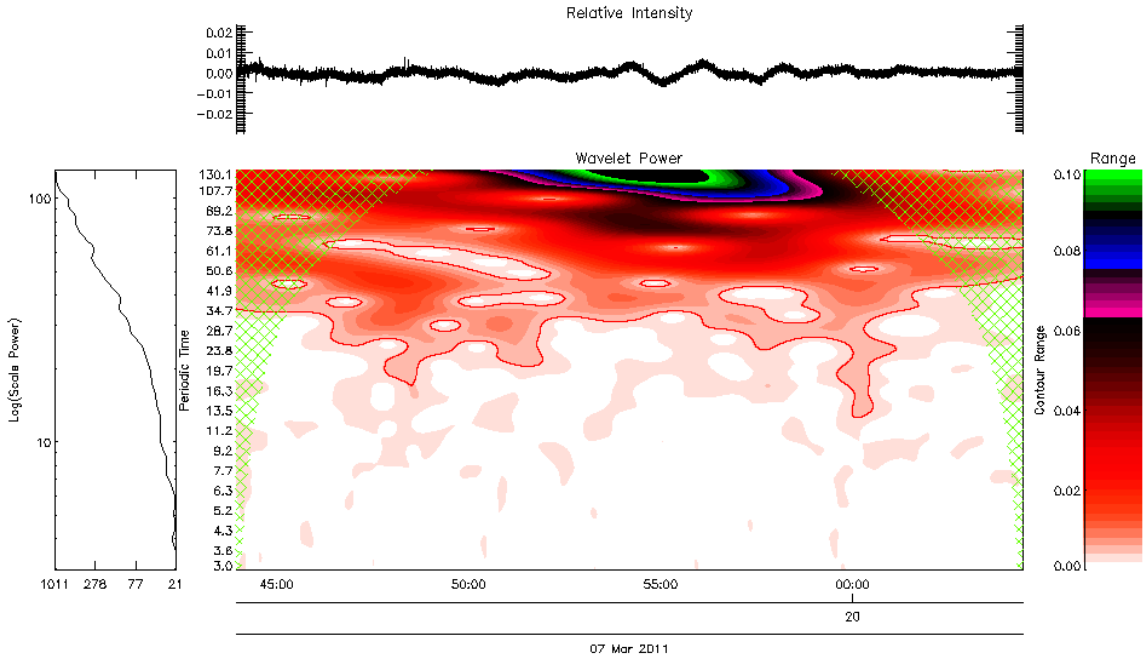


Figure 5.45: Wavelet analysis of the epoch occurring between 19:43 and 20:04 on March 7$^{\text{th}}$, 2011.

The oscillations present in the wavelet transform of this epoch can be seen in the original signal and relative intensity of the signal. Unfortunately, due to the Solarsoft archive not containing records for this interval of data, the strength of the event can not be determined. The existence of desired oscillatory components for this epoch is classed as certain.

5.6.3.2 March 8th, 2011

Two epochs detected as containing frequencies during this day of data, occurring between 03:35 and 03:55, and 10:37 and 10:58, are shown below. The original signal of the first epoch, beginning at 03:35, and corresponding wavelet analysis are shown in figures 5.46 and 5.47. The original signal of the second epoch, beginning at 10:37, and corresponding wavelet analysis are shown in figures 5.48 and 5.49.

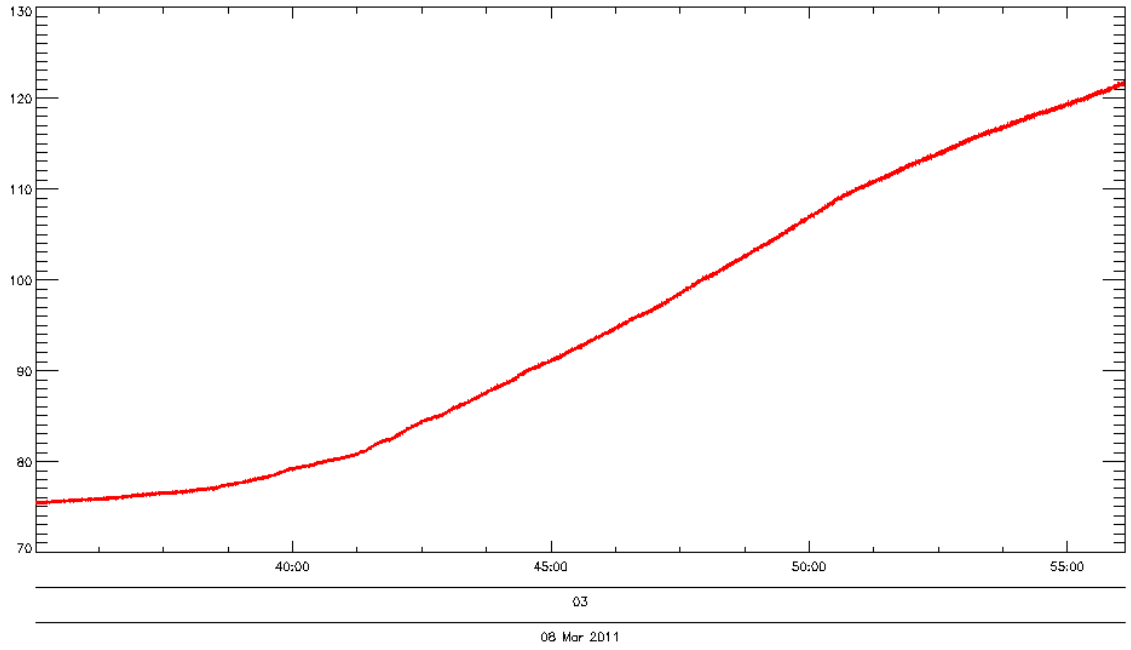


Figure 5.46: Original signal of the epoch occurring between 03:35 and 03:55 on March 8th, 2011.

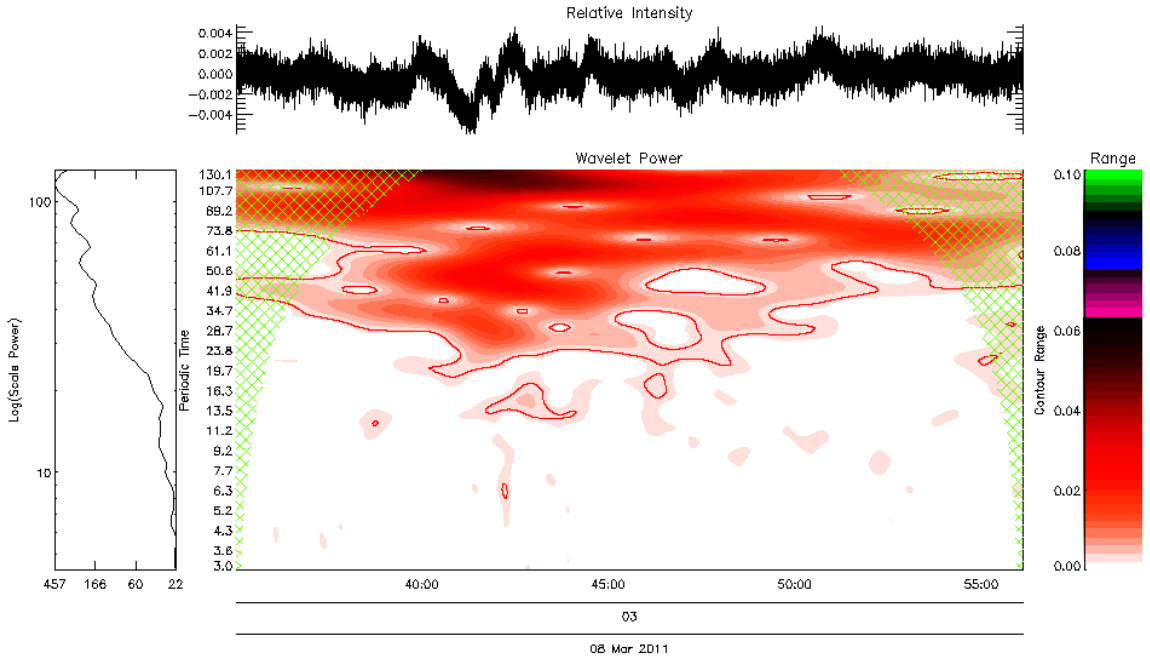


Figure 5.47: Wavelet analysis of the epoch occurring between 03:35 and 03:55 on March 8th, 2011.

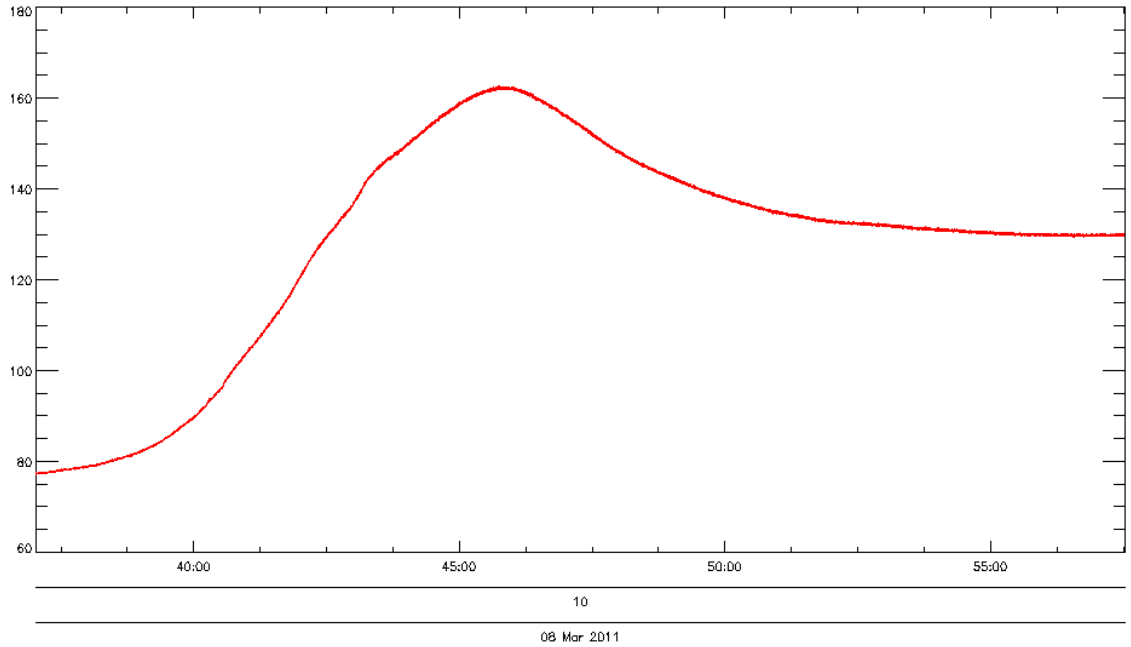


Figure 5.48: Original signal of the epoch occurring between 10:37 and 10:58 on March 8th, 2011.

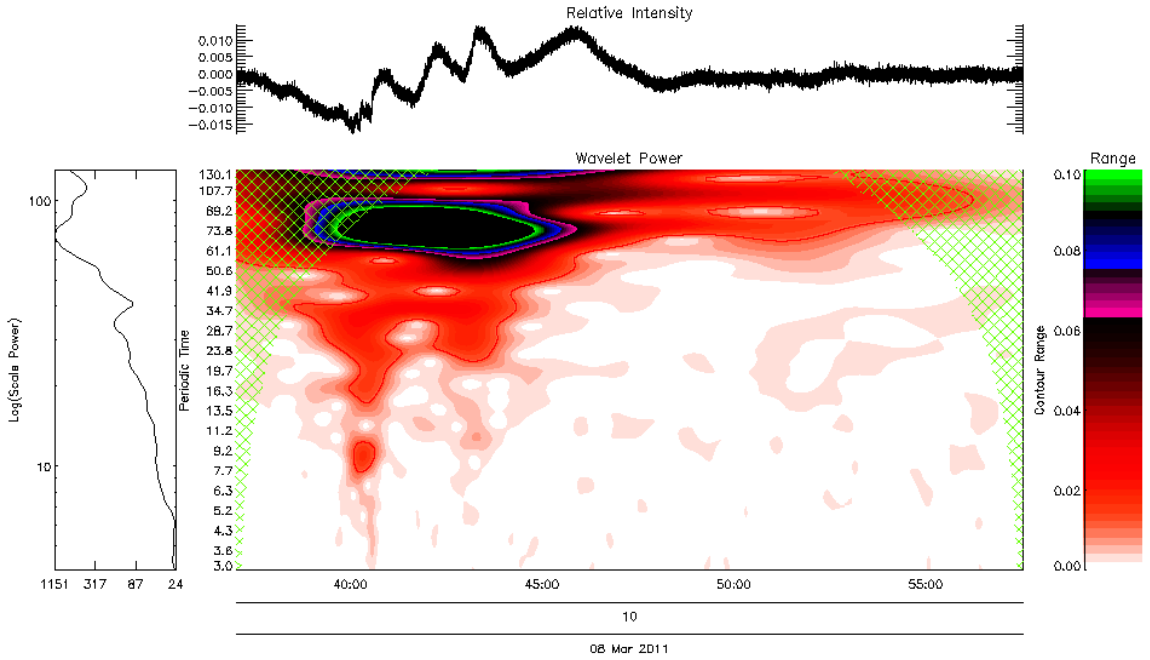


Figure 5.49: Wavelet analysis of the epoch occurring between 10:37 and 10:58 on March 8th, 2011.

The first epoch does not seem to contain a distinct oscillatory component. The existence of desired oscillatory components in the first epoch is classed as uncertain due to an unclear oscillatory component. The second epoch, shown in figures 5.48 and 5.49 shows a pronounced

oscillatory component during the rising phase of the flare. The existence of desired oscillations in this epoch is classed as certain.

5.6.3.3 March 9th, 2011

The epoch shown in this section occurs between 10:36 and 10:56 of March 9$^{\text{th}}$, 2011. The signal shown in this epoch seems to consist of a peak and decay phase of one event intersecting with the rising phase of another event; however this can't be verified due to the Solarsoft archives not being present. The original signal of this epoch and corresponding wavelet analysis are shown in figures 5.44 and 5.45.

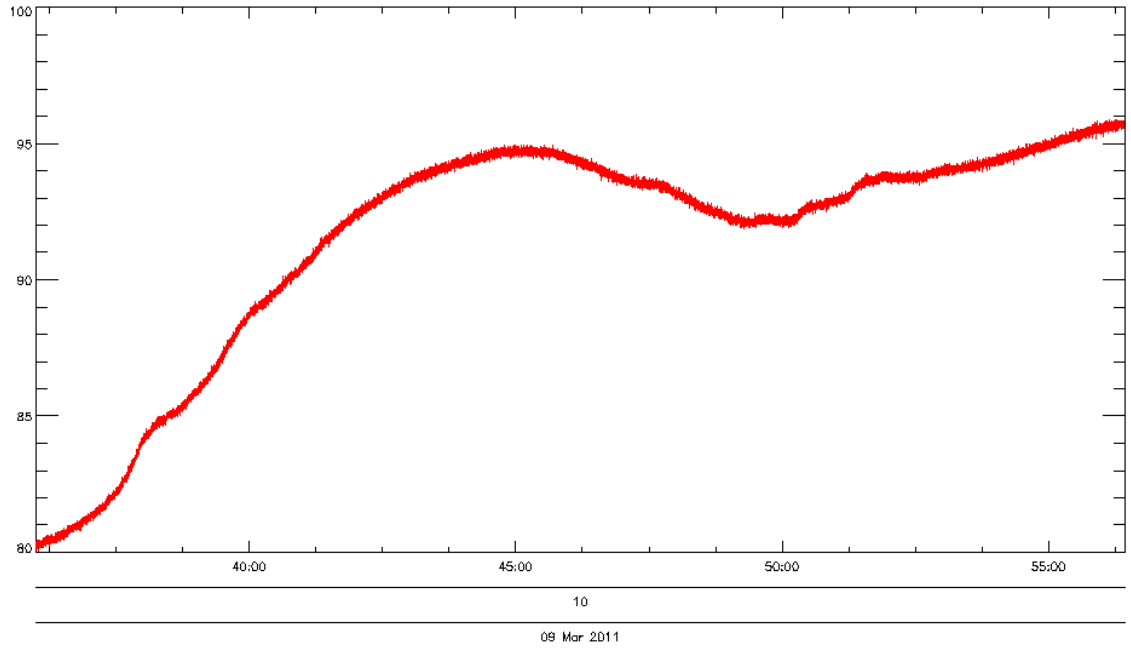


Figure 5.50: Original signal of the epoch occurring between 10:36 and 10:56 on March 9$^{\text{th}}$, 2011.

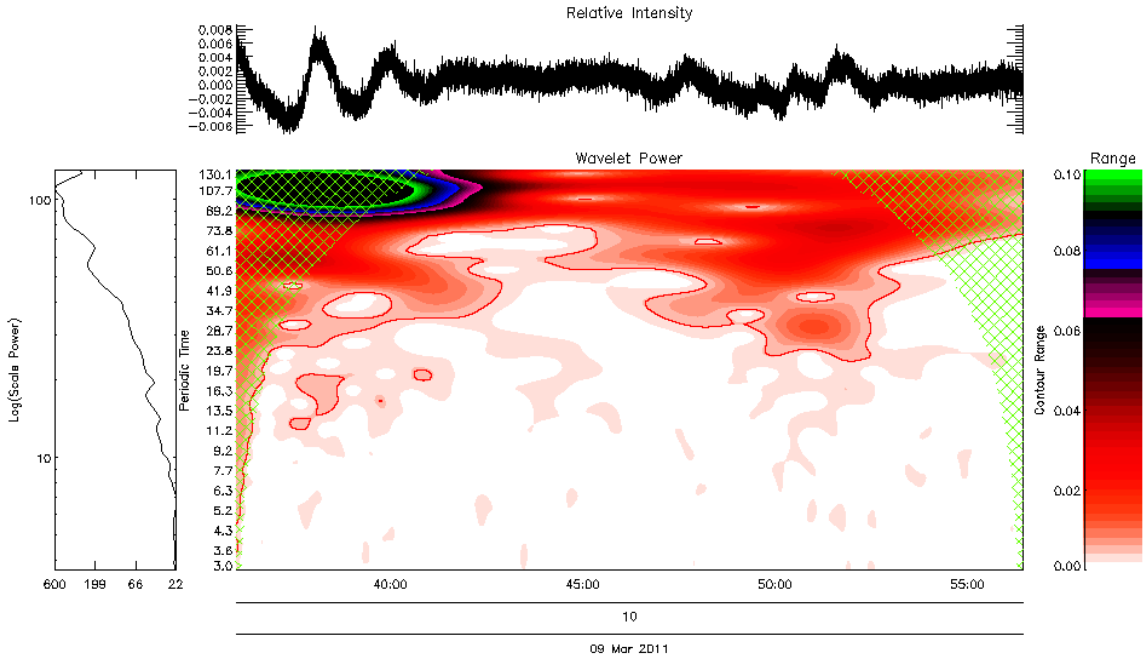


Figure 5.51: Wavelet analysis of the epoch occurring between 10:36 and 10:56 on March 9$^{\text{th}}$, 2011.

The oscillations present in the wavelet transform of this epoch can be seen in the original signal and relative intensity of the signal, occurring between 10:36 and 10:41. There are also oscillatory components present between 10:46 and 10:52 but these are not as pronounced as the oscillations near the beginning of the epoch. The existence of desired oscillations in this epoch is classed as certain.

5.6.3.4 March 24th, 2011

The epoch shown in this section occurs between 16:55 and 17:16 of March 24$^{\text{th}}$, 2011. The signal shown in this epoch seems to behave similarly to the second epoch discussed in section 5.6.3.2. The original signal of this epoch and corresponding wavelet analysis are shown in figures 5.52 and 5.53.

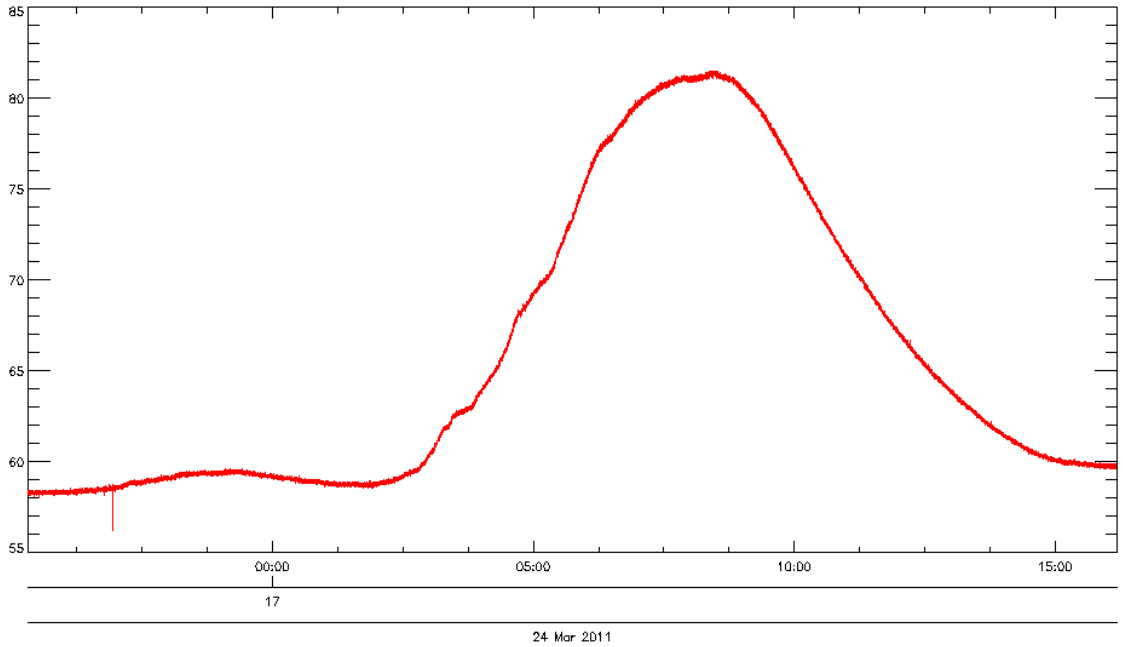


Figure 5.52: Original signal of the epoch occurring between 16:55 and 17:16 on March 24th, 2011.

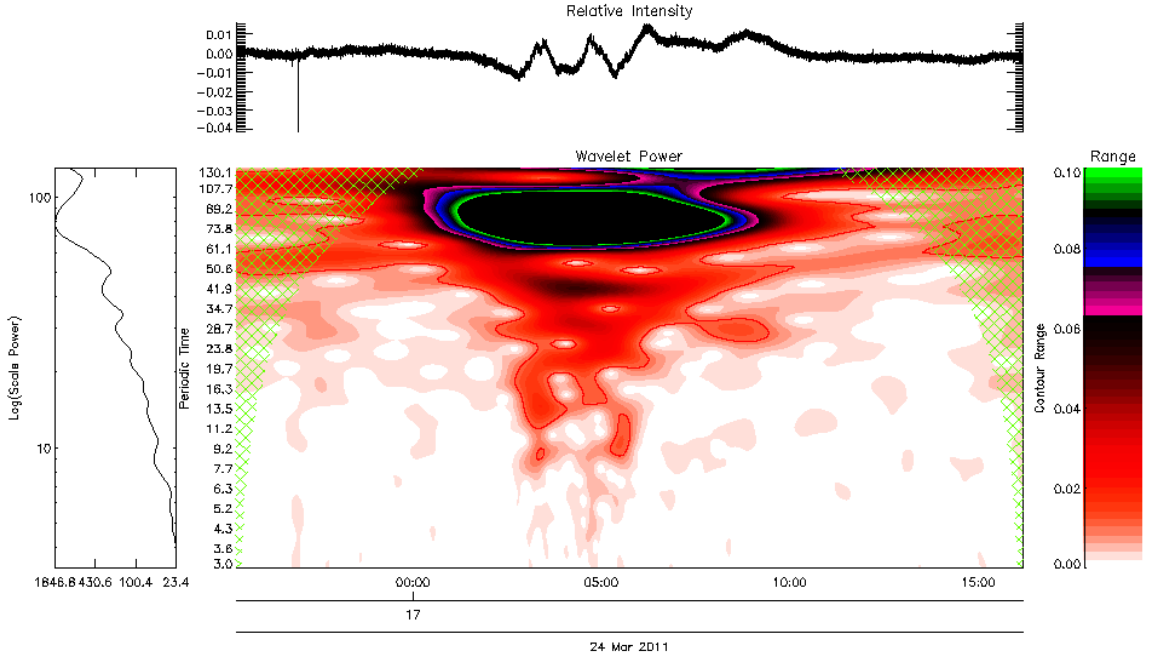


Figure 5.53: Wavelet analysis of the epoch occurring between 16:55 and 17:16 on March 24th, 2011.

The oscillations present in the wavelet transform of this epoch can be seen in the original signal and relative intensity of the signal, occurring between 17:02 and 17:08. The existence of desired oscillations in this epoch is classed as certain.

Chapter 6

Conclusions

6.1 Introduction

In this chapter, the results presented in chapter 5 will be discussed, and future work suggested. In the final section, a conclusion of the research project will be given.

6.2 Conclusion of Analysis

6.2.1 Introduction

In this section, the results of analysis presented in chapter 5 will be discussed in two steps; evaluation of analysis software, and interpretation of analysis results. Evaluation of analysis software will be discussed in sections 6.2.2 and 6.2.3 to highlight the performance of flare and frequency detection algorithms. This evaluation is presented as automated frequency analysis of this volume of LYRA data has not been performed prior to this research project.

6.2.2 Conclusion of Flare-Detection Algorithm

In analysis performed on LYRA data recorded during August as discussed in section 5.4, 18 flares were expected to be detected. There were 27 flares recorded by Solarsoft for this interval, but 8 of these events did not have corresponding LYRA data, and 1 flare occurred within the duration of another flare of the same class. Of these 18 flares, 12 were detected using the flare-detection algorithm. The 6 events which were not detected corresponded to B5.2, B5.3, B6.4, B6.5, B6.6, and B7.8 flares, respectively. Improvements that may be made to the flare-detection algorithm to reduce the number of false negatives will be discussed in section 6.3.

It was also seen in analysis that LYRA-based flare detection flagged epochs during the decay of flares. This is a positive result of LYRA-based flare detection as it suggests the start, end, and peak times of flares could be determined in future work, which is discussed in section 6.3.

6.2.3 Conclusion of Frequency-Detection Algorithm

In this section, only analysis performed from LYRA data recorded during August, as discussed in section 5.4.3.3, will be considered. Analysis performed during March can not be considered in evaluating frequency detection as the analysis performed was much more brief due to time constraints.

The results of frequency analysis performed in section 5.4.3.3 are summarised below in table 6.1.

Certain	Uncertain	False
1	7	6

Table 6.1: The number of epochs detected as containing frequencies classed as Certain, Uncertain, or False detections. The number of epochs correspond to analysis discussed in section 5.4.3.3.

The epoch containing desired frequency components classed as a certain detection corresponds to the M-class flare discussed in section 5.4.3.3. Unfortunately, only one event was flagged as reliably containing desired oscillatory components, and there was only M-class flare in the interval analysed. The 7 epochs flagged as possibly containing desired oscillatory components largely correspond to an unexpected type of event present in the LYRA data referred to as a erratic event; these events are discussed in section 6.2.7. Assuming that uncertain events are of no use to this research project, the rate of successful frequency detection is 1 in 13. This rate of success is quite poor and so, several improvements to the frequency-detection algorithms are suggested in section 6.3.

6.2.4 Conclusion of Analysis Results

In this section, all epochs flagged as reliably containing desired frequency components will be summarised, and their interpretation discussed. During this discussion, 'oscillatory components' refers to oscillations seen within the range of frequencies desired to be detected as specified by the configuration of the software.

6.2.4.1 Summary of Epochs Containing Frequency Components

In analysis performed on data recorded from August 1st to 20th, there was one epoch flagged as reliably containing oscillatory components; see section 5.2.3.3. This epoch spanned between 18:09 and 18:30 of August 7th, 2010, and corresponded to the rising phase of an M1.0 class flare. The oscillations detected were recorded with a peak amplitude at approximately 18:13.

Similarly, in analysis performed over June 13th, 2010, one epoch was flagged as reliably containing oscillatory components; see section 5.3.3.3. This epoch spanned between 05:27 and 05:48, and contained the entirety of an M1.0 class flare occurring between 05:30 and 05:44. Oscillations were detected in the rising phase of this flare, with a peak amplitude at 05:37.

Both of these epochs suggest that the desired oscillatory components seem to occur during the rising phase of flares; however, this can not be stated conclusively without further research into flares of a similar class.

As there were only two epochs flagged as reliably detecting oscillatory components in analysis over 20 days of August and 1 day from June, an extra set of analysis was performed on

data from March, 2011. This month was chosen as the Sun was more active during this period, with an X-class flare being recorded on February 14th, and the detector had recently ceased being affected by the Earth's ionosphere; see section 5.5.2. The epochs flagged as containing oscillatory components in this analysis are discussed in section 5.6.

A summary of epochs flagged as containing oscillatory components during analysis of data recorded from March, August, and June, is shown in table 6.2. A summary of the oscillatory components found in these epochs is given in table 6.3.

		Time (HH::MM)		
Epoch ID	Date	Start	End	Flare Strength
1	June 13th, 2010	05:27	05:48	M1.0
2	August 7th, 2010	18:09	18:30	M1.0
3	March 7th, 2011	19:43	20:04	N/A
4	March 8th, 2011	10:37	10:58	N/A
5	March 9th, 2011	10:36	10:56	N/A
6	March 24th, 2011	16:55	17:16	N/A

Table 6.2: Summary of epochs flagged as reliably containing oscillatory components in analysis performed on data from June 13th, 2010, August, 2010, and March, 2011.

		Time (HH::MM:SS)			
Epoch ID	Date	Flare Begin	Flare Peak	Δt (s)	T
1	June 13th, 2010	05:35:30	05:40:30	330	65
2	August 7th, 2010	17:56:00	18:30:00	2040	100
3	March 7th, 2011	19:49:00	20:39:00	3000	130
4	March 8th, 2011	10:39:30	10:45:30	360	65
5	March 9th, 2011	10:35:30	10:45:00	570	110
6	March 24th, 2011	17:02:30	17:08:00	330	80

Table 6.3: Summary of flare details corresponding to each epoch listed in table 6.2. The start and peak times for flares, and change in signal strength are approximations measured from the raw signal plots of the Zirconium channel. The duration in flare start and peak times is given by Δt. The periodic time of oscillations is given by T.

Using the details in table 6.3, a plot was generated of the periodic times of oscillatory components against the time (in seconds) between the start and peak times of the flare. This plot was generated to investigate if there were any relations present between the periodic time of oscillations and the duration of flares. The plot is shown in figure 6.2.

In order to measure the duration between the start and peak times of flares, a method had to be used based on LYRA data as the Solarsoft archives do not contain records during March,

2011. In order to measure this duration using LYRA data, the following steps were used:

1. The peak of the corresponding event was measured using a plot of the raw signal in each epoch.

2. A line was fitted to the linear portion of the rising phase of the flare.

3. The background signal strength was measured prior to the flare. In particularly noisy data, this was taken to be the lowest strength before the flare.

4. The start of the flare was marked as the point where the line fitted to the rising phase of the flare crossed with the background signal strength.

These steps are illustrated using the M-class flare occurring on June 13th, 2010, in figure 6.1.

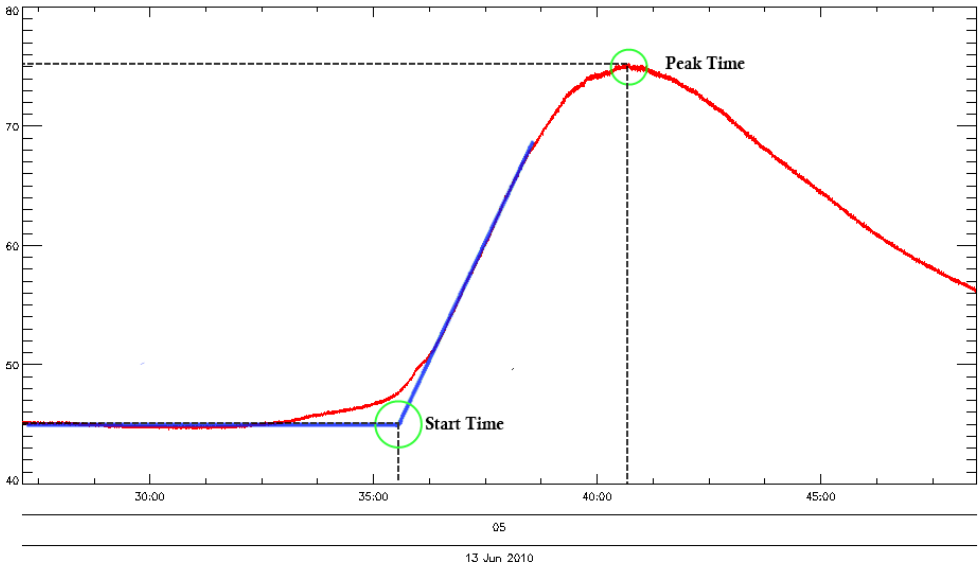


Figure 6.1: A demonstration of measuring the start and peak times of a flare using LYRA data.

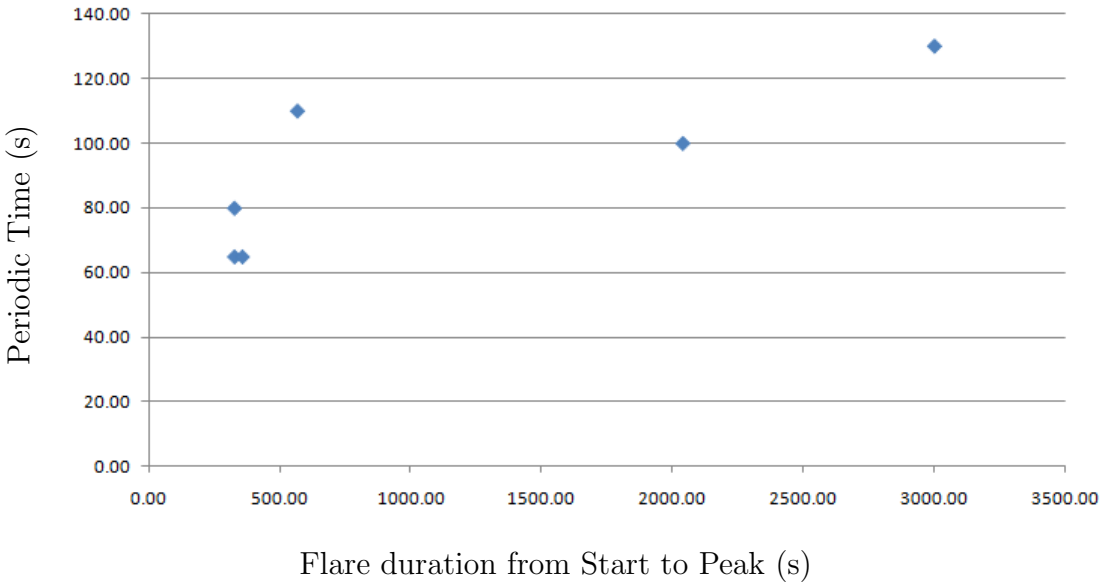


Figure 6.2: A plot of periodic times of oscillatory components against the duration (in seconds) between the start and peak times of corresponding flares.

Unfortunately, due to the deviations in the plot, and limited number of epochs flagged as reliably containing oscillatory components, a relationship between periodic times and start-peak durations of flares cannot be concluded.

6.2.4.2 Interpretation

Interpretation of the oscillatory components seen during flares, particularly larger, M-class flares, show that the oscillations are of a distinct periodic time during the event, and seem to occur in the same period of their corresponding flares, which suggests that the oscillatory components seen are not random deviations in signal strength.

However, analysis over a larger number of events is required before further investigations can be made in relations between flares and corresponding oscillatory components.

6.2.5 Comparison of Analysis from June 13th, 2010.

In this section, the results of analysis performed on LYRA data from June 13$^{\text{th}}$, 2010, will be compared against the results of work reported by J.J. Zender *et al.*

6.2.5.1 Work Reported by J.J. Zender et al

In work reported by J.J. Zender *et al* [6], the M-class flare occurring between 05:30 and 05:44 on June 13$^{\text{th}}$, 2010 , is discussed.

In wavelet analysis of the M-class flare in this paper, multiple frequency bands are reported; the most prominent of which is a $0.013 - 0.015$ Hz band corresponding to a band of approximately $67 - 77$ s. A second band is also reported with a band of $0.037 - 0.053$ Hz, corresponding to a band of approximately $19 - 27$ s.

6.2.6 Comparative Results

In analysis performed as part of this research project, separate oscillatory components were not investigated. Instead, a single oscillatory component was determined if there was a pronounced peak in scale-power reported in wavelet analysis. The wavelet analysis presented for this flare is shown in figure 6.3. From the scale-power plot seen to the left of the wavelet transform, a peak strength can be seen corresponding to a periodic time of approximately $65 - 70$ s; which agrees with the band of approximately $67 - 77$ s reported in J.J. Zender *et al.* The second band reported of approximately $19 - 27$ s can also be seen in scale-power plot as a peak occurring between 20 and 30 s.

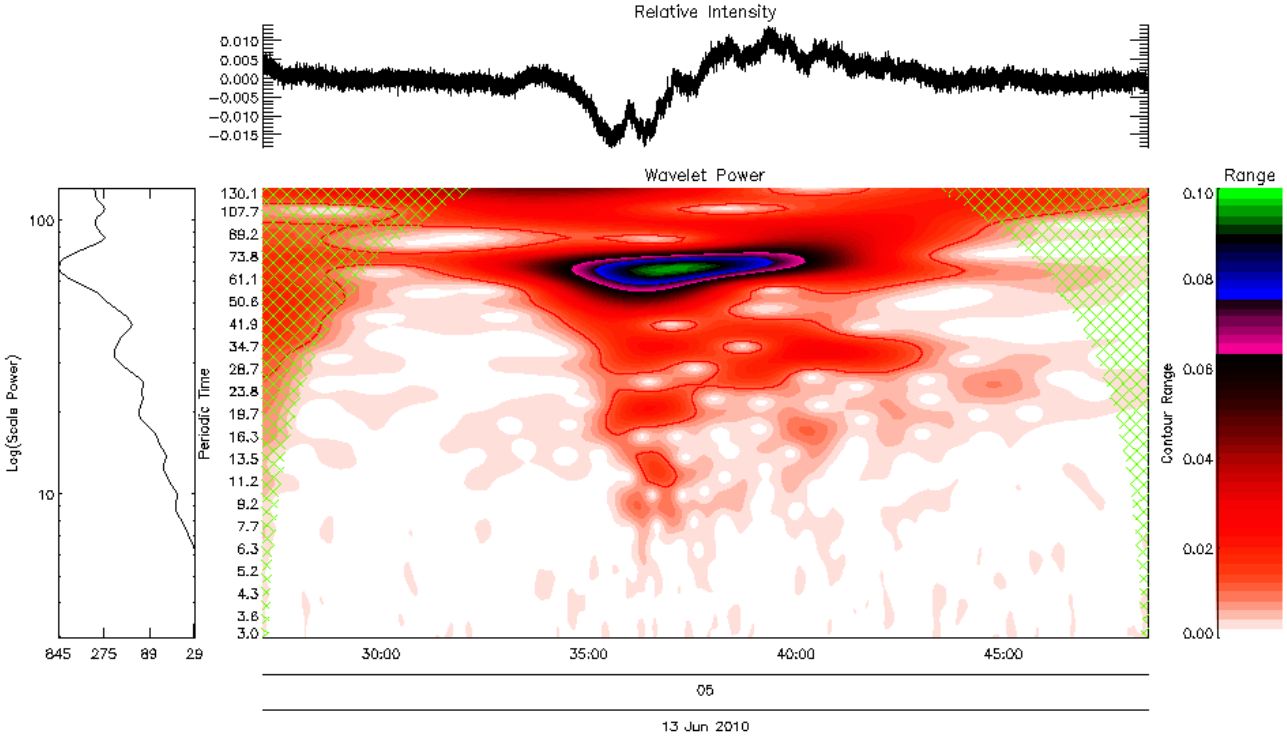


Figure 6.3: The wavelet transform of an epoch spanning between 05:27 and 05:48 on June 13$^{\text{th}}$, 2010, containing an M1.0 class flare occurring between 05:27 and 05:48.

In conclusion, analysis performed as part of this research project agrees with the findings reported in J.J. Zender *et al.*

6.2.7 Discussion of Erratic Events

During analysis of LYRA data, a type of event was seen in the data which did not seem to be of instrumental origins; this event was referred to as an erratic event. Two examples of these events are shown in figures 6.4 and 6.5. These events can be characterised by the following:

- Noisy signals, with strong sudden variances in signal strength over the order of seconds.

- Generally short lived, with durations of the order of hundreds of seconds.

These events are not recorded by Solarsoft archives, which suggests that the event occurs in a wavelength out of the detection range of GOES or that it is caused by events in the Earths atmosphere. In order to investigate the origin of these events, analysis was performed on August 3rd, 2010, using each of the 4 channels of the LYRA instrument. The signal was seen in the aluminium channel of LYRA, but was not seen in Lyman-Alpha and Herzberg channels; as shown in figure 6.6.

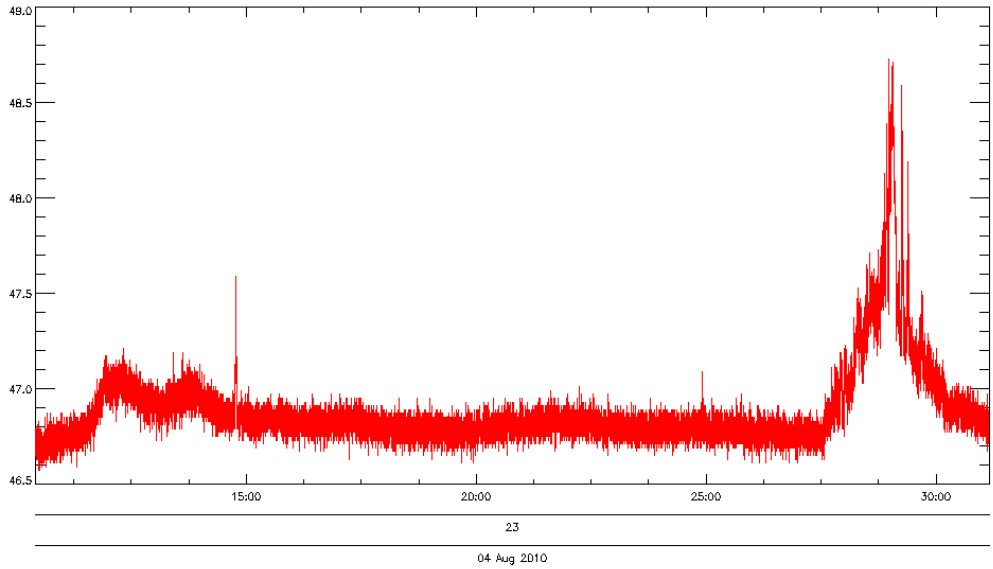


Figure 6.4: Plot of Zirconium channel between 23:10 and 23:31 of August 4th, 2010.

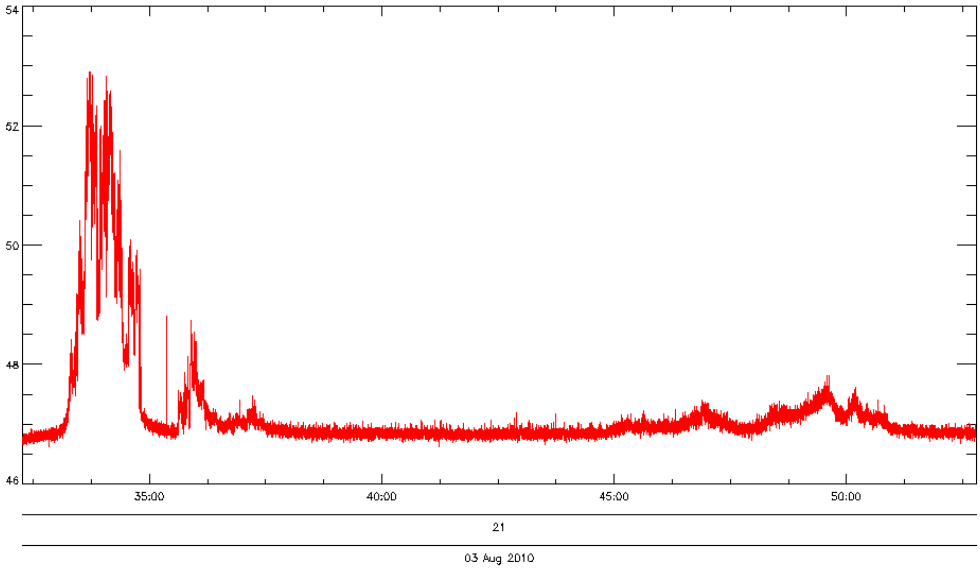


Figure 6.5: Plot of Zirconium channel between 21:32 and 21:52 of August 3rd, 2010.

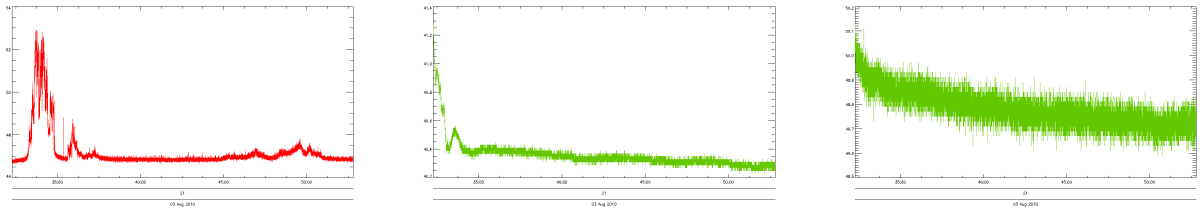


Figure 6.6: Plot of the Aluminium (Left), Herzberg (Centre), and Lyman-Alpha (Right) channels of LYRA between 21:32 and 21:52 of August 3rd, 2010.

6.2.7.1 Effects on Analysis

From investigation of epochs flagged as containing frequencies, it was seen that a lot of uncertain detections arose from these eratic events. These events are very prone to causing false positives in frequency analysis due to the poor generation of the relative intensity signal used by the wavelet transform; demonstrated in figure 6.7. The poor generation of relative intensity is caused by the dramatic and sudden deviations in these signals.

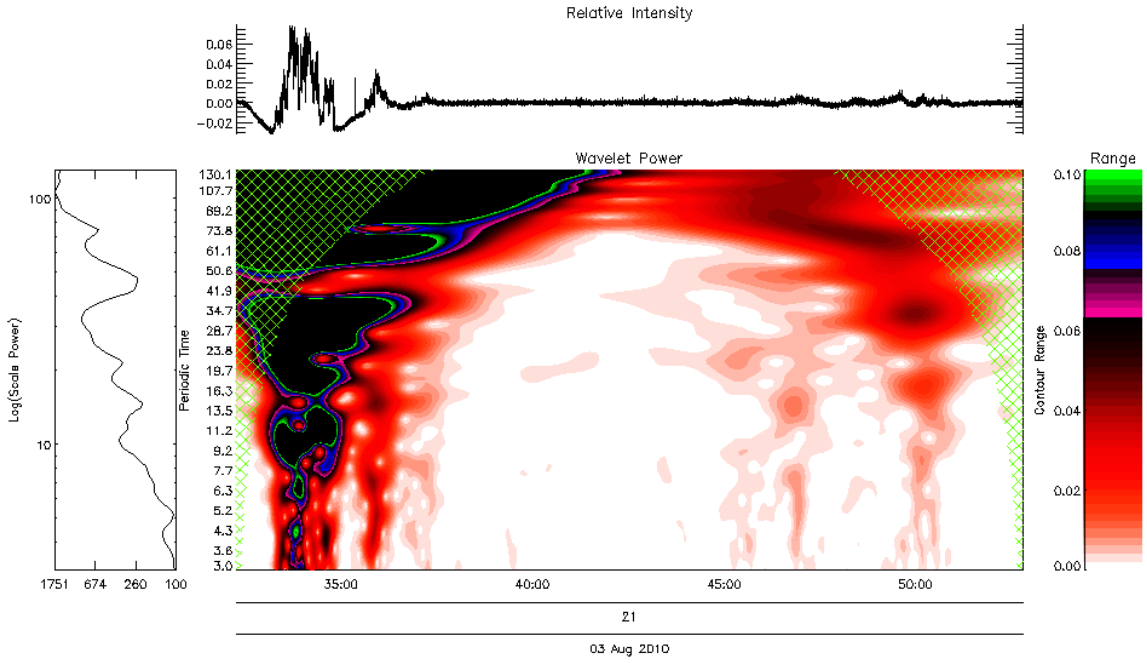


Figure 6.7: Wavelet analysis of Zirconium channel between 21:32 and 21:52 of August 3rd, 2010.

6.2.7.2 Conclusions

From analysis of erratic events, it can be concluded that the events are present in both the Aluminium and Zirconium channels of Unit 2 in LYRA, and that the events are not recorded by Solarsoft. However, further analysis into these events must be performed before their origin can be determined. SWAP images could be potentially used in this analysis, but the current

cadence of an image every 2 minutes would potentially overlook these events.

Suggestions to help reduce the effect of these events on analysis, and further investigation of these events are proposed in section 6.3.

6.3 Future Work

As this research project was a first attempt at analysing large sets of data from an experimental instrument, a series of events and behaviours were encountered which were not expected prior to development of the analysis software. The aim of this section is to introduce and discuss a series of suggestions aimed at improving analysis of this data as part of future work.

6.3.1 Use Calibrated Data

As of March 10^{th}, 2010, 2 new levels of calibrated LYRA data were made available. These were not used in work reported in this paper, but the analysis software could be easily configured to use the new sets of data in future work.

However, this data contains few significant changes that would affect analysis. The changes currently consist of removing the dark current of the photo-detectors, correcting the data for the degradation trend of detector amplitude over time, and converting the data into physical units of W/m^2.

6.3.2 Improve flare-detection

The current implementation of the flare-detection algorithm, as discussed in section 4.7, uses the median value of the days flare-response as a reference value in flare-detection. This was used due to the degradation of the LYRA detectors over time. If calibrated data is used in future work, this algorithm could be improved to be less reliant on the overall behaviour of a day of LYRA data. It may also be possible in future improvements to this algorithm, to incorporate the measurement of start, end, and peak times of flares. Implementing this feature would allow the duration of oscillatory components to be linked to different phases of flares in future work.

6.3.3 Improve frequency-detection

One of the main goals of this research project was to automatically detect transient oscillatory components within flares present in LYRA data. During analysis, a number of unexpected events led to suggestions to improvements of the frequency-detection algorithm; this section will discuss each of these improvements.

6.3.3.1 COI Influenced False Negatives

In the current implementation of the frequency-detection algorithm, the wavelet transform is only analysed where it is not affected by the Cone of Influence. This can potentially lead to incorrect measurements of oscillation durations in analysis causing false negatives in frequency detection. One improvement that may prevent false negatives in future analysis would be the consideration of oscillations seen in the wavelet transform provided that the peak strength is outside of the Cone of Influence. Edge effects would not be considered in oscillation detection as the peak strength in edge effects exists on the extreme edges of the transform.

6.3.3.2 Temporal Analysis

Temporal analysis could be incorporated in wavelet analysis by cross referencing the existence of oscillatory components with the original temporal data being analysed. This could allow the following:

- Linking the times of oscillations to particular phases of flares automatically.
- Discarding oscillatory components if they exist over a region that is changing dramatically in time (e.g. discarding oscillations detected due to an erratic event).

6.3.3.3 Reduce Effect Of Erratic Events

Erratic events were demonstrated in section 6.2.7 to have dramatic effects on wavelet analysis. These could be flagged as erratic events or removed prior to analysis by calculating the derivative of the signal prior to analysis. The derivative of the signal would be characteristic of the event due to the rapidly changing amplitudes within erratic events. Similar methods have been used in past work carried out during my student placement in ESA, (2010), in order to map regions over the Earth's surface affected by the SAA using LYRA data.

6.3.3.4 Addition of Statistical Output

Statistical output could be generated from frequency-detection to include the following:

- Peaks in scale-power of the wavelet transform.
- Durations of peaks in the wavelet transform.
- Frequency detection confidence levels using temporal analysis as well as COI mapping.

Having these statistics included in report generation would reduce the amount of analysis required by the user for a single day of LYRA data.

6.3.4 Report Improvements

During the initial stages of analysis, reporting was a very low priority with most analysis being performed on-the-fly in order to further develop analysis software. However, as the software became more developed, the importance of meaningful reporting was realised. This is what led to HTML report generation being the primary method of reporting; previous methods resulted in large files of text which were very difficult to parse through.

During analysis of LYRA software, some useful additions to report generation were realised; these will be discussed in this section.

6.3.4.1 Add SWAP Images

During analysis of LYRA data, images from SWAP were rarely used. However, when unexpected events were detected in LYRA (such as the atmospheric effects discussed in section 5.5.2), SWAP images were used to indicate the origins of the event. The inclusion of SWAP images would be of use in LYRA analysis reports as they could show the regions of flares, if the satellite was off-pointing, or event if the Sun is partially obscured by the moon. These images could be included as a short strip for each epoch analysed in a day of LYRA data.

6.3.4.2 Add Satellite Position Records

Another improvement suggested in reporting, was the addition of the satellites position over time. This could be included as a trace of the satellites position plotted over a map of the Earth during each epoch. Similar work has been performed in the past as part of work carried out during my student placement in ESA (2010), but this has not been implemented due to time constraints on analysis.

6.3.4.3 Output Summary Statistics Separately

One caveat of report generation is that, once analysis has been performed over a day of LYRA data, the results are only stored within the HTML report. This means that parsing results spanning several days requires manual analysis of each report. It also means that if settings are changed in HTML formatting, analysis must be performed again in order to generate the results needed to make a new report. One suggestion for future work in report generation is to completely separate report generation from analysis and create structures of analysis results which can be read by IDL. If this is done, analysis results can be collectively parsed to create reports of arbitrarily long intervals of data. It would also reduce time required in developing the algorithms used to generate HTML reports.

6.4 Final Conclusions

In analysis performed, the oscillations that were intended to be used in characterising solar flares were clearly identified in a number of solar events. However, it was also found that the periodic times of these oscillations can vary. A possible link between the periodic times of oscillations and flare duration can exist, but this can not be concluded without further analysis.

A different band of oscillations with atmospheric origins was also found in analysis. These oscillations may be linked to the oscillations detected in solar flares, as the flares may trigger vibrations in the ionosphere. However this relation is purely speculative, and will also require further analysis of data.

Several suggestions have also been put forward to enhance the analysis software, which will provide a much more thorough set of results in future work.

In conclusion, the original goal to characterise solar flares using frequency analysis was performed to a degree, but further analysis of large solar events is required to complete this task. However, although the analysis performed did not conclusively solve the goal of the project, further areas of analysis have been identified which could be pursued in the future and may even lead to understanding the reasons behind the oscillations originally detected.

Bibliography

[1] Diagram of the sun. http://www.nasa.gov/mission_pages/hinode/solar_020.html. [Online; accessed 12-Mar-2011].

[2] 2009 P. Carril. Rear artist view of proba-2 as it looks toward the sun. http://orbitalhub.com/?p=568, 2009. [Online; accessed 06-Jan-2011].

[3] Lyra instrument. http://proba2.sidc.be/index.html/science/lyra/article/lyra-instrument, . [Online; accessed 06-Jan-2011].

[4] A. Benmoussa, I. E. Dammasch, J.-F. Hochedez, U. Schühle, S. Koller, Y. Stockman, F. Scholze, M. Richter, U. Kroth, C. Laubis, M. Dominique, M. Kretzschmar, S. Mekaoui, S. Gissot, A. Theissen, B. Giordanengo, D. Bolsee, C. Hermans, D. Gillotay, J.-M. Defise, and W. Schmutz. Pre-flight calibration of LYRA, the solar VUV radiometer on board PROBA2. *Astronomy and Astrophysics*, 508:1085–1094, December 2009. doi: 10.1051/0004-6361/200913089.

[5] Progress of swap commissioning. http://proba2.sidc.be/index.html/swap/article/progress-of-swap-commissioning, . [Online; accessed 06-Jan-2011].

[6] J.J. Zender, B. Foing, D. Vagg, M. Dominique, and I. Dammasch. Temporal and Frequency Varioations of Flares observed by LYRA onboard PROBA-2. In preparation.

[7] E. O'Shea, K. Muglach, and B. Fleck. Oscillations above sunspots: Evidence for propagating waves? *Astronomy and Astrophysics*, 387:642–664, May 2002. doi: 10.1051/0004-6361:20020375.

[8] A. C. Katsiyannis, D. Berghmans, B. Nicula, J.-M. Defise, G. Lawrence, J.-H. Lecat, J.-F. Hochedez, and V. Slemzin. Swap: AN EUV Imager for Solar Monitoring on Board of PROBA2. In D. E. Innes, A. Lagg, & S. A. Solanki, editor, *Chromospheric and Coronal Magnetic Fields*, volume 596 of *ESA Special Publication*, November 2005.

[9] A survey of the solar atmospheric models. http://csep10.phys.utk.edu/astr162/lect/sun/photosphere.html, . [Online; accessed 28-Mar-2011].

[10] A survey of the solar atmospheric models. http://prints.iiap.res.in/handle/2248/510, . [Online; accessed 28-Mar-2011].

[11] What is a solar flare. http://hesperia.gsfc.nasa.gov/sftheory/flare.htm, . [Online; accessed 28-Mar-2011].

[12] ESA. About Proba-2. http://www.esa.int/esaMI/Proba/SEMJJ5ZVNUF_0.html, Nov 2010. [Online; accessed 06-Jan-2011].

[13] Y. Stockman. LYRA, Solar UV Radiometer on the Technology Demonstration Platform Proba-2. In *ESA Special Publication*, volume 621 of *ESA Special Publication*, June 2006.

[14] J.-F. Hochedez, W. Schmutz, Y. Stockman, U. Schühle, A. Benmoussa, S. Koller, K. Haenen, D. Berghmans, J.-M. Defise, J.-P. Halain, A. Theissen, V. Delouille, V. Slemzin, D. Gillotay, D. Fussen, M. Dominique, F. Vanhellemont, D. McMullin, M. Kretzschmar, A. Mitrofanov, B. Nicula, L. Wauters, H. Roth, E. Rozanov, I. Rüedi, C. Wehrli, A. Soltani, H. Amano, R. van der Linden, A. Zhukov, F. Clette, S. Koizumi, V. Mortet, Z. Remes, R. Petersen, M. Nesládek, M. D'Olieslaeger, J. Roggen, and P. Rochus. LYRA, a solar UV radiometer on Proba2. *Advances in Space Research*, 37:303–312, 2006. doi: 10.1016/j.asr.2005.10.041.

[15] Lyra analysis manual. `http://proba2.oma.be/index.html/science/lyra-analysis-manual/`. [Online; accessed 10-Mar-2011].

[16] J.-M. Defise, J.-H. Lecat, E. Mazy, P. Rochus, L. Rossi, T. Thibert, J.-M. Gillis, D. Berghmans, J.-F. Hochedez, and U. Schühle. SWAP: Sun watcher with a new EUV telescope on a technology demonstration platform. In B. Warmbein, editor, *5th International Conference on Space Optics*, volume 554 of *ESA Special Publication*, pages 257–262, June 2004.

[17] D. Berghmans, J. F. Hochedez, J. M. Defise, J. H. Lecat, B. Nicula, V. Slemzin, G. Lawrence, A. C. Katsyiannis, R. van der Linden, A. Zhukov, F. Clette, P. Rochus, E. Mazy, T. Thibert, P. Nicolosi, M.-G. Pelizzo, and U. Schühle. SWAP onboard PROBA 2, a new EUV imager for solar monitoring. *Advances in Space Research*, 38:1807–1811, 2006. doi: 10.1016/j.asr.2005.03.070.

[18] D. C. Wells, E. W. Greisen, and R. H. Harten. FITS - a Flexible Image Transport System. *Astronomy and Astrophysics, Supplement*, 44:363–+, June 1981.

[19] The fits support office. `http://fits.gsfc.nasa.gov/`. [Online; accessed 12-Mar-2011].

[20] Swap data products: Description of the fits files headers. `http://proba2.sidc.be/index.html/swap/swap-analysis-manual/article/data-products`, . [Online; accessed 12-Mar-2011].

[21] Lyra team members. `http://proba2.oma.be/index.html/science/lyra/article/lyra-team`, . [Online; accessed 10-Mar-2011].

[22] Swap team members. `http://proba2.oma.be/index.html/swap/article/swap-team-members`, . [Online; accessed 10-Mar-2011].

[23] Windowing functions improve fft results. `http://www.tmworld.com/article/322450-Windowing_Functions_Improve_FFT_Results_Part_I.php`. [Online; accessed 06-Mar-2011].

[24] M. Boltežar and J. Slavič. Enhancements to the continuous wavelet transform for damping identifications on short signals. *Mechanical Systems and Signal Processing*, 18:1065–1076, September 2004. doi: 10.1016/j.ymssp.2004.01.004.

[25] Mathworks: R2010b documentation wavelet toolbox. `http://www.mathworks.com/help/toolbox/wavelet/ug/f8-3495.html`. [Online; accessed 06-Mar-2011].

[26] C. Torrence and Gilbert. P. Comp. A Practical Guide to Wavelet Analysis. *Bulletin of the American Meteorological Society*, October 1997.

[27] S. M. Hill, V. J. Pizzo, C. C. Balch, D. A. Biesecker, P. Bornmann, E. Hildner, L. D. Lewis, R. N. Grubb, M. P. Husler, K. Prendergast, J. Vickroy, S. Greer, T. Defoor, D. C. Wilkinson, R. Hooker, P. Mulligan, E. Chipman, H. Bysal, J. P. Douglas, R. Reynolds, J. M. Davis, K. S. Wallace, K. Russell, K. Freestone, D. Bagdigian, T. Page, S. Kerns, R. Hoffman, S. A. Cauffman, M. A. Davis, R. Studer, F. E. Berthiaume, T. T. Saha, G. D. Berthiume, H. Farthing, and F. Zimmermann. The NOAA Goes-12 Solar X-Ray Imager (SXI) 1. Instrument, Operations, and Data. *Solar Physics*, 226:255–281, February 2005. doi: 10.1007/s11207-005-7416-x.

[28] The idl programming language. `http://www.ittvis.com/language/en-US/ProductsServices/IDL/IDLProgrammingLanguage.aspx`, . [Online; accessed 01-Apr-2011].

[29] The idl astronomy user's library. `http://idlastro.gsfc.nasa.gov/`, . [Online; accessed 01-Apr-2011].

[30] K. Zwintz, R. Kuschnig, W. W. Weiss, R. O. Gray, and H. Jenkner. Hubble Deep Field guide star photometry. *Astronomy and Astrophysics*, 343:899–903, March 1999.

[31] B. Bertucci and AMS-01 Collaboration. Leptons with E ¿ 200 MeV Trapped in the Earth's Radiation Belts Observed with the AMS Experiment. In *International Cosmic Ray Conference*, volume 4 of *International Cosmic Ray Conference*, pages 1761–+, July 2003.

[32] Y. Uchida. Diagnosis of Coronal Magnetic Structure by Flare-Associated Hydromagnetic Disturbances. *Publications of the ASJ*, 22:341–+, 1970.

[33] M. J. Aschwanden, B. Kliem, U. Schwarz, J. Kurths, B. R. Dennis, and R. A. Schwartz. Wavelet Analysis of Solar Flare Hard X-Rays. *Astrophysical Journal*, 505:941–956, October 1998. doi: 10.1086/306200.

[34] J. Rendtel, J. Staude, and W. Curdt. Observations of oscillations in the transition region above sunspots. *Astronomy and Astrophysics*, 410:315–321, October 2003. doi: 10.1051/0004-6361:20031189.

[35] M. S. Madjarska, J. G. Doyle, D. E. Innes, and W. Curdt. Jets or High-Velocity Flows Revealed in High-Cadence Spectrometer and Imager Co-observations? *"Astrophysical Journal, Letters"*, 670:L57–L60, November 2007. doi: 10.1086/524013.

[36] E. O'Shea, D. Banerjee, J. G. Doyle, B. Fleck, and F. Murtagh. Active region oscillations. *Astronomy and Astrophysics*, 368:1095–1107, March 2001. doi: 10.1051/0004-6361:20010073.

[37] Bash. `http://www.gnu.org/software/bash/`. [Online; accessed 16-Apr-2011].

[38] Open view of proba-2 payload with swap and lyra (image credit: Csl). `http://www.eoportal.org/directory/pres_PROBA2ProjectforOnBoardAutonomy2.html`. [Online; accessed 13-Mar-2011].

[39] The amazing structure of the sun. `http://stargazers.gsfc.nasa.gov/resources/amazing_structure.htm`, . [Online; accessed 13-Mar-2011].

[40] A. Fludra. Transition region oscillations above sunspots. *Astronomy and Astrophysics*, 368:639–651, March 2001. doi: 10.1051/0004-6361:20000574.

[41] Ionosphere and magnetosphere. `http://www.britannica.com/EBchecked/topic/1369043/ionosphere-and-magnetosphere`. [Online; accessed 16-Apr-2011].

Appendix A

Code Listing

A.1 Introduction

In this chapter, the source code of the analysis software will be listed as well as some sample pages from an analysis report.

A.2 Analysis Source Code

In this section the source code of all IDL routines written as part of this research project is listed. Each file has the extension of '.pro' and is listed in a corresponding subsection.

A.2.1 lyrascript

```
; Return a section of data to analyze
FUNCTION getDataWindow, data, chnl, timeWindow, smoothingTimePeriod
  ;+f* LYRASCRIPT/getDataWindow
  ; FUNCTION:
  ;       getDataWindow
  ;
  ; PURPOSE:
  ;       Return a structure containing data for a section of LYRA data.
  ;
  ; EXPLANATION:
  ;       Using the information contained within 'timeWindow', like the first
    and last indexes of data to view from a channel,
  ;       a section of data is generated. This section of data is returned
    within a structure along with variations of the section
  ;       such as relative intensity values.
  ;
  ; CALLING SEQUENCE:
  ;       Result = getDataWindow( Lyra_dataStructure, channel,
    timeWindowStructure, smoothingTimePeriod)
  ;
  ; INPUTS:
  ;       data            = A structure containing LYRA data channels.
  ;       chnl            = An integer specifying the channel to use from the
    LYRA data struct.
  ;       timeWindow      = A structure containing start and end times and
    indexes to be analysed.
```

```
;       smoothingTimePeriod = A double representing the length of data, in
    seconds, to create a moving average from.
;
; OUTPUTS:
;       dataWindow      = A structure containing a section of data from a
    single channel of the LYRA instrument.
;       dataWindow.data = A section of data from a single channel of the LYRA
    instrument as specified by the chnl parameter.
;       dataWindow.time = A section of the TIME array from the LYRA data
    corresponding to the section of data being returned.
;       dataWindow.data_relInt = A section of data from a single channel of
    the LYRA instrument as specified by the chnl parameter,
;       divided by the moving average of the original channel. This gives the
    relative intensity of oscillations in LYRA data.
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-

smoothingPointCount = (double(smoothingTimePeriod)/double(24*60*60)) *
    n_elements((*data).time) ; Finding the number of points required to smooth
    over

; Load the channel of LYRA data using the chnl parameter
if chnl eq '1' then $
    dataChannel = (*data).channel1
if chnl eq '2' then $
    dataChannel = (*data).channel2
if chnl eq '3' then $
    dataChannel = (*data).channel3
if chnl eq '4' then $
    dataChannel = (*data).channel4

; Generate the sections of data to return
time                    = ((*data).time)[ timeWindow.indexStart:timeWindow.
    indexEnd ]
dataSlice               = dataChannel[ timeWindow.indexStart:timeWindow.
    indexEnd ]
channelRelativeIntensity = (dataChannel / smooth(dataChannel,
    smoothingPointCount)) -1
dataRelativeIntensity   = channelRelativeIntensity[ timeWindow.indexStart:
    timeWindow.indexEnd ]

; Return a structure containing pointers to each section of data generated.
return, CREATE_STRUCT('data', ptr_new(dataSlice), 'time', ptr_new(time), '
    data_relInt', ptr_new(dataRelativeIntensity))
END
```

```
PRO lyrascript
        ;+p* LYRASCRIPT/lyrascript
        ; PROCEDURE:
        ;       lyrascript
        ;
        ; PURPOSE:
        ;       A modifyable script which automatically performs analysis of
           LYRA data.
        ;
        ; CALLING SEQUENCE:
        ;       $>IDL -e LYRASCRIPT --args Year Month Day
        ;
        ; INPUTS:
        ;       Year  =  The year component of the date of data to be analysed.
        ;       Month =  The month component of the date of data to be analysed.
        ;       Day   =  The day component of the date of data to be analysed.
        ;
        ; OUTPUTS:
        ;       The script can generate several images and HTML reports.
        ;
        ; EXAMPLE:
        ;       $>IDL -e LYRASCRIPT --args 2010 06 13
        ;
        ; NOTES:
        ;       As this file is essentially a script, there are several
           configurable
        ;                   parameters within the file itself. These are
           arranged into structures
        ;                   named corresponding to the properties they control,
        ;                   e.g. 'wavelet_parameters'.
        ;
        ; RESTRICTIONS:
        ;       Warning -- The contents of structures are not verified before
           use.
        ;
        ; MODIFICATION HISTORY:
        ;       Initial Version D. Vagg, February 2011
        ;-

  ; Parse command line information
      cmdArgs = COMMAND_LINE_ARGS()
      if ((size(cmdArgs))[0] gt 0) then begin
            yyyy=cmdArgs[0]
            mm=cmdArgs[1]
            dd=cmdArgs[2]
            timeStart     = JULDAY(mm, dd, yyyy, 0, 0, 0)
  timeEnd       = JULDAY(mm, dd, yyyy, 24, 0, 0)
      end else begin
```

```
        ; TESTING VALUES (REMOVE THESE)
        yyyy=2010
        mm=08
                dd=07
                timeStart     = JULDAY(mm, dd, yyyy, 0, 30, 0)
                timeEnd       = JULDAY(mm, dd, yyyy, 24, 20, 0)
        end

        window, 0, xsize=1000,ysize=600

; Convert the date specified to a julian date
        mDate     = JULDAY(mm, dd, yyyy, 0, 0, 0)
        smoothingPeriodTime=30
channel  = 4

; Create parameters to be used in the performing wavelet analysis
waveletParameters = CREATE_STRUCT(              $
                'wv_family',          'morlet', $
                'wv_order',           10,       $
                'wv_signal_T',        [60,100], $
                'wv_pad',             1,        $
                'wv_signif',          0.99,     $
                'signal_threshold', 0.015d      $
                )
; Create parameters to be used in flare detection
flareDetectionParameters = CREATE_STRUCT(       $
                'power_response',         0.d,    $
                'median_power_response', 0.d,    $
                'min_power_response',     0.06d $
                )
; Create parameters to be used in parsing the SolarSoft archive
solarSoftParameters = CREATE_STRUCT(                          $
                'analyse_time_before',    0.d/(24.d * 60.d), $
                'analyse_time_after',     0.d/(24.d * 60.d)  $
                )

; Set parameters to be used when plotting the LYRA signal
timePlotParameters = CREATE_STRUCT(                    $
                'plot_type',      'time',              $
                'save_plot',      1,                   $
                'color',          color24([255,0,0]),  $
                'plot_channel',   '4',                 $
                'ynozero',        1                    $
                )
; Set parameters to be used when plotting the LYRA signal
waveletplotParameters = CREATE_STRUCT(        $
                'plot_type',   'wv_contour', $
                'save_plot',   1,            $
```

```
                'range',       [0,0.1],       $
                'levels',      50,            $
                'ct_reverse', 1,              $
                'ct',          14,            $
                'signif_color', color24([255,2,0]),$
                'coi_color',   color24([100,255,0]) $
                )
; Set parameters to be used when plotting the LYRA signal
powerResponsePlotParameters = CREATE_STRUCT(     $
                'plot_type',   'power_response', $
                'save_plot',   1,                $
                'color',       color24([0,0,0])  $
                )

;-----------------------------------
; RETRIEVE DATA FOR ANALYSIS
;-----------------------------------
; Get the FITS data
data = p2lc_getFitsData(mDate, 'std')
if ptr_valid(data) eq 0 then $
  return

; Determine the time windows to analyze (i.e. windows of data between large
    angle rotations)
timeWindows = p2lc_getwindows(mDate, data, timeStart, timeEnd)
if size(timeWindows, /type) eq 2 then begin
  print, "NO DATA"
  return
end

; Get the list of flares corresponding to the desired date by parsing the
    solarsoft archive
solarSoftList = p2lc_getsolarsoftflarelist(string(yyyy,format='(I04)')+'/'+
    string(mm,format='(I02)')+'/'+string(dd,format='(I02)'), $
                                           solarSoftParameters.
                                               analyse_time_before,
                                               solarSoftParameters.
                                               analyse_time_after)
;===================================
; Add plot of entire day
;===================================
dateLabel = LABEL_DATE(DATE_FORMAT = $
            ['%I:%S', '%H', '%D_%M_%Y'])
!x.tickformat = $
          ['LABEL_DATE','LABEL_DATE','LABEL_DATE']
!x.tickunits  = $
          ['Time',        'Hour',       'Day']
!x.tickinterval = 5
```

```
!x.ticklayout    = 0
!p.background    = color24([255,255,255])
!p.multi         = 0
plot, (*data).time, smooth((*data).channel4, 50),/ynozero,color=color24
    ([0,0,0]), background=color24([255,255,255]),position=[0.05,0.15,.95,.95],
    xstyle=1,linestyle=0

for i=0, n_elements(solarSoftList) -1 do begin
  if ptr_valid(solarSoftList[i]) eq 0 then continue
  event = *solarSoftList[i]
  flareRegion = where( ((*data).time gt event.EVENTSTARTJD) and ((*data).time
      lt event.EVENTENDJD), count)
  if count gt 0 then begin
    flareData = ((*data).channel4)[flareRegion]
    timeData = ((*data).time)[flareRegion]
    oplot, timeData, flareData, color=color24([255,0,0])
  end
end
overViewPlotName = p2lc_savePlot(CREATE_STRUCT('plot_type', 'OVERVIEW'),
    ptr_new((*data).time))
;=============================================
; END feature, add plot of entire day
;=============================================

; Create an array of pointers to store analysis on the data windows
windowStats    = ptrarr(n_elements(timeWindows))

;=============================================
; ITERATING OVER DATA FOR ANALYSIS
;=============================================
; Iterate over the available windows of data and perform analysis;;;
      for tWindowInd = 0, n_elements(timeWindows)-1 do begin
    ; This structure is generated simply to store the filenames of plots that
        are saved by the tool for each window.
    plotNames  =  CREATE_STRUCT(              $
                  'time_plot_name',      '', $
                  'response_plot_name', '', $
                  'wavelet_plot_name',  '', $
                  'fft_plot_name',       ''  $
                  )

    ; Get the timewindow
    timeWindow = *timeWindows[tWindowInd]                    ; Get the
        time window specifications to be used for analysis
    if timeWindow.indexStart gt timeWindow.indexEnd then continue ; If there
        is strange LYRA data due to calibration campaigns during the day for
        example, perform next loop
```

```
print, "Analysing window: ", jd2utc(timeWindow.timeStart), " to ", jd2utc(
    timeWindow.timeEnd)
dataWindow     = getDataWindow(data, channel, timeWindow,
    smoothingPeriodTime)                    ; Get the data corresponding
     to the region indicated by the timeWindow.
preAnalysis    = p2lc_getdatastat(dataWindow)
                                                ; Generate some
    simple pre-analysis statistics of data.
flareStats     = p2lc_getflarestat(data, timeWindow,
    flareDetectionParameters, solarSoftList, mDate) ; Generate the power
    response statistics of the data.
waveletStat    = 0 ; ensure reset of stats from later steps

;       dataParameters  =  A strucutre of parameters that contain a
    parameters specifying the channel to analyse.
; Check if flare signal in either detection or SolarSoft archives
if (flareStats.PR_flare) or (flareStats.SS_flare > 0) then begin
    waveletSignal = p2lc_getwavelet(waveletParameters, dataWindow)
                                            ; Perform Wavelet transform.
    waveletStat    = p2lc_getwaveletanalysis(waveletSignal, dataWindow,
        waveletParameters)  ; Generate analysis of Wavelet transform.
    plotNames.wavelet_plot_name = p2lc_plotwavelet(waveletplotParameters,
        waveletSignal, dataWindow)      ; Plot wavelet transform.

    ; Free all pointers in list
    ptrsToFree = [waveletSignal.time, waveletSignal.w_coi, waveletSignal.
        w_power, waveletSignal.w_transform]
    for ptr = 0, n_elements(ptrsToFree)-1 do begin
      if ptr_valid(ptrsToFree[ptr]) then ptr_free, ptrsToFree[ptr]
    end
end

; Create plots of the time-series data and the power-response curve of the
     Al and Zc data.
plotNames.time_plot_name        = p2lc_plotdata(timePlotParameters,
            dataWindow.data,  dataWindow)
plotNames.response_plot_name    = p2lc_plotdata(powerResponsePlotParameters
    , flareStats,        dataWindow)

; Generate post-analysis statistics.
windowStats[tWindowInd] = p2lc_getwindowreport(timeWindow, dataStat,
    flareStats, waveletStat, plotNames, solarSoftList)
; Cleaning up some pointers.
ptrsToFree = [dataWindow.data, dataWindow.data_relInt, dataWindow.time,  $
              flareStats.azAdd, flareStats.azDiv]
for ptr = 0, n_elements(ptrsToFree)-1 do begin
  if ptr_valid(ptrsToFree[ptr]) then ptr_free, ptrsToFree[ptr]
end
```

```
        end

        ; Create a report on the statistics generated from analysis of each
            window.
        windowStats = windowStats[where(ptr_valid(windowStats))]

        final_report = p2lc_getdailystats(windowStats)
        ; Concatonate all parameters used for record keeping.
        analysisParameters = CREATE_STRUCT('timePlotParameters',
            timePlotParameters,             $
                                              'flareDetectionParameters',
                                                  flareDetectionParameters,      $
                                          'waveletplotParameters',
                                              waveletplotParameters,          $
                                          'powerResponsePlotParameters',
                                              powerResponsePlotParameters,    $
                                          'waveletParameters',
                                              waveletParameters               $
                                          )
    overviewStruct = CREATE_STRUCT('Plot_of_day', '<img_src="'+overViewPlotName+'
        "/>')
    ; Create a structure to hold any parameters or analysis.
    ; Note: this is largely for the benefit of report generation, which uses the
        structure of inputs to format the document.
    analysisDataStruct = CREATE_STRUCT('OVERVIEW', overviewStruct, $
                                          'SOLARSOFT_EVENTLIST', solarSoftList,
                                                  $
                                          'FINAL_REPORT',     final_report,
                                                  $
                                          'Parameters', analysisParameters)
    p2lc_writereport, analysisDataStruct, mDate

        ; Clean Up any pointers left.
        PTR_FREE, PTR_VALID()
        print, "---Script finished---"
end
```

A.2.2 p2lc_getdailystats

```
FUNCTION getWindowCorrelationArray, windowStats
    ;+f* p2lc_getdailystats/getWindowCorrelationArray
    ; FUNCTION:
    ;       getWindowCorrelationArray
    ;
    ; PURPOSE:
    ;       Generates an array representing the correlation between different
        variables across solar events as detected by SolarSoft.
    ;
```

```
; EXPLANATION:
;       The array of windowStats is iterated over to create a correlation
    table based on results from each window of analysis.
;
; CALLING SEQUENCE:
;       Result = getWindowCorrelationArray( windowStats )
;
; INPUTS:
;       windowStats = An array of pointers. Each pointer referencing a
    structure returned from 'p2lc_getwindowreport'
;
; OUTPUTS:
;       correlationArray = A 2 dimensional integer array containing the
    correlation values between different parameters in analysis.
;
; SEE ALSO:
;       p2lc_getwindowreport.pro
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-

  num = n_elements(windowStats)
  firstEntry = *(windowStats[0])
  correlationEntries = firstEntry.CORRELATION_WINDOWDATA
  fieldNames = tag_names(correlationEntries)
  numFields  = n_tags(correlationEntries)

  correlationArray = strarr(numFields+1, numFields+1)
  correlationArray[*,*] = '0'
  correlationArray[0,0] = 'Field'

  for i = 0, numFields-1 do begin
    correlationArray[i+1,0] = fieldNames[i]
    correlationArray[0,i+1] = fieldNames[i]
  end

  for i = 0, num-1 do begin
    if ptr_valid(windowStats[i]) eq 0 then continue
    entry = (*(windowStats[i])).CORRELATION_WINDOWDATA
    images = (*(windowStats[i])).plotNames
    for a = 0, n_tags(entry)-1 do begin
      for b = 0, n_tags(entry)-1 do begin
        if size(entry.(a), /type) gt 5 or size(entry.(b), /type) gt 5 then
            continue
        if (entry.(a) and entry.(b)) then begin
          correlationArray[a+1,b+1] = long(correlationArray[a+1,b+1]) + 1
        end
```

```
        end
      end
    end

    return, correlationArray
END

FUNCTION genWindowCorrelationImages, windowStats
  ;+f* p2lc_getdailystats/genWindowCorrelationImages
  ; FUNCTION:
  ;       genWindowCorrelationImages
  ;
  ; PURPOSE:
  ;       Generates an array representing the correlation between different
    variables across solar events as detected by SolarSoft.
  ;
  ; EXPLANATION:
  ;       The array of windowStats is iterated over to create a correlation
    table based on results from each window of analysis.
  ;
  ; CALLING SEQUENCE:
  ;       Result = genWindowCorrelationImages( windowStats )
  ;
  ; INPUTS:
  ;       windowStats = An array of pointers. Each pointer referencing a
    structure returned from 'p2lc_getwindowreport'
  ;
  ; OUTPUTS:
  ;       correlationArrayImages = A set of images corresponding to unique cells
     in the related correlation array.
  ;
  ; SEE ALSO:
  ;       p2lc_getwindowreport.pro
  ;
  ; MODIFICATION HISTORY:
  ;       Initial Version D. Vagg, February 2011
  ;-

    num = n_elements(windowStats)
    firstEntry = *(windowStats[0])
    correlationEntries = firstEntry.CORRELATION_WINDOWDATA
    fieldNames = tag_names(correlationEntries)
    numFields  = n_tags(correlationEntries)

    numUniqueCells = numFields * (numFields+1) / 2
    correlationArrayImages = ptrarr(numFields, numFields)

    for i = 0, num-1 do begin
```

```
    if ptr_valid((windowStats[i])) eq 0 then continue
    entry = (*(windowStats[i])).CORRELATION_WINDOWDATA
    for a = 0, n_tags(entry)-1 do begin
      for b = 0, a do begin
        if size(entry.(a), /type) gt 5 or size(entry.(b), /type) gt 5 then
            continue
        if (entry.(a) and entry.(b)) then begin
          if ptr_valid(correlationArrayImages[a,b]) then begin
            windowList = *(correlationArrayImages[a,b])
            ptr_free, correlationArrayImages[a,b]
            windowList = [windowList, windowStats[i]]
          end else begin
            windowList = [windowStats[i]]
          end
          correlationArrayImages[a,b] = ptr_new(windowList)
        end
      end
    end
  end

  ; Convert the 2D array to a set of structures containing correlated event
      information
  imageReport = CREATE_STRUCT('Number of Windows', num)
  for i=0, numFields-1 do begin
    for j=0, i do begin
      if ptr_valid(correlationArrayImages[i,j]) eq 0 then continue
      windowArr = *(correlationArrayImages[i,j])
      cellIdentifier = fieldNames[i] + '_' + fieldNames[j]
      cellImageList = ptrarr(n_elements(windowArr))

      for k=0, n_elements(windowArr)-1 do begin
        currentWind = *(windowArr[k])
        plotNames = currentWind.plotNames

        windowImageStruct = CREATE_STRUCT( $
                          'PLOT_TEMPORAL', '<img src="'+plotNames.time_plot_name
                              +'"/>', $
                          'PLOT_WAVELET', '<img src="'+plotNames.
                              wavelet_plot_name+'"/>', $
                          'PLOT_FLARE_RESPONSE', '<img src="'+plotNames.
                              response_plot_name+'"/>', $
                          'WINDOW_START', currentWind.timeStart, $
                          'WINDOW_END', currentWind.timeEnd)
        cellImageList[k] = ptr_new(windowImageStruct)
      end
      imageReport = CREATE_STRUCT(imageReport, cellIdentifier, cellImageList)
    end
  end
```

```
    return, imageReport
END

FUNCTION getEventCorrelationArray, windowStats
  ;+f* p2lc_getdailystats/getEventCorrelationArray
  ; FUNCTION:
  ;       getEventCorrelationArray
  ;
  ; PURPOSE:
  ;       Generates an array representing the correlation between different
    variables across solar events as detected by SolarSoft.
  ;
  ; EXPLANATION:
  ;       The array of windowStats is iterated over to create a correlation
    table based on events recorded by the SolarSoft archive.
  ;       So all windows corresponding to an event are analysed to measure the
    correlation between different results of analysis, such as frequency
    detection etc.
  ;       This clarifies analysis somewhat as frequencies may be detected in a
    single window, but a flare could last several windows. This could lead to
    the misunderstanding
  ;       that frequencies were not detected in several flare events as opposed
    to not being detected in portions of a single event.
  ;
  ; CALLING SEQUENCE:
  ;       Result = getEventCorrelationArray( windowStats )
  ;
  ; INPUTS:
  ;       windowStats = An array of pointers. Each pointer referencing a
    structure returned from 'p2lc_getwindowreport'
  ;
  ; OUTPUTS:
  ;       correlationArray = A 2 dimensional integer array containing the
    correlation values between different parameters in analysis.
  ;
  ; SEE ALSO:
  ;       p2lc_getwindowreport.pro
  ;
  ; MODIFICATION HISTORY:
  ;       Initial Version D. Vagg, February 2011
  ;-

    numWind = n_elements(windowStats)
    firstEntry = *(windowStats[0])
    correlationEntries = firstEntry.CORRELATION_EVENTDATA
    fieldNames = tag_names(correlationEntries)
    numFields  = n_tags(correlationEntries)
```

```
; Make the array
correlationArray = strarr(numFields+1, numFields+1)
correlationArray[*,*] = '0'
correlationArray[0,0] = 'Field'
; Put the fields along top and left side
for i = 0, numFields-1 do begin
  correlationArray[i+1,0] = fieldNames[i]
  correlationArray[0,i+1] = fieldNames[i]
end

correlationWindowTable = dblarr(numFields, numWind)
for i = 0, numWind-1 do begin
  if ptr_valid((windowStats[i])) eq 0 then continue
  windowData = (*(windowStats[i])).CORRELATION_EVENTDATA
  for j = 0, numFields-1 do begin
    correlationWindowTable[j,i] = windowData.(j)
  end
end

eventList = correlationWindowTable[0,*]
events = where(eventList gt 0, count)
if count eq 0 then return, correlationArray
eventList = eventList[events]

uniqueEventsInd = uniq(eventList, sort(eventList))
uniqueEvents = eventList[uniqueEventsInd]
eventCount = n_elements(uniqueEvents)

; Create a temporary array containging the (number of fields) * (unique
    events)
; Then fill this for later use
eventCorrelationTable = dblarr(numFields, eventCount)
for i = 0, eventCount-1 do begin
  eventNumber = uniqueEvents[i]
  windowsAtEvent = correlationWindowTable[*, where(correlationWindowTable
      [0,*] eq eventNumber)]
  for j = 0, numFields - 1 do begin
    eventCorrelationTable[j,i] = (max(windowsAtEvent[j,*]) gt 0) ; Ensures
        entry is boolean
  end
end


for i = 0, eventCount-1 do begin
  entry = eventCorrelationTable[*,i]
  for a = 0, n_elements(entry)-1 do begin
    for b = 0, n_elements(entry)-1 do begin
```

```
            if size(entry[a], /type) gt 5 or size(entry[b], /type) gt 5 then
                continue
            if (entry[a] gt 0 and entry[b] gt 0) then begin
              correlationArray[a+1,b+1] = long(correlationArray[a+1,b+1]) + 1
            end
          end
        end
      end

      return, correlationArray
END

FUNCTION getSummaryStats, windowStats
      num = n_elements(windowStats)
      firstTime = strjoin((*windowStats[0]).timeStart, ':')
      lastTime  = strjoin((*windowStats[num-1]).timeEnd, ':')
      summaryStats = CREATE_STRUCT('Window Count', num, $
                                   'time start', firstTime, $
                                   'time end', lastTime)
      return, summaryStats
END

FUNCTION p2lc_getdailystats, windowStats
   ;+f* LYRASCRIPT/p2lc_getdailystats
   ; FUNCTION:
   ;       p2lc_getdailystats
   ;
   ; PURPOSE:
   ;       Summarise analysis from a collection of stats performed on each
     section of LYRA data.
   ;
   ; EXPLANATION:
   ;       Analysis of LYRA data is performed a section at a time. Analysis is
     generated for each window, and the results are stored in a structure.
   ;       The results of these statistics are summarised in this function.
   ;
   ; CALLING SEQUENCE:
   ;       Result = p2lc_getdailystats( windowStats )
   ;
   ; INPUTS:
   ;       windowStats = An array of pointers. Each pointer referencing a
     structure returned from 'p2lc_getwindowreport'
   ;
   ; OUTPUTS:
   ;       analysis                    = A structure containing a summarised
     report of all analysis performed over multiple sections of LYRA data.
   ;       analysis.Summary_Stats      = A basic summary of some statistics,
     including window count, time range analysed.
```

```
;       analysis.Window_CORRELATION = An array showing the relation between
    multiple parameters, such as flare and frequency detection, across multiple
     windows.
;       analysis.Event_CORRELATION  = An array showing the relation between
    multiple parameters, such as flare and frequency detection, across solar
    events.
;       Solar events are determined using the solarsoft archives.
;
; SEE ALSO:
;       p2lc_getwindowreport.pro
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-

    if n_elements(windowStats) eq 0 or ptr_valid(windowStats[0]) eq 0 then $
      return, -1

    windowCorrelationArray  = getWindowCorrelationArray(windowStats)
    windowCorrelationImages = genWindowCorrelationImages(windowStats)
    eventCorrelationArray   = getEventCorrelationArray(windowStats)
    summaryStats            = getSummaryStats(windowStats)

    analysis = CREATE_STRUCT(                                    $
          'Summary_Stats', summaryStats,                         $
          'Window_CORRELATION', windowCorrelationArray,          $
          'Window_CORRELATION_IMAGES', windowCorrelationImages, $
          'Event_CORRELATION', eventCorrelationArray)

    return, ptr_new(analysis)
END
```

A.2.3 p2lc_getdatastat

```
FUNCTION p2lc_getdatastat, dataWindow
    ;+f* LYRASCRIPT/p2lc_getdatastat
    ; FUNCTION:
    ;       p2lc_getdatastat
    ;
    ; PURPOSE:
    ;       Return a structure containing basic statistics of a specified
       section of LYRA data. This is also a placeholder for future functionality
        to be added.
    ;
    ; CALLING SEQUENCE:
    ;       result = p2lc_getdatastat( dataWindow )
    ;
    ; INPUTS:
```

```
;       dataWindow = A structure returned from the 'getDataWindow' function
    containing both a section of data and corresponding time data.
;
; OUTPUTS:
;       Analysis = A pointer to a structure containing all statistics
    generated for the section of data.
;
; NOTES:
;       The data structure that is read is based on the output from using
    MRDFITS() to open the FITS data
;
; SEE ALSO:
;       getDataWindow
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-

data = *(dataWindow.data)

length  = n_elements(data)
maxVal  = max(data)
minVal  = min(data)
medVal  = median(data)
meanVal = mean(data)
stdDVal = stddev(data)

analysis = CREATE_STRUCT('length',   length, $
                         'maxVal',   maxVal, $
                         'minVal',   minVal, $
                         'medVal',   medVal, $
                         'meanVal',  meanVal,$
                         'stdDVal',  stdDVal $
                          )

return, analysis
END
```

A.2.4 p2lc_getfitsdata

```
PRO setAbsoluteTime, fitsData, header
        ;+p* p2lc_getfitsdata/setAbsoluteTime
    ; FUNCTION:
    ;       setAbsoluteTime
    ;
    ; PURPOSE:
    ;       Modifies the 'time' component of a day of LYRA data to give absolute
        time in JulianDay rather than beginning at 0
```

```
;
; EXPLANATION:
;       The time column of a LYRA FITS file gives the time in JulianDays
    beginning from 0.0
;       This procedure adds an offset to the time data to give the absolute
    time in JulianDays.
;
; CALLING SEQUENCE:
;       setAbsoluteTime, fitsData, header
;
; INPUTS:
;       fitsData    =       A pointer to the FITS structure returned from
    MRDFITS.
;       header              =   The header of the FITS file.
;
; OUTPUTS:
;       None.
;
; NOTES:
;       The time correction is based on the 'DATE-OBS' parameter in the
    header file. The separation
;               between DATE and OBS has changed once in the past from '-'
    to '_' within different FITS files.
;               The string is matched using regular expressions to prevent
    this having an effect.
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-

    ; Find line in header showing first time of observation
    dateStr = stregex(header, 'DATE.OBS')
    dateStr = header[where(dateStr ne -1)]

    ; Extract the time and date from that string
    date = stregex(dateStr, ".{4}-.{2}-.{2}", /extract)
    time = stregex(dateStr, ".{2}:.{2}:.{2}", /extract)

    ; Split both date and time into components
    date = strsplit(date, '-', /extract)
    time = strsplit(time, ':', /extract)

    ; Create a julian day value based on the observation start time
    jdAddition = julday(date[1], date[2], date[0], time[0], time[1], time
        [2])

    ; Add the JD value of observation start time to the time array in the
        data
```

```
        (*fitsData).time = ((*fitsData).time / (24.d * 60.d * 60.d)) +
            jdAddition
END

FUNCTION p2lc_getfitsdata, date, type
        ;+* LYRASCRIPT/p2lc_getfitsdata
    ; FUNCTION:
    ;       p2lc_getfitsdata
    ;
    ; PURPOSE:
    ;       Returns a pointer to a structure containing LYRA data
    ;
    ; CALLING SEQUENCE:
    ;       result = p2lc_getfitsdata(date, type)
    ;
    ; INPUTS:
    ;       date    =   The date of data to be returned in JulianDay format.
    ;       type        =   A string containing the 3 characters of the LYRA
       FITS data specifying the type, i.e. 'std'.
    ;
    ; OUTPUTS:
    ;       A pointer to the structure loaded by reading the FITS file
       corresponding to the date and type of data using the MRDFITS function.
    ;
    ; NOTES:
    ;       The data structure that is read is based on the output from using
       MRDFITS() to open the FITS data.
    ;       The variable 'type' is used to specify which file to load by finding
        the filename which contains the string.
    ;
    ; SEE ALSO:
    ;       MRDFITS
    ;
    ; RESTRICTIONS:
    ;       Currently only reads data is 'std' type is given
    ;
    ; MODIFICATION HISTORY:
    ;       Initial Version D. Vagg, February 2011
    ;-

        ; Separate 'date' to month, day, year
        CALDAT, date, mm, dd, yyyy

        ; Determine name of FITS file
        fileDir = settings('data_dir') + $
        string(yyyy,format='(I04)')+'/'+string(mm,format='(I02)')+'/'+string(dd,
            format='(I02)')+'/'
  fname = fileDir+'*'+type+'*.fits'
```

```
        if ((file_test(fileDir) eq 0) or (file_test(fname) eq 0)) then begin
                error, 'fits_file_missing', fname
                return, 0
        end

        fitsData = ptr_new(mrdfits(fname, 1))

        setAbsoluteTime, fitsData, headfits(fname)

        return, fitsData
END
```

A.2.5 p2lc_getflarestat

```
FUNCTION getMedianResponse, data
   ;+f* p2lc_getflarestat/getMedianResponse
   ; FUNCTION:
   ;               getMedianResponse
   ;
   ; PURPOSE:
   ;               Get the median 'power response' using the aluminium and
      zirconium channels of the LYRA data.
   ;
   ; EXPLANATION:
   ;    When a flare occurs, the amplitudes of the Zirconium and Aluminium
      channels of LYRA rise.
   ;    However, there is a lag between the rise in the Zirconium channel and the
      Aluminium channel.
   ;    This lag is used in flare detection algorithms to automatically trigger
      flare analysis.
   ;    The median value is used for comparisons in response changes.
   ;    Value are compared against this median due to the degredation of LYRA
      detectors over time in Unit 2.
   ;
   ; CALLING SEQUENCE:
   ;               result  =       getMedianResponse( data )
   ;
   ; INPUTS:
   ;               data    =       A structure containing of LYRA data returned
      from the MRDFITS function.
   ;
   ; OUTPUTS:
   ;      medianResponse  =       The median power response for a day of LYRA data
       determined using the Aluminium and Zirconium channels
   ;                                                                   (fields '
      channel3' and 'channel4', respectively)
   ;
```

```
  ; NOTES:
  ;             The 'power response' is determined by obtaining the addition of
     the aluminium and zirconium signal amplitudes and the division of the two
     channels, then diving the addition by the division.
  ;             During a flare, there is a lag between the response in the
     zirconium and alumium channels, this lag in response is used to determine
     the 'power response'.
  ;             The data used is smoothed before-hand to remove anomalous spikes
      such as those caused by the South Atlantic Anomaly.
  ;
  ; MODIFICATION HISTORY:
  ;             Initial Version D. Vagg, February 2011
  ;-

        aluData = smooth((*data).channel3,500)
        zrcData = smooth((*data).channel4,500)

        AZadd = aluData+zrcData
        AZdiv = zrcData/aluData

        medianAdd = median(AZadd, /double)
        medianDiv = median(AZdiv, /double)

        return, medianAdd/medianDiv
END

; Returns a structure containing wavelet transformations and data
FUNCTION p2lc_getflarestat, data, timeWindow, flareParameters, solarSoftList,
    date
  ;+f* LYRASCRIPT/p2lc_getflarestat
        ; FUNCTION:
        ;               p2lc_getflarestat
        ;
        ; PURPOSE:
        ;               Returns a structure containing several variables
           relating to flare detection including two boolean flags,
        ;               'PR_flare' and 'SS_flare' which are used to indicate if
           a flare was detected using Power Response or SolarSoft.
        ;
        ; EXPLANATION:
        ;   Flare detection is attempted using LYRA data. A list of events from
           SolarSoft is read at the same time to ensure no flares are missed.
        ;   The flare detection is based on the lag between the responses in the
            Aluminium and Zirconium channels of the LYRA instrument.
        ;   The response is measured as a ratio of the difference between the
           max and median against the median value.
        ;   Once this ratio reaches above a configurable threshold, flare
           analysis is performed.
```

```
;
; CALLING SEQUENCE:
;   result = p2lc_getflarestat(data, timeWindow, flareParameters,
    solarSoftList, date)
;
; INPUTS:
;               data                =       A structure containing
   LYRA data channels.
;               timeWindow          =   A structure containing details
    of the range of LYRA data to view including 'start' and 'end' indexes
    .
;               flareParameters =       A structure of parameters that
    contain a parameters specifying the channel to analyse.
;               solarSoftList   = A structure containing details of any
    events in the SolarSoft archive for a particular date.
;               date            = The date of data being analysed.
;
; OUTPUTS:
;               flareStat                       =   A structure
    containing various results from flare detection routines.
;   flareStat.AZdiv             = The division of Aluminium and
    Zirconium channels.
;   flareStat.AZadd             = The addition of Aluminium and Zirconium
   channels.
;   flareStat.PR_median_day     = The median value for Power Response for a
   day of LYRA data.
;   flareStat.PR_delta          = The difference between the maximum power
   response and the median value.
;   flareStat.PR_delta_threshold  = The threshold value required before a
   detection is flagged.
;   flareStat.PR_pc_median      = The ratio of the 'PR_delta' against '
   PR_median_day'.
;   flareStat.PR_flare          = A boolean set to 1 if Power Response
   algorithms detected a flare.
;   flareStat.SS_flare          = A boolean set to 1 if a flare was
   specified in the SolarSoft archives.
;   flareStat.SS_startEvent     = A boolean set to 1 if the window of data
   being considered contained the beginning of a SolarSoft event.
;   flareStat.SS_endEvent       = A boolean set to 1 if the window of data
   being considered contained the ending of a SolarSoft event.
;   flareStat.PR_matched_SS_event = A boolean set to 1 if the Power Response
   algorithms and SolarSoft archives detected a flare.
;
; NOTES:
;               Flare detection using LYRA data is using an experimental
    procedure and is not guaranteed to detect all flares.
;
; SEE ALSO:
```

```
        ;    p2lc_getsolarsoftflarelist
        ;
        ; MODIFICATION HISTORY:
        ;    Initial Version D. Vagg, February 2011
        ;-
        CALDAT, date, month, day, year

        if (flareParameters.median_power_response eq 0) then $
           flareParameters.median_power_response = getMedianResponse(data)

        aluChannel = smooth((*data).channel3,500)
zrcChannel = smooth((*data).channel4,500)
zrcData = zrcChannel[ timeWindow.indexStart:timeWindow.indexEnd ]
aluData = aluChannel[ timeWindow.indexStart:timeWindow.indexEnd ]

AZadd = aluData+zrcData
AZdiv = zrcData/aluData

medianPR     = flareParameters.median_power_response
minPR        = flareParameters.min_power_response

powerRespDelta = max(AZadd/AZdiv - flareParameters.median_power_response)
PRDeltaRatio = powerRespDelta/medianPR

flarePR = (PRDeltaRatio gt minPR) ; if the ratio of powerResponse delta is
    greater than the threshold, set this flag to 1

; Check if the SolarSoft archive indicates a flare occuring somewhere in this
    window
SS_flare = 0
SS_startEvent = 0
SS_endEvent = 0
matchedCurrentEvent = 0
if size(solarSoftList,/type) eq 10 then begin
  for i=0, n_elements(solarSoftList)-1 do begin
    flareDetails = *(solarSoftList[i])

    flareStart = flareDetails.eventStartJD
    flareEnd   = flareDetails.eventEndJD

    if ((timeWindow.timeStart gt flareStart) and (timeWindow.timeEnd lt
       flareEnd))   OR $          ; Window between flare start & end
       ((timeWindow.timeStart lt flareStart) and (timeWindow.timeEnd gt
          flareStart)) OR $          ; Flare starting in window
       ((timeWindow.timeStart lt flareEnd)    and (timeWindow.timeEnd gt
          flareEnd))   then begin  ; Flare ending in window
        SS_flare = i+1
```

```
            SS_startEvent = ((timeWindow.timeStart lt flareStart)  and (timeWindow
                .timeEnd gt flareStart))
            SS_endEvent = ((timeWindow.timeStart lt flareEnd)    and (timeWindow.
                timeEnd gt flareEnd))
            if flarePR then begin ; Replace the current event in the structure
                with a new one indicating that it has been flagged by Power
                Response algorithms.
              ptr_free, solarSoftList[i]
              matchedCurrentEvent = 1 - flareDetails.lag_matched
              flareDetails.lag_matched = 1
              solarSoftList[i] = ptr_new(flareDetails)
            end
            break
        end
      end
    end

    flareStat = CREATE_STRUCT('AZdiv',                ptr_new(AZdiv), $
                              'AZadd',                ptr_new(AZadd), $
                              'PR_median_day',        medianPR,       $
                              'PR_delta_threshold', minPR,            $
                              'PR_delta',             powerRespDelta, $
                              'PR_pc_median',         PRDeltaRatio,   $
                              'PR_flare',             flarePR,        $
                              'SS_flare',             SS_flare,       $
                              'SS_startEvent',        SS_startEvent,  $
                              'SS_endEvent',          SS_endEvent,    $
                              'PR_matched_SS_event',matchedCurrentEvent  $
                              )
        return, flareStat
END
```

A.2.6 p2lc_getsolarsoftflarelist

```
FUNCTION getHtml, url
  ;+f* p2lc_getsolarsoftflarelist/getHtml
  ; FUNCTION:
  ;       getHtml
  ;
  ; PURPOSE:
  ;       Returns the html from a URL.
  ;
  ; CALLING SEQUENCE:
  ;       HTML = getHtml(url).
  ;
  ; INPUTS:
  ;       url = The URL of the page to return the HTML from.
  ;
```

```
  ; OUTPUTS:
  ;       HTML = the raw HTML from the page.
  ;
  ; MODIFICATION HISTORY:
  ;       Initial Version D. Vagg, February 2011
  ;-
  print, "--------Retrieving HTML---------"
  print, "URL: " + string(url)
  response = webget(url,timeout=5)
  html = response.text
  info = response.header
  print, "HEADER: " + info
  return, html
END

FUNCTION parseEvents, html, rowIdentifierString
  ;+f* p2lc_getsolarsoftflarelist/parseEvents
  ; FUNCTION:
  ;       parseEvents
  ;
  ; PURPOSE:
  ;       Separates the HTML of a page into blocks using the '
    rowIdentifierString' parameter as a separator.
  ;
  ; CALLING SEQUENCE:
  ;       HTMLblocks = parseEvents(html, rowIdentifierString)
  ;
  ; INPUTS:
  ;       html = The raw HTML of the page to split up.
  ;       rowIdentifierString = A string used to split the HTML into blocks.
  ;
  ; OUTPUTS:
  ;       HTMLBlocks = Blocks of HTML separated using the 'rowIdentifierString'
    string
  ;
  ; MODIFICATION HISTORY:
  ;       Initial Version D. Vagg, February 2011
  ;-

  print, "---------Retrieving Latest Events HTML---------"
  EventBlocks = strarr(n_elements(html))
  lineBuffer = ""

  for lInd = 0, n_elements(html)-1 do begin
    line = html[lInd]
    if strcmp(line, rowIdentifierString, strlen(rowIdentifierString)) eq 1 then
      begin
      EventBlocks[lInd] = lineBuffer
```

```
      lineBuffer = line
    end else begin
      lineBuffer += line
    end
  end
  goodEvents = where(EventBlocks ne '',count)
  if count eq 0 then return, -1
  EventBlocks = EventBlocks[goodEvents]
  return, EventBlocks
END

FUNCTION getEventPage, eventList, dateRegex, eventUrlIdentifier, aDate
  ;+f* p2lc_getsolarsoftflarelist/getEventPage
  ; FUNCTION:
  ;       getEventPage
  ;
  ; PURPOSE:
  ;       Parses an 'eventList' corresponding to each table row of the page
    showing events for each date in the solarsoft archive.
  ;
  ; CALLING SEQUENCE:
  ;       URLComponent = getEventPage(eventList, dateRegex, eventUrlIdentifier,
    aDate)
  ;
  ; INPUTS:
  ;       eventList = An array of strings corresponding to the table rows of
    flare records for each date.
  ;       dateRegex = The regular expression to match date strings
  ;       eventUrlIdentifier = The regular expression to match the URL specified
     in the hyperlink in each row
  ;       aDate = The date for events to be found for in the format: yyyy/mm/dd
  ;
  ; OUTPUTS:
  ;       eventUrl = The string found in the hyperlink on the desired row
  ;
  ; MODIFICATION HISTORY:
  ;       Initial Version D. Vagg, February 2011
  ;-
  analyseDate = strsplit(aDate, '/',/extract)
  analyseDate = julday(analyseDate[1], analyseDate[2], analyseDate[0])
  for eventInd = 0, n_elements(eventList)-1 do begin
    event = eventList[eventInd]
    event = strsplit(event, '<',/extract)
    dates = stregex(event, dateRegex, /extract)
    dateMatches = where(dates ne '', dCount)
    if dCount lt 2 then $
      continue
    dates = dates[dateMatches]
```

```
    eventStart = dates[0]
    eventEnd   = dates[1]
    eventStart = strsplit(eventStart, '/',/extract)
    eventEnd   = strsplit(eventEnd, '/',/extract)
    eventStart = julday(eventStart[1], eventStart[2], eventStart[0])
    eventEnd   = julday(eventEnd[1], eventEnd[2], eventEnd[0])
    if (eventStart lt analyseDate) and (eventEnd gt analyseDate) then begin
      event = eventList[eventInd]
      eventUrl = stregex(event, eventUrlIdentifier, /extract, /subexpr)
      if n_elements(eventUrl) eq 2 then $
        return, eventUrl[1]
    end
  end
  return, -1
END

FUNCTION parseEventDetails, eventHtmlBlocks, timeRegex, flareStrengthRegex,
    analyseDate, trimBack, trimForward
  ;+f* p2lc_getsolarsoftflarelist/parseEventDetails
  ; FUNCTION:
  ;       parseEventDetails
  ;
  ; PURPOSE:
  ;       Parses strings of HTML for flare details of a specified date.
  ;
  ; CALLING SEQUENCE:
  ;       events = parseEventDetails(eventHtmlBlocks, timeRegex,
    flareStrengthRegex, analyseDate)
  ;
  ; INPUTS:
  ;       eventHtmlBlocks     = An array of strings corresponding to the table
    rows of a flare page for a specified date.
  ;       timeRegex           = The regular expression to match time strings
  ;       flareStrengthRegex  = The regular expression to match flare strength
  ;       analyseDate         = The date for events to be found for in the
    format: yyyy/mm/dd
  ;       trimBack            = Amount of time to trim back from the ending time
     of the solarSoft event in JulianDays.
  ;       trimForward         = Amount of time to trim forward from the
    beginning time of the solarSoft event in JulianDays.
  ;
  ; OUTPUTS:
  ;       eventList                = A structure of pointers. Each pointer
    referencing a structure containing the start/end/peak times of the flare as
     well as strength (GOES)
  ;       eventList.eventStart     = Starting time of an event in UTC format.
  ;       eventList.eventEnd       = Ending time of an event in UTC format.
  ;       eventList.eventStartJD   = Starting time of an event in JulianDays.
```

```
;       eventList.eventEndJD    = Ending time of an event in JulianDays.
;       eventList.eventPeak     = Time of event peak as parsed from the online
    archive.
;       eventList.eventStrength = Strength of event as string (i.e. 'B3.0').
;       eventList.freq_found    = Boolean value that can be set to 1 if
   frequencies are detected in later analysis.
;       eventList.lag_matched   = Boolean value that can be set to 1 if flare
   detection using lag between LYRA data detects the event.
;
; NOTES:
;       More parameters can be calculated/parsed as required.
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-

eventList = ptrarr(n_elements(eventHtmlBlocks))
for eInd = 0, n_elements(eventHtmlBlocks)-1 do begin
  event = eventHtmlBlocks[eInd]
  if (stregex(event, analyseDate, /boolean)) then begin
    event = strsplit(event, '<td>', /extract)
    times = stregex(event, timeRegex, /extract)
    strength = stregex(event, flareStrengthRegex, /extract)
    times = times[where(times ne '')]
    strength = strength[where(strength ne '')]
    if n_elements(times) lt 3 then $
      continue

    dateStr = strSplit(analyseDate, '/', /extract)
    month = dateStr[1]
    day = dateStr[2]
    year = dateStr[0]

    eventStartStr = times[0]
    eventEndStr  = times[1]
    eventPeak  = times[2]
    strength   = strength[0]
    eventStartJD = strsplit(eventStartStr, ':', /extract)
    eventStartJD = julday(month, day, year, eventStartJD[0], eventStartJD[1],
       eventStartJD[2])
    eventEndJD   = strsplit(eventEndStr, ':', /extract)
    eventEndJD   = julday(month, day, year, eventEndJD[0], eventEndJD[1],
       eventEndJD[2])
    eventStartJD += trimForward
    eventEndJD   -= trimBack
    eventStart = jd2utc(eventStartJD)
    eventEnd   = jd2utc(eventEndJD)
```

```
      eventList[eInd] = ptr_new(CREATE_STRUCT('eventStart', eventStart, $
                                              'eventEnd', eventEnd, $
                                              'eventStartJD', eventStartJD, $
                                              'eventEndJD', eventEndJD, $
                                              'eventPeak', eventPeak, $
                                              'eventStrength', strength, $
                                              'lag_matched', 0))
    end
  end
  validEvents = where(ptr_valid(eventList), eventCount)
  if eventCount gt 0 then begin
    return, eventList[validEvents]
  end else $
    return, -1
END

FUNCTION p2lc_getsolarsoftflarelist, aDate, trimBack, trimForward
  ;+f* LYRASCRIPT/p2lc_getsolarsoftflarelist
  ; FUNCTION:
  ;       p2lc_getsolarsoftflarelist
  ;
  ; PURPOSE:
  ;       Parses the solarsoft archive (http://www.lmsal.com/solarsoft) for
    events occuring on the date specified.
  ;
  ; CALLING SEQUENCE:
  ;       result = p2lc_getsolarsoftflarelist('yyyy/mm/dd', trimBack,
    trimForward)
  ;
  ; INPUTS:
  ;       aDate       = A string of the date for events to be found for. Format:
     yyyy/mm/dd.
  ;       trimBack    = Amount of time to trim back from the ending time of the
    solarSoft event in JulianDays.
  ;       trimForward = Amount of time to trim forward from the beginning time
    of the solarSoft event in JulianDays.
  ;
  ; OUTPUTS:
  ;       eventList = A structure of pointers. Each pointer referencing a
    structure containing the start/end/peak times of the flare as well as
    strength as detected using GOES.
  ;
  ; NOTES:
  ;       The current implementation of this function gets events parsed
    directly from the HTML code from the online archive using regular
    expressions.
  ;       For this reason, it's effectiveness is subject to change if the
    website changes or is edited in any way.
```

```
  ;
  ; MODIFICATION HISTORY:
  ;       Initial Version D. Vagg, February 2011
  ;-

  ; This loads an info file for parsing
  ; Test if file exists
  LatestEvents = 'latest_events_archive.html'
  eventPagePrefix = 'http://www.lmsal.com/solarsoft/'
  rowIdentifierString = '<tr'
  eventUrlIdentifier = ('HREF="(last_events[^"]*)"')
  dateRegex = ('([0-9]{4}/[0-9]{2}/[0-9]{2})')
  timeRegex = ('([0-9]{2}\:[0-9]{2}\:[0-9]{2})')
  flareStrengthRegex = ('[ABCMX]{1}[0-9]{1}\.[0-9]{1}')

  ; Grab the html from the solarsoft archive page
  eventHtml = getHtml(eventPagePrefix + LatestEvents)
  ; Parse the html to individual table entries based on the table row identifier
  events = parseEvents(eventHtml, rowIdentifierString)
  ; Iterate over each 'event' or row of table
  eventPageForDay = getEventPage(events, dateRegex, eventUrlIdentifier, aDate)
  if (eventPageForDay eq -1) then return, -1
  ; Grab html of event page for day of interest
  eventDayHtml = getHtml(eventPagePrefix + eventPageForDay)
  ; Parse this for the html of each row
  dayEventBlocks = parseEvents(eventDayHtml, rowIdentifierString)
  ; Parse this for event details, ready to be used with IDL
  eventList = parseEventDetails(dayEventBlocks, timeRegex, flareStrengthRegex,
     aDate, trimBack, trimForward)

  return, eventList
END
```

A.2.7 p2lc_getwavelet

```
; Determine the scaling parameters to be used in wavelet analysis
FUNCTION getScaling, waveletParameters, dataParameters, data
        ;+f* p2lc_getwavelet/getScaling
        ; FUNCTION:
        ;       getScaling
        ;
        ; PURPOSE:
        ;       Generate a set of scaling parameters to be used in wavelet
           analysis.
        ;
        ; EXPLANATION:
        ;       This function only returns static values for scaling in the
           current implementation but it intended to be improved in later
```

```
        iterations.
;       At the moment, it is largely a placeholder for more advanced
        functionality.
;
; CALLING SEQUENCE:
;       result = getScaling(waveletParameters, dataParameters, data)
;
; INPUTS:
;                   waveletParameters    =    A structure containing
        parameters to be used in wavelet analysis (i.e. wavelet family and
        order).
;                   dataParameters       = A strucutre of parameters
        that contain a parameters specifying the channel to analyse.
;       data                                                  =    A
        structure containing LYRA data channels.
;
; OUTPUTS:
;       waveScaling     =       A structure containing wavelet scaling
        parameters to be used for wavelet analysis.
;
; NOTES:
;       This function is returning static values until further
        development is possible.
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-
waveScaling = CREATE_STRUCT(        $
              'start',    100,    $    ; The scale to begin with
        'delta',    0.05,       $    ; The logarithmic increase between
              each scale
        'n_scales', 110     $    ; The number of scales to be used


return, waveScaling
END

FUNCTION doWavelet, waveletParameters, scaleParameters, dataWindow, scales
        ;+f* p2lc_getwavelet/doWavelet
        ; FUNCTION:
        ;       doWavelet
        ;
        ; PURPOSE:
        ;       Get the continuous wavelet transform of the section of LYRA data
            in 'dataWindow'.
        ;
        ; CALLING SEQUENCE:
```

```
;       result = doWavelet(waveletParameters, scaleParameters,
    dataWindow, scales)
;
; INPUTS:
;                   waveletParameters = A structure containing
    parameters to be used in wavelet analysis (i.e. wavelet family and
    order)
;                   scaleParameters          =      A structure
    containing the parameters to be used for scales including start scale
    , dscale (delta scale), and number of scales.
;       dataWindow                                 =      A structure
    containing a subsection of LYRA data and time data.
;       scales                                              =      A
    pointer used to store the scales used in the transform.
;
; OUTPUTS:
;       wave            =       A pointer to the wavelet transform.
;       scales  = A pointer to a 1D array of scales used in the wavelet
    transform.
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-

    wave = wv_cwt(*(dataWindow.data_relInt),      $   ; Data to be analyzed
                        waveletParameters.wv_family,          $   ;
                            Wavelet Family
                        waveletParameters.wv_order,           $   ;
                            Wavelet Order
                        start_scale = scaleParameters.start,  $   ;
                            Starting Scale of wavelet
                        dscale      = scaleParameters.delta,  $   ;
                            Log Increment of scale
                        nscale      = scaleParameters.n_scales, $   ;
                            Number of Scales to use
                        scale       = scales,
                                      $   ; Store scales used in a
                            variable called 'scales'
                        pad         = waveletParameters.wv_pad  $   ;
                            Should padding be performed
                        )
    return, ptr_new(wave)
END

FUNCTION scaleToPeriods, waveletScales, time, wv_family, wv_order
  ;+f* p2lc_getwavelet/scaleToPeriods
  ; FUNCTION:
  ;       scaleToPeriods
```

```
;
; PURPOSE:
;       Convert the scales used in the continuous wavelet transform to Fourier
    periods
;       (Reference: C. Torrence and G.P. Compo et al, A Practical Guide to
    Wavelet Analysis).
;
; CALLING SEQUENCE:
;       result = doWavelet(waveletParameters, scaleParameters, dataWindow,
    scales)
;
; INPUTS:
;       wv_family     = A string containing the name of the mother-wavelet
    family used to obtain the wavelet transform.
;       wv_order      = An integer of the order of the mother-wavelet family
    used to obtain the wavelet transform.
;       time          = A pointer to the corresponding time array of the
    section of LYRA data that was converted using the wavelet transform.
;       waveletScales = A pointer to the scales used in the wavelet transform
    .
;
; OUTPUTS:
;       fPeriods  = An array of identical dimensions to that in 'waveletScales
    ' containing the fourier periods related to the corresponding entries in '
    waveletScales'
;
; NOTES:
;       Requires Morlet mother-wavelet to be used in analysis.
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-

time = *time
if (wv_family ne 'morlet') then begin
  print, 'Unable to convert scale using ' + wv_family + ' wavlet'
  return, waveletScales
end
conversionPeriodScale = (4*!Pi)/(wv_order + sqrt(2+(wv_order*wv_order))) ; One
    wavelength = converstionPeriodScale * scale
numSec = (time[n_elements(time)-1] - time[0])*24*60*60 ; CAREFUL! If you put
    24*60*60 in brackets; i.e. (24*60*60), it will cause issues..
numPointsPerSec = ceil(n_elements(time)/numSec)

fPeriods = waveletScales * conversionPeriodScale
fPeriods = fPeriods / numPointsPerSec
return, fPeriods
END
```

```
FUNCTION getCOI, waveletPtr, scales, time
  ;+f* p2lc_getwavelet/getCOI
  ; FUNCTION:
  ;       getCOI
  ;
  ; PURPOSE:
  ;       Generate an array of identical dimensions to the wavelet transform
    containing boolean values indicating if a point in the array
  ;       is affected by the Cone of Influence (1) or not (0).
  ;       (Reference: C. Torrence and G.P. Compo et al, A Practical Guide to
    Wavelet Analysis).
  ;
  ; CALLING SEQUENCE:
  ;       coneOfInfluence = getCOI( waveletPtr, scales, time )
  ;
  ; INPUTS:
  ;       waveletPtr = A pointer to the wavelet transform.
  ;       scales     = An array containing the scales used in the wavelet
    transform.
  ;       time       = A pointer to the corresponding time array of the section
     of LYRA data that was converted using the wavelet transform.
  ;
  ; OUTPUTS:
  ;       coneOfInfluence = A pointer to a 2 dimensional boolean array of
    identical size to the wavelet transform indicating the regions of the Cone
    of Influence.
  ;
  ; NOTES:
  ;       Requires Morlet mother-wavelet to be used in analysis.
  ;
  ; MODIFICATION HISTORY:
  ;       Initial Version D. Vagg, February 2011
  ;-

  coneOfInfluence = (*waveletPtr)
  coneOfInfluence[*] = 0
  time = *time

  for i = 0, n_elements(scales)-1 do begin
    singleWave = coneOfInfluence[*,i]
    invalidLength = sqrt(2) * scales[i]
    if invalidLength gt n_elements(singleWave) then invalidLength = n_elements(
       singleWave)

    singleWave[0:invalidLength-1] = 1
    singleWave[n_elements(singleWave)-(invalidLength):n_elements(singleWave)-1]
       = 1
```

```
    coneOfInfluence[*,i] = singleWave
  end

  return, ptr_new(coneOfInfluence)
END

FUNCTION getWaveletPower, waveletPtr, coi, scales
  ;+f* p2lc_getwavelet/getWaveletPower
  ; FUNCTION:
  ;       getWaveletPower
  ;
  ; PURPOSE:
  ;       Sums the power of the wavelet transform at each scale along the region
      outside of the Cone of Influence.
  ;
  ; CALLING SEQUENCE:
  ;       power = getWaveletPower( waveletPtr, scales, coi )
  ;
  ; INPUTS:
  ;       waveletPtr =  A pointer to the wavelet transform.
  ;       coi        =  A pointer to the Cone of Influence for the wavelet
     transform.
  ;       scales     =  An array containing the scales used in the wavelet
     transform.
  ;
  ; OUTPUTS:
  ;       power = A pointer to a single dimensional double array the same length
      as the scales array containing the total power of each scale outside of
     the Cone of Influence.
  ;
  ; MODIFICATION HISTORY:
  ;       Initial Version D. Vagg, February 2011
  ;-

  power = dblarr(n_elements(scales))
  invCOI = abs(*coi - 1) ; converts COI from 1 in COI and 0 in good region, to 0
      in COI and 1 in good region. Convenient for mult operation
  for i = 0, n_elements(scales) - 1 do begin
    power[i] = total((abs(*waveletPtr) * invCOI)[*,i])
  end
  return, ptr_new(power)
END

FUNCTION getWaveSignif, dataWindow, scales, waveletParameters
  ;+f* p2lc_getwavelet/getWaveSignif
  ; FUNCTION:
  ;       getWaveSignif
  ;
```

```
; PURPOSE:
;       Generate an array of identical dimensions to the wavelet transform
    containing boolean values indicating if a point in the array
;       is affected by the Cone of Influence (1) or not (0).
;       (Reference: C. Torrence and G.P. Compo et al, A Practical Guide to
    Wavelet Analysis).
;
; CALLING SEQUENCE:
;       Result = getWaveSignif( dataWindow, scales, waveletParameters )
;
; INPUTS:
;       dataWindow        = A structure containing the section of data being
    analysed.
;       scales            = An array containing the scales used in the
    wavelet transform.
;       waveletParameters = A pointer to a structure of parameters containing
     the family, and order of the mother-wavelet
;       along with the significance level clamped between 0 and 1 (i.e. 0.99 =
     99% significance).
;
; OUTPUTS:
;       signif = An array of doubles indicating the regions of significance in
     the wavelet transform.
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-
data = *(dataWindow.data_relInt)
time = *(dataWindow.time)

numSec = (time[n_elements(time)-1] - time[0])*24*60*60 ; CAREFUL! If you put
    24*60*60 in brackets; i.e. (24*60*60), it will cause issues..
numPointsPerSec = ceil(n_elements(time)/numSec)
dt = 1/numPointsPerSec

signif = wave_signif(*(dataWindow.data_relInt),          $
                     dt, scales,                         $
                     mother=waveletParameters.wv_family, $
                     param=waveletParameters.wv_order,   $
                     siglvl=waveletParameters.wv_signif)
return, signif
END

FUNCTION p2lc_getwavelet, waveletParameters, dataWindow
        ;+f* LYRASCRIPT/p2lc_getwavelet
        ; FUNCTION:
        ;       p2lc_getwavelet
        ;
```

```
        ; PURPOSE:
        ;       Generate the wavelet transform of data and return a structure
            containing the results.
        ;
        ; CALLING SEQUENCE:
        ;       result = p2lc_getwavelet(waveletParameters, dataParameters, data
            , timeWindow)
        ;
        ; INPUTS:
        ;                   waveletParameters = A structure containing
            parameters to be used in wavelet analysis (e.g. wavelet family and
            order).
        ;       dataWindow                          = A structure containing the
            section of LYRA data to analyse along with the corresponding time
            data.
        ;
        ; OUTPUTS:
        ;       waveletResults =        A structure containing the wavelet
            transform of input data, the scales used in the transform, and the
            time range that was analysed.
        ;       waveletResults.w_scales = Scales used in the wavelet transform.
        ;       waveletResults.w_periods = Periods corresponding to scales used
            in wavelet transform.
        ;       waveletResults.w_coi     = Cone of influence corresponding to
            the wavelet transform.
        ;       waveletResults.w_power   = Total power of the absolute of the
            wavelet transform outside of the COI.
        ;       waveletResults.w_signif  = Regions above a configurable
            significance level.
        ;       waveletResults.time      = The corresponding time data for the
            wavelet transform.
        ;
        ; MODIFICATION HISTORY:
        ;       Initial Version D. Vagg, February 2011
        ;-
        scaleParameters  = getScaling(waveletParameters, dataWindow)       ;
            Determine scaling parameters

        waveletTransform = doWavelet(waveletParameters, scaleParameters, $  ;
            Perform wavelet transform
                                        dataWindow, scales)

waveletPeriods   = scaleToPeriods(scales, dataWindow.time, waveletParameters.
    wv_family, waveletParameters.wv_order)

coneOfInfluence  = getCOI(waveletTransform, scales, dataWindow.time)

waveletPower     = getWaveletPower(waveletTransform, coneOfInfluence, scales)
```

```
    waveletSignif     = getWaveSignif(dataWindow, scales, waveletParameters)

    waveletTime       = *(dataWindow.time)
    ; Create a structure of wavelet information to be returned
          waveletResults    = CREATE_STRUCT(                    $
                              'w_transform', waveletTransform,  $   ; Wavelet
                                 transform
                              'w_scales', scales,               $   ; Scales used
                                 in wavelet transform
                              'w_periods', waveletPeriods,      $   ; Scales
                                 converted to periods
                              'w_coi', coneOfInfluence,         $   ; Cone of
                                 influence in result
                              'w_power', waveletPower,          $   ; Total power
                                 of absolute transform at each scale
                              'w_signif', waveletSignif,        $
                              'time', ptr_new(waveletTime)      $
                              )

          return, waveletResults
END
```

A.2.8 p2lc_getwaveletanalysis

```
FUNCTION p2lc_getwaveletanalysis, waveletSignal, dataWindow, waveletParameters
          ;+f* LYRASCRIPT/p2lc_getwaveletanalysis
          ; FUNCTION:
          ;      p2lc_getwaveletanalysis
          ;
          ; PURPOSE:
          ;      To perform analysis on the wavelet transform to determine if
             frequencies were present.
          ;
          ; CALLING SEQUENCE:
          ;      result     =        p2lc_getwaveletanalysis( waveletSignal,
             dataWindow, waveletParameters )
          ;
          ; INPUTS:
          ;      waveletSignal       = A pointer to the results of 'p2lc_getwavelet'
          ;      dataWindow          = A structure returned from the 'getDataWindow'
              function containing both a section of data and corresponding time
             data.
          ;                  waveletParameters  = A structure of parameters that
             contain fft windowing parameters.
          ;
          ; OUTPUTS:
```

```
;       waveletStats                        = A structure containing several
    fields for detected frequencies.
;       waveletStats.signal_start           = The value of the time array
    where the signal began.
;       waveletStats.signal_end             = The value of the time array
    where the signal ends.
;       waveletStats.signal_duration        = The duration of the signal in
    seconds.
;       waveletStats.detectedFrequencies = A boolean flag set to 1 if the
    duration of frequency components (if detected) is above the threshold
     specified in detection parameters.
;       waveletStats.signal_max             = The maximum value of the
    frequency component.
;
; SEE ALSO:
;       p2lc_getwavelet
;       getDataWindow
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-
waveletStats =          CREATE_STRUCT( 'signal_start', 0.d,$
                                       'signal_end',   0.d,  $
                                       'signal_duration', 0.d, $
                                       'detectedFrequencies', 0, $
                                       'signal_max',   0.d)

periodicTRange   = waveletParameters.wv_signal_t
waveletTransform  = abs(*(waveletSignal.w_transform))
waveletCOI        = *(waveletSignal.w_COI)
waveletPeriods    = waveletSignal.w_periods
signalThreshold   = waveletParameters.signal_threshold

time = *(dataWindow.time)
data = *(dataWindow.data)

startTime = time[0]
endTime   = time[ n_elements(time)-1 ]

scaleRange = where( (waveletPeriods lt max(periodicTRange)) and (
    waveletPeriods gt min(periodicTRange)), count)
largestScaleIndexInRange   = max(scaleRange)
smallestScaleIndexInRange  = min(scaleRange)

if (count eq 0) then return, waveletStats

; Find the collective maximum of all scales within the region specified by the
    user
```

```
  transformSlice   = waveletTransform[*, scaleRange]          ; Get slice
  inverseCOI       = abs(waveletCOI - 1)                      ; Invert Cone of
     influence to allow multiplication of both arrays, resulting in area OUTSIDE
      of COI, instead of INSIDE
  invCOISlice         = inverseCOI[*,scaleRange]
  transformCOISlice = transformSlice * invCOISlice            ; Multiply
     slice of wavelet by slice of inverted COI
  transformCOISliceMax = max(transformCOISlice, dimension=2)  ; Get maximum of
     scales within slice

  ;Find the region of the [maximum values of the transform within the scales set
      by user * COI] above threshold
  signalAboveThreshold = where(transformCOISliceMax gt signalThreshold, count)
  if count eq 0 then return, waveletStats

  ;Determine length of COI at maximum periodicity that is being detected.
  longestCOI = max(waveletCOI[*,scaleRange], dimension=2)
  transformCOISlicePoints = where(longestCOI eq 1, count)
  ratioOfCOI = double(count)/n_elements(transformCOISliceMax)
  durationOfWindowJD = (endTime - startTime)*24.d*60.d*60.d
  durationOfCOI = durationOfWindowJD*ratioOfCOI

  ;Populate structure with statistics
  waveletStats.signal_start      = min(time[signalAboveThreshold])
  waveletStats.signal_end        = max(time[signalAboveThreshold])
  waveletStats.signal_duration   = (waveletStats.signal_end - waveletStats.
     signal_start)*24*60*60
  waveletStats.signal_max        = max(transformCOISliceMax(signalAboveThreshold)
     )
  waveletStats.detectedFrequencies = (waveletStats.signal_duration gt
     durationOfCOI)

  return, ptr_new(waveletStats)
END
```

A.2.9 p2lc_getwindowreport

```
FUNCTION p2lc_getwindowreport, timeWindow, dataStat, flareStats, waveletStatPtr,
    plotNames, solarSoftList
  ;+f* LYRASCRIPT/p2lc_getwindowreport
  ; FUNCTION:
  ;       p2lc_getwindowreport
  ;
  ; PURPOSE:
  ;       Generate a standard set of analysis from several sets of data.
  ;
  ; EXPLANATION:
```

```
;       During analysis there are several different sets of analysis and
    statistics generated. These can be difficult to keep track of during
    implementation.
;       This function serves to create a uniform set of analysis for each
    window of data, which simplifies later functions which summarise results of
     entire days of analysis.
;
; INPUTS:
;       timeWindow      = The timeWindow structure used during analysis of a
    window.
;       dataStat        = Pre-analysis structure of statistics generated
    before analysis.
;       flareStats      = Statistics used for flare detection including a
    boolean flag indicating if a flare occurred.
;       waveletStatPtr  = Statistics generated from analysis of the wavelet
    transform of the window of data.
;       plotNames       = The names of any plots generated during analysis of
    a window.
;       solarSoftList   = Structure of solar events as recorded in SolarSoft
    archives.
;
; OUTPUTS:
;       analysis                    = A structure containing summarised
    statistics from all results of analysis.
;       analysis.CORRELATIONDATA    = A structure of data to be used in
    generating a correlation table.
;       analysis.WindowStat         = A structure of data similar to
    CORRELATIONDATA but contains event numbers of SolarSoft events which are
    used when generating event correlation tables.
;       analysis.timeStart          = The start time of the window of data.
;       analysis.timeEnd            = The end time of the window of data.
;
; NOTES:
;       This function is designed for specific use with 'p2lc_getdailystats'.
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-

  solarSoftFlag      = flareStats.SS_flare
  startSolarSoftEvent = flareStats.SS_startEvent
  endSolarSoftEvent = flareStats.SS_endEvent
  solarSoftEventDetected = flareStats.PR_matched_SS_event
  lagDetectionFlag  = flareStats.PR_flare
  freqDetect = 0

  if ptr_valid(waveletStatPtr) then begin
    waveletStats = (*waveletStatPtr)
```

```
    freqDetect     = waveletStats.detectedFrequencies
  end

  timeStart  = jd2utc(timeWindow.timestart)
  timeEnd    = jd2utc(timeWindow.timeend)

  CORRELATION_WINDOWDATA = CREATE_STRUCT('SSW',    solarSoftFlag gt 0,  $
                                         'SSW_n',  solarSoftFlag eq 0,  $
                                         'LDW',    lagDetectionFlag gt 0,  $
                                         'LDW_n',  lagDetectionFlag eq 0,  $
                                         'FDW', freqDetect eq 1,     $
                                         'FDW_n', freqDetect eq 0)

  CORRELATION_EVENTDATA = CREATE_STRUCT('SSE', solarSoftFlag,     $
                                        'LDE', lagDetectionFlag gt 0, $
                                        'FDE', freqDetect gt 0)

  analysis = CREATE_STRUCT('CORRELATION_WINDOWDATA', CORRELATION_WINDOWDATA, $
                           'CORRELATION_EVENTDATA', CORRELATION_EVENTDATA, $
                           'plotNames', plotNames, $
                           'timeStart', timeStart, $
                           'timeEnd', timeEnd)

  return, ptr_new(analysis)
END
```

A.2.10 p2lc_getwindows

```
FUNCTION loadInfoFile, filename
  ;+f* p2lc_getwindows/loadInfoFile
  ; FUNCTION:
  ;       loadInfoFile
  ;
  ; PURPOSE:
  ;       Loads a 'commanding' log file for the PROBA-2 satellite and creates an
     array representing each line of the file.
  ; s
  ; CALLING SEQUENCE:
  ;       result = loadInfoFile(filename)
  ;
  ; INPUTS:
  ;       filename  = The full path of the log file
  ;
  ; OUTPUTS:
  ;       infoArr = A pointer to an array containing each line of a command log
    file. One line per entry. Will return -1 if the file does not exist.
  ;
  ; NOTES:
```

```
;       The command log files for the PROBA-2 satellite do not contain all
    instructions for the date represented in the file name.
;       For this reason, it is recommended that commanding logs are loaded
    from at least one day prior to the date intended to be analysed.
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-

; This loads an info file for parsing
; Test if file exists
if file_test(filename) eq 1 then begin
  ; Create a string array based on the number of lines in the file
  fileLines = file_lines(filename)
  infoArr = strarr(fileLines)
  ; open the file and store data in the string array
  openr, 2, filename
  readf, 2, infoArr
  close, 2
  ; Return this array
  return, ptr_new(infoArr)
end else begin
  ; Print error message
  error, 'info_file_missing', filename
  return, -1
end
END

FUNCTION getLargeAngleRotationTimes, infoArr, month, day, year
;+f* p2lc_getwindows/getLargeAngleRotationTimes
; FUNCTION:
;       getLargeAngleRotationTimes
;
; PURPOSE:
;       Parses an array of strings from a LYRA commanding log to determine the
     times of Large Angle Rotations, indicated with the 'CHANGE ATTITUDE'
    string.
;
; CALLING SEQUENCE:
;       LARTimes = getLargeAngleRotationTimes(infoFileArray, month, day, year)
;
; INPUTS:
;       infoArr = A string array generated from a command log file for the
   PROBA-2 satellite.
;       month   = The month component of the date for which Large Angle
    Rotations are to be listed.
;       day     = The day component of the date for which Large Angle
    Rotations are to be listed.
```

```
;       year    = The year component of the date for which Large Angle
    Rotations are to be listed.
;
; OUTPUTS:
;       LARTimes  = An array containing times in JulianDay format representing
     the times of each Large Angle Rotation.
;       Will return -1 if there was a parsing error.
;
; SEE ALSO:
;       julday
;       loadInfoFile
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-

; PARSES OVER INFO FILE AND LOCATES LARGE ANGLE ROTATIONS

; Determine the strings to search for. (Change attitude, and the correct date)
LarCommandStr = 'CHANGE_ATTITUDE'
dateStr       = string(year, FORMAT='(I4)') + '/' + string(month, FORMAT='(I2)
    ') + '/' + string(day, FORMAT='(I2)')

; Get lines from info file  where LARs occur by checking for the string '
    CHANGE ATTITUDE'
ChangeIndices = stregex(*infoArr, LarCommandStr)
LARs = (*infoArr)[where(ChangeIndices ne -1)]

; Select LAR lines that correspond to the date passed to this method
; If lines cannot be found containing the correct dates, then print an error
    message
DateIndices = stregex(LARs, dateStr)
correctDates = WHERE(DateIndices ne -1, lineCount)
if (lineCount gt 0) then begin
  LARs = LARs[correctDates]
end else begin
  error, 'parsing_error', dateStr
  return, -1
end

; Extract the time information from each line
LARs = stregex(LARs, '.{2}:.{2}:.{2}', /EXTRACT)
; Create a float array for storing the jd values to high accuracy
LarTimes = dblarr(n_elements(LARs))

; Parse each time into a Julian Day
for i = 0, n_elements(LARs)-1 do begin
  LarTime = strsplit(LARs[i], ':', /extract)
```

```
    LarTimes[i] = julday(month, day, year, LarTime[0], LarTime[1], LarTime[2])
  end

  return, LarTimes
END

FUNCTION createWindows, LarTimes, data, timeStart, timeEnd
  ;+f* p2lc_getwindows/createWindows
  ; FUNCTION:
  ;       createWindows
  ;
  ; PURPOSE:
  ;       Returns an array of structures, containing parameters used to specify
    sections of LYRA data outside of Large Angle Rotations.
  ;
  ; CALLING SEQUENCE:
  ;       validWindows = createWindows( LarTimes, data, timeStart, timeEnd )
  ;
  ; INPUTS:
  ;   LarTimes  = An array of times in JulianDay format representing each Large
    Angle Rotation of the PROBA-2 satellite.
  ;   data      = A structure containing a pointer to the TIME channel of LYRA
    data for the date to be analysed.
  ;   timeStart = The beginning of the time region to generate 'windows' for.
    Represented in JulianDay format.
  ;   timeEnd   = The end of the time region to generate 'windows' for.
    Represented in JulianDay format.
  ;
  ; OUTPUTS:
  ;   validWindows            = An array of pointers to structures containing 4
    elements to represent the start and end indexes and times of a window using
     the following fields:(timeStart, timeEnd, indexStart, indexEnd).
  ;   validWindows.timeStart  = The start of the window in JulainDay format.
  ;   validWindows.timeEnd    = The end of the window in JulainDay format.
  ;   validWindows.indexStart = The index of the starting position of the window
     given a full channel of data.
  ;   validWindows.indexEnd   = The index of the final position of the window
    given a full channel of data.
  ;
  ; NOTES:
  ;   The MRDFITS procedure will return FITS data in the correct format for this
     procedure automatically.
  ;
  ; SEE ALSO:
  ;     JULDAY
  ;     LarTimes
  ;     MRDFITS
  ;     loadInfoFile
```

```
;
; MODIFICATION HISTORY:
;       Initial Version D. Vagg, February 2011
;-

timeStart = timeStart
timeEnd = timeEnd - settings('LAR_trimBack')

;Get the indexes of the two time values that will completely enclose the
    specified times
firstIndex = max(where(LarTimes lt timeStart))    ;Find the index of the latest
    LAR that occurs before 'timeStart'
endIndex = max(where(LarTimes lt timeEnd))        ;Find the index of the
    earliest LAR that occurs after 'timeEnd'

if firstIndex lt 0 then firstIndex = 0
;Create an array to store timeWindows (+1 as the indexes of LARs are inclusive
    )
timeWindows = ptrarr((endIndex - firstIndex) + 1)


; Determine windows, and populate an array of simple pair structures
; containing a start time (timeStart) and an end time (timeEnd)
for LarIndex = 0, endIndex-firstIndex-1 do begin
  startTime  = LarTimes[firstIndex+larIndex] + settings('LAR_trimFront')
  endTime    = LarTimes[firstIndex+larIndex+1] - settings('LAR_trimBack')

  startIndex = max(where((*data).time lt startTime))
  endIndex   = max(where((*data).time lt endTime))

  if startIndex gt 0 and endIndex gt 0 then begin
    endIndex -= ((endIndex-startIndex)+1) mod 8  ; subtract up to 7 units of
        data to make the data divisible by 8 (for quick fft separations)
    timeWindows[larIndex] = ptr_new(CREATE_STRUCT('timeStart', startTime, '
        timeEnd', endTime, $
                                                  'indexStart', startIndex, '
                                                      indexEnd', endIndex))
  end
end
; Remove elements of the array where data was non existant
validPointers = where( ptr_valid(timeWindows) eq 1, countValid)
if countValid eq 0 then $
 return, -1

validWindows  = timeWindows[validPointers]
;Return a pointer array containing pointers to each 'time window' structure
return, validWindows
END
```

```
FUNCTION p2lc_getwindows, date, data, timeStart, timeEnd
  ;+f* LYRASCRIPT/p2lc_getwindows
  ; FUNCTION:
  ;       p2lc_getwindows
  ;
  ; PURPOSE:
  ;       Generate a series of structures specifying time and index values to
    avoid Large Angle Rotations by parsing a 'commanding' log file for the
    PROBA-2 satellite.
  ;
  ; CALLING SEQUENCE:
  ;       result = p2lc_getwindows(date, data, timeStart, timeEnd)
  ;
  ; INPUTS:
  ;       date        = The date to generate timeWindows for.
  ;       data        = A structure containing LYRA data channels.
  ;       timeStart   = The start time in JulianDays.
  ;       timEnd      = The end time in JulianDays.
  ;
  ; OUTPUTS:
  ;       dataWindows   = An array of structures containing start and end times/
    indexes for each window.
  ;
  ; MODIFICATION HISTORY:
  ;       Initial Version D. Vagg, February 2011
  ;-

  CALDAT, date, mm, dd, yyyy          ; Get month, day, and year values from the
      date passed in

  fileDir = settings('info_dir') + $  ; Generate the file path for the info file
    string(yyyy,format='(I04)') + '/' + string(mm,format='(I02)') + '/' + string
      (dd,format='(I02)') + '/'

  fname = fileDir+'*Info*txt'

  infoData = loadInfoFile(fname)      ; Load the info file for parsing
  if (ptr_valid(infoData) eq 0) then $
    return, 0

  ; Get each time that a Large Angle Rotation occurs from a log file.
  LarTimes =  getLargeAngleRotationTimes(infoData, mm, dd, yyyy)

  ; Get an array of start and end indexes for each window between large angle
    rotations
  dataWindows = createWindows(LarTimes, data, timeStart, timeEnd)
```

```
  return, dataWindows
END
```

A.2.11 p2lc_plotdata

```
PRO setPlotParams, type
  ;+p* p2lc_plotdata/setPlotParams
  ; PROCEDURE:
  ;   setPlotParams
  ;
  ; PURPOSE:
  ;   Set window parameters to be used in plotting data based on the 'type'
      parameter.
  ;
  ; CALLING SEQUENCE:
  ;   setPlotParams, type
  ;
  ; INPUTS:
  ;   type  = A string specifying the type of plot to set up.
  ;
  ; MODIFICATION HISTORY:
  ;   Initial Version D. Vagg, February 2011
  ;-

  !x.tickformat=''
  !x.tickunits=''
  !x.tickinterval=0
  !x.ticklayout=0

  if type eq "time" then begin
    dateLabel = LABEL_DATE(DATE_FORMAT = $
              ['%I:%S', '%H', '%D_%M_%Y'])
    !x.tickformat = $
              ['LABEL_DATE','LABEL_DATE','LABEL_DATE']
    !x.tickunits  = $
              ['Time',        'Hour',       'Day']
    !x.tickinterval = 5
    !x.ticklayout   = 0
    !p.background   = color24([255,255,255])
    !p.multi        = 0
  end

  if type eq "contour" then begin
    dateLabel = LABEL_DATE(DATE_FORMAT = $
              ['%I:%S', '%H', '%D_%M_%Y'])
    !x.tickformat = $
              ['LABEL_DATE','LABEL_DATE','LABEL_DATE']
    !x.tickunits  = $
```

```
                ['Time',        'Hour',      'Day']
    !x.tickinterval = 5
    !x.ticklayout   = 0
    !p.background   = color24([255,255,255])
    !p.multi        = 0
  end

  if type eq "power_response" then begin
    dateLabel = ''
    !x.tickname = ''
    !p.background   = color24([255,255,255])
    !p.multi = 0
  end

END


PRO timePlot, parameters, data, time
  ;+p* p2lc_plotdata/timePlot
  ; PROCEDURE:
  ;    timePlot
  ;
  ; PURPOSE:
  ;    Plot the time-series LYRA data using parameters specified externally.
  ;
  ; CALLING SEQUENCE:
  ;    timePlot, parameters, data, time
  ;
  ; INPUTS:
  ;    parameters  = Parameters used to specify the plot details.
  ;    data        = An array containing the data to be plotted.
  ;    time        = An array containing time values corresponding to data.
  ;
  ; MODIFICATION HISTORY:
  ;    Initial Version D. Vagg, February 2011
  ;-

  ; set default plot settings
  setPlotParams, "time"

  ; Determine range of chosen channels in both x and y axis
  range = [0,max(*data)]
  if (parameters.ynozero eq 1) then range[0] = min(*data)

  ; Create an empty plot
  plot, (*time), [150,200], /nodata,  $
        background =color24([255,255,255]), $
        color      =color24([0,0,0]), $
```

```
        yrange      = range, $
        xstyle=1,   $
        POSITION = [0.05, 0.15, 0.95, 0.9]

    ; Plot the time-series data
    oplot, *time, *data, color=parameters.color
END


PRO powerResponsePlot, parameters, data
    ;+p* p2lc_plotdata/powerResponsePlot
    ; PROCEDURE:
    ;    powerResponsePlot
    ;
    ; PURPOSE:
    ;    Generate a 2-dimensional plot of the addition of Aluminium and Zirconium
        channels of the LYRA instrument against the division of the respective
        channels.
    ;
    ; CALLING SEQUENCE:
    ;    result = powerResponsePlot, parameters, data
    ;
    ; INPUTS:
    ;    parameters     = Parameters used to specify the power-response plot details
        .
    ;    data           = A structure containing two arrays; one containing the
        division of the LYRA Aluminium data by the Zirconium data,
    ;                     and the other array containing the addition of the two
        sets of data.
    ;
    ; MODIFICATION HISTORY:
    ;    Initial Version D. Vagg, February 2011
    ;-

    ; set default plot settings
    setPlotParams, "power_response"

    ; Create an empty plot
    plot, *(data.AZdiv), *(data.AZadd), $
          background = color24([255,255,255]), $
          color      = parameters.color, $
          title      = 'Power Response', $
          xstyle=1,  $
          /ynozero,  $
          POSITION = [0.05, 0.15, 0.95, 0.9]
END

FUNCTION p2lc_plotdata, parameters, data, dataWindow
```

```
;+f* LYRASCRIPT/p2lc_plotdata
; FUNCTION:
;    p2lc_plotdata
;
; PURPOSE:
;    Plot various types of data using standard plots, and contour plots.
;
; CALLING SEQUENCE:
;    result = p2lc_plotdata (parameters, data, time)
;
; INPUTS:
;    parameters     = Parameters used to specify the plot details.
;    data           = The data to be plotted.
;    dataWindow     = A window of data containing corresponding time data for
    the data being plotted.
;
; OUTPUTS:
;    plotFileName  = The file name of the plot (if saved).
;
; MODIFICATION HISTORY:
;    Initial Version D. Vagg, February 2011
;-

time = dataWindow.time
; Determine the type of plot using the 'plot_type' field in the data structure
    .
if (parameters.plot_type eq "time") then $
 timePlot, parameters, data, time

if (parameters.plot_type eq "power_response") then $
 powerResponsePlot, parameters, data

if (parameters.save_plot) then $
 plotFileName = p2lc_savePlot(parameters, time) $
else $
 plotFileName = ''

return, plotFileName
END
```

A.2.12 p2lc_plotwavelet

```
PRO waveletPlot, parameters, waveletStructure, time, data
;+p* p2lc_plotwavelet/contourPlot
; PROCEDURE:
;    waveletPlot
;
; PURPOSE:
```

```
;    Plots the wavelet transform of the section of LYRA data, as well as the
    contour legend, data used to generate the transform, and the power of each
    scale.
;
; CALLING SEQUENCE:
;    waveletPlot, parameters, waveletStructure, time, data
;
; INPUTS:
;    parameters        = Parameters used to specify the wavelet plot details.
;    waveletStructure  = The wavelet structure returned by '
    p2lc_getwaveletanalysis'.
;    time              = The time corresponding to the wavelet transform.
;    data              = The LYRA data corresponding to the wavelet transform.
;
; MODIFICATION HISTORY:
;    Initial Version D. Vagg, February 2011
;-
dateLabel = LABEL_DATE(DATE_FORMAT = $
          ['%I:%S', '%H', '%D_%M_%Y'])
!x.tickformat = $
          ['LABEL_DATE','LABEL_DATE','LABEL_DATE']
!x.tickunits  = $
          ['Time',        'Hour',      'Day']
!x.tickinterval = 5
!x.ticklayout   = 0
!p.background   = color24([255,255,255])
!p.multi        = 0

range  = parameters.range     ; The depth of the scalogram to view
levelN = parameters.levels    ; The number of levels to render in the contour
    plot

n      = range[1]-range[0]       ; Create a range of levels based on the depths
     and number of levels to view
levels = ((indgen(levelN,/double)/(levelN))*n) + range[0]

; Plot the waelet result
wavelet = *(waveletStructure.w_transform)
coi     = *(waveletStructure.w_coi)
periods = waveletStructure.w_periods
signif  = waveletStructure.w_signif
power   = *(waveletStructure.w_power)
power   = power[where(finite(power))]

; PLOT REGION FOR CONTOUR
contour, abs(wavelet), *time, periods, /nodata, $
  xstyle=1,ystyle=1,          $
  title='Wavelet Power',      $
```

```
  ytitle='Periodic Time',     $
  yticks=20,                  $
  position=[.2,.2,.85,.75], $
  LEVELS=levels, /ylog,       $
  /fill,background=color24([255,255,255]),      $
  color=0

device, decomposed=0            ; Configure the screen for colour plot
loadct, parameters.ct          ; Load a color table for the contour plot

bkgnd = 255
fgnd = 0
if (parameters.ct_reverse eq 1) then begin
  bkgnd = 0
  fgnd = 255
  tvlct, r,g,b,/get
  gg = reverse(g)
  rr = reverse(r)
  bb = reverse(b)
  tvlct, rr,gg,bb
end

; PLOT WAVELET
contour, abs(wavelet), *time, periods, $
  xstyle=21,ystyle=21,          $
  yticks=20, /noerase,        $
  position=[.2,.2,.85,.75], $
  LEVELS=levels, /ylog,       $
  /fill, color=fgnd

; PLOT LEGEND
bar = levels # REPLICATE(1B, n_elements(levels))
bar = rotate(bar, 1)
contour, bar, $
  xstyle=21,ystyle=21,         $
  xtickformat='',$
  title='Range',      $
  position=[.90,.2,.95,.75], $
  LEVELS=levels, /noerase,  $
  /fill,background=255, color=fgnd
axis, 0.9, YAXIS=0, YRANGE = range, YSTYLE = 1, $
 YTITLE = 'Contour Range', fgnd
device, decomposed=1

; PLOT SIGNIFICANT LEVELS
signif = rebin(transpose(signif),n_elements(*time),n_elements(periods))
contour, abs(wavelet)^2/signif, *time, periods, $
  xstyle=21,ystyle=21,          $
```

```
    position=[.2,.2,.85,.75], $
    color=parameters.signif_color, $
    level=1.0, /noerase, /ylog

  ; PLOT HATCHING
  contour, coi,levels=[1,2], $
  xstyle=21,ystyle=21,          $
  /noerase, c_orientation=[45], $
  color=parameters.coi_color,     $
  position=[.2,.2,.85,.75]
  contour, coi,levels=[1,2], $
  xstyle=21,ystyle=21,          $
  /noerase, c_orientation=[-45], $
  color=parameters.coi_color,     $
  position=[.2,.2,.85,.75]


  ; PLOT ORIGINAL DATA
  plot, *time, *data,       $
  ystyle=1, xstyle=21,      $
  color=color24([0,0,0]),   $
  /ynozero, /noerase,       $
  position=[.2,.80,.85,.95],$
  title='Relative Intensity'

  !x.tickformat=''
  !x.tickunits=''
  !x.tickinterval=0
  !x.ticklayout=0

  ; PLOT WAVELET POWER
  plot, power, periods,             $
  ystyle=1, xstyle=1,               $
  color=color24([0,0,0]),           $
  xrange=[max(power), min(power)+1],$
  position=[.05,.2,.15,.75],        $
  xticks = 3,                       $
  /ylog, /xlog, ytitle='Log(Scale Power)', $
  /noerase
END

FUNCTION p2lc_plotwavelet, parameters, waveletStructure, dataWindow
  ;+f* LYRASCRIPT/p2lc_plotwavelet
  ; FUNCTION:
  ;    p2lc_plotwavelet
  ;
  ; PURPOSE:
  ;    Plot the wavelet transform of LYRA data.
```

```
  ;
  ; CALLING SEQUENCE:
  ;    result = p2lc_plotdata( parameters, waveletStructure, dataWindow )
  ;
  ; INPUTS:
  ;    parameters        = Parameters used to specify the wavelet plot details.
  ;    waveletStructure  = The wavelet structure returned by '
      p2lc_getwaveletanalysis '.
  ;    dataWindow        = A structure containing the corresponding LYRA data for
       the wavelet transform.
  ;
  ; OUTPUTS:
  ;    plotFileName  = The file name of the wavelet plot (if saved).
  ;
  ; MODIFICATION HISTORY:
  ;    Initial Version D. Vagg, February 2011
  ;-

  time = dataWindow.time
  data = dataWindow.data_relInt
  waveletPlot, parameters, waveletStructure, time, data
  if (parameters.save_plot) then $
   plotFileName = p2lc_savePlot(parameters, time)

  return, plotFileName
END
```

A.2.13 p2lc_saveplot

```
FUNCTION p2lc_savePlot, parameters, time
  ;+f* p2lc_plotdata/savePlot
  ; FUNCTION:
  ;    savePlot
  ;
  ; PURPOSE:
  ;    Save the plot to a file.
  ;
  ; CALLING SEQUENCE:
  ;    p2lc_plotdata, parameters, variableDump, time
  ;
  ; INPUTS:
  ;    parameters     = Parameters used to specify the plot details.
  ;    time           = A single dimension array specifying the time of each point
       of data.
  ;
  ; MODIFICATION HISTORY:
  ;    Initial Version D. Vagg, February 2011
  ;-
```

```
  timeStart = (*time)[0]
  timeEnd   = (*time)[n_elements(*time) - 1]

  caldat, timeStart , mm1, dd1, yyyy1, h1, m1, s1
  caldat, timeEnd   , mm2, dd2, yyyy2, h2, m2, s2

  timeIdentifier =  'S' + string(h1,format='(I02)') + '-' + string(m1,format='(
     I02)') + '-' + string(s1,format='(I02)') + '_' + $
                    'E' + string(h2,format='(I02)') + '-' + string(m2,format='(
                       I02)') + '-' + string(s2,format='(I02)')

  dateDir = settings('output_dir')  + $
            string(yyyy1,format='(I04)') + '/' + $
            string(mm1,format='(I02)')   + '/' + $
            string(dd1,format='(I02)')   + '/'

  imageDir = dateDir + timeIdentifier + '/'

  if file_test(imageDir) eq 0 then file_mkdir, imageDir

  fname = imageDir + timeIdentifier + '_T' + parameters.plot_type + '.png'

  write_png, fname, tvrd(true=1), _Extra=extra

  return, timeIdentifier + '/' + timeIdentifier + '_T' + parameters.plot_type +
     '.png'
END
```

A.2.14 p2lc_writereport

```
FUNCTION getFileName, analysisParameters, dataDate
        ;+f* p2lc_writereport/getFileName
        ; FUNCTION:
        ;       getFileName
        ;
        ; PURPOSE:
        ;       Determine an appropriate file name, including path, based on a
           structure of parameters.
        ;
        ; CALLING SEQUENCE:
        ;       Result = getFileName( analysisParameters, dataDate )
        ;
        ; INPUTS:
        ;       analysisParameters =  A set of parameters containing a date
           specifying the date that the report should be analysed.
        ;  dataDate           = The date of data used in analysis.
        ;
```

```
        ; OUTPUTS:
        ;       fname =       filename including full path to be used to wite
            the report document.
        ;
        ; MODIFICATION HISTORY:
        ;       Initial Version D. Vagg, February 2011
        ;-

  CALDAT, dataDate, mm, dd, yyyy

  fileDir = settings('output_dir') + $
  string(yyyy,format='(I04)') + '/' + string(mm,format='(I02)') + '/' + string(
      dd,format='(I02)') + '/'
  fname = fileDir + 'dataReport.html'

  return, fname
END

FUNCTION loadHTMLTemplate, filename
  ;+f* p2lc_writereport/loadHTMLTemplate
  ; FUNCTION:
  ;  loadHTMLTemplate
  ;
  ; PURPOSE:
  ;  Load a template HTML document to base the report on.
  ;
  ; CALLING SEQUENCE:
  ;  Result = loadHTMLTemplate( filename )
  ;
  ; INPUTS:
  ;   filename = The full filename of the template document to load.
  ;
  ; OUTPUTS:
  ;   lineArr = A string array containing each line of the template document.
  ;
  ; MODIFICATION HISTORY:
  ;  Initial Version D. Vagg, February 2011
  ;-
  ; Test if file exists
  if file_test(filename) eq 1 then begin
    ; Create a string array based on the number of lines in the file
    fileLines = file_lines(filename)
    lineArr = strarr(fileLines)
    ; open the file and store data in the string array
    openr, 2, filename
    readf, 2, lineArr
    close, 2
    ; Return this array
```

```
    return, ptr_new(lineArr)
  end else begin
    ; Print error message
    error, 'info_file_missing', filename
    return, -1
  end
END

PRO writeHTMLReport, lun, identifier, name, property, level, variableType
  ;+p* p2lc_writereport/writeHTMLReport
  ; PROCEDURE:
  ;   writeHTMLReport
  ;
  ; PURPOSE:
  ;   A recursive procedure to write nested structures of information to a HTML
      report.
  ;
  ; CALLING SEQUENCE:
  ;   writeHTMLReport, lun, identifier, name, property, level, variableType
  ;
  ; INPUTS:
  ;   lun          = A unique identifier for the output file
  ;   name         = The name of the property
  ;   property     = A property to write to file, there is no enforced type.
  ;   level        = Indicates the current level within the nested structure.
  ;   variableType = A string recording the type of entry currently being
      iterated through, such as arrays or structures.
  ;
  ; NOTES:
  ;   'level' is currently unused, and is kept for future improvements to the
      implementation.
  ;
  ;   Due to the nature of generating a HTML report, there are some hardcoded
      strings used for HTML generation within the code.
  ;   If these strings are changed and an error exists in the report, the report
      will not be viewable in a browser.
  ;
  ; MODIFICATION HISTORY:
  ;   Initial Version D. Vagg, February 2011
  ;-

  propertyType = size(property, /type)
  if propertyType eq 0 then begin
    printf, lun, '<p>NULL</p>'
    return
  end

  propertyCount = n_elements(property)
```

```
if (propertyCount gt 1) then begin
  printf, lun, '<p><a onclick="switchMenu(''''+identifier+''');" title="Expand
     "><i>'+name+'</i></a></p>'
  printf, lun, '<div id="'+identifier+'" class="collapse" style="display: none
     ;">'
  printf, lun, '<table border="1">'
  cols = (size(property))[1]
  for colInd = 0, cols -1 do begin
    printf, lun, '<tr>'
    rows = (size(property))[2]
    if size(property, /n_dimensions) eq 1 then rows = 1
    for rowInd = 0, rows -1 do begin
      printf, lun, '<td width="50" height="50">'
      writeHTMLReport, lun, identifier + string(colInd, rowInd), name,
         property[colInd, rowInd], level, 'table'
      printf, lun, '</td>'
    end
    printf, lun, '</tr>'
  end
  printf, lun, '</table>'
  printf, lun, '</div>'
  return
end

if (propertyType eq 10) then begin
  if ptr_valid(property) then $
    writeHTMLReport, lun, identifier, name, *(property[0]), level,
       variableType
  return
end

if propertyType eq 8 then begin
  propNames = tag_names(property)
  printf, lun, '<p><a onclick="switchMenu(''''+identifier+''');" title="Expand
     "><i>'+name+'</i></a></p>'
  printf, lun, '<div id="'+identifier+'" class="collapse" style="display: none
     ;">'
  for i = 0, n_tags(property)-1 do begin
    writeHTMLReport, lun, identifier+propNames[i], propNames[i], property.(i),
       level+1, 'struct'
  end
  printf, lun, '</div>'
  return
end

if variableType eq 'struct' then begin
  printf, lun, '<p><b>'+string(name)+': </b>'+string(property)+'</p>'
end else if variableType eq 'table' then begin
```

```
    printf, lun, '<p><center>'+string(property)+'</center></p>'
  end else begin
    printf, lun, '<p><i>'+string(name)+': </i>'+string(property)+'</p>'
  end
END

; ------------------------------------------------------------
; ------------------------------------------------------------
; ------------------------------------------------------------
PRO p2lc_writereport, reportStructure, date
        ;+p* LYRASCRIPT/p2lc_writereport
        ; PROCEDURE:
        ;       p2lc_writereport
        ;
        ; PURPOSE:
        ;       Automatically generate a report containing details of LYRA data
            analysis.
        ;
        ; EXPLANATION:
        ;  This procedure loads a template HTML report.
        ;  It then saves the report line-by-line into a new file.
        ;  Once it reaches a special string while saving the template document,
            it begins to generate the report.
        ;
        ; CALLING SEQUENCE:
        ;       p2lc_writereport, reportStructure, date
        ;
        ; INPUTS:
        ;       reportStructure =     A structure containing the report data.
        ;       date                  =       The date of data used in
            analysis.
        ;
        ;      NOTES:
        ;       The layout and heirarchy of the reportStructure is used to
            generate the report document.
        ;       There are no restrictions on the structure of the
            reportStructure variable; it can contain as many levels as required,
            of any type of data.
        ;
        ; MODIFICATION HISTORY:
        ;       Initial Version D. Vagg, February 2011
        ;-

  reportFileName = getFileName(reportStructure, date)
  Get_LUN, lun
  OpenW, lun, reportFileName, 'html'
  templateHTML_fileName = settings('HTML_template')
  templateHTML_lines    = loadHTMLTemplate(templateHTML_filename)
```

```
  for i = 0, n_elements(*templateHTML_lines)-1 do begin
    html_line = (*templateHTML_lines)[i]
    if html_line eq '<IDL>write_data</IDL>' then begin
      writeHTMLReport, lun, 'DATA', 'Generated_Report', reportStructure, 0, 'h2'
    end else $
      printf, lun, html_line
  end
  Free_LUN, lun
  ptr_free, templateHTML_lines
END
```

A.2.15 settings

```
FUNCTION settings, settingName
        ;+f* LYRASCRIPT/settings
        ; FUNCTION:
        ;       settings
        ;
        ; PURPOSE:
        ;       Return a static setting based on an input string, 'settingName
            ', and is available throughout a program.
        ;
        ; CALLING SEQUENCE:
        ;       setting = settings( settingName )
        ;
        ; INPUTS:
        ;       settingName     =       A string specifying the setting to
            return.
        ;
        ; OUTPUTS:
        ;       A value based on the settingName.
        ;
        ; NOTES:
        ;       The return values are intended to be modified once for initial
            setup of the program.
        ;
        ; MODIFICATION HISTORY:
        ;       Initial Version D. Vagg, February 2011
        ;-
        if (settingName eq 'data_dir') then $
                return, '/LYRA/DATA/'
        if (settingName eq 'info_dir') then $
                return, '/LYRA/DATA/'
        if (settingName eq 'output_dir') then $
                return, '/LYRA/ANALYSIS/'
        if (settingName eq 'LAR_trimBack') then $
                return, 4.d/(24.d * 60.d) ; Represents 4 minutes as a fraction
                    of a day
```

```
        if (settingName eq 'LAR_trimFront') then $
    return, 0.d/(24.d * 60.d)
  if (settingName eq 'HTML_template') then $
    return, '/home/ezeakeal/Dropbox/IDLWorkspace/lyraScript/src/reportTemplate.
        html'


  error, 'setting_missing', settingName
  return, 0
END
```

A.3 LYRA Data Downloading Script

In the following sections, the source code of the BASH scripts used to download LYRA data and command logs are listed. There are two files which can be used separately to download the LYRA data or command logs. A third file is used to call both of these commands for convenience.

A.3.1 getLyraAria.sh

```
#!/bin/bash
#
# Download the lyra data of one day
#
 if [ $# -ne 3 ]
 then
        year=`date -d yesterday +%d`
        month=`date -d yesterday +%m`
        day=`date -d yesterday +%D`
# Decrement the day so on autorun it retrieves the most recent data
 else
        year=$1
        month=$2
        day=$3
 fi
 path=${year}'/'${month}'/'${day}
 url='http://proba2.oma.be/lyra/data/eng/'${year}'/'${month}'/'${day}

 echo $path

aria2c ${url}/lyra_${year}${month}${day}-000000_lev[1-3]_{std,met,cal,std,bst,
    bca,rej}.fits -P --dir=/LYRA/DATA/${path} --split=5 -Z
```

A.3.2 getlyraLogs.sh

```
#!/bin/bash
#
```

```
# Download the lyra logs of one day
#
 if [ $# -ne 3 ]
 then
        year=`date -d yesterday +%Y`
        month=`date -d yesterday +%m`
        day=`date -d yesterday +%d`
# Decrement the day so on autorun it retrieves the most recent data
 else
        year=$1
        month=$2
        day=$3
 fi
 path='/LYRA/DATA/'${year}'/'${month}'/'${day}

 d=`date --date "-2 days $year-$month-$day" +%d`
 m=`date --date "-2 days $year-$month-$day" +%m`
 y=`date --date "-2 days $year-$month-$day" +%Y`

 url='http://194.78.233.110/proba2/products/Eclipse/PROBA2_Info_'${y}${m}${d}'.
    txt'
 aria2c $url --dir=$path

 url='http://194.78.233.110/proba2/products/Eclipse/PROBA2_LTAN_'${y}${m}${d}'.
    txt'
 aria2c $url --dir=$path
```

A.3.3 getLyra.sh

```
#!/bin/bash
#
# Download the lyra data of one day
#
 if [ $# -ne 3 ]
 then
        year=`date -d yesterday +%Y`
        month=`date -d yesterday +%m`
        day=`date -d yesterday +%d`
# Decrement the day so on autorun it retrieves the most recent data
 else
        year=$1
        month=$2
        day=$3
 fi

bash getlyraAria.sh $year $month $day
bash getlyraLogs.sh $year $month $day
```

A.4 Sample Report Pages

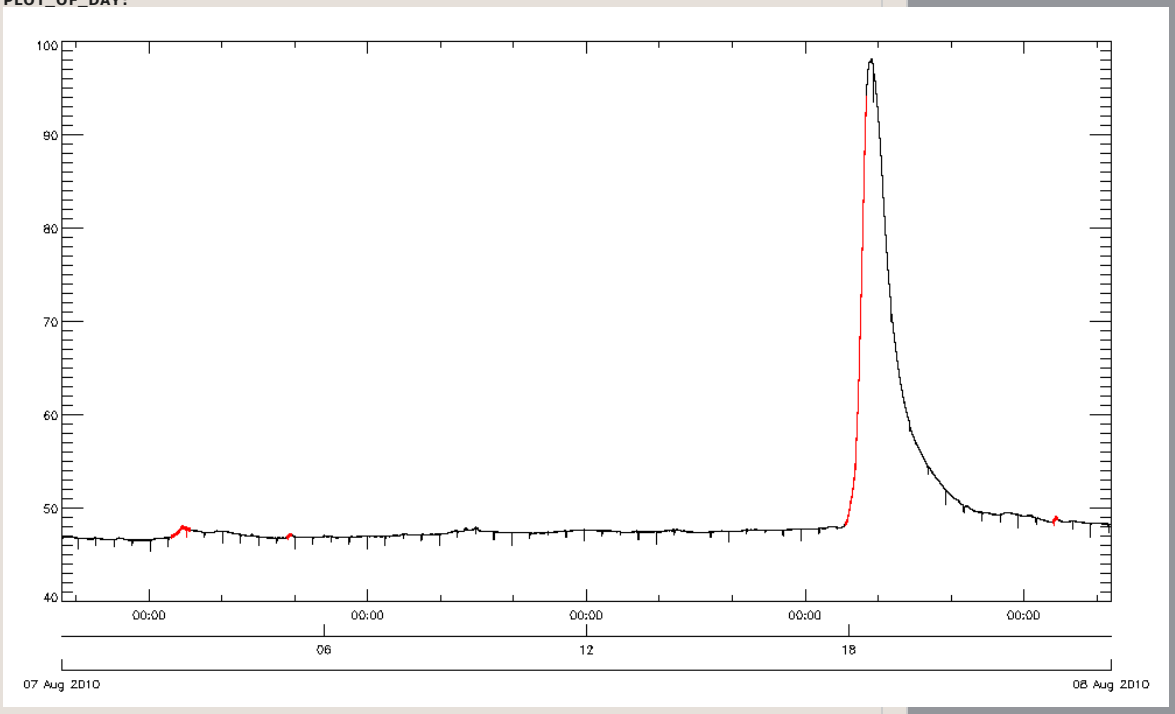
The following pages are extracted from a report corresponding to analysis performed from August 7th, 2010. This report was converted to PDF format prior to extraction.

Data Report

Generated Report

OVERVIEW

PLOT_OF_DAY:



SOLARSOFT_EVENTLIST

SOLARSOFT_EVENTLIST

EVENTSTART: 2010-08-07T02:30:00.000

EVENTEND: 2010-08-07T02:56:00.000

EVENTSTARTJD: 2455415.6

EVENTENDJD: 2455415.6

EVENTPEAK: 02:44:00

EVENTSTRENGTH: B3.0

LAG_MATCHED: 0

SOLARSOFT_EVENTLIST

EVENTSTART: 2010-08-07T05:09:00.000

EVENTEND: 2010-08-07T05:15:00.000

EVENTSTARTJD: 2455415.7

EVENTENDJD: 2455415.7

EVENTPEAK: 05:13:00

EVENTSTRENGTH: B1.7

LAG_MATCHED: 0

SOLARSOFT_EVENTLIST

EVENTSTART: 2010-08-07T17:55:00.000

EVENTEND: 2010-08-07T18:24:00.000

EVENTSTARTJD: 2455416.2

EVENTENDJD: 2455416.3

EVENTPEAK: 18:24:00

EVENTSTRENGTH: M1.0

LAG_MATCHED: 1

SOLARSOFT_EVENTLIST

EVENTSTART: 2010-08-07T22:40:00.000

EVENTEND: 2010-08-07T22:46:00.000

EVENTSTARTJD: 2455416.4

EVENTENDJD: 2455416.4

EVENTPEAK: 22:43:00

EVENTSTRENGTH: B2.9

LAG_MATCHED: 0

FINAL_REPORT

SUMMARY_STATS

WINDOW_COUNT: 56

TIME_START: 2010-08-07T00:22:35.000

TIME_END: 2010-08-07T23:27:50.000

WINDOW_CORRELATION

Field	SSW	SSW_N	LDW	LDW_N	FPW	FPW_N
SSW	6	0	2	4	1	5
SSW_N	0	50	6	44	0	50
LDW	2	6	8	0	1	7
LDW_N	4	44	0	48	0	48
FPW	1	0	1	0	1	0
FPW_N	5	50	7	48	0	55

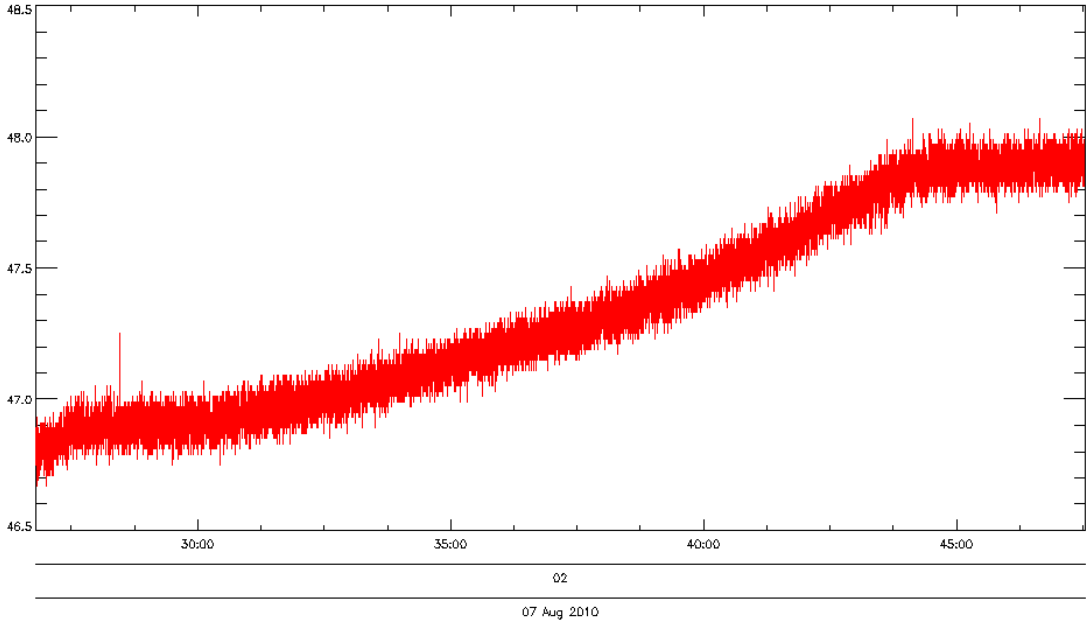
WINDOW_CORRELATION_IMAGES

NUMBER_OF_WINDOWS: 56

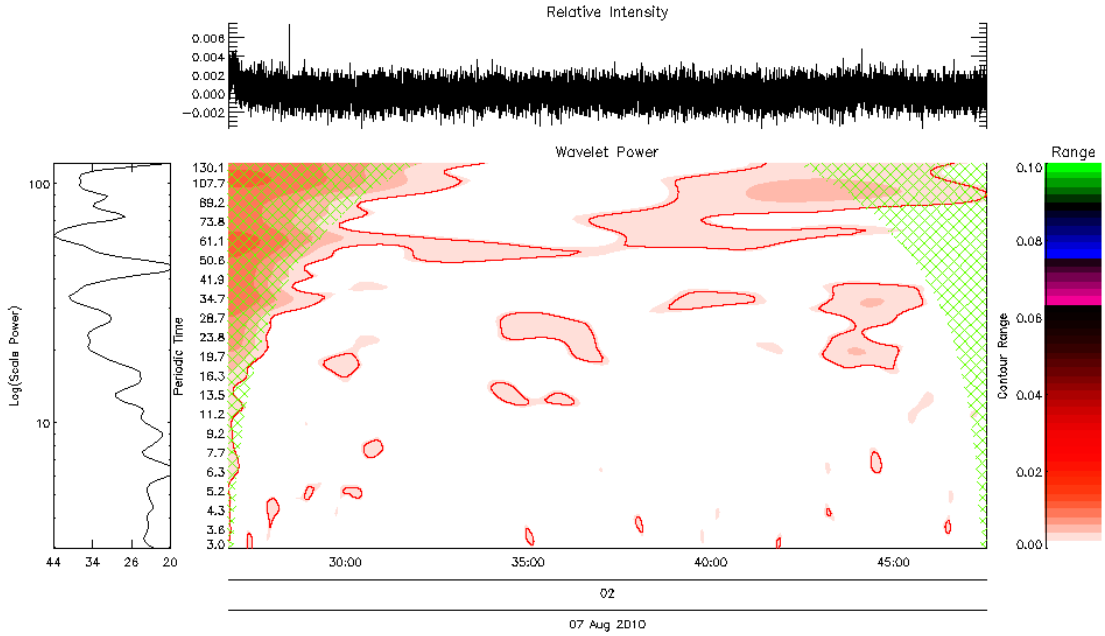
SSW_SSW

SSW_SSW

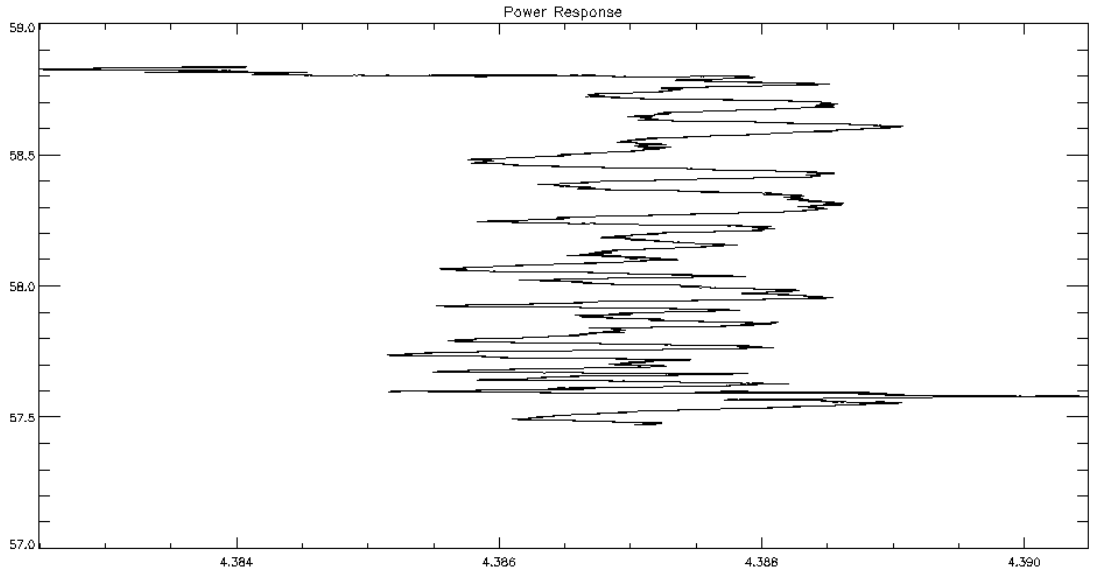
PLOT_TEMPORAL:



PLOT_WAVELET:



PLOT_FLARE_RESPONSE:

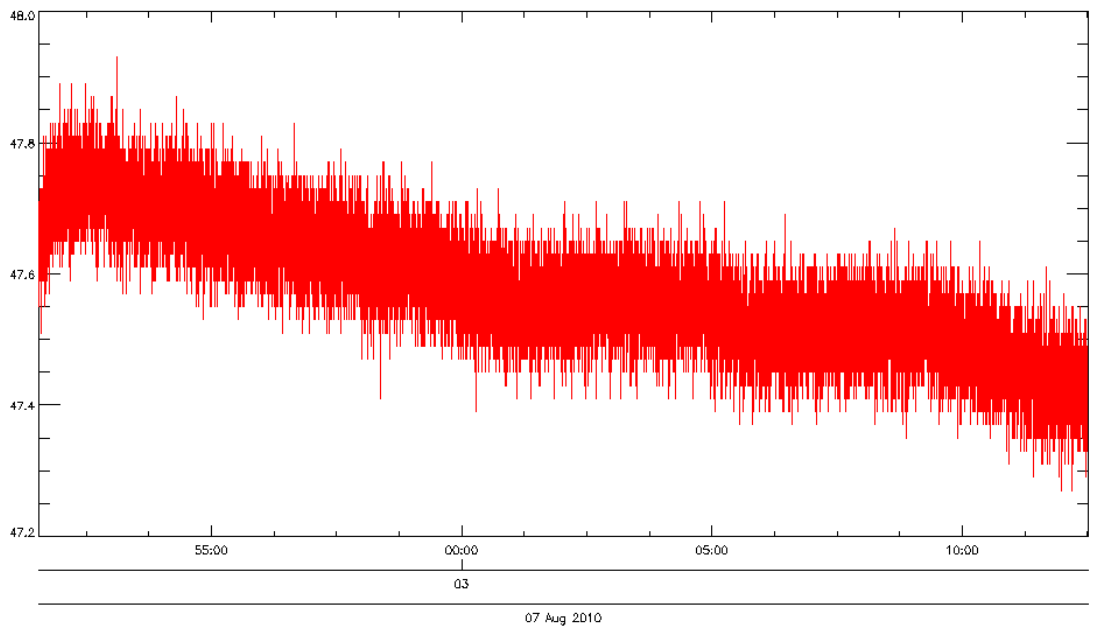


WINDOW_START: 2010-08-07T02:26:48.000

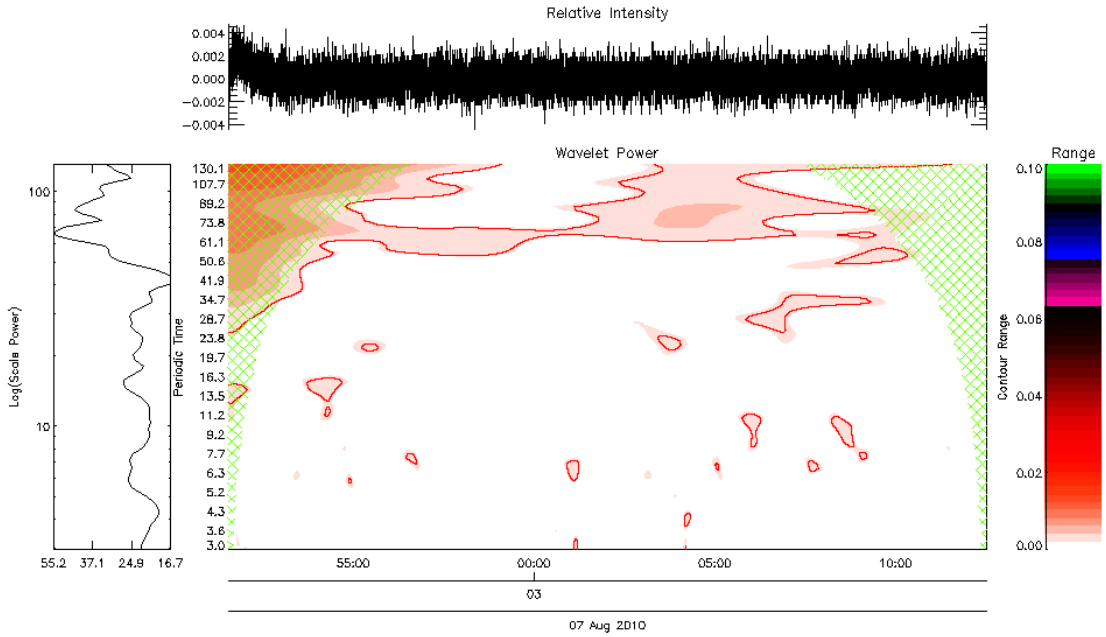
WINDOW_END: 2010-08-07T02:47:33.000

SSW_SSW

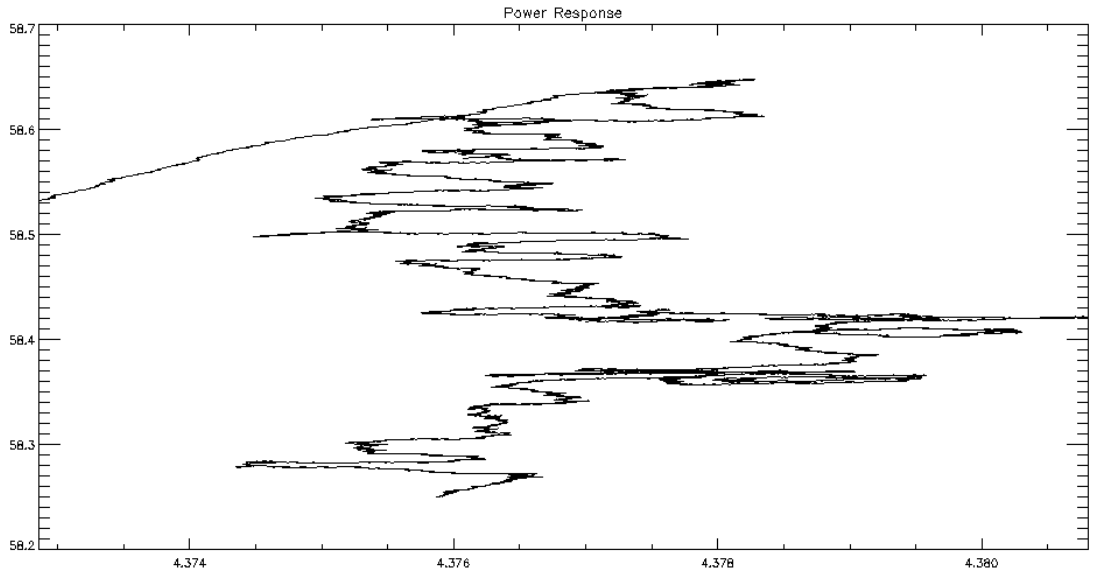
PLOT_TEMPORAL:



PLOT_WAVELET:



PLOT_FLARE_RESPONSE:

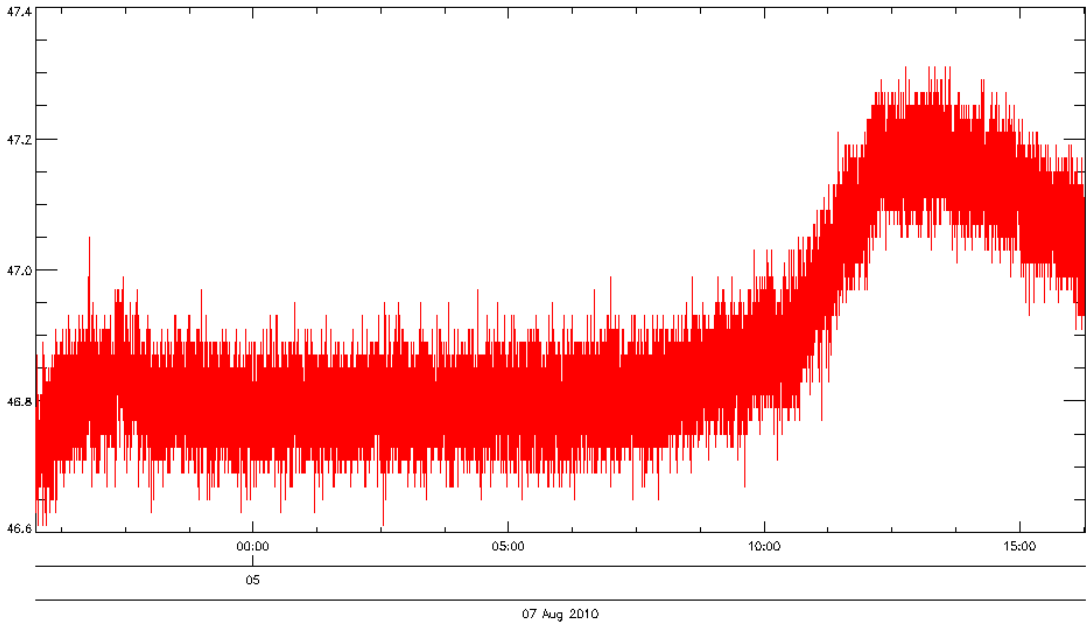


WINDOW_START: 2010-08-07T02:51:33.000

WINDOW_END: 2010-08-07T03:12:31.000

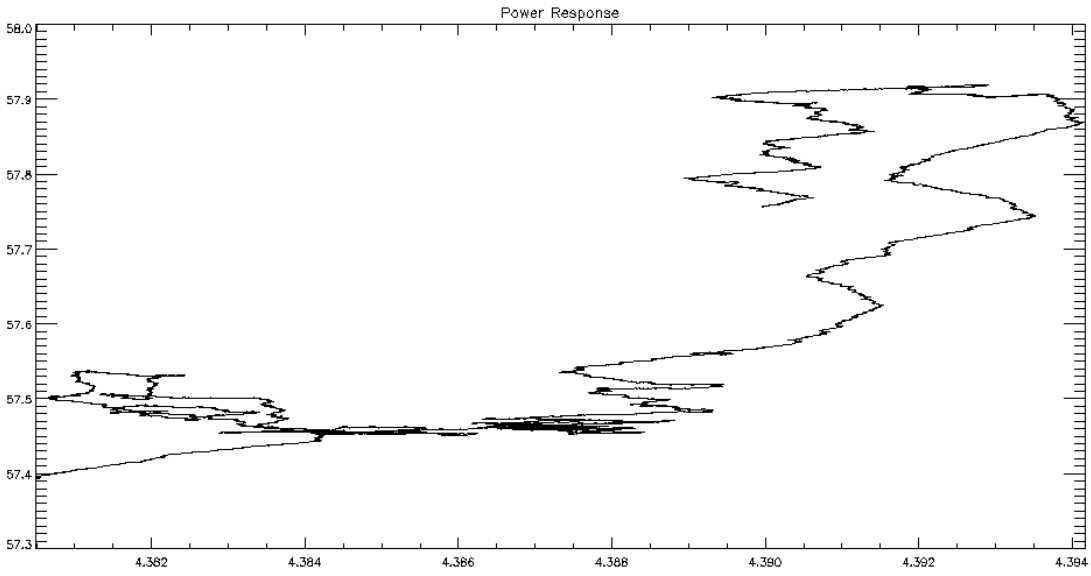
SSW_SSW

PLOT_TEMPORAL:



PLOT_WAVELET:

PLOT_FLARE_RESPONSE:

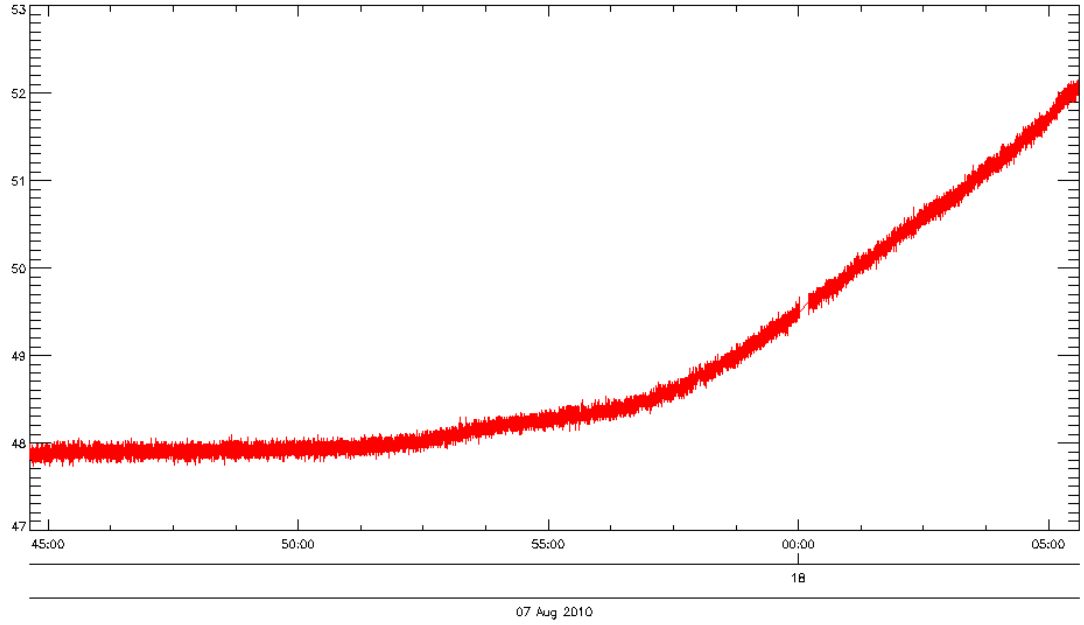


WINDOW_START: 2010-08-07T04:55:45.000

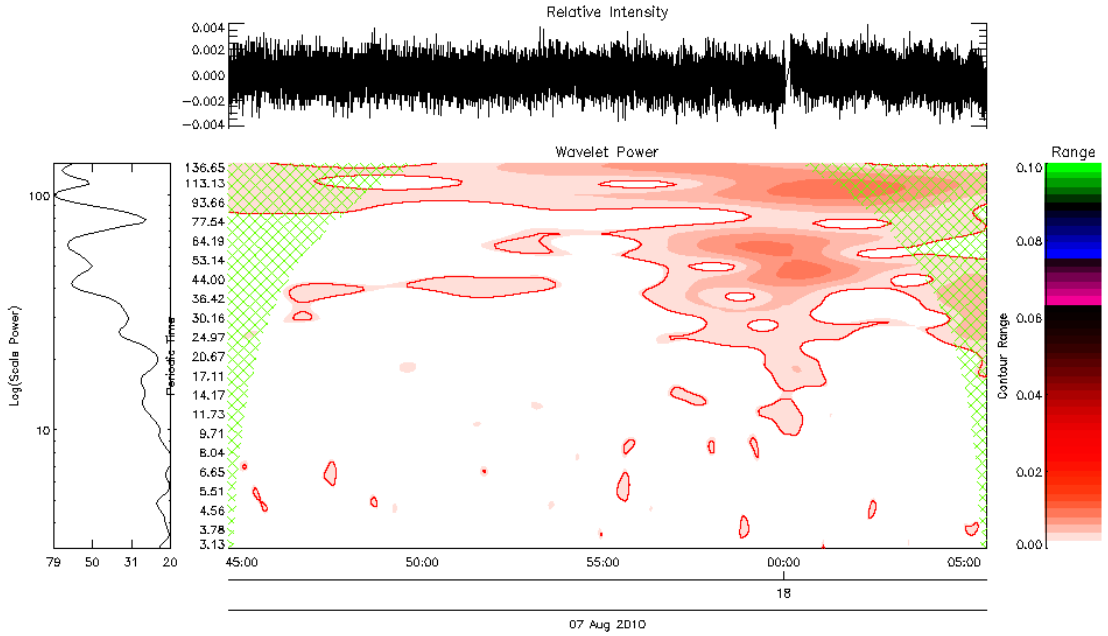
WINDOW_END: 2010-08-07T05:16:17.000

SSW_SSW

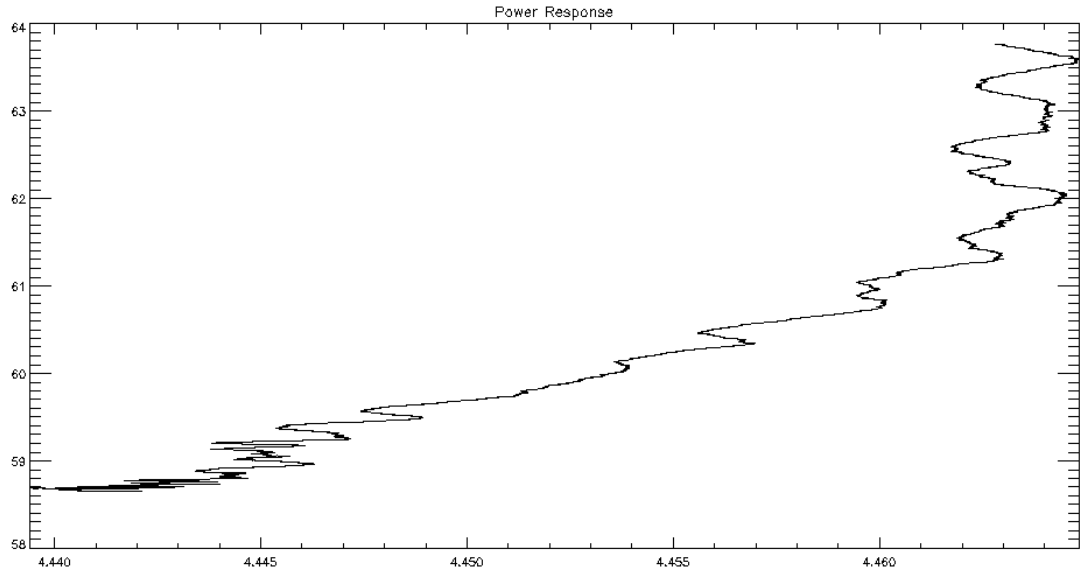
PLOT_TEMPORAL:



PLOT_WAVELET:



PLOT_FLARE_RESPONSE:

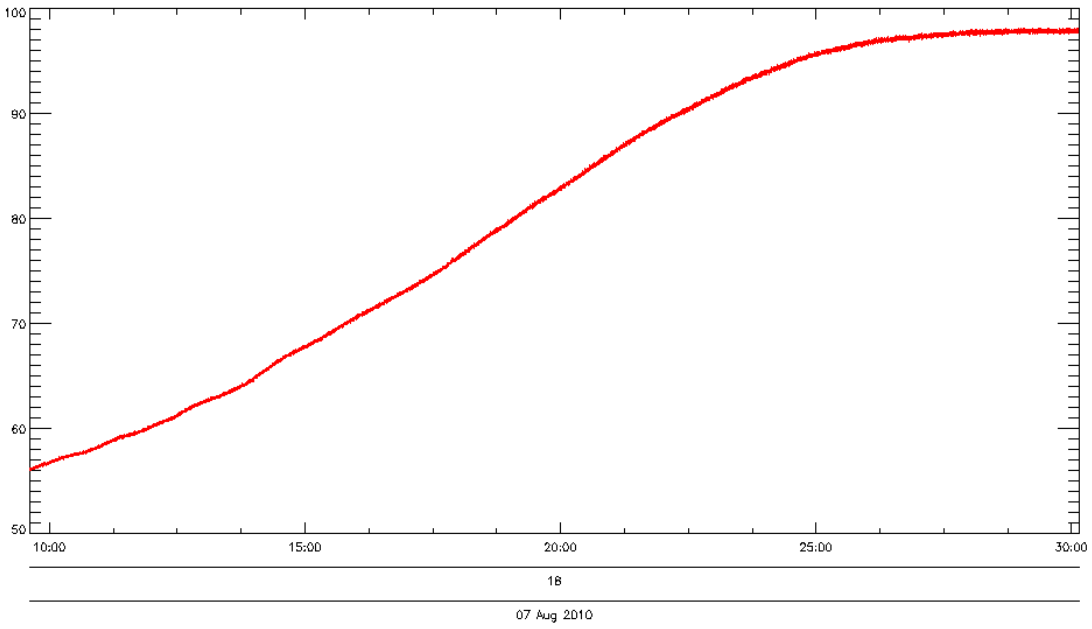


WINDOW_START: 2010-08-07T17:44:38.000

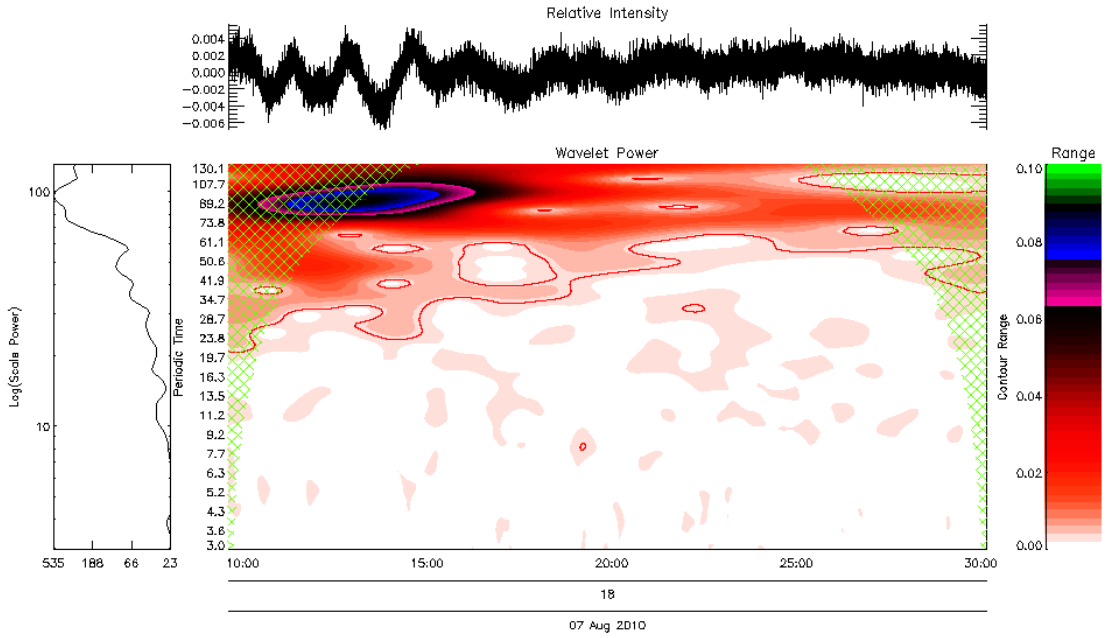
WINDOW_END: 2010-08-07T18:05:37.000

SSW_SSW

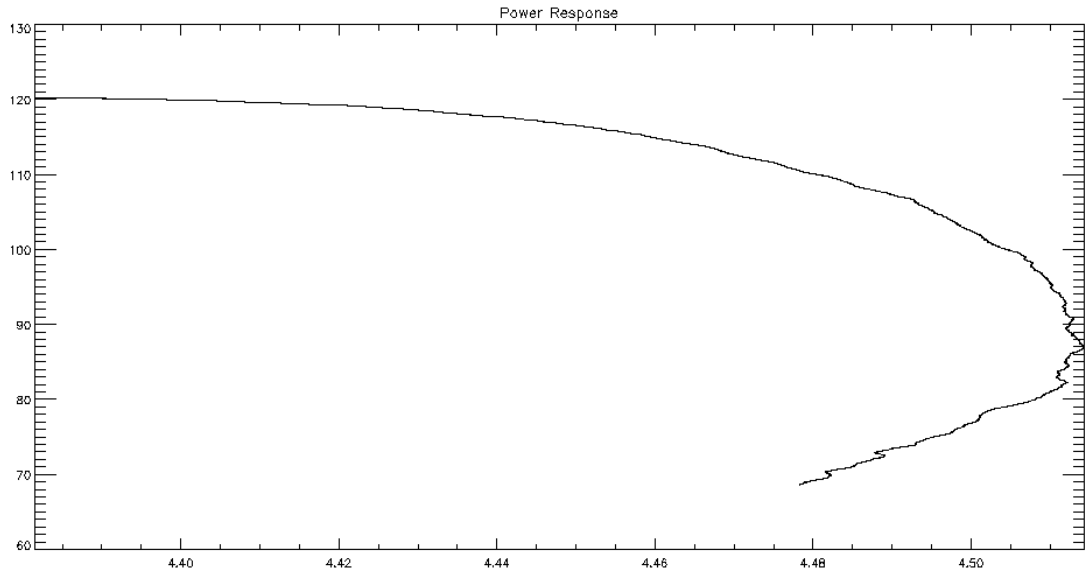
PLOT_TEMPORAL:



PLOT_WAVELET:



PLOT_FLARE_RESPONSE:

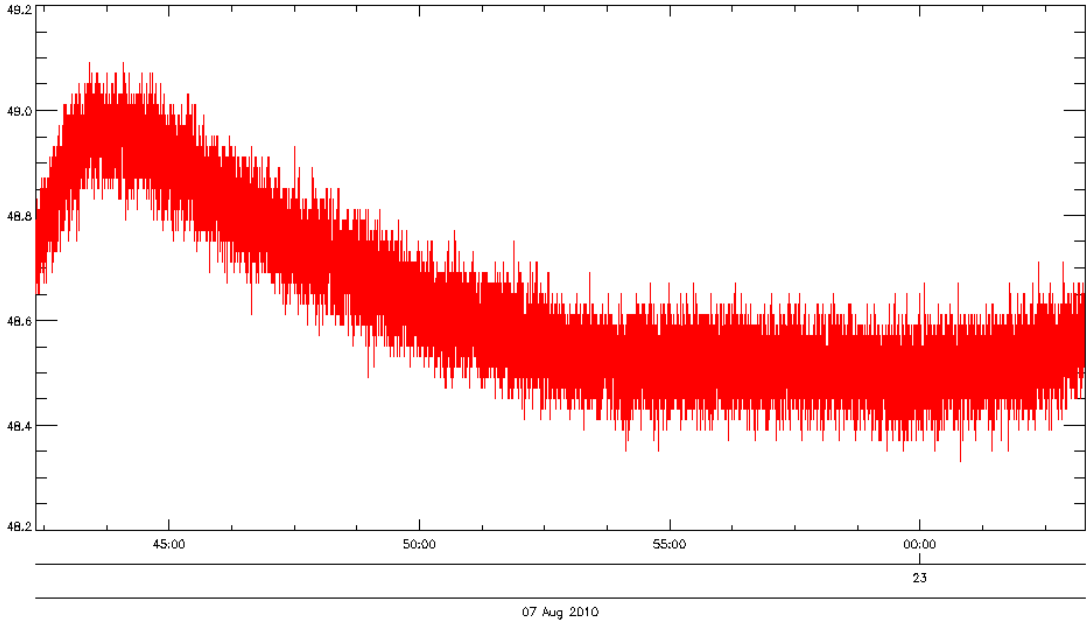


WINDOW_START: 2010-08-07T18:09:37.000

WINDOW_END: 2010-08-07T18:30:09.000

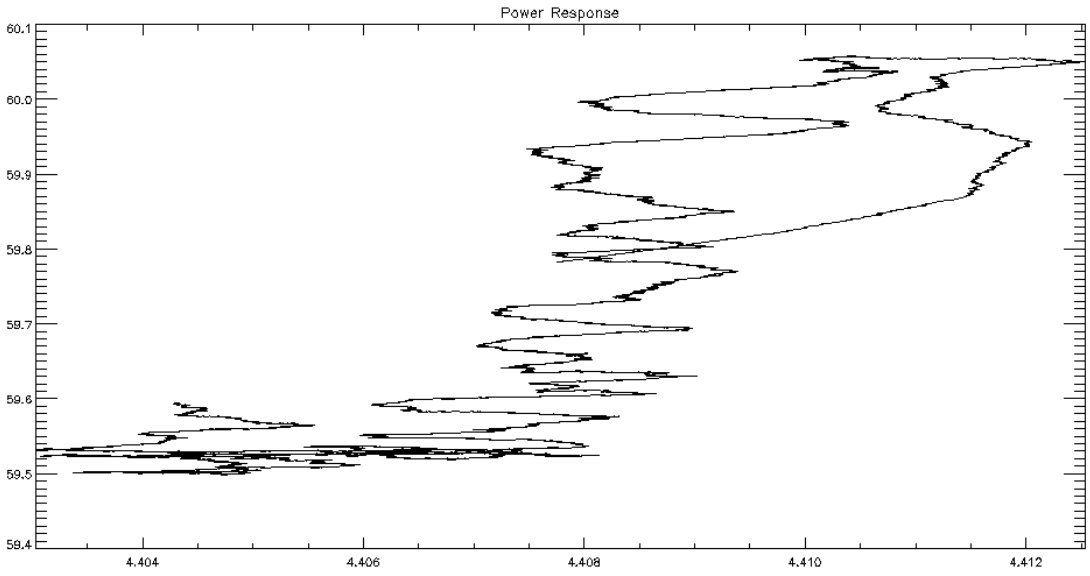
SSW_SSW

PLOT_TEMPORAL:



PLOT_WAVELET:

PLOT_FLARE_RESPONSE:



WINDOW_START: 2010-08-07T22:42:20.000

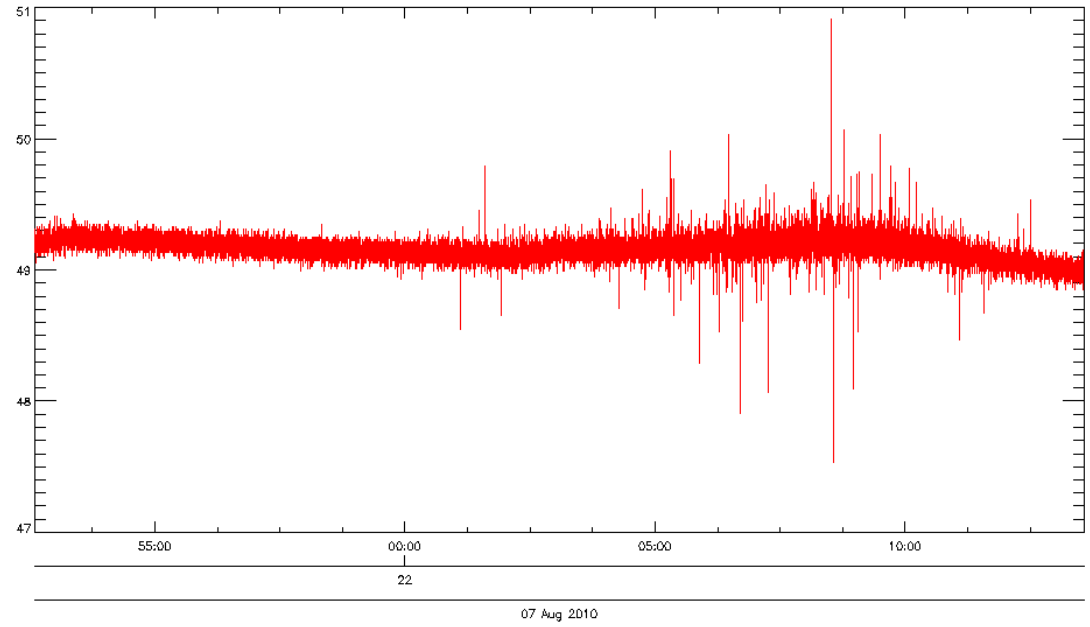
WINDOW_END: 2010-08-07T23:03:18.000

SSW_N_SSW_N

SSW_N_SSW_N

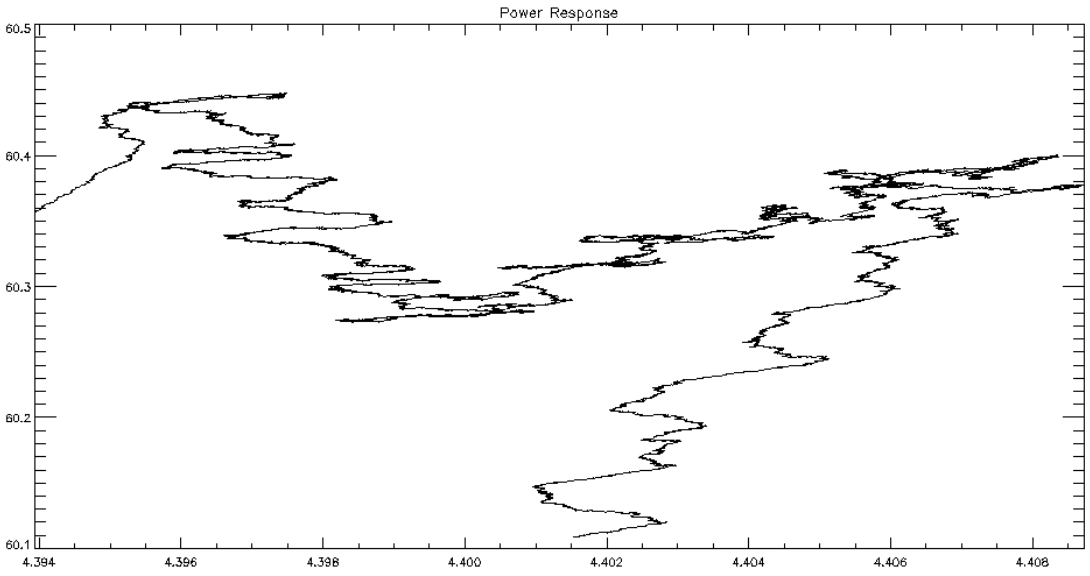
PLOT_TEMPORAL:

Pages 11– 364 omitted



PLOT_WAVELET:

PLOT_FLARE_RESPONSE:

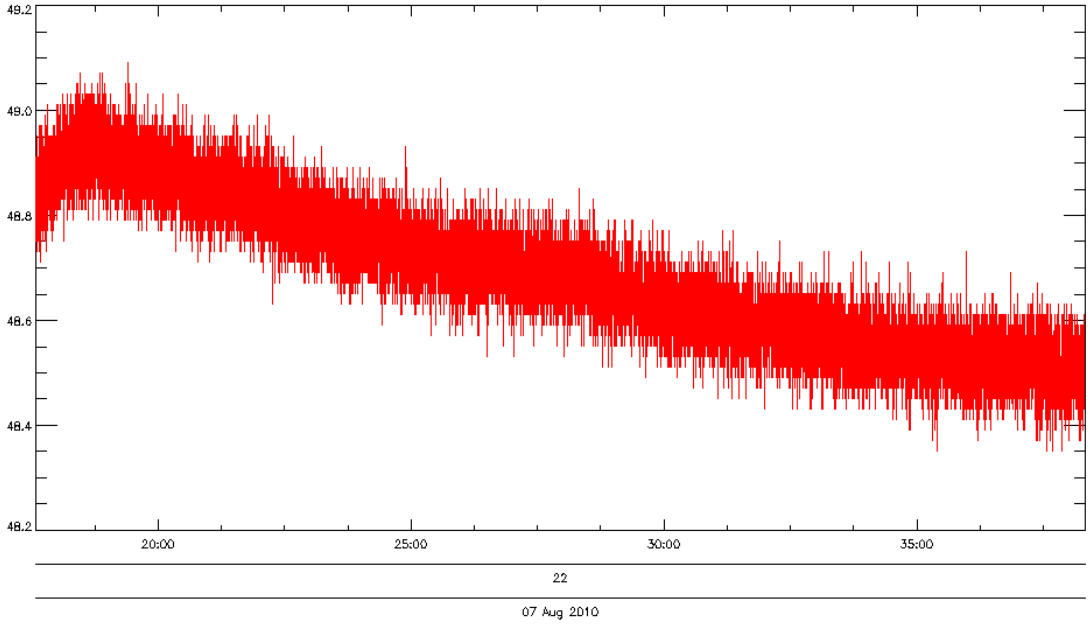


WINDOW_START: 2010-08-07T21:52:36.000

WINDOW_END: 2010-08-07T22:13:35.000

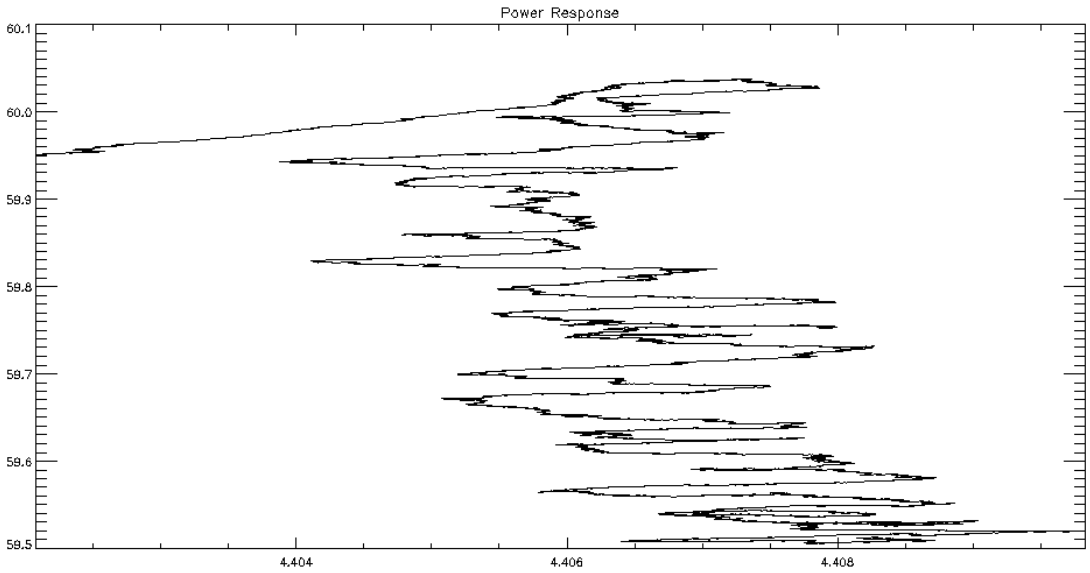
FPW_N_FPW_N

PLOT_TEMPORAL:



PLOT_WAVELET:

PLOT_FLARE_RESPONSE:

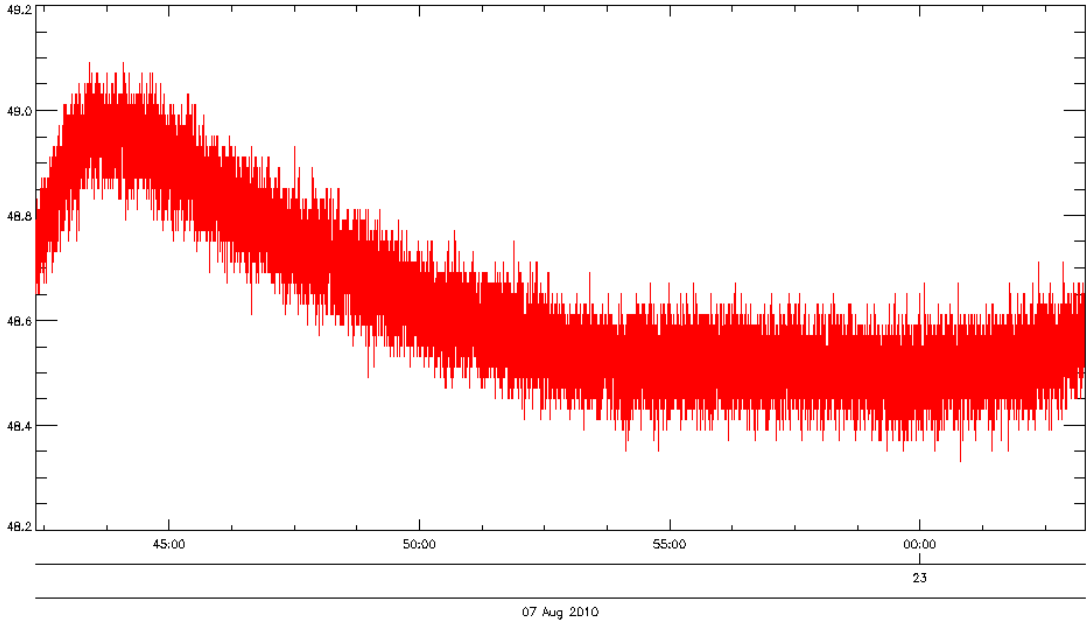


WINDOW_START: 2010-08-07T22:17:35.000

WINDOW_END: 2010-08-07T22:38:20.000

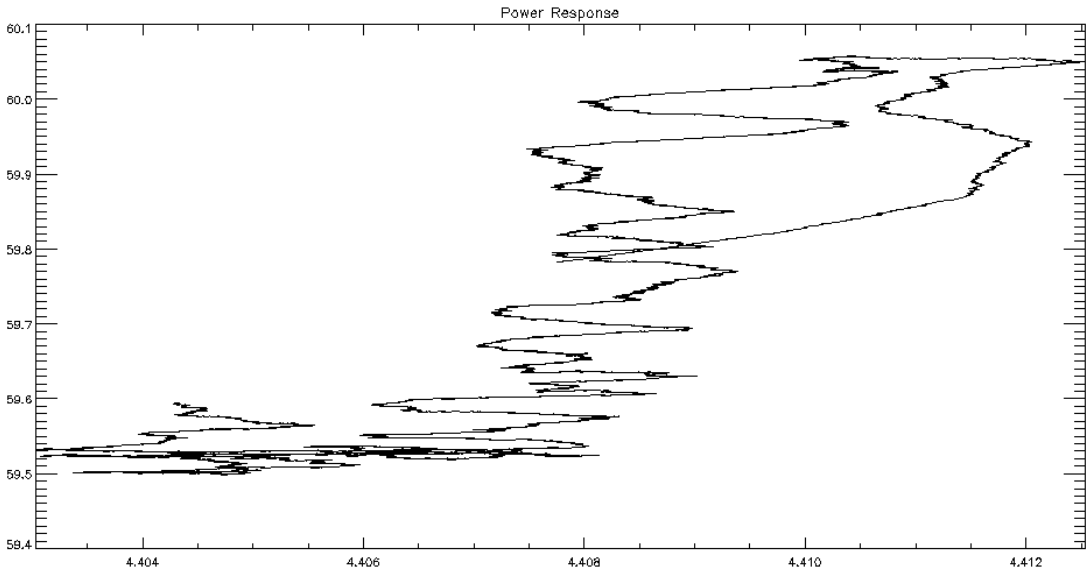
FPW_N_FPW_N

PLOT_TEMPORAL:



PLOT_WAVELET:

PLOT_FLARE_RESPONSE:

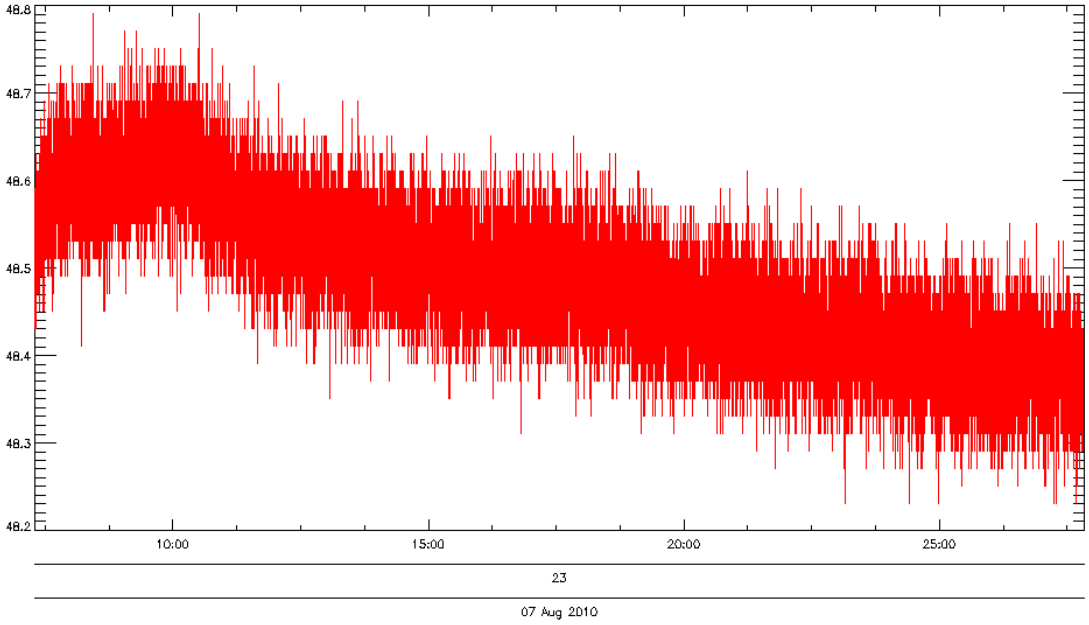


WINDOW_START: 2010-08-07T22:42:20.000

WINDOW_END: 2010-08-07T23:03:18.000

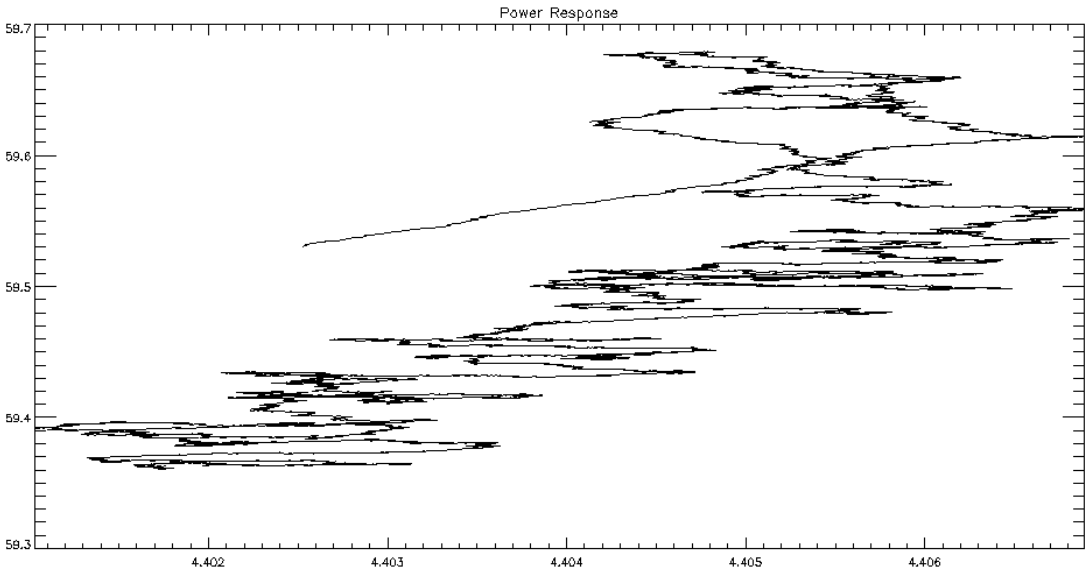
FPW_N_FPW_N

PLOT_TEMPORAL:



PLOT_WAVELET:

PLOT_FLARE_RESPONSE:



WINDOW_START: 2010-08-07T23:07:18.000

WINDOW_END: 2010-08-07T23:27:50.000

EVENT_CORRELATION

Field	SSE	LDE	LDE_N	FDE	FDE_N
SSE	4	1	3	1	4
LDE	1	1	0	1	1
LDE_N	3	0	3	0	3
FDE	1	1	0	1	1
FDE_N	4	1	3	1	4

PARAMETERS

TIMEPLOTPARAMETERS

PLOT_TYPE: time
SAVE_PLOT: 1
COLOR: 255
PLOT_CHANNEL: 4
YNOZERO: 1

FLAREDETECTIONPARAMETERS

POWER_RESPONSE: 0.0000000
MEDIAN_POWER_RESPONSE: 13.206733
MIN_POWER_RESPONSE: 0.060000000

WAVELETPLOTPARAMETERS

PLOT_TYPE: wv_contour
SAVE_PLOT: 1
RANGE

0.00000
0.100000

LEVELS: 50
CT_REVERSE: 1
CT: 14
SIGNIF_COLOR: 767
COI_COLOR: 65380

POWERRESPONSEPLOTPARAMETERS

PLOT_TYPE: power_response
SAVE_PLOT: 1
COLOR: 0

WAVELETPARAMETERS

WV_FAMILY: morlet
WV_ORDER: 10
WV_SIGNAL_T

60
100

WV_PAD: 1
WV_SIGNIF: 0.990000
SIGNAL_THRESHOLD: 0.015000000