

# **Supplementary wavelength calibration methods for SALT/RSS spectropolarimetric observations**

Justin Cooper

B.Sc. (Hons)

Submitted in fulfillment of the requirements for the degree

Magister Scientiæ

in the Faculty of Natural and Agricultural Sciences

Department of Physics

University of the Free State

South Africa

Date of submission: June 4, 2024

Supervised by: Prof. B. van Soelen, Department of Physics

# Abstract

TODO:

- Done last
- Flow from use of SALT and pipeline and basics of its science implementations into why a more streamlined wavelength calibration is an improvement.
- Give summary of results.
- Aim for a paragraph (~ 600) without going too in-depth into anything specific.
- Brian's comment: Abstract should summarize paper. Include results, conclusions, etc.

Keywords:

TODO:

- Add Keywords → look up the astronomy journal keywords
- Look up keywords for pipeline development and data reduction.
- I.E. Polarization: optical, Calibration: wavelength, galaxies: AGN, Blazars, Pipeline, SALT, etc.

## Acknowledgements

I hereby acknowledge and express my sincere gratitude to the following parties for their valuable contributions:

- **TODO: Add acknowledgements!**



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Spectropolarimetry and the SALT RSS</b>	<b>3</b>
2.1	Spectroscopy . . . . .	3
2.1.1	Telescope Optics . . . . .	3
2.1.2	Slit . . . . .	4
2.1.3	Collimator . . . . .	4
2.1.4	Dispersion Element . . . . .	4
2.1.5	Camera Optics . . . . .	5
2.1.6	Detector . . . . .	5
2.1.7	Dispersion of Light . . . . .	5
2.1.8	Detector and Spectroscopic Calibrations . . . . .	9
2.2	Polarimetry . . . . .	15
2.2.1	Polarization . . . . .	16
2.2.2	Polarization Measurement . . . . .	18
2.2.3	Polarimetric Calibrations . . . . .	22
2.3	Spectropolarimetry . . . . .	23
2.3.1	Spectropolarimetric Measurement . . . . .	24
2.3.2	Spectropolarimetric Calibrations . . . . .	25
2.4	The Southern African Large Telescope . . . . .	25
2.4.1	The Primary Mirror . . . . .	26
2.4.2	Tracker and Tracking . . . . .	26
2.4.3	SALT Instrumentation . . . . .	27
<b>3</b>	<b>Existing and Developed Software</b>	<b>31</b>
3.1	POLSALT . . . . .	31
3.1.1	Basic CCD Reductions . . . . .	32
3.1.2	Wavelength Calibrations . . . . .	32
3.1.3	Spectral Extraction . . . . .	33
3.1.4	Raw Stokes Calculations . . . . .	33
3.1.5	Final Stokes Calculations . . . . .	34
3.1.6	Visualization . . . . .	34
3.1.7	Post-Processing Analysis . . . . .	34
3.1.8	POLSALT Limitations and the Need for Supplementary Tools . . . . .	35
3.2	IRAF . . . . .	36
3.2.1	Identify . . . . .	37

3.2.2	Reidentify . . . . .	39
3.2.3	Fitcoords . . . . .	39
3.2.4	Transform . . . . .	40
3.3	STOPS . . . . .	40
3.3.1	Splitting . . . . .	41
3.3.2	Joining . . . . .	43
3.3.3	Sky Line Checks . . . . .	45
3.3.4	Cross Correlation . . . . .	48
3.4	General Reduction Procedure . . . . .	49
3.4.1	Initial Setup . . . . .	50
3.4.2	POLSLT Pre-Reductions . . . . .	51
3.4.3	Wavelength Calibration . . . . .	51
3.4.4	POLSLT Reduction Completion . . . . .	53
<b>4</b>	<b>Testing and Application</b>	<b>57</b>
<b>5</b>	<b>Science Applications</b>	<b>59</b>
5.1	Application to Spectropolarimetric Standards . . . . .	59
5.2	Application in Publications . . . . .	59
<b>6</b>	<b>Conclusions</b>	<b>61</b>
6.1	Future Work . . . . .	61
<b>I</b>	<b>The Modified Reduction Process</b>	<b>63</b>
<b>II</b>	<b>Supplementary Tools for POLSLT Spectropolarimetry (STOPS) Source Code</b>	<b>71</b>
	<b>Bibliography</b>	<b>115</b>
	<b>List of Acronyms</b>	<b>123</b>

# Chapter 1

## Introduction

TODO: Very short intro to Spectroscopy, Polarization, and Spectropolarimetry and their importance in astronomy

TODO: Problem Statement, VERY IMPORTANT, roughly a sentence but problem thoroughly fleshed out.

TODO: Focus on AGN implications and implementations such as the types of objects and a short history for each type of object, Blazar focus with specification on BL Lacs and FSRQs, the Unified Model, ~~The Blazar sequence~~

TODO: Brian's comment: Highlight importance of polarimetry for understanding emission and how that plays a role in AGN.

TODO: Basics of modelling (Different energy/wavelength ranges used and what the models tell us about emission processes/structure) so that Hester's results can be noted for applications of the pipeline.

TODO: General layout of Dissertation



# Chapter 2

## Spectropolarimetry and the SALT RSS

This chapter gives an overview of the basics of spectropolarimetry (§ 2.3), and how it functions, following from the principles of both spectroscopy (§ 2.1) and polarimetry (§ 2.2). Further, it is discussed how these techniques are practically implemented for Southern African Large Telescope (SALT) (§ 2.4), using the Robert Stobie Spectrograph (RSS) (§ 2.4.3), and how the spectropolarimetric reduction process is completed (§ 2.4.3).

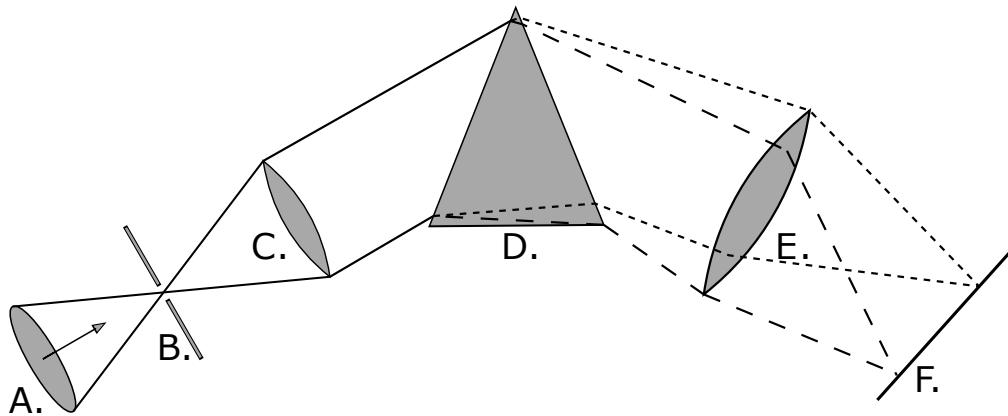
### 2.1 Spectroscopy

Spectroscopy originated in its most basic form with Newton's examinations of sunlight through a prism (Newton and Innys, 1730) but came to prominence as a field of scientific study with Wollaston's improvements to the optics elements (Wollaston, 1802), Fraunhofer's use of a diffraction grating instead of a prism (der Wissenschaften, 1824), and Bunsen and Kirchoff's classifications of spectral features to their respective chemical elements (Kirchhoff and Bunsen, 1861).

The simplest spectrometer schematic, as shown in Figure 2.1, consists of incident light collected from the telescope's optics, labelled A, being focused onto a slit, B, and passed through a collimator, C. The collimator collimates the light allowing a dispersion element, D, to disperse the light into its constituent wavelengths. The resultant spectrum is focused by camera optics, E, onto a focal plane, F. Viewing optics are situated at the focal plane in the case of a spectroscope and a detector is situated at the focal plane in the case of a spectrograph.

#### 2.1.1 Telescope Optics

The telescope optics refers simply to all the components of a telescope necessary to acquire a focal point at the spectrometer entrance, labelled B. The focal point in most traditional telescope designs is fixed relative to the telescope and so the spectrometer may be mounted at that point. In cases where the telescope is designed to have a moving focal point relative to the telescope (see Buckley et al., 2006; Cohen, 2009; Ramsey et al., 1998), the spectrometer, or a signal transfer method such as a fibre feed to the spectrometer, must also move along the telescope's focal path.



**Figure 2.1:** Layout depicting the light path through a spectrometer. Diagram adapted from Birney et al. (2006).

### 2.1.2 Slit

The slit's function is to control the amount of incident light entering a spectrometer and, along with the exposure time of the detector, prevents over-exposures of bright sources on highly sensitive detectors (Tonkin, 2013). If a source is spatially resolvable, or larger than the seeing conditions, the slit additionally acts to spatially limit the source to increase the spectral resolution, resulting in sharper features in the resultant spectrum. Without the slit the spectral resolution would be determined by the projected width of the source on the detector, or the seeing if the source was a star-like point source. Increasing the spectral resolution comes with the trade-off of decreasing the light collected from the source and thus acquiring a less intense resultant spectrum. Multiple spectra may be acquired simultaneously when the slit is positioned such that collinear sources lie along the slit.

The spectrometer is usually situated at the focal point. In cases where this is not feasible due to restrictions, for example restrictions of weight or size, a fibre feed may be situated behind the slit on the telescope. This allows the signal to be routed away from the telescope to a controlled environment with only minuscule losses.

### 2.1.3 Collimator

The collimators function is to collimate the focused light from the telescope, ensuring that all light rays run parallel before reaching the dispersion element. The focal ratio of the collimator ( $f_c/D_c$ , where  $f$  refers to the focal length and  $D$  refers to the diameter) should ideally match the focal ratio of the telescope ( $f_T/D_T$ ).

### 2.1.4 Dispersion Element

Including a dispersion element in the optical path is what defines a spectrometer. As the name suggests, a dispersion element disperses the light incident on it into its constituent wavelengths and produces a spectrum. There are two types of dispersion elements, namely the prism and the diffraction grating, which operate on different principles, as discussed in § 2.1.7.

### 2.1.5 Camera Optics

The lens functions similarly to that of the telescope's optics but in this case focuses the dispersed light onto a receiver situated at the focal plane. As mentioned previously, an eye piece is fixed to the focal point for a spectroscope while a spectrograph employs a detector.

### 2.1.6 Detector

The two most prevalent detector types in spectroscopy are the Charged-Coupled Device (CCD) and Complementary Metal-Oxide-Semiconductor (CMOS) detectors. In astronomical spectroscopy however, sources are fainter and exposure times are much longer and so the CCD detectors are by far the preferred detector as their output has a higher-quality and lower-noise when compared to CMOS cameras under the same conditions (Janesick et al., 2006).

The CCD is a detector composed of many thousands of pixels which can store a charge so long as a voltage is maintained across the pixels. Each pixel detects incoming photons using photo-sensitive capacitors through the photoelectric effect and converts the photons to a charge (Buil, 1991). There are also thermal agitation effects which introduce noise to the charge accumulated by a pixel, further discussed in § 2.1.8. Once the exposure is finished the accumulated charge is read column by column, row by row, through an Analog-to-Digital Converter (ADC) which produces a two-dimensional array of 'counts'.

### 2.1.7 Dispersion of Light

Light can be broken up into its constituent wavelengths through two different physical phenomena, namely dispersion and diffraction, which dispersive elements use to create spectra. Dispersive prisms and diffractive gratings each have their strengths and weaknesses and a wide spectrum of instruments exist which implement either, or both, concepts. Regardless of the specific element, dispersive elements all have a resolving power,  $R$ , and an angular dispersion. Generally, while the angular dispersion is a more involved process to determine, the resolving power of a spectrograph can be measured as:

$$R = \frac{\lambda}{FWHM}, \quad (2.1)$$

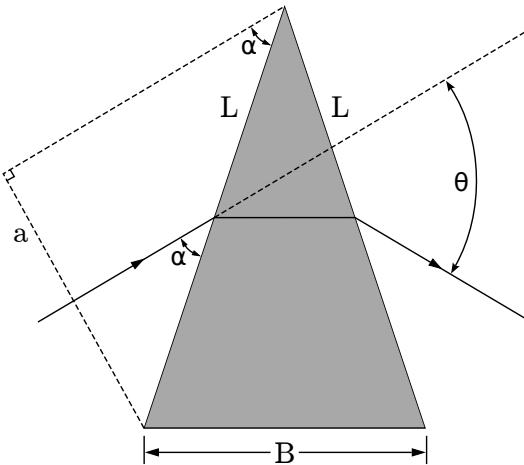
where  $\lambda$  is the wavelength of an incident monochromatic beam and Full Width at Half Maximum (FWHM) refers to the width of the feature on the detector at half of its maximum intensity.

#### Prism

The prism operates on the principle that the refractive index of light,  $n$ , varies as a function of its wavelength,  $\lambda$ . Prisms were the only dispersive elements available for early spectroscopic studies, but they were not without flaw. The angular dispersion of a prism is given by:

$$\frac{\partial\theta}{\partial\lambda} = \frac{B}{a} \frac{dn}{d\lambda}, \quad (2.2)$$

where  $\theta$  is the angle at which the refracted light differs from the incident light,  $\lambda$  is the wavelength of the incident light,  $B$  is the longest distance the beam would travel through



**Figure 2.2:** Geometry of a prism refracting an incident monochromatic beam at a minimum deviation angle. Diagram adapted from Birney et al. (2006).

the prism.  $a = L \sin(\alpha)$  is the maximal beam width that would fit onto a prism with a transmissive surface of length  $L$  for a given angle,  $\alpha$ , at which a beam would strike the transmissive surface, as shown in Figure 2.2.

The refractive index of a material as a function of its wavelength,  $n(\lambda)$ , can be approximated by Cauchy's equation:

$$n(\lambda) = A_C + \frac{B_C}{\lambda^2} + \frac{C_C}{\lambda^4} + \dots, \quad (2.3)$$

where  $A_C, B_C, C_C$  are the Cauchy coefficients and have known values for certain materials. Cauchy's equation is a much simpler approximation of the refractive index that remains very accurate at visible wavelengths (Jenkins and White, 1976). Taking only the first term of the derivative of the Cauchy equation allows us to approximate the angular dispersion of a prism,

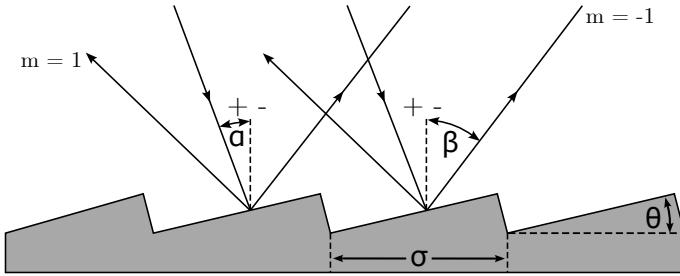
$$\frac{\partial \theta}{\partial \lambda} = -\frac{B}{a} \frac{2B_C}{\lambda^3} \propto -\lambda^{-3}, \quad (2.4)$$

which shows that the angular dispersion of a prism is wavelength dependent and furthermore that longer wavelengths are dispersed less than shorter wavelengths (Birney et al., 2006; Hecht, 2017). The dependence of the angular dispersion,  $d\theta/d\lambda$ , on the wavelength,  $\lambda$ , is crucial for the formation of a spectrum but this cubic, non-linear, relation results in a non-linear spectrum. Since prisms rely on the refractive index of the material they are made of, they have low angular dispersions.

Multiple prisms can be used to increase the angular dispersion but as the dispersion is non-linear it becomes increasingly more difficult to calibrate. The more material and material boundaries the light must pass through, the more its intensity decreases due to attenuation effects and Fresnel losses. Even so, the transmittance of modern prisms for their selected wavelength range is generally very high due to improved manufacturing methods as well as improved transmitting materials.<sup>1</sup>

---

<sup>1</sup>See manufacturers technical specifications, THORLABS, or Edmund Optics for example.



**Figure 2.3:** Geometry of a reflective blazed grating refracting an incident monochromatic beam. Diagram adapted from Birney et al. (2006).

## Diffraction Grating

The alternative dispersing element is a diffraction grating, which operates on the principle that as light interacts with a grating where the groove size is comparable to the light's wavelength, the light is dispersed through constructive and destructive interference. This interference results in multiple diffracted beams  $m$ , called orders, either side of a central reflected, or transmitted, beam such that  $m \in \mathbb{Z}$ , where  $m = 0$  is the non-dispersed, or reflected, beam.

An example of a reflective blazed grating is illustrated in Figure 2.3. Here a monochromatic beam is incident on the grating at an angle of  $\alpha$  from the grating normal. Due to the interference, a diffracted beam of wavelength  $\lambda$  is found at an angle of  $\beta$  from the grating normal. The relation between the incident and diffracted beams is given by the grating equation:

$$m\lambda = \sigma(\sin(\alpha) \pm \sin(\beta)), \quad (2.5)$$

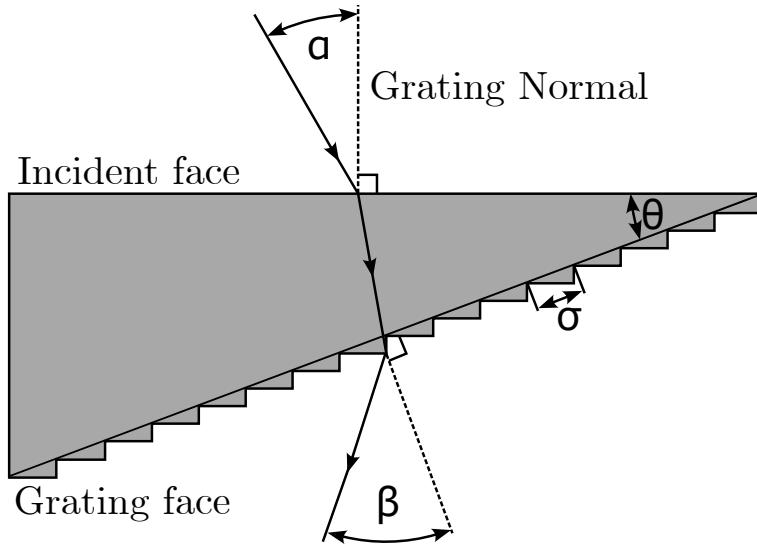
where  $\sigma$  is the groove spacing of the grating and  $m$  is the order of the diffracted beam being considered. The grating equation also applies to transmission gratings, though care should be taken for the signs of  $\alpha$  and  $\beta$ .

Equation 2.5 also shows that different diffracted beams may share an angle of dispersion for beams not in the same order. The regions of an order that do not overlap with another order are called free spectral ranges. An order-blocking filter may be used to account for the overlaps and increase the free spectral range. A diffraction grating can also be blazed by an angle  $\theta$ , as illustrated in Figure 2.3. Blazing refers to the fact that the grooves on the surface of the grating are not symmetrical. The asymmetry of the grooves diffracts the incident beam such that most of the beam's intensity is found in a reflected, zeroth order, beam. The wavelength at which a blazed spectrograph is most effective is called the blaze wavelength,  $\lambda_b$ , which is determined by:

$$\begin{aligned} m\lambda_b &= 2\sigma \sin(\theta) \cos(\alpha - \theta), \text{ where} \\ 2\theta &= \alpha + \beta. \end{aligned} \quad (2.6)$$

Taking the derivative of Equation 2.5 with respect to  $\lambda$  while keeping  $\alpha$  constant, allows us to determine the angular dispersion of a diffraction grating,

$$\frac{\partial \beta}{\partial \lambda} = \frac{m}{\sigma \cos(\beta)}. \quad (2.7)$$



**Figure 2.4:** Diagram of a grism for an incident monochromatic beam of light and a diffracted beam of order  $m = 1$ . Diagram adapted from Birney et al. (2006).

Substituting  $m/\sigma$  with the grating equation results in

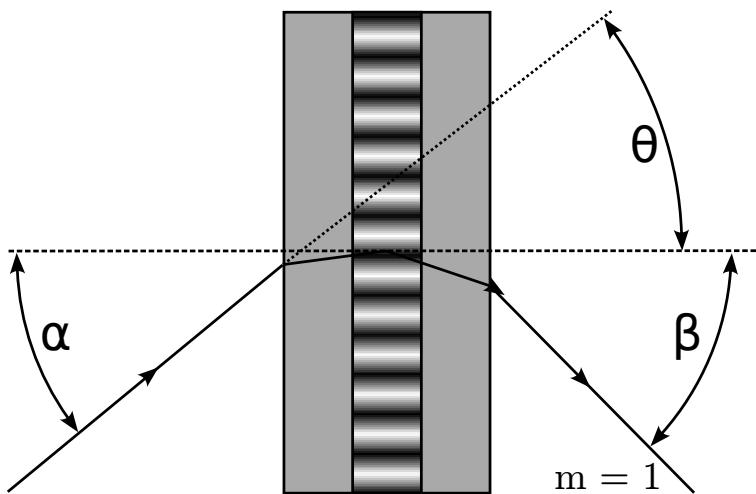
$$\frac{\partial \beta}{\partial \lambda} = \frac{\sin(\alpha) + \sin(\beta)}{\lambda \cos(\beta)} \propto \lambda^{-1}. \quad (2.8)$$

Similar to the dispersion of a prism, Equation 2.8 shows that the dispersion of a grating is wavelength dependent, but this dependence is only inversely proportional and thus more uniform across a wavelength range than that of a prism. Furthermore, shorter wavelengths are refracted less than longer wavelengths since there is no negative relation between the angular dispersion and the wavelength (Birney et al., 2006; Hecht, 2017).

### Alternate Diffraction Elements

As mentioned before, multiple subgroups exist for both dispersive prisms and diffractive gratings. For prisms, along with the single and multiple prism setups mentioned, there also exists grisms and immersed gratings. A grism (Grating Prism), as shown in Figure 2.4, refers to a transmissive grating etched onto one of the transmissive faces of a prism and allows a single camera to capture both spectroscopic and photometric images without needing to be moved, with and without the grism in the path of the beam of light, respectively. An immersed grating refers to a grism modified such that the transmissive grating is coated with reflective material. The primary source of dispersion for both grisms and immersive gratings is the grating and any aberration effects from the prism are negligible in comparison.

Other types of gratings include the Volume Phase Holographic (VPH) grating as well as the echelle grating. The VPH grating consists of a photoresist, which is a light-sensitive material, sandwiched between two glass substrates. Diffraction is possible since the photoresist's refractive index varies near-sinusoidally perpendicularly to the gratings lines, as seen in Figure 2.5. This allows for sharper diffraction orders and low stray light scattering as compared to more traditional gratings but since blazing is not possible the



**Figure 2.5:** Diagram of a VPH grating for an incident monochromatic beam of light. Diagram adapted from Birney et al. (2006).

efficiency is decreased. An echelle grating refers to a diffraction grating with higher groove spacing which is optimized for use at high orders. The high order of the diffracted beam allows for greater angular dispersion which is most useful when combined with another dispersion element to cross-disperse a spectrum, resulting in a high resolution spectrum.

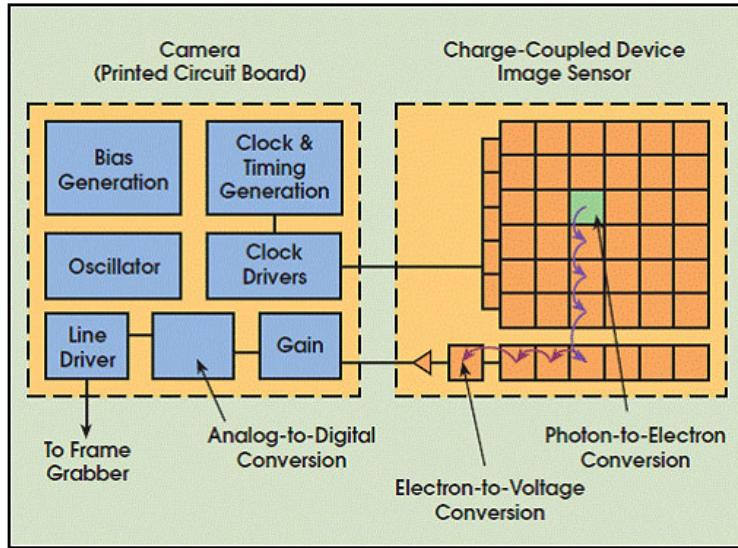
### 2.1.8 Detector and Spectroscopic Calibrations

Acquiring a spectrum from observations is more involved than simply reading out the data recorded on the CCD. A raw science image, which is the raw counts of the observed source read from the CCD with no calibrations applied, has on it a combination of useful science data as well as noise. The noise is a combination of random noise introduced through statistical processes and systematic noise introduced through the instrumentation and the observation conditions the source was observed under. This noise causes an uncertainty in the useful data and can be minimized, predominantly by calibrating for the systematic noise, but never fully removed (Howell, 2006).

The dominant source of noise in a raw image is detector noise. CCDs are manufactured to have a small base charge in each pixel, called the ‘bias’ current which allows the readout noise, a type of random noise, to better be sampled. There is also an unintentional additional charge which is linearly proportional to the exposure time and originates from thermal agitation of the CCD material, called the ‘dark’ current. The dark current can be minimized and possibly ignored if the CCD is adequately cooled. These types of noise add to the charge held by a pixel and are thus considered additive.

The CCD is not a perfect detector and the efficiency of it and the optics of the telescope also contribute noise to the image. The efficiency of a CCD is referred to as the Quantum Efficiency, and it is a measure of what percentage of light striking the detector is actually recorded and converted to a charge. The efficiency of the CCD and telescope optics is also wavelength dependent and so the noise that results from them is more complex than that of additive noise. This type of noise is referred to as multiplicative noise.

Additive noise, such as bias and dark currents, is inherent to CCD images, and as such



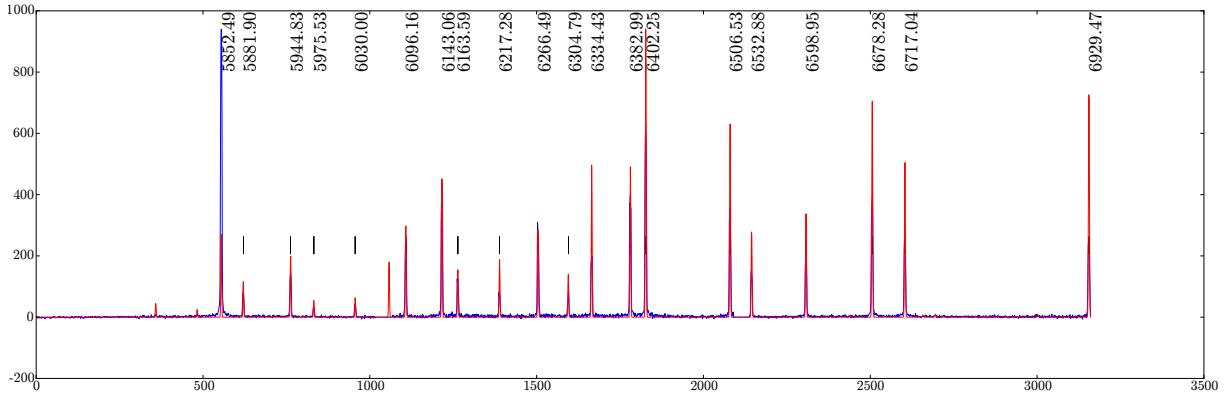
**Figure 2.6:** Diagram of the inner logic of a CCD. Figure adapted from Litwiller (2001).

needs to be subtracted out first when performing calibrations. Bias currents can be found by taking a bias image or by adding an overscan region to each image. A bias image is an image where the charges on the CCD are reset and then immediately read off without exposing anything on the detector, effectively taking an image with zero exposure time. Alternatively, to save time during an observational run, overscan regions may be added to the images. An overscan region refers to adding a few cycles to the readout of each column of the CCD such that the base current is read out and appended to each image.

Dark currents can be found by taking an image with nothing exposed onto the detector for a certain exposure time. This resultant dark image can then be scaled to the science images exposure time since the dark current should be linearly proportional to exposure time. When the detector is capable of being held at precise temperatures, dark images may be taken over multiple hours during the day to produce a high quality master dark image that may then be scaled and subtracted from all subsequent images.

Next, multiplicative noise, such as a CCD’s pixel-to-pixel response, should be accounted for. This pixel-to-pixel response should be uniform across the image and to achieve this an average response may be divided out. The average response is referred to as a ‘flat’ image or flat-field and may be acquired by observing a uniformly illuminated surface to determine the pixel-to-pixel response.

Dome flats are images taken of a relatively flat surface, usually the inside a telescopes dome, and are used in both photometry and spectroscopy. The surface is uniformly and indirectly illuminated by a projector lamp, ideal for flat-field images. Alternate flat-fielding methods, such as night sky and twilight flats, are available but are suited solely for photometry. Night sky flats are produced from science images containing mostly sky. The science images are combined using the ‘mode’ statistic which removes any celestial objects at the cost of a low Signal-to-Noise Ratio (S/N) flat-field. Twilight flats are produced from images of the twilight (or dawn) sky. They are taken when the Sun has just set, in the opposite direction, at  $\sim 20^\circ$  from zenith and provide a better S/Ns at the cost of careful timing of the images.



**Figure 2.7:** Example of an arc spectrum for NeAr taken with SALT’s RSS using the PG1800 grating at a grating angle of  $34.625^\circ$ , an articulation angle of  $69.258^\circ$ , and covering a wavelength range of  $\sim 5600 - 6900 \text{ \AA}$ . Plot adapted from SALT’s published Longslit Line Atlases, (2023).<sup>2</sup>

A flat-field must be normalized before being used to correct any science images since it only acts to account for the pixel-to-pixel response and not for the additive errors. A normalized spectroscopic flat image,  $F_\lambda^n(x, y)$ , can be calculated as:

$$F_\lambda^n(x, y) = \frac{F_\lambda(x, y) - B(x, y) - (\frac{t_S}{t_D})D(x, y)}{\text{med}_{lp}(F_\lambda(x, y) - B(x, y) - (\frac{t_S}{t_D})D(x, y))}, \quad (2.9)$$

where  $F_\lambda(x, y)$  is the non-corrected flat image,  $B(x, y)$  is the bias image,  $D(x, y)$  is the dark image which is scaled by the exposure time of the science image,  $t_S$ , and the dark image,  $t_D$ .  $\text{med}_{lp}$  is a low-pass median filter which smoothes out any rapid changes in the pixel-to-pixel response, removing the illumination contribution.

The calibrated science image,  $S_\lambda^*(x, y)$ , which accounts for the bias and dark currents as well as the flat fielding can then be calculated as:

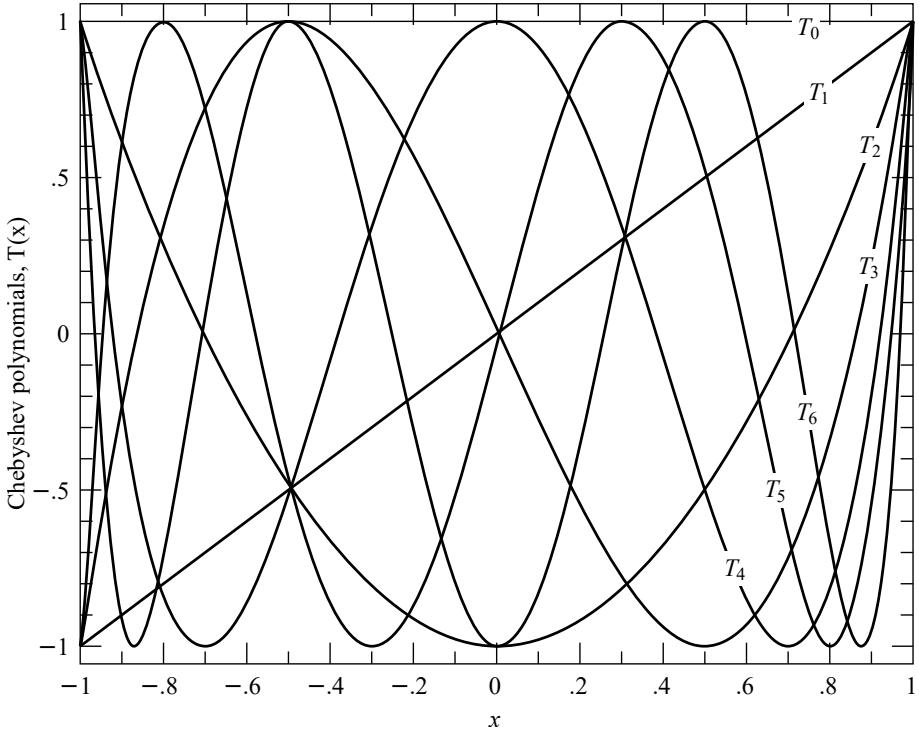
$$S_\lambda^*(x, y) = \frac{S_\lambda(x, y) - B(x, y) - (\frac{t_S}{t_D})D(x, y)}{F_\lambda^n(x, y)}. \quad (2.10)$$

When multichannel CCDs are used, which consist of multiple CCDs or a CCD with multiple output amplifiers, additional calibrations, specifically cross-talk corrections and mosaicking, are required. Cross-talk noise refers to contamination that occurs during readout in one channel from another channel with a high signal and occurs because the signals can not be completely isolated from one another. Cross-talk corrections therefore account for this signal contamination between channels being read out at the same time (Freyhammer et al., 2001). Mosaicking is necessary for multichannel CCDs since the digitized signal read out from the detector has no reference of the physical location of the pixel it was detected at. Mosaicking, therefore, correctly orients the data acquired from a multichannel detector so that a single correctly oriented image is produced.

## Wavelength Calibration

Finally, since the dispersion element breaks the incident light into its constituent wavelengths non-linearly (§ 2.1.7), the relation between the pixel on a detector and the wavelength of the light incident on it is unknown. Ideally, the spectrometer’s optics would be

<sup>2</sup>NeAr plot sourced from <https://astronomers.salt.ac.za/data/salt-longslit-line-atlas/>



**Figure 2.8:** The first seven Chebyshev polynomials ( $T_0$  through  $T_6$ ) as defined by Equation 2.12 over the region  $[-1, 1]$  for which they are orthogonal. Plot adapted from (Press et al., 2007) (2023)<sup>3</sup>

modelled to produce a reliable pixel to wavelength calibration (see E.g. Liu and Hennelly, 2022), but this becomes increasingly more difficult for spectrometers with complex, non-sedentary, optical paths. Alternatively, a source with well-defined spectral features, with said features evenly populating the wavelength region of interest, such as in Figure 2.7 may be observed. The observed frame is commonly referred to as an ‘arc’ frame, after the arc-lamps used to acquire the spectra, and should be observed alongside the science frames over the course of an observation run. It is important that the arc frame is observed at the same observing conditions and parameters as the science frames since the optical path will vary over the course of an observing run and for different observing parameters, invalidating previously acquired arc frames.

The wavelength calibrations then consist of defining a two-dimensional pixel-to-wavelength conversion function from the arc frame which may later be applied to calibrate the science frames. The two most common approximations for wavelength calibrations are the Chebyshev and Legendre polynomial approximations.

**Chebyshev Polynomials** The Chebyshev polynomials are defined explicitly as:

$$T_n(x) = \cos(n \cos^{-1}(x)) , \text{ or recursively as} \quad (2.11)$$

$$\begin{aligned} T_0(x) &= 1 , \\ T_1(x) &= x , \text{ and} \end{aligned} \quad (2.12)$$

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) , \text{ for } n \geq 1 ,$$

---

<sup>3</sup>Excellent resources on Chebyshev and Legendre polynomials are available digitally at [www.numerical.recipes/book](http://www.numerical.recipes/book).

where  $T$  is a Chebyshev polynomial of order  $n$ .<sup>4</sup> An important property of Chebyshev polynomials is that they are orthogonal polynomials. This means that the inner product of any two differing Chebyshev polynomials,  $T_i(x)$  and  $T_j(x)$ , over the range  $[-1, 1]$  is zero, as shown by:

$$\int_{-1}^1 T_i(x)T_j(x) \frac{1}{\sqrt{1-x^2}} dx = \begin{cases} 0, & i \neq j \\ \pi/2, & i = j \neq 0 \\ \pi, & i = j = 0 \end{cases}, \quad (2.13)$$

where  $1/\sqrt{1-x^2}$  is the weighting factor for Chebyshev polynomials. This property is important because it means that the coefficients in the Chebyshev polynomial expansion are independent of one another, allowing for a unique solution when approximating an unknown function (Arfken and Weber, 1999; Press et al., 2007).

An approximation, using Chebyshev polynomials, of an unknown wavelength calibration function is given by:

$$f(x) \approx \sum_{i=0}^N c_i T_i(u), \text{ or} \quad (2.14)$$

$$F(x, y) \approx \sum_{i=0}^N \sum_{j=0}^M c_{ij} T_i(u) T_j(v), \quad (2.15)$$

for a one- or a two-dimensional wavelength surface function, respectively. Here  $N$  and  $M$  are the desired  $x$  and  $y$  orders, and  $c_i$  and  $c_{ij}$  are the Chebyshev polynomial coefficients (Florinsky and Pankratov, 2015; Leng, 1997). Since the orthogonality property of the Chebyshev polynomials only holds true over the range  $[-1, 1]$ , the  $(x, y) \in ([0, a], [0, b])$  pixel coordinates must be remapped to  $u, v \in [-1, 1]$  following the relation:

$$(u, v) = \frac{2(x, y) - a - b}{b - a}. \quad (2.16)$$

The Chebyshev polynomials are more suited for wavelength calibrations than standard polynomials since they are orthogonal and have minima and maxima located at  $[-1, 1]$ , as seen in Figure 2.8. This means that the Chebyshev approximation is exact when  $x = x_n$ , where  $x_n$  are the positions of the  $n - 1$   $x$ -intercepts of  $T_N(x)$ . These properties greatly minimize the error in the Chebyshev approximation, even at lower order approximations (Arfken and Weber, 1999).

**Legendre Polynomials** Similar to the Chebyshev polynomials, the Legendre polynomials may be defined explicitly as:

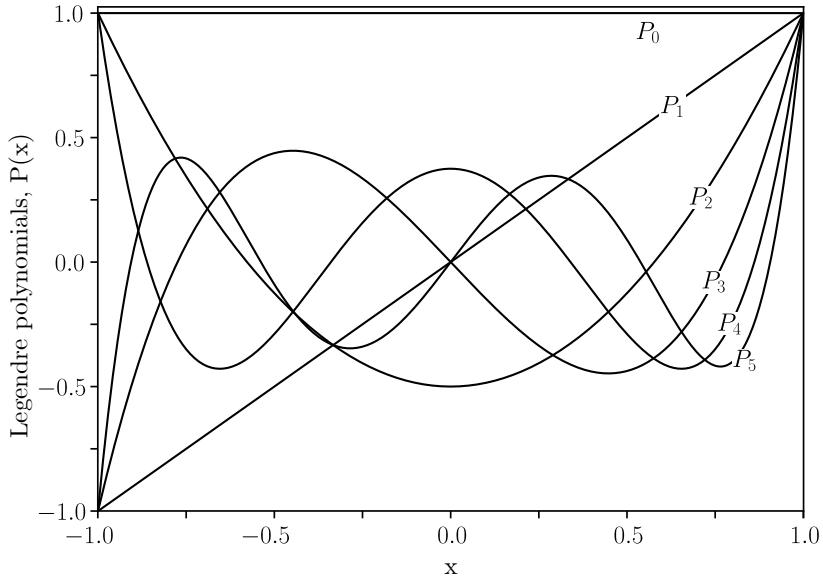
$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} (x^2 - 1)^n, \text{ or recursively as} \quad (2.17)$$

$$\begin{aligned} P_0(x) &= 1, \\ P_1(x) &= x, \text{ and} \end{aligned} \quad (2.18)$$

$$(n + 1)P_{n+1}(x) = (2n + 1)xP_n(x) - nP_{n-1}(x), \text{ for } n \geq 1,$$

---

<sup>4</sup>Chebyshev polynomials are denoted  $T$  as a hold-over from the alternate spelling of ‘Tchebycheff’.



**Figure 2.9:** The first six Legendre polynomials ( $P_0$  through  $P_5$ ) as defined by Equation 2.20 over the region  $[-1, 1]$  for which they are orthogonal. Plot adapted from Geek3, CC BY-SA 3.0, via Wikimedia Commons (2023).

where  $P$  is a Legendre polynomial of order  $n$ . Legendre polynomials also hold the property of orthogonality. This means that the inner product of any two differing Legendre polynomials,  $P_i(x)$  and  $P_j(x)$ , over the range  $[-1, 1]$  is zero, as shown by:

$$\int_{-1}^1 P_i(x) P_j(x) dx = \begin{cases} 0, & i \neq j \\ \frac{2}{2n+1}, & i = j \end{cases}, \quad (2.19)$$

where a weight of 1 is the weighting factor for Legendre polynomials (Dahlquist and Björck, 2003; Press et al., 2007).

An approximation, using Legendre polynomials, of an unknown wavelength calibration function is given by:

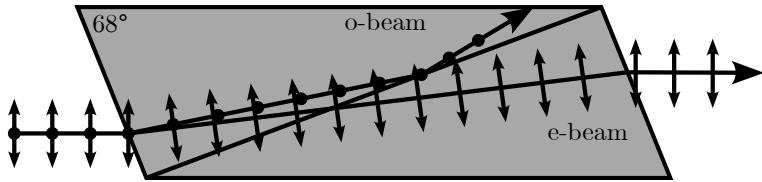
$$f(x) \approx \sum_{n=0}^N a_n P_n(u), \text{ or} \quad (2.20)$$

$$F(x, y) \approx \sum_{i=0}^N \sum_{j=0}^M a_{ij} P_i(u) P_j(v), \quad (2.21)$$

for a one-dimensional wavelength function or a two-dimensional surface function, respectively. Here  $N$  and  $M$  are the desired  $x$  and  $y$  orders,  $u$  and  $v$  are the same mapping variable as in Equation 2.16, and  $a_{ij}$  are the Legendre polynomial coefficients.

Legendre polynomials benefit from having the orthogonality condition with no weight necessary ( $w = 1$ ) which makes their coefficients computationally easier to compute but increases the error in a Legendre approximation when compared to that of the error in a Chebyshev approximation for functions of the same order,  $N$  (Ismail, 2005).

Regardless of which method of polynomial approximation is chosen, the polynomials are fit by varying the relevant coefficients using the least squares method. The resultant



**Figure 2.10:** Diagram of a Nicol prism for incident non-polarized light. Diagram adapted from Fred the Oyster, CC BY-SA 4.0, via Wikimedia Commons (2023).

minimized function may then be used to convert the science frames from an ( $x$ -pixel,  $y$ -pixel) coordinate system to a ( $\lambda$ ,  $y$ -pixel) coordinate system.

## 2.2 Polarimetry

Both Huygens and Newton came to the conclusion that light demonstrates transversal properties (Huygens, 1690; Newton and Innys, 1730), which was later further investigated and coined as ‘polarization’ by Malus (Malus, 1809). Malus also investigated the polarization effects of multiple materials including some of which were birefringent, such as optical calcite, which he referred to as Iceland spar after Bartholinus’ investigations of the material (Bartholinus, 1670).

Fresnel built on Malus’ work showing that two beams of light, polarized at a right angle to one another, do not interfere, conclusively proving that light is transversal in nature, opposing the widely accepted longitudinal nature of light due to the prevalent belief in the ether. He later went on to correctly describe how polarized light is reflected and refracted at the surface of optical dielectric interfaces, without knowledge of the electromagnetic nature of light. Fresnel’s equations for the reflectance and transmittance,  $R$  and  $T$ , are defined as:

$$\begin{aligned} R_s &= \left| \frac{Z_2 \cos \theta_i - Z_1 \cos \theta_t}{Z_2 \cos \theta_i + Z_1 \cos \theta_t} \right|^2, \\ R_p &= \left| \frac{Z_2 \cos \theta_t - Z_1 \cos \theta_i}{Z_2 \cos \theta_t + Z_1 \cos \theta_i} \right|^2, \\ T_s &= 1 - R_s, \text{ and} \\ T_p &= 1 - R_p, \end{aligned} \quad (2.22)$$

where  $s$  and  $p$  are the two polarized components of light perpendicular to one another,  $Z_1$  and  $Z_2$  are the impedance of the two media, and  $\theta_i$ ,  $\theta_t$ , and  $\theta_r$  are the angles of incidence, transmission, and reflection, respectively (Fresnel, 1870).

Nicol was the first to create a polarizer, aptly named the Nicol prism, where the incident light is split into its two perpendicular polarization components, namely the ordinary and extraordinary beams. Faraday discovered the phenomenon where the polarization plane of light is rotated when under the influence of a magnetic field, known as the Faraday effect. Brewster calculated the angle of incidence,  $\theta_B = \arctan n_2/n_1$ , at which incident polarized light is perfectly transmitted through a transparent surface, with refractive indexes of  $n_1$  and  $n_2$ , while non-polarized incident light is perfectly polarized when reflected and partially polarized when refracted.

Stokes’ work created the first consistent description of polarization and gave us the

Stokes parameters which describe an operational approach to measuring polarization (discussed further in § 2.2.1) (Stokes, 1852). Hale was the first to apply polarization to astronomical observations, using a Fresnel rhomb and Nicol prism as a quarter-wave plate and polarizer, respectively (Hale, 1908, 1979). Wollaston also created a prism, similarly named the Wollaston prism, which allowed simultaneous observation of the ordinary and extraordinary beams due to the smaller deviation angle (Wollaston, 1802). Finally, Chandrasekhar's work furthered our understanding of astrophysical polarimetry by explaining the origin of polarization observed in starlight as well as mathematically modeling the polarization of rotating stars, which came to be named Chandrasekhar polarization (Chandrasekhar, 1950).

### 2.2.1 Polarization

Maxwell's equations for an electromagnetic field propagating through a vacuum are given as:

$$\begin{aligned}\nabla \cdot \mathbf{E} &= 0, \\ \nabla \cdot \mathbf{B} &= 0, \\ \nabla \times \mathbf{E} &= -\frac{1}{c} \frac{\partial \mathbf{B}}{\partial t}, \text{ and} \\ \nabla \times \mathbf{B} &= \frac{1}{c} \frac{\partial \mathbf{E}}{\partial t},\end{aligned}\tag{2.23}$$

where  $\mathbf{E}$  and  $\mathbf{B}$  are the electric and magnetic field vectors, and  $c$  is the speed of light. In a right-handed  $(x, y, z)$  coordinate system, a non-trivial solution of an electromagnetic wave following Maxwell's Equations propagating along the  $z$ -axis, towards a hypothetical observer, is described by:

$$\begin{aligned}\mathbf{E} &= E_x \cos(kz - \omega t + \Phi_x) \hat{x} + E_y \cos(kz - \omega t + \Phi_y) \hat{y}, \text{ and} \\ \mathbf{B} &= \frac{1}{c} E_y \cos(kz - \omega t + \Phi_y) \hat{x} + \frac{1}{c} E_x \cos(kz - \omega t + \Phi_x) \hat{y},\end{aligned}\tag{2.24}$$

where  $E_x$ ,  $E_y$ ,  $\Phi_x$ , and  $\Phi_y$  are all parameters describing the amplitude and phase of the electric field vector in the  $(x, y)$  plane, and with the magnetic field vector proportional and perpendicular to the electric field vector (Griffiths, 2005).

Considering only the electric field component and rewriting Equation 2.24 using complex values allows us to simplify the form of the solution to:

$$\mathbf{E} = \Re(\mathbf{E}_0 e^{-i\omega t}),\tag{2.25}$$

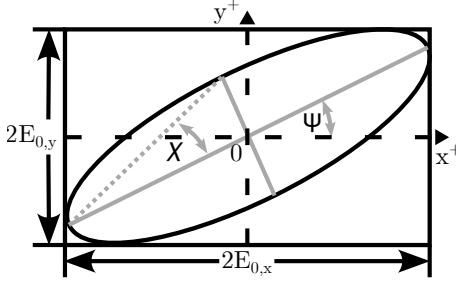
where we only consider the real part of the equation, and where  $\mathbf{E}_0$  is defined as:

$$\mathbf{E}_0 = E_x e^{i\Phi_x} \hat{x} + E_y e^{i\Phi_y} \hat{y},\tag{2.26}$$

and is referred to as the polarization vector since it neatly contains the parameters responsible for the polarization properties (Degl'Innocenti, 2014).

For an electric field vector with oscillations in some combination of the  $x$  and  $y$  axes, the tip of the vector sweeps out an ellipse, as depicted in Figure 2.11. This ellipse is referred to as the polarization ellipse and has the form:

$$\left( \frac{\mathbf{E}_x}{\mathbf{E}_{0,x}} \right)^2 + \left( \frac{\mathbf{E}_y}{\mathbf{E}_{0,y}} \right)^2 - \frac{2\mathbf{E}_x \mathbf{E}_y}{\mathbf{E}_{0,x} \mathbf{E}_{0,y}} \cos \Phi = \sin^2 \Phi,\tag{2.27}$$



**Figure 2.11:** The polarization ellipse for an electric field vector propagating through free space. Diagram adapted from Inductiveload, PDM 1.0, via Wikimedia Commons (2023).

where  $\Phi = \Phi_x - \Phi_y$  is the phase difference between the  $x$  and  $y$  phase parameters. The degree of polarization for the polarization ellipse is related to the eccentricity of the ellipse and the angle at which it is rotated relates to the polarization angle. Since  $\mathbf{E}_{0,x}$ ,  $\mathbf{E}_{0,y}$ ,  $\Phi_x$ , and  $\Phi_y$  describe the wave, the polarization ellipse that results from these parameters is fixed as the wave continues to propagate.

Since observations consist of images taken over a desired exposure time, time averaging of Equation 2.27 over the exposure time is necessary. Given the periodical nature and high frequencies of the fields, the time averaging may be found over a single oscillation using:

$$\langle \mathbf{E}_i \mathbf{E}_j \rangle = \lim_{dt \rightarrow \infty} \frac{1}{T} \int_0^T \mathbf{E}_i \mathbf{E}_j dt, \quad \text{for } i, j \in (x, y), \quad (2.28)$$

where  $T$  is the total averaging time over the electric field vectors  $\mathbf{E}_i$  and  $\mathbf{E}_j$  (Collett, 2005). Applying the time averaging to Equation 2.27 and simplifying results in:

$$(E_{0x}^2 + E_{0y}^2)^2 - (E_{0x}^2 - E_{0y}^2)^2 - (2E_x E_y \cos \Phi)^2 = (2E_x E_y \sin \Phi)^2. \quad (2.29)$$

The expressions inside the parentheses can be found through observation and may also be represented as:

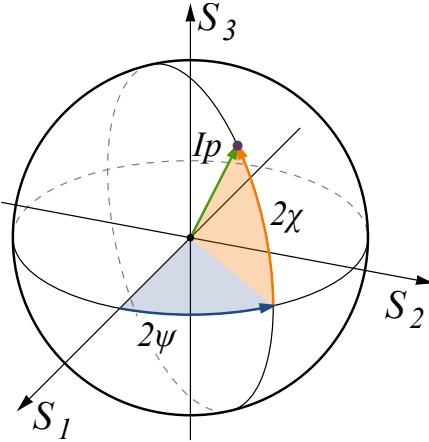
$$\mathbf{S} = \begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{pmatrix} = \begin{pmatrix} I \\ Q \\ U \\ V \end{pmatrix} = \begin{pmatrix} E_{0x}^2 + E_{0y}^2 \\ E_{0x}^2 - E_{0y}^2 \\ 2E_{0x} E_{0y} \cos \Phi \\ 2E_{0x} E_{0y} \sin \Phi \end{pmatrix}, \quad (2.30)$$

where  $S_0$  to  $S_3$  are referred to as the Stokes (polarization) parameters. The parameters describe the:  $S_0$ , total intensity (often normalized to 1);  $S_1$ , ratio of the Linear Horizontally Polarized (LHP) to Linear Vertically Polarized (LVP) light;  $S_2$ , ratio of the Linear  $+45^\circ$  Polarized ( $L+45^\circ$ ) to Linear  $-45^\circ$  Polarized ( $L-45^\circ$ ) light; and  $S_3$ , ratio of the Right Circularly Polarized (RCP) (clockwise) to Left Circularly Polarized (LCP) (counter-clockwise) light. When the intensity is normalized, the Stokes parameters range from 1 to  $-1$ , based on the dominating component of the parameter (Chandrasekhar, 1950; Stokes, 1852).

From Equation 2.29 and 2.30, the polarization parameters are related by:

$$I^2 = Q^2 + U^2 + V^2, \quad (2.31)$$

for entirely polarized light. Only beams of completely polarized light could be accounted for before Stokes' work on polarization. Using the Stokes parameters, we can now account



**Figure 2.12:** The Poincaré sphere describing the polarization properties of a wave-packet propagating through free space. Diagram adapted from Inductiveload, PDM 1.0, via Wikimedia Commons (2023).

for partially polarized light such that:

$$I^2 \geq Q^2 + U^2 + V^2, \quad (2.32)$$

where  $I$ ,  $Q$ ,  $U$ , and  $V$  are the normalized polarization parameters, often symbolized as

$$\bar{Q} = \frac{Q}{I}, \quad \bar{U} = \frac{U}{I}, \quad \text{and} \quad \bar{V} = \frac{V}{I}. \quad (2.33)$$

Similar to the polarization ellipse, the Stokes parameters may be depicted using the Poincaré sphere in spherical coordinates  $(IP, 2\Psi, 2\chi)$ , such that:

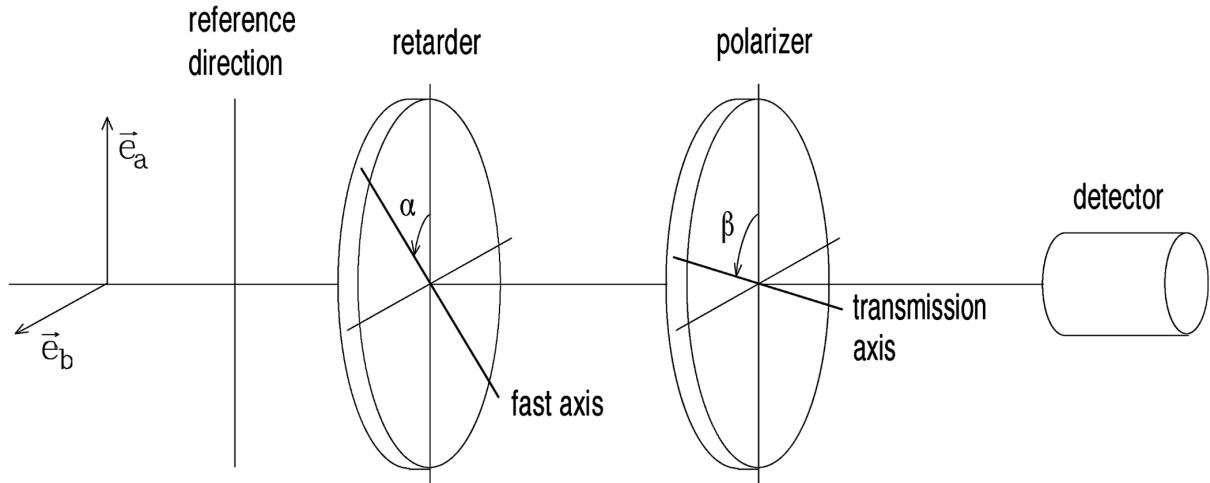
$$\begin{aligned} I &= S_0, \\ P &= \frac{\sqrt{S_1^2 + S_2^2 + S_3^2}}{S_0}, \text{ for } 0 \leq P \leq 1, \\ 2\Psi &= \arctan \frac{S_3}{\sqrt{S_1^2 + S_2^2}}, \text{ and} \\ 2\chi &= \arctan \frac{S_2}{S_1}, \end{aligned} \quad (2.34)$$

where  $I$  denotes the total intensity,  $P$  denotes the degree of polarization, or the ratio of polarized to non-polarized light in the wave-packet,  $\chi$  denotes the polarization angle, and  $\Psi$  denotes the ellipticity angle of the polarization ellipse.

## 2.2.2 Polarization Measurement

Except for polarimetry in the radio-wavelength regime, the polarization of a beam can not be directly measured. The polarization properties may, however, be recovered from the beam through the manipulation of the four parameters given in Equation 2.24. This so-called manipulation is achieved by passing the beam through optical elements which vary the beam for differing amplitudes and phases. These matrix operations may be represented by their corresponding Mueller matrices.

For ideal components, the resultant beam  $\mathbf{S}'$  after passing through an optical element is given by  $\mathbf{S}' = \mathbf{MS}$ , where  $\mathbf{S}$  is the beam incident on the optical element and  $\mathbf{M}$



**Figure 2.13:** A diagram of an ideal polarimeter. Diagram adapted from Degl'Innocenti and Landolfi (2004).

represents the  $4 \times 4$  Mueller matrix representing the optical element. Mueller matrices are especially useful when dealing with paths through optical elements as they observe the ‘train’ property (Priebe, 1969). This means that an incoming beam  $\mathbf{S}$  passing, in order, through elements with known Mueller matrices ( $\mathbf{M}_0, \dots, \mathbf{M}_N$ ) results in an outgoing beam  $\mathbf{S}'$  such that:

$$\mathbf{S}' = \mathbf{M}_N \dots \mathbf{M}_0 \mathbf{S}. \quad (2.35)$$

Some Mueller Matrices are given below with angles related to those in Figure 2.13, measured counter-clockwise in a right-handed coordinate system.

**General Rotation** The Mueller matrix for coordinate space rotations about the origin by an angle  $\theta$ ,

$$\mathbf{R}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 2\theta & \sin 2\theta & 0 \\ 0 & -\sin 2\theta & \cos 2\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2.36)$$

**General Linear Retardance** The Mueller matrix for retardance where  $\alpha$  is the angle between the incoming vector and fast axis, and  $\delta$  is the retardance introduced by the retarder,

$$\mathbf{W}(\alpha, \delta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos^2 2\alpha + \sin^2 2\alpha \cos \delta & \cos 2\alpha \sin 2\alpha(1 - \cos \delta) & \sin 2\alpha \sin \delta \\ 0 & \cos 2\alpha \sin 2\alpha(1 - \cos \delta) & \cos^2 2\alpha \cos \delta + \sin^2 2\alpha & -\cos 2\alpha \sin \delta \\ 0 & -\sin 2\alpha \sin \delta & \cos 2\alpha \sin \delta & \cos \delta \end{bmatrix}. \quad (2.37)$$

The retarder is often referred to by this retardance, e.g. if the retardance is  $\delta = \pi$  or  $\pi/2$ , the retarder is referred to as a half- or quarter-wave plate, respectively.

**General Linear Polarization** The Mueller matrix for linear polarization where  $\beta$  is the angle between the incoming vector and transmission axis,

$$\mathbf{P}(\beta) = \frac{1}{2} \begin{bmatrix} 1 & \cos 2\beta & \sin 2\beta & 0 \\ \cos 2\beta & \cos^2 2\beta & \cos 2\beta \sin 2\beta & 0 \\ \sin 2\beta & \sin 2\beta \cos 2\beta & \sin^2 2\beta & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \quad (2.38)$$

These matrices in combination with Equation 2.35 allow us to describe how the incoming Stokes parameters would change when passing through the various optical elements. For a setup similar to Figure 2.13, the detected Stokes parameters can be described by:

$$\begin{aligned} S'(\alpha, \beta, \gamma) \propto \frac{1}{2} \{ & I + [Q \cos 2\alpha + U \sin 2\alpha] \cos(2\beta - 2\alpha) \\ & - [Q \sin 2\alpha + U \cos 2\alpha] \sin(2\beta - 2\alpha) \cos \gamma \\ & + V \sin(2\beta - 2\alpha) \sin \gamma \}, \end{aligned} \quad (2.39)$$

where the retardance angle,  $\alpha$ , polarization angle,  $\beta$ , for a wave plate with a relative phase difference,  $\gamma$ , may be varied to acquire a system of equations that can be solved to retrieve the Stokes polarization parameters (Bagnulo et al., 2009).

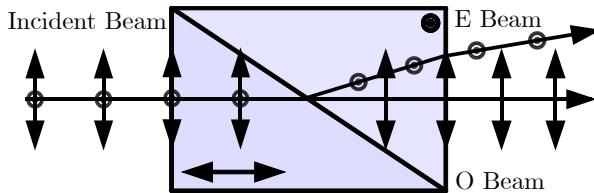
Several or more frames taken under differing configurations may be used to reduce a system of equations to extract all four Stokes polarization parameters, but it is possible to extract the  $I$ ,  $Q$  and  $U$  polarization parameters using only four frames, or two dual-beam frames, for well-chosen configurations and assuming ideal components. This ideal configuration varies the retarder angle such that  $\Delta\alpha = \pi/8$  while keeping the polarizer stationary. More frames for additional retarder angles are advisable and often necessary, however, as they correct for any differences in sensitivity, such as may arise in a polarized flat field and which is further discussed in § 2.2.3 (Patat and Romaniello, 2006).

From Equation 2.39 we see that the linear retarder element is the driving element of a polarizer as the first three Stokes parameters ( $S_{0-2}$ , or  $I$ ,  $Q$ , and  $U$ ) may be found by changing only the angle of retardance,  $\alpha$ .

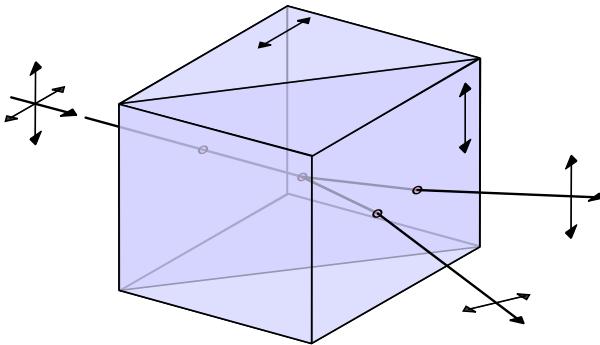
**Wave Plates** Wave plates, also commonly referred to as retarders, are generally made from optically transparent birefringent crystals. A wave plate has a fast and slow axis, which are perpendicular to one another and both perpendicular to an incident beam. Due to the birefringence of the wave plate medium, the phase velocity of the beam polarized parallel to the fast axis, namely the extraordinary beam, slightly increases while that of the beam polarized parallel to the slow axis, namely the ordinary beam, remains unaffected. This difference in the perpendicular component's phase velocities introduces a relative phase difference between the two beams,  $\gamma$ , which is given by:

$$\gamma = \frac{2\pi\Delta n L}{\lambda_0} \quad (2.40)$$

where  $\Delta n$  and  $L$  refer to the birefringence and thickness of the wave plate medium, respectively, and  $\lambda_0$  refers to the vacuum wavelength of the beam (Hecht, 2017).



**Figure 2.14:** Diagram of a Rochon prism. Included in the diagram are the optical axes of the differing sections of the birefringent material as well as polarizing directions of the incident beam, denoted using the  $\leftrightarrow$  and  $\odot$  symbols, for the  $O$ - and  $E$ -beams, respectively. Figure adapted from ChrisHedgesUK, CC BY-SA 3.0, via Wikimedia Commons (2023).



**Figure 2.15:** Diagram of a Wollaston prism. Included in the diagram are the optical axes of the differing sections of the birefringent material as well as polarizing directions of the incident beam, denoted using the  $\leftrightarrow$  and  $\ddagger$  symbols, for the  $O$ - and  $E$ -beams, respectively. Diagram adapted from fgalore, CC BY-SA 3.0, via Wikimedia Commons (2023).

This relative phase difference determines the name of the wave plate, such that the  $\gamma = m(\pi/2)$  and  $\gamma = m(\pi/4)$  phase differences, for  $m \in \mathbb{Z}^+$ , refer to the half- and quarter-wave plates (which are the most common wave plate phases), respectively. Phase differences with an integer multiple of one another relate to the same phase difference and are referred to as multiple-order wave plates, while wave plates with a phase difference less than an integer multiple are referred to as zero-order wave plates. Several multiple-order wave plates can be combined by alternatively aligning the fast axis of one to the slow axis of another to create a compound zero-order wave plate (Hale and Day, 1988).

**Polarizers** Polarizers are typically made from two prisms, of a birefringent material, cemented together with an optically transparent adhesive. The actual effect of separating the perpendicular polarization components is achieved using varying effects, namely through:

- absorption of one of the polarized components, such as in Polaroid polarizing filters,
- total internal reflection of a single polarized component, such as in a Nicol prism (Figure 2.10),
- Refraction of a single polarized component, such as in a Rochon prism (Figure 2.14), or
- Refraction of both polarization components in differing directions, such as in a Wollaston prism (Figure 2.15).

**Wollaston Prisms** The Wollaston prism consists of two right-angle prisms consisting of a birefringent monoaxial material, cemented together with an optically transparent adhesive along their hypotenuses with their optical axes orthogonal, as seen in Figure 2.15. The Wollaston prism is a common optical polarizing element in astrophysical polarimetry

which separates an incident beam into two linearly polarized *O*- and *E*-beams, orthogonal to one another, and deviated from their common axis equally. The deviation angle of the polarized beams is determined by the wedge angle which is defined as the angle from the common hypotenuse to that of the outer transmission face of either prism.

Wollaston prisms benefit over simpler elements (such as those listed in the polarizer paragraph) since a single frame allows for the observation of both orthogonal polarization components. This halves the observational time required to collect enough data to calculate the Stokes parameters, at the cost of an increase in calibration and reduction difficulty (Simon, 1986).

### 2.2.3 Polarimetric Calibrations

The raw science images acquired during polarimetric observations contain a combination of useful science data as well as noise, similar to § 2.1.8. Corrections and calibrations related to the detector remain unchanged from those described in § 2.1.8, while those related to correcting for the optical elements relate to corrections for spurious polarization effects.

#### Flat Fielding

Once the CCD calibrations have been completed, the polarization intrinsic to the optical elements needs to be accounted for such that the pixel-to-pixel response is made uniform. Flat-fielding is, once again, used to correct for this. The flats taken for polarimetry, however, introduce an additional challenge as the targets for conventional flats are polarized, such as twilight and dome flats which are polarized by light scattering in the atmosphere and the reflective surface of the dome, respectively.

If no unpolarized flat images can be taken for flat field calibrations then, when possible due to the polarimeter design, the wave plate may be constantly rotated to act as a depolarizing element; this is effective so long as the wave plate rotation period is much faster than the flat's exposure time. Alternatively, polarized flats may be taken at the same set of half-wave plate angles used for science observations and averaged together to achieve a similar depolarizing effect.

Observing additional ‘redundant’ exposures for the science and flat images increases the depolarizing effect up to the maximum of 16 half-wave plate positions, where exposures with a half-wave plate angle differing by  $\pi/4$  from another are considered redundant due to the *O*- and *E*-beams swapping between the related exposures.

Increasing the amount of redundant observations proportionally increases the time needed to observe all the exposures, which in turn introduces time-dependent effects such as fringing or intensity variations of the flat source. As such, a middle ground must be found for the amount of redundant frames observed. (Patat and Romanielo, 2006; Peinado et al., 2010).

### Dual-Beam Extraction and Alignment

After calibrations for the CCD and light path are accounted for, the  $O$ - and  $E$ -beams can be extracted and further reduced. The extraction depends heavily on the layout of the polarimeter but often a simple cropping of the differing sections is enough to separate the two images.

After extracting the  $O$ - and  $E$ -beams for a specific half-wave plate angle, the images need to be aligned such that the sources present in them overlap. The Wollaston prism needs to be corrected for as it introduces a beam deviation which differs across both images. The aligning of the  $O$ - and  $E$ -beams is crucial as the comparison of the dual images is what allows for the calculation of the polarization properties.

### Sky Subtraction

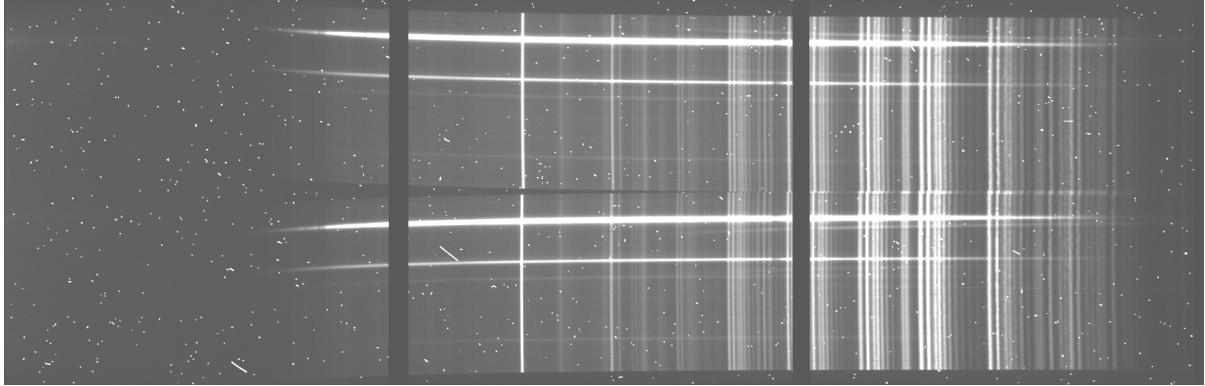
The polarization introduced by the sky introduces a difference in the intensity of the background sky and needs to be removed as it will influence the polarization results of the target source. Thankfully, the background polarization is an additive type of noise and may be subtracted out across the frames. This subtraction is done independently for both beams in a frame and for each frame since the background intensity of all observed polarimetric beams will differ based on the observational parameters.

## 2.3 Spectropolarimetry

As the name suggests, spectropolarimetry is the measurement of the polarization of light for a chosen spectral range and provides polarimetric results as a function of wavelength. As spectropolarimetry is so closely reliant on both spectroscopy and polarimetry, advancements in spectropolarimeters have always been gated by the advancements of spectrometers and polarimeters (as described in § 2.1 and § 2.2).

The most notable historical contributions of spectropolarimetry are those of spectropolarimetric studies instead of instrumental developments. Spectropolarimetry provides further insights into a materials physical structure, chemical composition, and magnetic field, allowing spectropolarimetry to be useful across multiple disciplines. In astronomy in particular, spectropolarimetry has been used to study the magnetic field, chemical composition, and underlying structure and emission processes of multiple types of celestial objects (see for example Antonucci and Miller, 1985; Donati et al., 1997; Wang and Wheeler, 2008).

Along with common points of consideration when developing any instrumentation for observational astronomy, such as resolution and sensitivity, spectropolarimeters need also consider the spectral response of the polarimetric components as well as the polarization response of the spectroscopic components as both are simultaneously in the light-path during observations and have noticeable affects on one another. Time is another constraint for spectropolarimetry as the incident light is separated both by wavelength and by polarization states. This division of the incident light results in increased exposure times for both target observations and observations necessary for calibrations.



**Figure 2.16:** A spectropolarimetric target exposure as observed by the SALT RSS in spectropolarimetry mode.

Figure 2.16 illustrates a typical science image taken with a spectropolarimeter. The image contains the  $O$ - and  $E$ -beams which are both dispersed into their spectra. Spectropolarimetric results are acquired from measurements and calibrations of these images alongside any necessary calibration images.

### 2.3.1 Spectropolarimetric Measurement

The derived relations given in § 2.2.1, such as the Stokes parameters, describe polarization in general and are valid for both polarimetry and spectropolarimetry. Due to the time averaging of the observed light (Equation 2.28), any minor temporal variation, partial polarization, or monochromatic nature of the spectropolarimetric polarization parameters are accounted for.

For linear spectropolarimetry using a dual-beam polarizing element, an exposure measures the  $O$ - and  $E$ -beam wavelength dependent intensities,  $f_{O,i}(\lambda)$  and  $f_{E,i}(\lambda)$ , for a given wave plate angle  $\theta_i$  at angle  $i$ . These intensities thus relate to the wavelength dependent Stokes parameters as:

$$\begin{aligned} f_{O,i}(\lambda) &= \frac{1}{2}[I(\lambda) + Q(\lambda) \cos(4\theta_i) + U(\lambda) \sin(4\theta_i)], \text{ and} \\ f_{E,i}(\lambda) &= \frac{1}{2}[I(\lambda) - Q(\lambda) \cos(4\theta_i) - U(\lambda) \sin(4\theta_i)]. \end{aligned} \quad (2.41)$$

At least four linear equations are required to solve for three variables in a system of linear equations and thus at least two exposures must be taken to solve for the linear ( $I(\lambda)$ ,  $Q(\lambda)$ , and  $U(\lambda)$ ) polarization parameters (Degl’Innocenti et al., 2006; Keller, 2002).

The first Stokes parameter,  $I(\lambda)$ , may be recovered for each dual-beam exposure using

$$I_i(\lambda) = f_{O,i}(\lambda) + f_{E,i}(\lambda). \quad (2.42)$$

By calculating the  $I_i(\lambda)$  Stokes parameter for each wave plate position  $i$ , the variation of the target over the course of observation may be corrected for, resulting in the  $I(\lambda)$  Stokes parameter.

Next, the  $Q(\lambda)$  and  $U(\lambda)$  Stokes parameters are found by first defining the normalized

difference in relative intensities,  $F_i(\lambda)$ , as:

$$F_i(\lambda) \equiv \frac{f_{O,i}(\lambda) - f_{E,i}(\lambda)}{f_{O,i}(\lambda) + f_{E,i}(\lambda)}, \quad (2.43)$$

which allows Equation 2.41 to be written, as

$$F_i(\lambda) = \bar{Q}(\lambda) \cos(4\theta_i) + \bar{U}(\lambda) \sin(4\theta_i) = P \cos(4\theta_i - 2\chi), \quad (2.44)$$

in terms of the normalized Stokes parameters, or, alternatively, the degree of polarization,  $P$ , and polarization angle,  $\chi$  (as described in Equation 2.33 and 2.34).

The optimal change in wave plate angle is  $\Delta\theta_i = \pi/8$  as it allows the normalized Stokes polarization parameters to be calculated as:

$$\begin{aligned} \bar{Q}(\lambda) &= \frac{2}{N} \sum_{i=0}^{N-1} F_i(\lambda) \cos\left(\frac{\pi}{2}i\right), \text{ and} \\ \bar{U}(\lambda) &= \frac{2}{N} \sum_{i=0}^{N-1} F_i(\lambda) \sin\left(\frac{\pi}{2}i\right), \end{aligned} \quad (2.45)$$

where  $N$  is the number of exposures taken, limited such that  $N \in [2, 16]$  (Patat and Romaniello, 2006).

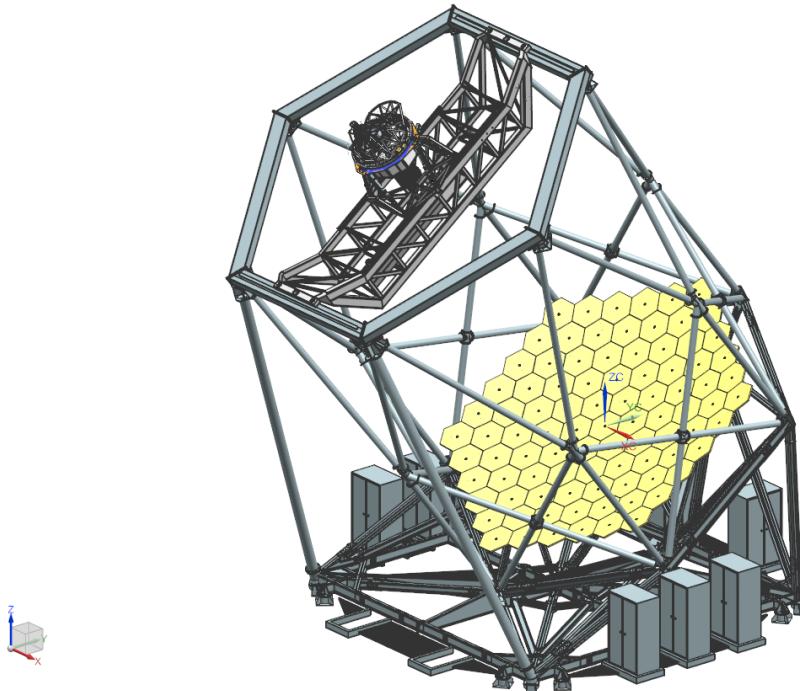
### 2.3.2 Spectropolarimetric Calibrations

Just as the elements of a spectropolarimeter are an amalgamation of both a spectrometer and polarimeter, it naturally follows that the calibrations necessary to reduce spectropolarimetric data are a combination of the calibrations needed for spectroscopy and polarimetry, discussed further in § 2.1.8 and § 2.2.3. Even though the spectrometer and polarimeter components both have an effect on an incident beam following the light-path through the spectropolarimeter, the calibration procedures for both methods remain mostly independent of one another and as such need not be repeated here.

Spectropolarimetric calibrations are, however, more involved when compared to the same calibrations for either spectroscopy or polarimetry. Minor deviations in the calibrations across both the spectra and the polarized beam compound, especially when dealing with the wavelength calibration, resulting in poor Signal-to-Noise Ratio (S/N)'s. Generally, more exposures over longer timespans are required to acquire enough redundancy and signal for the calculation of the Stokes parameters on top of the time necessary for calibrations to be completed. It should therefore be noted just how important the calibrations are when dealing with spectropolarimetry.

## 2.4 The Southern African Large Telescope

Southern African Large Telescope (SALT) is a 10 m class optical/near-infrared telescope situated at the South African Astronomical Observatory (SAAO) field station near Sutherland, South Africa (Burgh et al., 2003). The operational design was based on the Hobby-Eberly Telescope (HET) situated at McDonald Observatory, Texas, which limits



**Figure 2.17:** The tracker, supporting structure, and primary mirror of SALT. Figure adapted from the SALT call for proposals (2022).<sup>5</sup>

the pointing of the telescope’s primary mirror to a fixed elevation ( $37^\circ$  from zenith in the case of SALT) while still allowing for full azimuthal rotation (Ramsey et al., 1998). Both SALT and HET utilize a spherical primary mirror which is stationary during observations and a tracker housing most of the instrumentation that tracks the primary mirrors spherically shaped focal path. Figure 2.17 depicts SALT’s tracker (top left), supporting structure, and primary mirror (bottom right).

### 2.4.1 The Primary Mirror

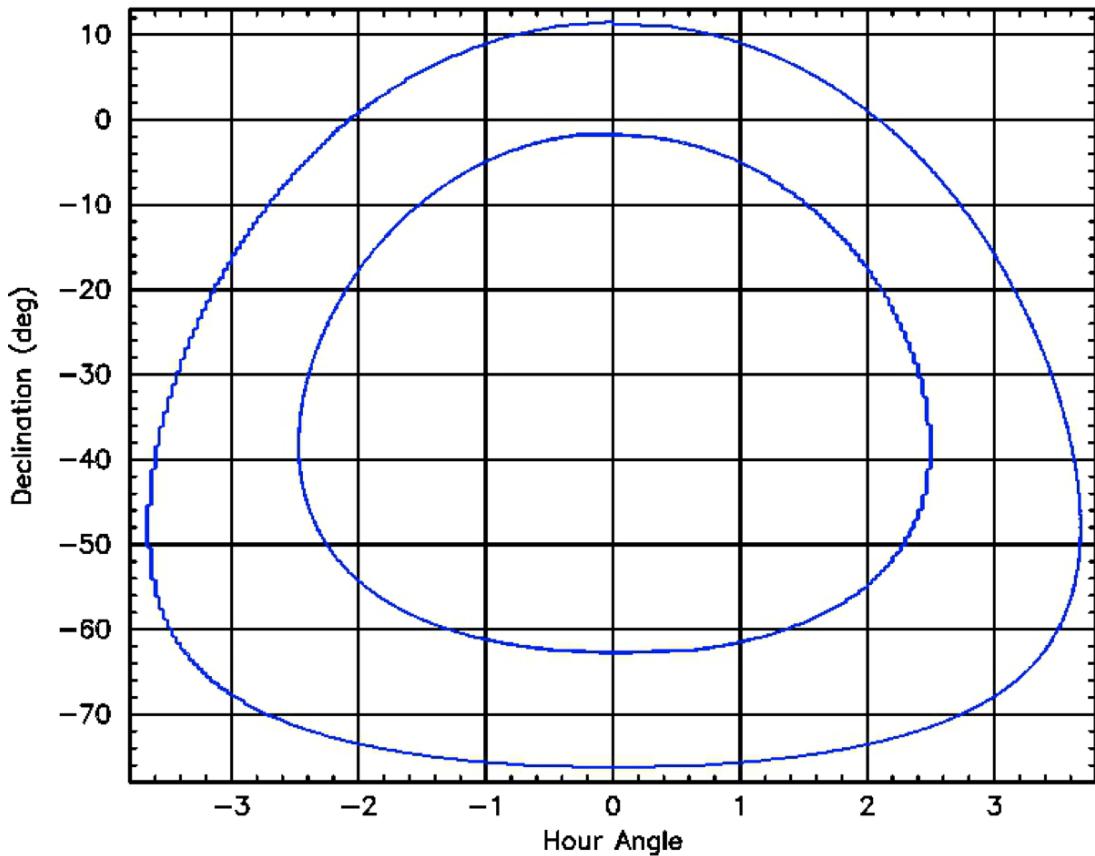
The primary mirror is composed of 91 individual 1 m hexagonal mirrors which together form an 11 m segmented spherical mirror. Each mirror segment can be adjusted by actuators allowing the individual mirrors to approximate a single monolithic spherical mirror. The fixed elevation means that SALT’s primary mirror has a fixed gravity vector allowing for a lighter, cost-effective supporting structure when compared to those of a more traditional altitude-azimuthal mount but with the trade-off that the control mechanism and tracking have increased complexity (Buckley et al., 2006).

### 2.4.2 Tracker and Tracking

During observations the primary mirror is stationary and the tracker tracks celestial objects across the sky by moving along the primary focus. The tracker is capable of 6 degrees of freedom with an accuracy of  $5 \mu\text{m}$  and is capable of tracking  $\pm 6^\circ$  from the optimal central track position. Targets at declinations from  $10.5^\circ$  to  $-75.3^\circ$ , as shown in Figure 2.18 are accessible during windows of opportunity. As the tracker moves along the track the effective collecting area varies and thus SALT has a varying effective diameter

---

<sup>5</sup>[http://pysalt.salt.ac.za/proposal\\_calls/current/ProposalCall.html](http://pysalt.salt.ac.za/proposal_calls/current/ProposalCall.html)



**Figure 2.18:** The visibility annulus of objects observable by SALT. Figure adapted from the SALT call for proposals (2013).<sup>6</sup>

of  $\sim 7$  m to 9 m when the tracker is furthest and closest to the optimal central position, respectively.

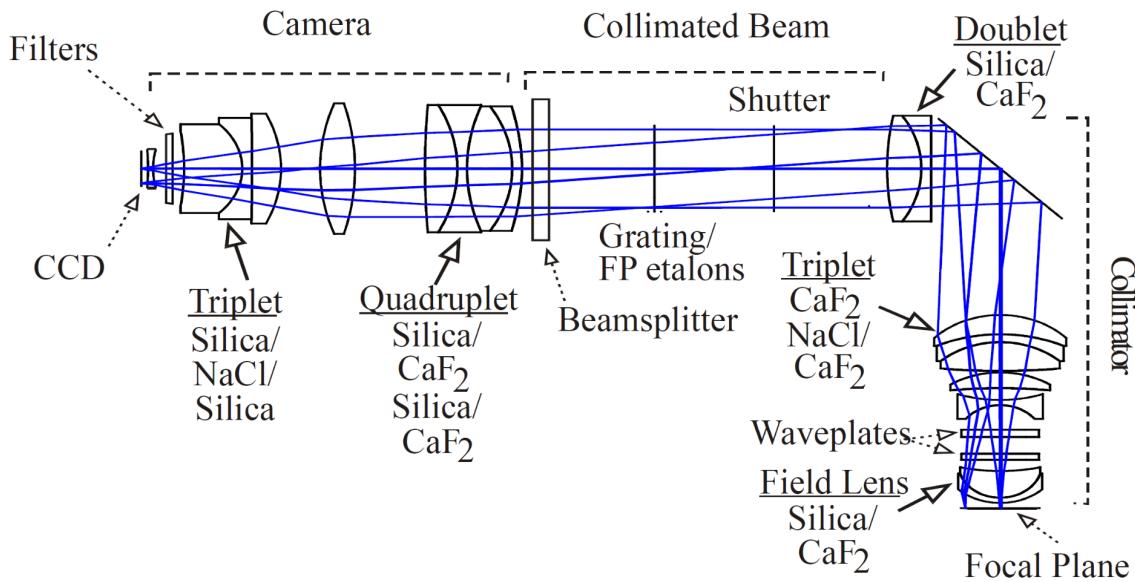
The tracker is equipped with a spherical aberration corrector (O'Donoghue, 2000), and an atmospheric dispersion compensator (O'Donoghue, 2002), which corrects for the spherical aberration caused by the geometry of the primary mirror and allows access to wavelengths as short as 3200 Å. These return a corrected flat focal plane with an 8' diameter field of view at prime focus on to the science instruments, with a 1' annulus around it used by the Tracker in a closed-loop guidance system.

### 2.4.3 SALT Instrumentation

SALT is equipped with the SALT Imaging Camera (SALTICAM) and the RSS science instruments onboard the tracker, and the High Resolution Spectrograph (HRS) and Near Infra-Red Washburn Labs Spectrograph (NIRWALS) science instruments which are fibre-fed from the tracker to their own climate controlled rooms. The RSS is currently the only instrument used for spectropolarimetry.

<sup>6</sup>[https://pysalt.salt.ac.za/proposal\\_calls/2013-2/](https://pysalt.salt.ac.za/proposal_calls/2013-2/)

<sup>7</sup>[https://pysalt.salt.ac.za/proposal\\_calls/current/ProposalCall.html](https://pysalt.salt.ac.za/proposal_calls/current/ProposalCall.html)



**Figure 2.19:** The optical path of the SALT RSS. Figure adapted from the SALT call for proposals (2023).<sup>7</sup>

## NIRWALS

The Near Infra-Red Washburn Labs Spectrograph (NIRWALS) is currently being commissioned and will have a wavelength coverage of 8000 to 17000 Å, providing medium resolution spectroscopy at  $R = 2000$  to  $5000$  over Near Infra-Red (NIR) wavelengths (Brink et al., 2022; Wolf et al., 2022). NIRWALS is fibre-fed from its integral field unit, containing 212 object fibers, along with a separate sky bundle, containing 36 fibers, housed in the SALT fibre instrument feed. It is ideally suited for studies of nearby galaxies.

## HRS

The High Resolution Spectrograph (HRS) echelle spectrograph was designed for high resolution spectroscopy at  $R = 37000$  -  $67000$  covering a wavelength range of 3700 - 8900 Å and consists of a dichroic beam splitter and two VPH gratings (Nordsieck et al., 2003). This instrument is capable of stellar atmospheric and radial velocity analysis.

## SALTICAM

The SALT Imaging Camera (SALTICAM) functions as the acquisition camera and simple science imager with various imaging modes, such as full-mode and slot-mode imaging, and supports low exposure times, down to 50 ms (O'Donoghue et al., 2006). This enables photometry of faint objects, especially at fast exposure times.

## RSS

The Robert Stobie Spectrograph (RSS) functions as the primary spectrograph on SALT and can operate in long-slit spectroscopy and spectropolarimetry modes, a narrowband imaging mode, and multi-object and high resolution spectroscopy modes (for an in-depth

Grating Name	Wavelength Coverage (Å)	Usable Angles (°)	Bandpass per tilt (Å)	Resolving Power (1.25'' slit)
PG0300 <sup>8</sup>	3700 – 9000		3900/4400	250 – 600
PG0700 <sup>8</sup>	3200 – 9000	3.0 – 7.5	4000 – 3200	400 – 1200
PG0900	3200 – 9000	12 – 20	~ 3000	600 – 2000
PG1300	3900 – 9000	19 – 32	~ 2000	1000 – 3200
PG1800	4500 – 9000	28.5 – 50	1500 – 1000	2000 – 5500
PG2300	3800 – 7000	30.5 – 50	1000 – 800	2200 – 5500
PG3000	3200 – 5400	32 – 50	800 – 600	2200 – 5500

**Table 2.1:** Gratings available for use with the RSS. Table adapted from the SALT call for proposals (2023).

discussion on operational modes see Kobulnicky et al., 2003, or the latest call for proposals).

**The Detector** The RSS detector consists of a mosaic of 3 CCD chips with a total pixel scale of  $0.1267''$  per unbinned pixel with varying readout times depending on the binning and readout mode. The mosaicking results in a characteristic double ‘gap’ in the frames and resultant spectra taken with the RSS, as seen in Figure 2.16.

**The Available Gratings** The RSS is equipped with a rotatable magazine of six VPH gratings, as listed in Table 2.1. Observations may be planned using simulator tools provided by SALT and are performed in the first order only. The RSS has a clear filter, as well as three Ultraviolet (UV) (with differing lower filtering ranges) and one blue order blocking filter available, used in conjunction with the various gratings to block out contamination from the second order.

**RSS Spectropolarimetry** Spectropolarimetry using the RSS is currently commissioned for long-slit linear spectropolarimetry, ( $I, Q, U$ ), where observations are taken following the waveplate pattern lists as in Table 2.2. Circular, ( $I, V$ ), and all-Stokes, ( $I, Q, U, V$ ), spectropolarimetry modes are in commissioning with observations including redundant half-wave plate pairs to be commissioned thereafter.<sup>9</sup>

---

<sup>8</sup>The PG0300 surface relief grating has been replaced with the PG0700 VPH grating as of November 2022 but has been included here as observations using the PG0300 are used in later sections.

<sup>9</sup>Commission status sighted from the latest ‘Polarimetry Observers Guide’ (2024).

Linear ( $^{\circ}$ )		Linear-Hi ( $^{\circ}$ )		Circular ( $^{\circ}$ )		Circular-Hi ( $^{\circ}$ )		All Stokes ( $^{\circ}$ )	
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$
0	-	0	-	0	45	0	45	0	0
45	-	45	-	0	-45	0	-45	45	0
22.5	-	22.5	-			22.5	-45	22.5	0
67.5	-	67.5	-			22.5	45	67.5	0
	-	11.25	-			45	45	0	45
	-	56.25	-			45	-45	0	-45
	-	33.75	-			67.5	-45		
		78.75	-			67.5	45		

**Table 2.2:** Spectropolarimetry waveplate patterns defined for the RSS. The stated angles refer to the angle of the half ( $\frac{1}{2}$ -) and quarter ( $\frac{1}{4}$ -) waveplate's optical axis from the perpendicular of the dispersion axis. Table adapted from the SALT call for proposals (2023).

# Chapter 3

## Existing and Developed Software: An overview of Polarimetric reductions for SALT (POLSLT), Image Reduction and Analysis Facility (IRAF), and STOPS

This chapter contains an overview of POLSLT (§ 3.1) and the limitations faced during POLSLT wavelength calibrations (§ 3.1.8), a brief overview of the IRAF tasks relevant to spectropolarimetric wavelength calibrations (§ 3.2), and an overview of STOPS, the software developed to supplement the POLSLT reduction process (§ 3.3). Finally, a discussion of the updated reduction process, an example of which may be found in Appendix I, is included (§ 3.4).

### 3.1 POLSLT - Polarimetric reductions for SALT

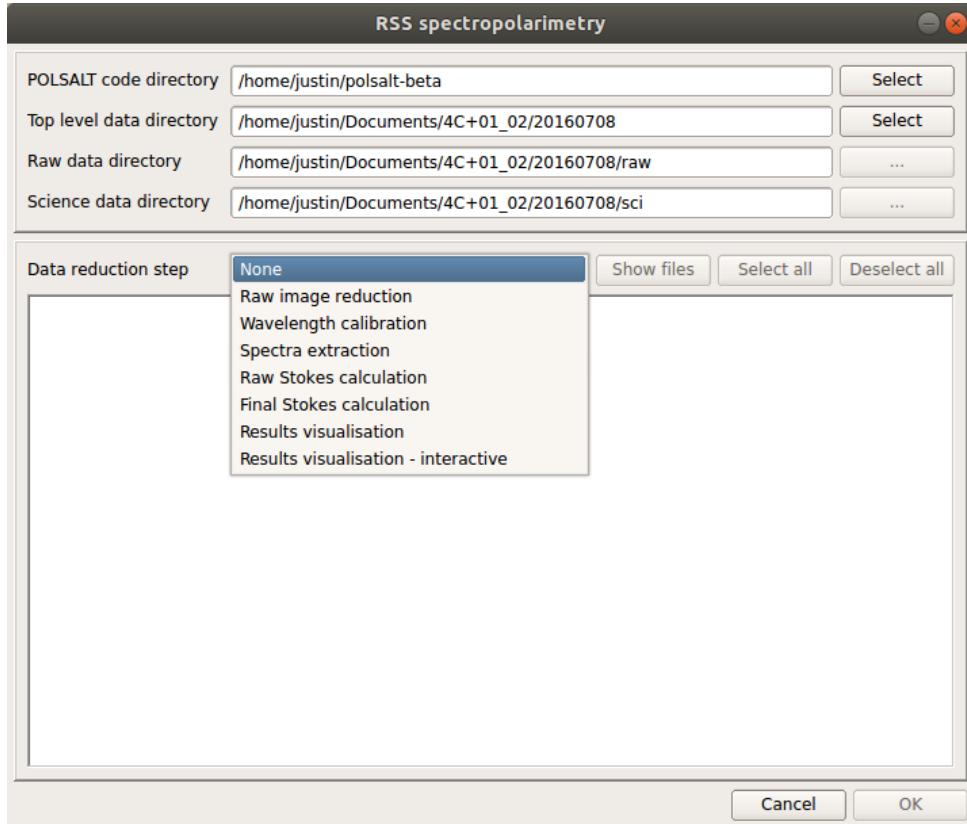
Polarimetric reductions for SALT (POLSLT) is developed by SALT and is the official reduction pipeline for spectropolarimetric data taken using the SALT RSS.<sup>1</sup> The newest version of the software, aptly named the ‘beta version’ (‘version’ 23 January 2020), includes a GUI as well as limited interactivity during key steps in the reduction process and was the version adapted in this study.<sup>2</sup>

The steps that make up the POLSLT reduction pipeline include basic CCD reductions, wavelength calibrations, background subtraction and spectral extraction, raw Stokes calculations, final Stokes calculations, and visualization of the results. Accurate reductions in each step are crucial for accurate results and thus briefly discussed below. Further details for the reduction process may be found at the POLSLT GitHub wiki.<sup>3</sup>

<sup>1</sup> POLSLT is made freely available via the POLSLT GitHub repository, available at <https://github.com/saltastro/polsalt>. It is strongly advised to follow the wiki for installation instructions.

<sup>2</sup> Installation files and instructions for the ‘beta version’ utilizing the GUI are available at <http://www.saao.ac.za/~ejk/polsalt/code/> in a TAR GZIP file.

<sup>3</sup> The GitHub wiki for POLSLT is available at <https://github.com/saltastro/polsalt/wiki>.



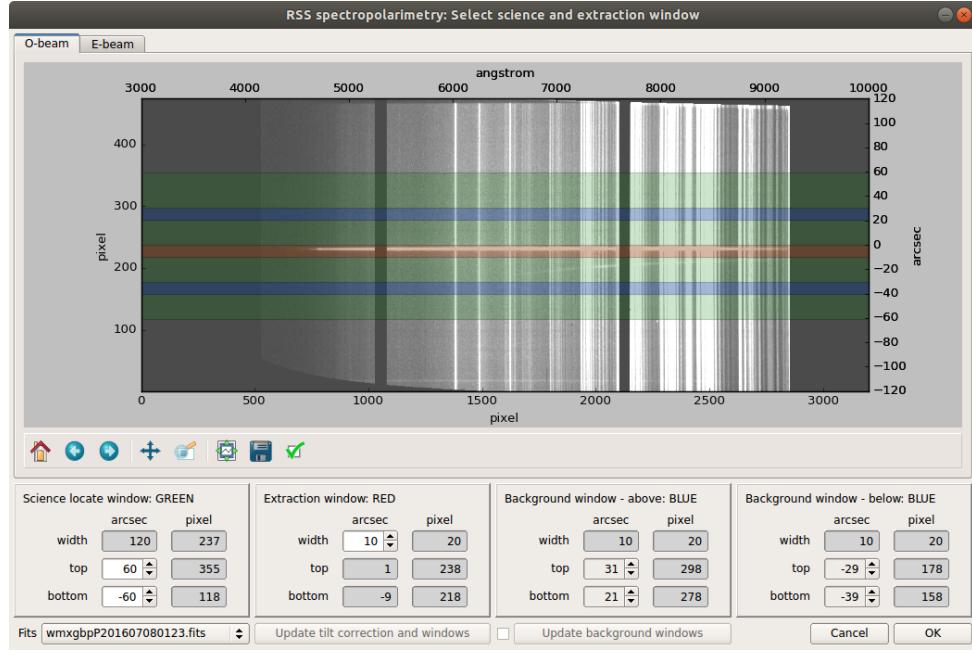
**Figure 3.1:** The layout of the `POLSALT` Graphical User Interface (GUI), including the contents of the reduction steps accessible via the dropdown menu. Note that there is no trailing forward slash after the ‘Top level data directory’. Figure created from local instance of the `POLSALT` GUI.

### 3.1.1 Basic CCD Reductions

Basic CCD reductions are run via `imred.py` and apply the necessary basic reductions to the raw data before any calibrations are applied. These reductions include overscan subtractions, gain corrections, crosstalk corrections, and mosaicking as well as attaching the bad pixel maps and pixel variance information. Files with basic reductions performed have “`mxgbp`” prepended to their names. As of February 2022, basic CCD reductions are automatically run for all RSS spectropolarimetric observations as part of the default SALT basic reduction pipeline that is run daily.

### 3.1.2 Wavelength Calibrations

Wavelength calibration and cosmic-ray rejection is performed via `specpolwavmap.py` and separately calibrates the *O*- and *E*-beams, based on the arc frames, and applies a simple cosmic-ray rejection for all science frames. This step is interactive and allows the user to individually fit wavelength calibration maps to each beam. The importance of an accurate correlation between both beams has been touched on previously (§ 2.3.2) and will be further discussed in § 3.1.8. The wavelength calibrated results are saved as an additional extension to each science FITS file, which are prefixed with a “`w`”, and the *O*- and *E*-beams of the extensions are split into their own sub-extensions.



**Figure 3.2:** The layout of the interactive POLSALT spectra extraction GUI after selecting the ‘update tilt correction and windows’ button along the bottom border of the window. Figure created from local instance of the POLSALT GUI.

### 3.1.3 Spectral Extraction

Background subtraction and spectral extraction is run via `specpoleextract_dev.py` which corrects for the beam-splitter distortion and tilt, performs sky subtraction, and extracts a one dimensional wavelength dependent spectrum for each beam sub-extension. This step is interactive and, using the brightest trace in the images, allows the user to define regions which span the wavelength axis and which define the background and trace regions for the sky subtraction and spectral extraction. Files with corrections applied are saved with “c” prepended to their names and files which contain the extracted one dimensional spectrum have “e” further prepended to their names.

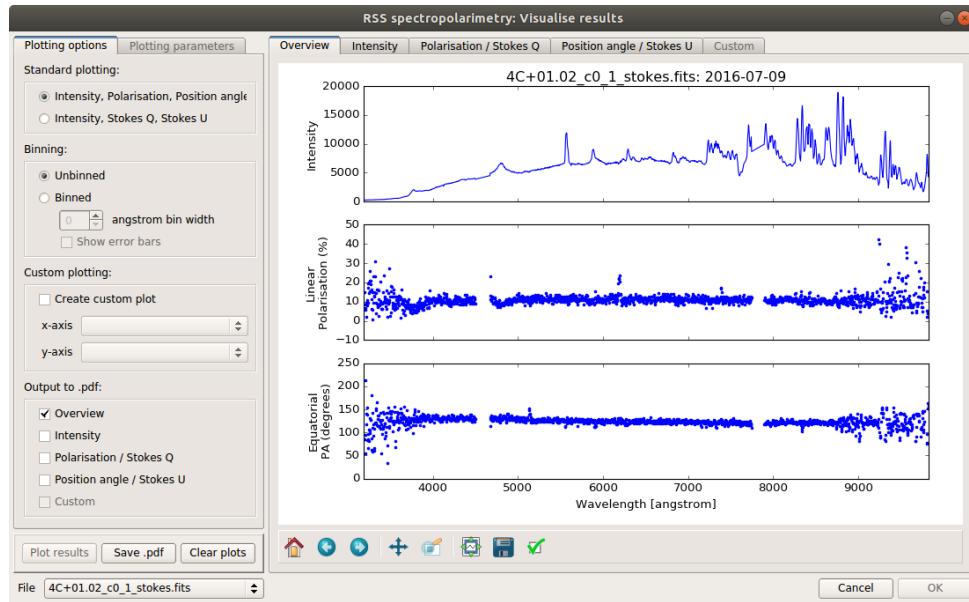
### 3.1.4 Raw Stokes Calculations

Raw Stokes calculations are performed via `specpolrawstokes_dev.py` and identify waveplate pairs for which the intensity,  $I$ , and a ‘raw Stokes’ signal,  $S$ , are calculated as:

$$I = \frac{1}{2}(O_1 + O_2 + E_1 + E_2), \text{ and} \quad (3.1)$$

$$S = \frac{1}{2} \left[ \left( \frac{O_1 - O_2}{O_1 + O_2} \right) - \left( \frac{E_1 - E_2}{E_1 + E_2} \right) \right]. \quad (3.2)$$

The raw Stokes signal is calculated as the normalized difference of the  $O$ - and  $E$ -beams, for a waveplate pair, taken perpendicular to one another. The files generated containing the raw Stokes information have a very specific naming style, with most notably the pair of frames being used included in the file names.



**Figure 3.3:** The layout of the interactive POLSALT visualization GUI after selecting the ‘Plot results’ button along the bottom border of the window. Figure created from local instance of the POLSALT GUI.

### 3.1.5 Final Stokes Calculations

The Final Stokes calculations are performed via `specpolfinalstokes.py` and, using the waveplate pattern along with the raw Stokes signals, calibrates for the polarimetric zero-point and waveplate efficiency, and calculates the final Stokes parameters. Before the final Stokes calculations are performed and when a sufficient amount of redundant exposures are taken, the raw Stokes signals are culled to eliminate outlier signals which may arise due to, for example, temporary atmospheric conditions affecting the signal. The data culling compares observation cycles against one another, compares the deviation of the signal means which estimate the baseline systematic polarization fluctuations (due to imperfections in repeatability), and performs a  $\chi^2$  analysis to eliminate outliers.

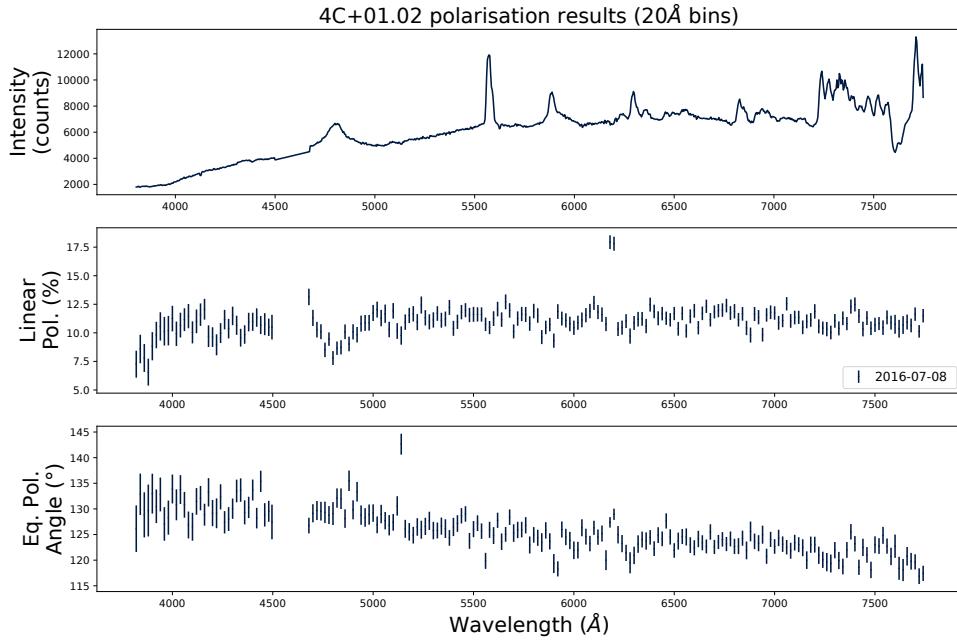
### 3.1.6 Visualization

Plotting the results of the spectropolarimetric reduction process uses `specpolview.py` and generates a plot of the Intensity, Linear Polarization (%), and Equatorial Polarization Angle ( $^\circ$ ) against a shared wavelength axis, as seen in Figure 3.4. This step is interactive and various options, such as the wavelength range, binning, etc., are available.

### 3.1.7 Post-Processing Analysis

Generally, the plot of the spectropolarimetric results is the stopping point for most reduction procedures as it contains or creates the desired results. However, additional tools exist which may be used after the polarization reductions, and which are not represented in the GUI, namely, flux calibration and synthetic filtering.

Flux-calibrations are performed via `specpolflux.py` and are only intended for shape corrections of the spectrum. Additionally, a flux database file must exist for the observed standard and must be included in the working science directory.



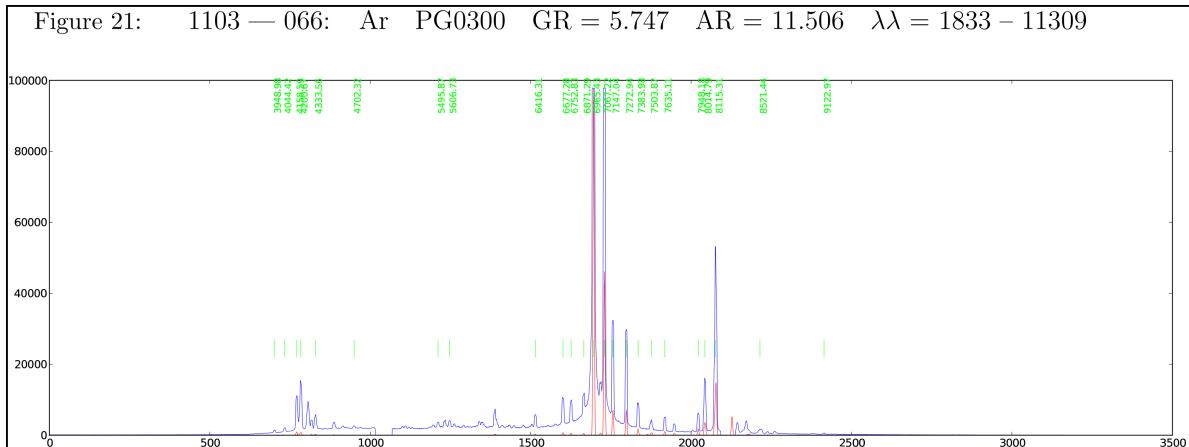
**Figure 3.4:** A typical plot resulting from the reduction process. Figure adapted from (Cooper et al., 2022).

Synthetic filtering is calculated via `specpolfilter.py` and computes the synthetically filtered polarization results. The filters which can be synthesized are the Johnson  $U$ ,  $B$ , and  $V$  filter curves from the SALTICAM filters, as well as the Cousins  $R$  and  $I$  filter curves, along with any user defined wavelength dependent throughput filter curves.

### 3.1.8 POLSALT Limitations and the Need for Supplementary Tools

The creation of supplementary tools for POLSALT spectropolarimetric reductions stemmed from the limitations of the wavelength calibration process and a need to compare wavelength solutions across the perpendicular  $O$  and  $E$  polarization beams. The process of calibrating wavelength solutions using the POLSALT pipeline is time-consuming for the average user, and often results in unexpected crashes when receiving erroneous inputs or key presses. Due to the time-consuming process of recalibrating the wavelength solutions it is not feasible to perform the wavelength calibrations time and time again for any amount of reductions larger than a handful of observations. This is particularly true for observations performed with the SALT PG0300 grating as the sparse spectral features of the Ar arc lamp are not handled well by the POLSALT pipeline.

Since PG0300 provided the widest wavelength range and highest throughput, it was almost exclusively used for observations of flaring blazars, resulting in a large backlog of unanalyzed data. The only arc available for the PG0300 grating with a close enough articulation and grating angle ( $\sim 10.68^\circ$  and  $\sim 5.38^\circ$ , respectively), was the Argon arc lamp which displays sparse spectral features with large gaps over the wavelength range at these grating and articulation angles (Figure 3.5). This often leads the POLSALT pipeline to create inconsistent wavelength solutions, or to fail to create a wavelength solution altogether, since minor deviations of identified spectral features result in large deviations in regions with no spectral features. To only further compound the difficulty of the wavelength calibrations, the spectrum of the Ar arc lamp contains a partial overlap of a differing order at higher wavelengths (§ 2.1.7, Equation 2.5).



**Figure 3.5:** One of many Argon arc lamp spectra as provided by SALT for line identification. Plot adapted from SALT’s published Longslit Line Atlases (as of 2024), resized to fit within the document margins but otherwise unchanged.<sup>4</sup>

The chosen solution to overcome the limitations of the wavelength calibration process was to use well established wavelength calibration software which allows for rapid recalibrations and provides a familiar interface. IRAF provides this familiar environment and reliability, in part thanks to its continued community development.

Unfortunately, IRAF is unable to natively parse the data structure implemented by POLSALT ‘as is’ and so the files must be restructured. This restructuring works both ways as once the IRAF reductions are complete the data structure must be restructured to match that of the POLSALT `wavelength calibration` output such that the reduction process may be completed in POLSALT.

## 3.2 IRAF - Image Reduction and Analysis Facility

Image Reduction and Analysis Facility (IRAF) is a collection of software designed specifically for the reduction and analysis of astronomical images and spectra (Tody, 1986, 1993). The software consists of many tasks which perform specific operations and which are grouped into relevant packages. Only a brief overview of the tasks will be provided here. Help documentation for any of the IRAF tasks may be found online under <https://iraf.net/irafdocs/> or through the IRAF Command Line Interface (CLI) through the `?` or `:help` ‘cursor commands’ when running interactive tasks, with more specific help documentation provided in the relevant section.

Useful IRAF tasks that deserve a brief mention before the wavelength calibration related tasks are: the `mkscript` task in the `system` package which allows a user to create and save a task along with the defined parameters as a file which can later be called as a script,<sup>5</sup> the `implot` task in the `plot` package which allows the rows or columns of an image to be interactively displayed,<sup>6</sup> and the `eparam` task in the `language` package which

<sup>4</sup>The ‘low resolution’ Ar plot sourced from <https://astronomers.salt.ac.za/data/salt-longslit-line-atlas/>

<sup>5</sup>Help documentation for the `mkscript` task may be found at [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/system.mkscript.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/system.mkscript.html).

<sup>6</sup>Help documentation for the `implot` task may be found at [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/plot.implot.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/plot.implot.html).

allows the parameters of a task to be edited within the IRAF CLI.<sup>7</sup>

For wavelength calibrations taken with the SALT RSS, the relevant tasks, in order, are the **identify** and **reidentify** tasks located in the `noao.onedspec` package, and the **fitcoords** and optionally the **transform** tasks located under the `noao.twodspec.longslit` package. These tasks produce a two-dimensional wavelength solution which must be obtained separately for the *O*- and *E*-beam.

### 3.2.1 Identify

The **identify** task is used to interactively determine a one-dimensional wavelength function across a chosen row of an arc exposure by identifying features in the spectrum with known wavelengths.<sup>8</sup> The task creates the first approximation of the wavelength solution as well as a local database in which the solution is saved. The initial identify solution is built on in subsequent tasks, and it is therefore imperative that the initial solution is well-fit to minimize errors further along the calibration process.

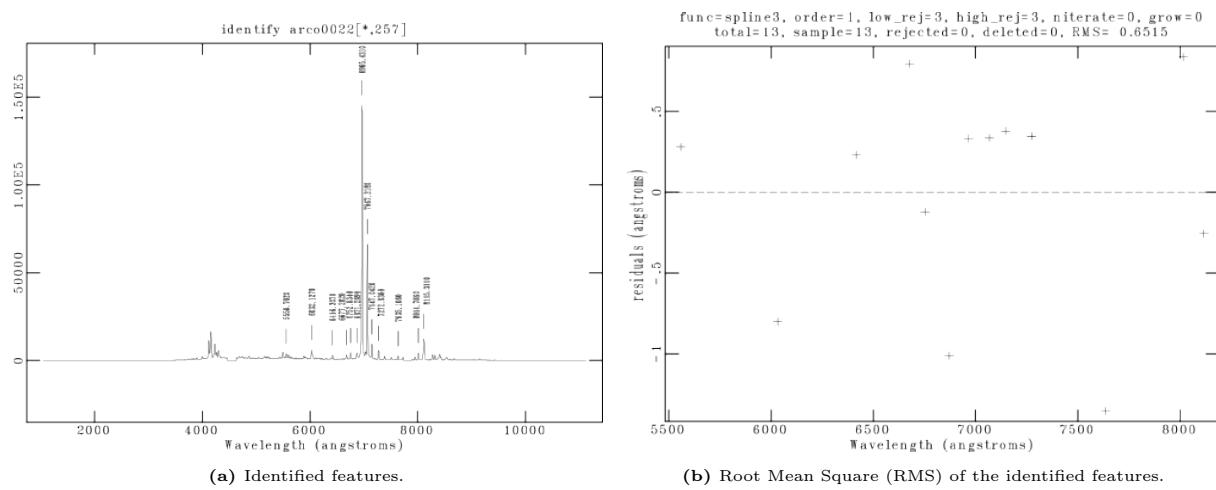
The process of using **identify** consists of identifying known features spanning the entire wavelength range and then removing identified features which negatively impact the wavelength solution. A balance must be found between the number of identified features and parameters of the fit against the deviation of the fit from the known features.

**Listing 3.1:** An example of the **identify** database contents.<sup>9</sup>

```
# Thu 15:19:16 13-May-2021
begin    identify arc00057 [* ,237]
  id      arc00057
  task    identify
  image   arc00057 [* ,237]
  units   Angstroms
  features 35
      53.61 5944.74989 5944.834 13.0 1 1 15257
    140.19 6029.9793 6029.997 13.0 1 1 4652
    185.30 6074.34644 6074.338 13.0 1 1 13396
    207.49 6096.15873 6096.161 13.0 1 1 21700
    255.23 6143.0493 6143.063 13.0 1 1 33330
    276.13 6163.56995 6163.594 13.0 1 1 11344
    330.89 6217.29293 6217.281 13.0 1 1 13705
    381.10 6266.51524 6266.495 13.0 1 1 21747
    420.21 6304.8113 6304.789 13.0 1 1 10226
    450.49 6334.45415 6334.428 13.0 1 1 36235
    500.18 6383.04826 6382.991 13.0 1 1 35824
    519.85 6402.26802 6402.248 13.0 1 1 70163
    626.70 6506.56147 6506.528 13.0 1 1 46165
    653.73 6532.91083 6532.882 13.0 1 1 21413
    721.60 6598.98642 6598.953 13.0 1 1 26396
    803.21 6678.31069 6678.277 13.0 1 1 51338
    843.18 6717.0732 6717.043 13.0 1 1 36780
    1099.95 6965.36335 6965.431 13.0 1 1 5618.4 ar
    1169.57 7032.38598 7032.413 13.0 1 1 100000
    1317.05 7173.89814 7173.938 13.0 1 1 5000 decrease
    1391.52 7245.11148 7245.167 13.0 1 1 73545
    1537.20 7383.93022 7383.981 13.0 1 1 5557.5 ar
    1595.02 7438.83545 7438.898 13.0 1 1 15000 decrease
    1663.64 7503.86263 7503.869 13.0 1 1 30000 ar; increase
    1697.46 7535.84584 7535.774 13.0 1 1 8000 increase
    1802.64 7635.07335 7635.106 13.0 1 1 20000 ar; decrease
    2209.19 8014.79559 8014.786 13.0 1 1 3000 ar; decrease
    2604.58 8377.66137 8377.607 13.0 1 1 14543
    2734.54 8495.41423 8495.359 13.0 1 1 8765
    2763.48 8521.52355 8521.442 13.0 1 1 4537.5 ar
    2840.92 8591.20799 8591.258 13.0 1 1 2000 decrease
    2889.39 8634.67334 8634.647 13.0 1 1 3059
    2911.42 8654.39264 8654.383 13.0 1 1 3000 decrease
    2926.56 8667.93501 8667.944 13.0 1 1 702.5 ar
    3135.72 8853.77575 8853.867 13.0 1 1 1820
  function legendre
  order 4
  sample *
```

<sup>7</sup>Help documentation for the `eparam` task may be found at [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/language.eparam.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/language.eparam.html).

<sup>8</sup>Help documentation for the **identify** task may be found at [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/noao.onedspec.identify.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.onedspec.identify.html).



**Figure 3.6:** A plot and the RMS of the identified features found using the IRAF `identify` task. Figures created using the IRAF `identify` task.

```

nverage 1
niterate 0
low_reject 3.
high_reject 3.
grow 0.
coefficients      8
 2.
 4.
53.60757446289061
3135.715576171875
7425.420339270724
1457.513831286474
-26.15751926622308
-3.000903509842187

```

---

<sup>9</sup>See also <https://iraf.net/irafdocs/formats/identify.php> for an explanation of the database contents.

### 3.2.2 Reidentify

The `reidentify` task is used to run the `identify` task autonomously and repeatedly across the entirety of the arc exposure at a defined interval.<sup>10</sup> The task uses the one-dimensional wavelength solution stored in the database created by the initial `identify` call and refits the previously identified points to match the new positions of the relevant spectral features. The task may fail based on a number of conditions, most common of which is the loss of features as the task moves further from the row at which the user manually ran `identify`.

When running `reidentify` non-interactively, it is recommended to set the `verbose` parameter to ‘yes’ as this will provide immediate confirmation of if the task quit early. Regardless of whether or not the task quit successfully, the newly defined wavelength solutions are appended to the local database following the `identify` task database format (see Listing 3.1).

### 3.2.3 Fitcoords

The `fitcoords` task is used to combine the identified one-dimensional wavelength solutions in the database directory into a two-dimensional surface function.<sup>11</sup> The usage of `fitcoords` is similar to that of `identify` and consists of examining the distribution of identified points and eliminating any points that `reidentify` may have misidentified.

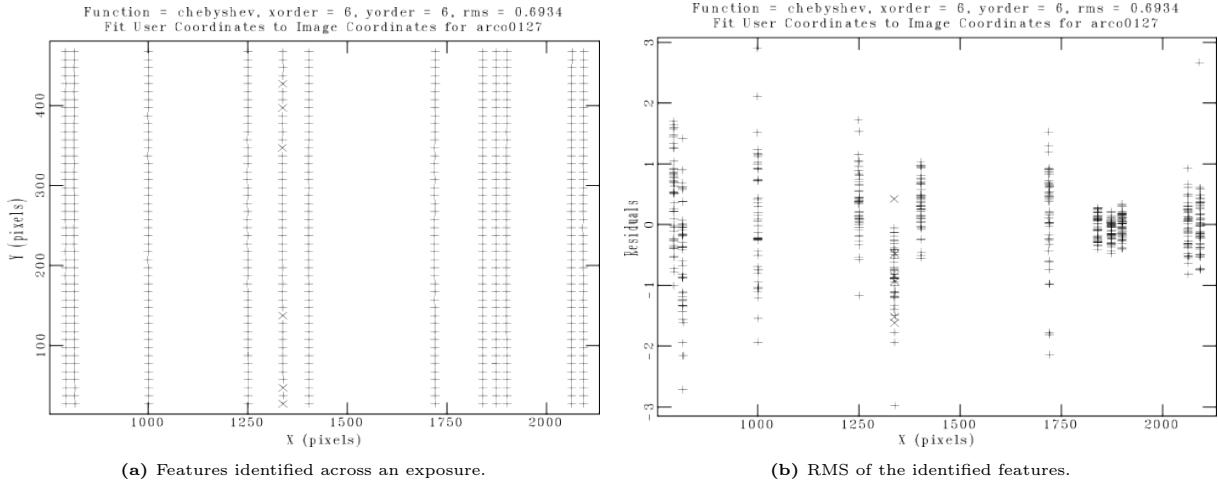
By eliminating outliers with bad residuals and modifying the two-dimensional surface function’s type and degree, the overall error of the fit is decreased, aligning more closely to what the ‘true’ wavelength solution is. This surface function is the final two-dimensional wavelength solution and is what is needed to convert the IRAF formatted wavelength calibrated Flexible Image Transport System (FITS) files back into the POLSALT format. The solution is stored in the local IRAF database and is the solution used by the STOPS `join` method.

**Listing 3.2:** An example of the `fitcoords` database contents.<sup>12</sup>

```
# Thu 15:26:55 13-May-2021
begin    arc00057
task      fitcoords
axis      1
units    angstroms
surface  33
      1.
      5.
      5.
      1.
      1.
      3199.
      1.
      474.
      7419.096745914063
      1510.03933621895
      -21.10886852752348
      -2.079553916887794
      0.06772631420528228
      0.7720164913117386
      -1.506773900054024
      0.1341878190232142
      -0.01659697703758917
      0.0251087019569153
      -3.318493303995171
      -0.3612632489821799
      0.003270665801371641
```

<sup>10</sup>Help documentation for the `reidentify` task may be found at [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/noao.onedspec.reidentify.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.onedspec.reidentify.html).

<sup>11</sup>Help documentation for the `fitcoords` task may be found at [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/noao.twodspec.longslit.fitcoords.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.twodspec.longslit.fitcoords.html).



**Figure 3.7:** A plot and the RMS of the features identified across the exposure using the IRAF `fitcoords` task. Figures created using the IRAF `fitcoords` task.

```

-0.0157962041414068
-0.003073690871589242
0.007533453962924031
0.02839687304474069
-0.003233465769521899
0.00174111456659807
0.00645177595090841
0.0105080093855621
-0.01157827440314294
-0.007789479002470706
-0.006562085282926231
-0.002321476801926803

```

### 3.2.4 Transform

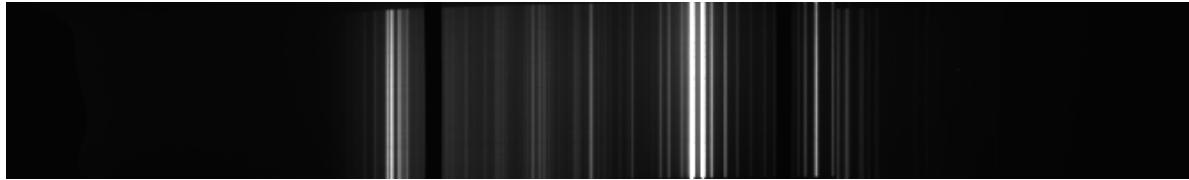
The `transform` task is the optional final step in the IRAF wavelength calibration process.<sup>13</sup> Simply put, `transform` converts the (pixel, pixel) units of an exposure to (wavelength, pixel) units which allows for an immediate check whether the wavelength solution is consistent across the frame. Any general error in the wavelength solution may be spotted in the transformed images (see Figure 3.8); ranging from minor errors, such as the arc exposure's arc lines or science exposure's sky lines not being straight across the columns of the frame, to more major errors, such as an incorrect wavelength solution skewing the exposure beyond recognition.

## 3.3 STOPS - Supplementary Tools for POLSALT Spectropolarimetry

Supplementary Tools for POLSALT Spectropolarimetry (STOPS) provides supplementary tools which convert the POLSALT and IRAF formats back and forth, allowing IRAF to be used for wavelength calibrations of SALT spectropolarimetric data. It also provides additional tools to check the accuracy of the wavelength calibration. STOPS is written in Python and requires Python 3.11+ to run, as well as `Astropy` (6.0.0+) (Astropy Collaboration et al., 2013, 2018, 2022), `ccdproc` (2.4.1+) (Craig et al., 2017), `Matplotlib`

<sup>12</sup>See also <https://iraf.net/irafdocs/formats/fitcoords.php> for an explanation of the database contents.

<sup>13</sup>Help documentation for the `transform` task may be found at [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/noao.twodspec.longslit.transform.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.twodspec.longslit.transform.html).



(a) A well-fit wavelength solution.



(b) A poorly fit wavelength solution with a region of no exposure (top of frame) displaying a terribly fit wavelength solution.

**Figure 3.8:** Examples of a well- and poor fit wavelength solution applied to the *O*- (a) and *E*-beams (b) of an arc image. The figures were scaled to best capture any deviation of the arc lines. Figures created by the IRAF `transform` task.

(3.5.2+) (Hunter, 2007), NumPy (1.26.4+) (Harris et al., 2020), and SciPy (1.13.0+) (Virtanen et al., 2020).

The parsing of POLSALT data into an IRAF usable format and the reformatting of the IRAF wavelength calibrated data back into a POLSALT usable format, referred to as *splitting* and *joining*, is performed by the STOPS `split` and `join` methods, respectively.

Methods to verify the validity of the alternate wavelength calibrations were also added to STOPS. The `skyline` method checks the sky line wavelength (*x*) positions across the frame as well as the variation of the sky lines across the positional (*y*) axis of the frame. The `correlate` method checks the correlation of the *O*- and *E*-beams either within a given FITS file or across multiple files (comparing only the *O*- and *E*-beams for each). With these two additional methods, a user is able to verify that the wavelength solutions do not conflict across the *O*- and *E*-beams and that no unexpected deviations are included in the wavelength solutions.

Help for the STOPS CLI commands may be found either by running:

---

```
$ python ~/STOPS --help
```

---

or in Listing II.1 (in Appendix II). Help for the STOPS specific mode commands may be found by running:

---

```
$ python ~/STOPS [split|join|correlate|skylines] --help
```

---

Finally, help documentation for the STOPS methods may be found within this section (Listing 3.3 to 3.6) or in Appendix II.

### 3.3.1 Splitting

**Listing 3.3:** The ‘docstring’ for `split.py`

---

```
26
27 """
28 The `Split` class allows for the splitting of `polsalt` FITS files
29 based on the polarization beam. The FITS files must have basic
30 `polsalt` pre-reductions already applied (`mxbp...` FITS files).
31
32 Parameters
```

---

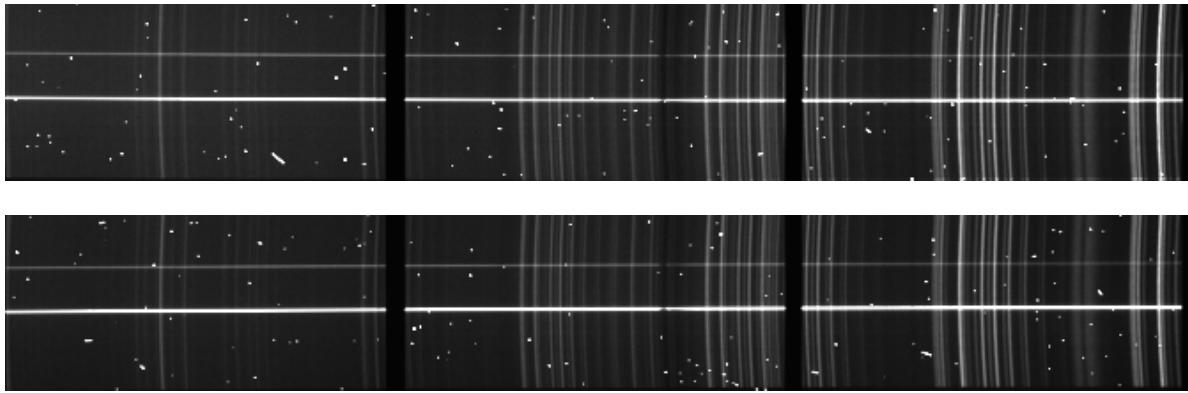
```

33 -----
34 data_dir : str
35   The path to the data to be split
36 fits_list : list[str], optional
37   A list of pre-reduced ‘polsalt’ FITS files to be split within `data_dir`.
38   (The default is None, ‘Split’ will search for ‘magbp*.fits’ files)
39 split_row : int, optional
40   The row along which to split the data of each extension in the FITS file.
41   (The default is SPLIT_ROW (See Notes), the SALT RSS CCD’s middle row)
42 no_arc : bool, optional
43   Decides whether the arc frames should be recombined.
44   (The default is False, ‘polsalt’ has no use for the arc after wavelength calibrations)
45 save_prefix : dict[str, list[str]], optional
46   The prefix with which to save the ‘O & E’ beams.
47   Setting ‘save_prefix’ = ‘None’ does not save the split ‘O & E’ beams.
48   (The default is SAVE_PREFIX (See Notes))
49
50 Attributes
51 -----
52 arc : str
53   Name of arc FITS file within `data_dir`.
54   ‘arc’ = “” if ‘no_arc’ or not detected in ‘data_dir’.
55 o_files, e_files : list[str]
56   A list of the ‘O’- and ‘E’-beam FITS file names.
57   The first entry is the arc file if ‘arc’ defined.
58 data_dir
59 fits_list
60 split_row
61 save_prefix
62
63 Methods
64 -----
65 split_file(file: os.PathLike)
66   -> tuple[astropy.io.fits.HDUList]
67   Handles creation and saving the separated FITS files
68 split_ext(hdulist: astropy.io.fits.HDUList, ext: str = ‘SCI’)
69   -> astropy.io.fits.HDUList
70   Splits the data in the ‘ext’ extension along the ‘split_row’
71 crop_file(hdulist: astropy.io.fits.HDUList, crop: int = CROP_DEFAULT (See Notes))
72   -> tuple[numumpy.ndarray]
73   Crops the data along the edge of the frame, that is,
74   ‘O’-beam cropped as [crop:], and
75   ‘E’-beam cropped as [:crop].
76 update_beam_lists(o_name: str, e_name: str)
77   -> None
78   Updates ‘o_files’ and ‘e_files’.
79 save_beam_lists(file_suffix: str = ‘frames’)
80   -> None
81   Creates (Overwrites if exists) and writes the ‘o_files’ and ‘e_files’ to files named
82   ‘o_{file_suffix}’ and ‘e_{file_suffix}’, respectively.
83 process()
84   -> None
85   Calls ‘split_file’ and ‘save_beam_lists’ on each file in ‘fits_list’ for automation.
86
87 Other Parameters
88 -----
89 **kwargs : dict
90   keyword arguments. Allows for passing unpacked dictionary to the class constructor.
91
92 Notes
93 -----
94 Constants Imported (See utils.Constants):
95   SAVE_PREFIX
96   CROP_DEFAULT
97   SPLIT_ROW
98
99 """

```

As mentioned previously, the format of the FITS file created by POLSALT after basic CCD reductions and the format expected by IRAF to be used for the wavelength calibrations are incompatible. Basic POLSALT CCD reductions return FITS files which contain a primary header along with extensions for the science, variance, and Bad Pixel Map (BPM) images. These extensions carry the image of the trace for both polarimetry beams (see Figure 3.9), the variance of the image, and a map of the pixels to be masked out, respectively.

IRAF is capable of dealing with multiple traces in an extension or lists of input files but is not as capable when dealing with multiple wavelength solutions contained in a single extension (as expected by the POLSALT `wavelength calibration`) or extensions containing sub-extensions (as expected by the POLSALT `spectral extraction`). To simplify the IRAF reduction procedure it was decided to separate the perpendicular polarization beams into their own files.



**Figure 3.9:** The split *O*- and *E*-beams as handed to IRAF. Figure created from the `STOPS` `split` method output.

The files with POLSALT pre-reductions applied, namely FITS files with an ‘`mxgbp`’ prefix (§ 3.1), are used as the starting point for the supplementary tool’s `split` method. Running `split` finds all the FITS files for wavelength calibration within the working directory, creates two empty Header Data Unit (HDU) structures for each sub-extension of the FITS file, and appends all science and header data necessary for wavelength calibration to the relevant HDU structure.

As the intent was always to parse the wavelength function back into POLSALT it was decided to keep these temporary FITS files as small as possible. This is especially necessary when considering the amount of exposures that are taken for a single spectropolarimetric observation run, and how the number of observations increases for long term studies.

To aid the scripting of the IRAF wavelength calibration process, the `split` method also performs row cropping to exclude CCD regions with no exposure and creates files listing the split *O*- and *E*-beam FITS files to simplify the IRAF task inputs.<sup>14</sup> Otherwise, defaults, such as which row to split the beams along, were kept as close to the POLSALT pipeline as possible.

### 3.3.2 Joining

**Listing 3.4:** The ‘docstring’ for `join.py`

```

32 """
33 The `Join` class allows for the joining of previously
34 split files and the appending of an external wavelength
35 solution in the `polsalt` FITS file format.
36
37 Parameters
38 -----
39 data_dir : str
40     The path to the data to be joined
41 database : str, optional
42     The name of the `IRAF` database folder.
43     (The default is "database")
44 fits_list : list[str], optional
45     A list of pre-reduced `polsalt` FITS files to be joined within `data_dir`.
46     (The default is ``None``, `Join` will search for `*mxgbp*.fits` files)
47 solutions_list: list[str], optional
48     A list of solution filenames from which the wavelength solution is created.
49     (The default is ``None``, `Join` will search for `fc*` files within the `database` directory)
50 split_row : int, optional
51     The row along which the data of each extension in the FITS file was split.
52

```

<sup>14</sup>Row cropping was decided on as IRAF does not handle rows with no exposure well, specifically when it comes to the `reidentify` task.

```

53     Necessary when Joining cropped files.
54     (The default is 517, the SALT RSS CCD's middle row)
55 save_prefix : dict[str, list[str]], optional
56     The prefix with which the previously split 'O'- & 'E'-beams were saved.
57     Used for detecting if cropping was applied during the splitting procedure.
58     (The default is SAVE_PREFIX (See Notes))
59 verbose : int, optional
60     The level of verbosity to use for the Cosmic ray rejection
61     (The default is 30, I.E. logging.INFO)
62
63 Attributes
64 -----
65 fc_files : list[str]
66     Valid solutions found from `solutions_list`.
67 custom : bool
68     Internal flag for whether `solutions_list` uses the `IRAF` or a custom format.
69     See Notes for custom solution formatting.
70     (Default (inherited from `solutions_list`) is False)
71 arc : str
72     Deprecated. Name of arc FITS file within `data_dir`.
73 data_dir
74 database
75 fits_list
76 split_row
77 save_prefix
78
79 Methods
80 -----
81 get_solutions(wavlist: list / None, prefix: str = "fc")
82     -> (fc_files, custom)
83     Parse `solutions_list` and return valid solution files and if they are non-'IRAF' solutions.
84 parse_solution(fc_file: str, xshape: int, yshape: int)
85     -> tuple[dict[str, int], np.ndarray]
86     Loads the wavelength solution file and parses keywords necessary for creating the wavelength extension.
87 join_file(file: os.PathLike)
88     -> None
89     Joins the files,
90     attaches the wavelength solutions,
91     performs cosmic ray cleaning,
92     masks the extension,
93     and checks cropping performed in `Split`.
94     Writes the FITS file in a `polsalt` valid format.
95 check_crop(hdu: pyfits.HDUList, o_file: str, e_file: str)
96     -> int
97     Opens the split 'O'- and 'E'-beam FITS files and returns the amount of cropping that was performed.
98 process()
99     -> None
100    Calls `join_file` on each file in `fits_list` for automation.
101
102 Other Parameters
103 -----
104 no_arc : bool, optional
105     Deprecated. Decides whether the arc frames should be processed.
106     (The default is False, 'polsalt' has no use for the arc after wavelength calibrations)
107 **kwargs : dict
108     keyword arguments. Allows for passing unpacked dictionary to the class constructor.
109
110 Notes
111 -----
112 Constants Imported (See utils.Constants):
113     DATADIR
114     SAVE_PREFIX
115     SPLIT_ROW
116     CR_PARAMS
117
118 Custom wavelength solutions must be formatted as:
119     'x',
120     'y',
121     *coefficients...
122 where the solutions are of order ('x' by 'y') and contain x*y coefficients.
123 The name of the custom wavelength solution file must contain either "cheb" or "leg"
124 for Chebychev or Legendre wavelength solutions, respectively.
125
126 Cosmic ray rejection is performed using lacosmic [1]_, implemented in ccdproc via astroscrappy [2]_.
127
128 References
129 -----
130 .. [1] van Dokkum 2001, PASP, 113, 789, 1420 (article : http://adsabs.harvard.edu/abs/2001PASP..113.1420V)
131 .. [2] https://zenodo.org/records/1482019
132 """
133
134
135 """

```

As mentioned previously, the FITS file formats created by IRAF after wavelength calibrations versus that expected by the POLSALT spectra extraction are incompatible. The STOPS join method is used at this point since all pieces necessary to recreate the POLSALT wavelength calibrated FITS files exist once `fitcoords` has been successfully run. A typical FITS file expected by the POLSALT spectra extraction contains a primary header along with the various image extensions, with most importantly a newly created

wavelength extension.

Running `join` finds all the relevant FITS and local IRAF database files, creates an empty HDU structure for each pair of matching spectropolarimetric beams, copies over the extensions and their respective image and header information, appends a new extension and parses the database wavelength solutions into the POLSALT intensity-wavelength format, performs cosmic ray cleaning, and masks the BPM to reflect the wavelength calibrated region.

Firstly, `join` parses the wavelength database file, described in § 3.2.3, for specifically either ‘Chebyshev’ or ‘Legendre’ solutions, and creates a function to convert a (*pixel*, *pixel*) position to a wavelength value. This is used to fill the pixels of the wavelength extension with their respective wavelength, as seen in Figure 3.10.

Next, the `join` method cleans the science extension of cosmic rays using the `lacosmic` python package which uses the L.A. Cosmic algorithm, based on Laplacian edge detection. The read noise and gain parameters used for cosmic ray cleaning were chosen based on the properties of the RSS, while the rest of the parameters were left as the default, following the publication and suggestions<sup>15</sup> by the algorithm’s creator, as well as the implementation of the algorithm in the python `ccdproc` package (McCully et al., 2018; van Dokkum, 2001). The chosen parameters work well for most cosmic rays, as can be seen when comparing Figure 3.9 to Figure 3.11, but may be modified as needed.

Thereafter, `join` updates the headers to be near-identical to those created by the POLSALT wavelength calibration, most notably updating the data shape, ‘CTYPE3’, and data type, ‘BITPIX’, keywords. The only difference in the header is the ‘NAXIS2’ keyword, due to the cropping performed by `split`. The cropped region could be reintroduced but would be masked out and further discarded in the following POLSALT spectra extraction process, making it redundant.

Finally, the wavelength extension is masked to remove any uncalibrated wavelength regions as well as masked for the skewing of the trace introduced by the wollaston element. The masking of the wollaston skewing is necessary since POLSALT introduces a wollaston correction in the spectra extraction process. The BPM extension is masked to reflect the valid wavelength calibrated region, and the files are saved with the POLSALT wavelength calibrated ‘wmxgbp’ prefix.

### 3.3.3 Sky Line Checks

**Listing 3.5:** The ‘docstring’ for `skylines.py`

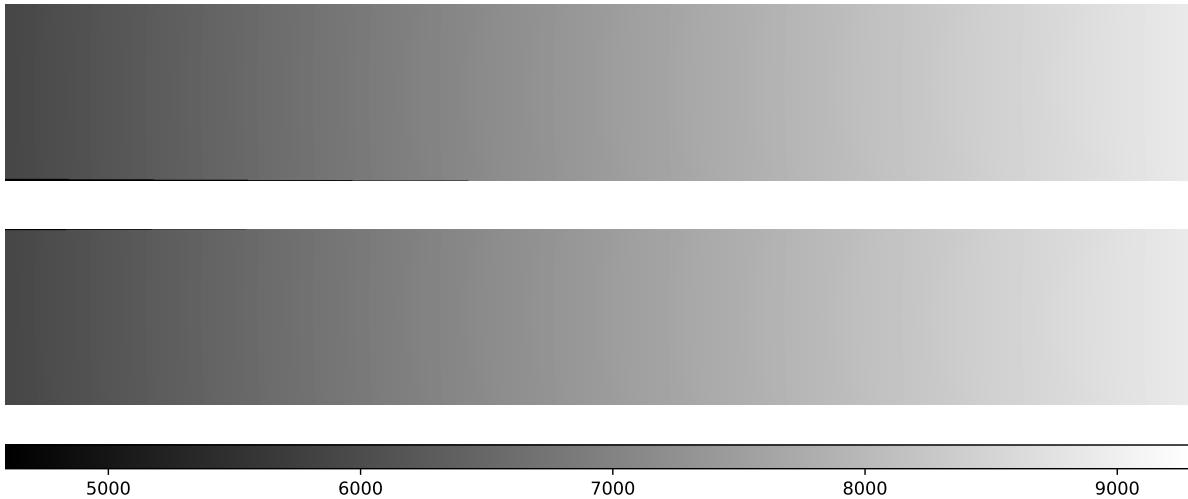
```

30
31 """
32 Class representing the Skylines object.
33
34 Parameters
35 -----
36 data_dir : Path
37     The directory containing the data files.
38 filenames : list[str]
39     The list of filenames to be processed.
40 beam : str, optional

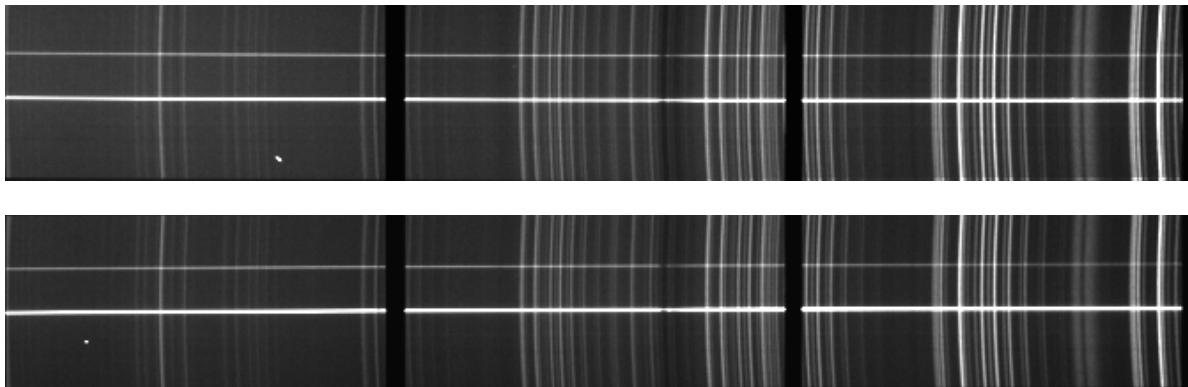
```

---

<sup>15</sup>Suggested parameters for the `lacosmic` algorithm may be found at <http://www.astro.yale.edu/dokkum/lacosmic/pars.html>.



**Figure 3.10:** The wavelength extension of a FITS file ready to be handed back to the POLSALT pipeline. The color bar displays the wavelength in Å as displayed by the *O*- and *E*-beam sub-extensions. Note that regions which fall outside the exposed region are masked by setting the corresponding pixel values of the wavelength and BPM extensions to 0. Figure created from the STOPS join method output.



**Figure 3.11:** A representative science extension of a FITS file capable of being parsed by the POLSALT pipeline. The intensity of the *O*- and *E*-beam sub-extensions are displayed via the grayscale value at each pixel. Note the inclusion of regions of: solid black demarcating the chip gaps, horizontal stripes of high intensity demarcating the trace, and (curved) vertical stripes of high intensity demarcating the sky lines. Figure created from the STOPS STOPSSjoin method output.

```

41     The beam mode, by default "OE".
42     plot : bool, optional
43         Flag indicating whether to plot the continuum, by default False.
44     save_prefix : Path / None, optional
45         The prefix for saving the data, by default None.
46     **kwargs
47         Additional keyword arguments.
48
49 Attributes
50 -----
51     data_dir : Path
52         The directory containing the data files.
53     fits_list : list[str]
54         The list of fits file paths.
55     beam : str
56         The beam mode.
57     can_plot : bool
58         Flag indicating whether to plot the continuum.
59     save_prefix : Path / None
60         The prefix for saving the data.
61     wav_unit : str
62         The unit of wavelength.
63     rawWav : np.ndarray
64         The raw wavelength data.
65     rawSpec : np.ndarray
66         The raw spectral data.
67     rawBpm : np.ndarray
68         The raw bad pixel mask data.
69     corrWav : np.ndarray
70         The corrected wavelength data.
71     corrSpec : np.ndarray

```

### TODO: Include example of skyline result

**Figure 3.12:** The resultant output plot of the STOPS `skylines` method. Figure created via the STOPS `skyline` method.

```

72     The corrected spectral data.
73     spec : np.ndarray
74         The median spectrum.
75     normSpec : np.ndarray
76         The normalized spectrum.
77
78     Methods
79 -----
80     checkLoad(self, path1: str) -> np.ndarray:
81         Checks and loads the data from the given path.
82     transform(self, wav_sol: np.ndarray, spec: np.ndarray) -> np.ndarray:
83         Transforms the input wavelength and spectral data based on the given wavelength solution.
84     rmvCont(self) -> np.ndarray:
85         Removes the continuum from the spectrum.
86     skylines(self) -> None:
87         Placeholder method for processing skylines.
88     process(self) -> None:
89         Placeholder method for processing the data.
90     """

```

Sky line comparisons allow the user to confirm the accuracy of the wavelength solution across both the rows and columns of the frame. The `skyline` method naively transforms the wavelength calibrated files,<sup>16</sup> allowing the deviation of a skyline spanning multiple rows to be measured, and compares the observed sky line wavelength positions to those measured by SALT, allowing the wavelength deviation of the skylines to be measured.<sup>17</sup>

The file used for sky line comparisons may either be the IRAF `transform` FITS file or the ‘wmxgbp’ FITS files created by `join`. The `skyline` method loads the wavelength calibrated files, transforms the frames if the frame was not transformed by `transform`, removes the continuum, compares the width of the sky lines before and after transformation, and compares the wavelength position of the sky lines to the reference sky lines as measured by SALT.

Firstly, determining if there is any inaccuracy in the wavelength solution in the spatial ( $y$ , or vertical) axis is relatively straightforward as a perfect wavelength solution will remove any horizontal variation of a sky line spanning multiple rows. The variation may be seen in the transformed frame, as mentioned in § 3.2.4, but is more accurately measured by the `skyline` method which compares the averaged sky line width before and after transformation. A wavelength solution exhibiting a poor fit across the spatial axis will display broader averaged sky lines than those of a relatively good fit.

Lastly, determining if there is any inaccuracy in the wavelength solution in the wavelength ( $x$ , or horizontal) axis is more challenging as no features, other than the trace of the observed sources, are exposed across a frame. Thankfully, SALT has published a sky line atlas which may be used to verify and measure the deviation of the observed sky line wavelength positions. Minor variations in the comparison of the sky lines are expected, but any uniform trends indicate an underlying poor fit across the horizontal axis of the wavelength solution. A poor horizontal fit is difficult to spot without supplementary tools and has adverse effect on the final polarization results.

<sup>16</sup>The transformation applied by the `skyline` method uses linear interpolation and is thus less accurate at flux conservation than the transformation applied by the `transform` method.

<sup>17</sup>The first iteration of a sky line atlas is available at <https://astronomers.salt.ac.za/data/salt-longslit-line-atlas/>

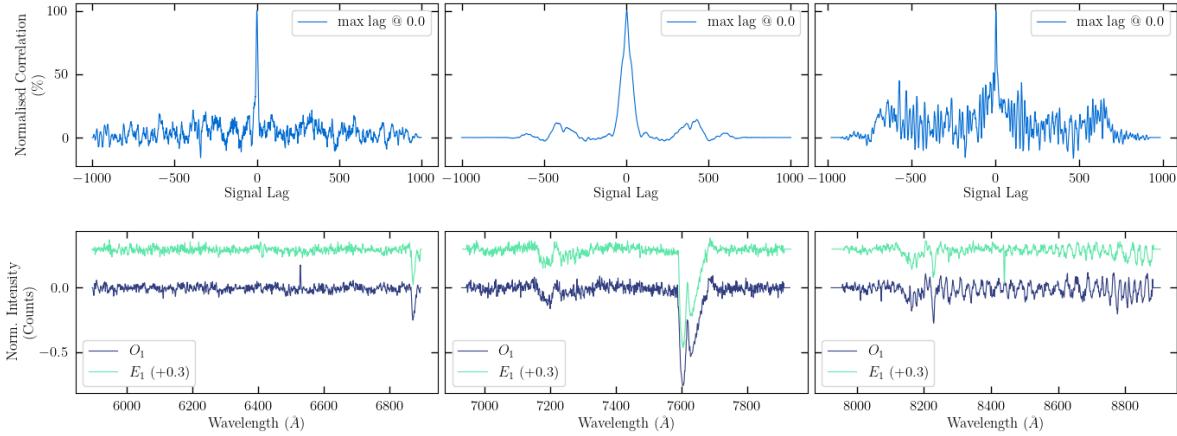
### 3.3.4 Cross Correlation

**Listing 3.6:** The ‘docstring’ for `cross_correlate.py`

```

34 """
35
36 Cross correlate allows for comparing the extensions of multiple
37 FITS files, or comparing the O and E beams of a single FITS file.
38
39 Parameters
40 -----
41 data_dir : str
42     The path to the data to be cross correlated
43 filenames : list[str]
44     The ecumagbp*.fits files to be cross correlated.
45 If only one filename is defined, correlation is done against the two polarization beams.
46 split_ccd : bool, optional
47     Decides whether the CCD regions should each be individually cross correlated.
48 (The default is True, which splits the spectrum up into its separate CCD regions)
49 cont_ord : int, optional
50     The degree of a chebyshev to fit to the continuum.
51 (The default is 11)
52 plot : bool, optional
53     Decides whether or not the continuum fitting should be plotted
54 (The default is False, so no continuum plots are displayed)
55 save_prefix : str, optional
56     The name or directory to save the figure produced to.
57     "." saves a default name to the current working. A default name is also used when save_prefix is a directory.
58 (The default is None, I.E. The figure is not saved, only displayed)
59
60 Attributes
61 -----
62 data_dir
63 fits_list
64 beams : str
65     The mode of correlation.
66 'OE' for same file, and 'O' or 'E' for different files but same ext's.
67 ccds : int
68     The number of CCD's in the data. Used to split the CCD's if split_ccd is True.
69 cont_ord : int
70     The degree of the chebyshev to fit to the continuum.
71 can_plot : bool
72     Decides whether or not the continuum fitting should be plotted
73 offset : int
74     The amount the spectrum is shifted, mainly to test the effect of the cross correlation
75 (The default is 0, I.E. no offset introduced)
76 save_prefix
77 wav_unit : str
78     The units of the wavelength axis.
79 (The default is Angstroms)
80 wav_cdelta : int
81     The wavelength increment.
82 (The default is 1)
83 alt : Callable
84     An alternate method of cross correlating the data.
85 (The default is None)
86
87 Methods
88 -----
89 load_file(filename: Path)
90     -> tuple[np.ndarray, np.ndarray, np.ndarray]
91     Loads the data from a FITS file.
92 get_bounds(bpm: np.ndarray)
93     -> np.ndarray
94     Finds the bounds for the CCD regions.
95 remove_cont(spec: list, wav: list, bpm: list, plotCont: bool)
96     -> None
97     Removes the continuum from the data.
98 correlate(filename1: Path, filename2: Path | None = None)
99     -> None
100    Cross correlates the data.
101 FTCS(filename1: Path, filename2: Path | None = None)
102     -> None
103     Cross correlates the data using the Fourier Transform.
104 plot(spec, wav, bpm, corrdb, lagsdb)
105     -> None
106     Plots the data.
107 process()
108     -> None
109     Processes the data.
110
111 Other Parameters
112 -----
113 offset : int, optional
114     The amount the spectrum is shifted, mainly to test the effect of the cross correlation
115 (The default is 0, I.E. no offset introduced)
116 **kwargs : dict
117     keyword arguments. Allows for passing unpacked dictionary to the class constructor.
118 FTCS : bool, optional
119     Decides whether the Fourier Transform should be used for cross correlation.
120
121 See Also
122 -----
123 scipy
124     https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.correlate.html
125
126
127

```



**Figure 3.13:** The resultant output plot of the `STOPS correlate` method. Figure created via the `STOPS correlate` method.

```

128 Notes
129 -----
130 Constants Imported (See utils.Constants):
131   SAVE_CORR
132
133 """

```

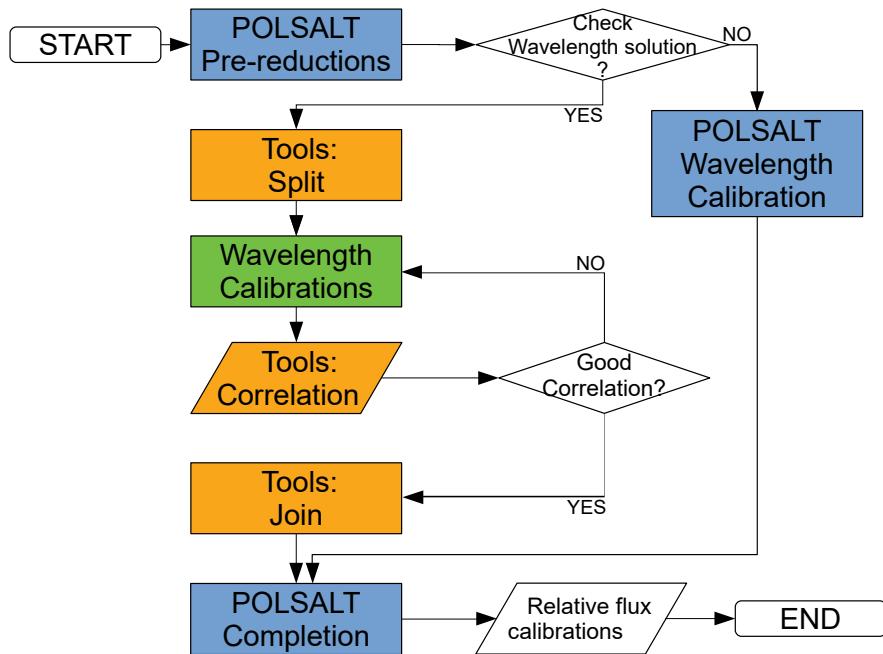
The `skyline` method allows for confirmation of a single wavelength solution, but has no means for comparing how the wavelength solutions of two polarization beams differ from each other. As the Stokes results, and thus final polarization results, are determined and are heavily influenced by the differences in the spectra of the different *O*- and *E*-beams, a direct comparison is not appropriate. Any observed unpolarized light, however, will reflect equally in both polarization beams and so the general trend of the two spectra may reasonably be expected to follow one another. The `correlate` method was created to allow for comparisons between the wavelength solutions of the *O*- and *E*-beams of a single exposure or the *O*- or *E*-beams of differing exposures by cross correlating the spectra.

The `correlate` method loads the POLSALT spectra extraction FITS files, removes the continuum and separates the CCD regions. The relevant CCD regions are cross correlated and the correlation peak is plotted and specified in the plot legend, as seen in Figure 3.13.

Sources under spectropolarimetric observation are generally expected to vary over time and, as such, the ratio of polarized to unpolarized light is also expected to vary. The accuracy of correlation may decrease as features with differences in the polarized component of the polarization beams change, and it is up to the user to determine the validity of the correlation result taking into consideration the two spectra being correlated. The differences in the features of the different spectra are often negligible when compared to the overall continuum of the spectra and are generally only reflected in a change in the intensity of said features when the continuum is removed.

## 3.4 General Reduction Procedure

This section aims to provide a comprehensive discussion of the modified reduction procedure, an example of which is provided in Appendix I. As users all employ a variety of



**Figure 3.14:** A general workflow for data reductions using a combination of `POLSLT`, `IRAF`, and the developed supplementary tools. Diagram adapted from Cooper et al. (2022).

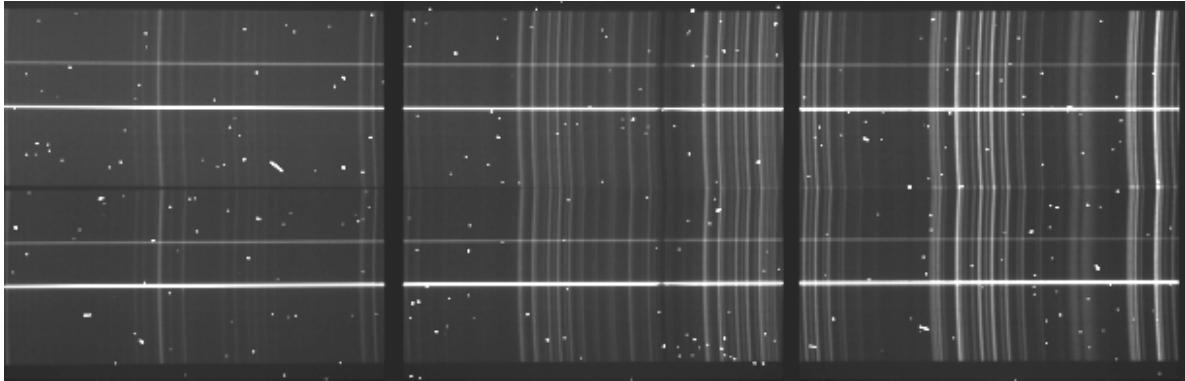
operating systems, language environments, and software setups, not much emphasis will be placed on how to get the software running or the managing of files: instead, the commands necessary to complete each step of the reduction process are discussed, assuming that the software is running as intended.

### 3.4.1 Initial Setup

It is important to note that while POLSALT was developed in Python 2.7, the STOPS supplementary tools were developed in, and require, Python 3.11+, as well as other requirements mentioned in § 3.3. While managing multiple versions of Python introduces some extra complication, it would not have been reasonable to develop STOPS in Python 2, as it is no longer supported, nor would it have been reasonable to update POLSALT to Python 3.

It is therefore recommended that the different versions of Python are managed using separate virtual environments. While the `anaconda` package manager was used in this study and is recommended, any package manager may be used. The `anaconda` environments are aliased ‘`salt`’ for Python 2.7 and ‘`stops`’ for Python 3.11. When Listings are provided (see for example Listing I.1 or the Listing in § 3.4.2 below), the `anaconda` environment is activated at the start of the Listing, otherwise it is assumed the previously specified environment is still active.

It is recommended to use POLSALT through the GUI as it provides a user-friendly environment while also sequentially listing each step of the reduction process in a dropdown menu, as seen in Figure 3.1. Reductions are possible, however, purely through the CLI using the POLSALT ‘beta’ scripts.



**Figure 3.15:** The science extension of a typical spectropolarimetric FITS file taken with the SALT RSS, after basic POLSALT CCD reductions have been completed. Figure created from the `STOPs split` output.

### 3.4.2 POLSALT Pre-Reductions

The POLSALT reduction process requires a file structure such that the raw data received from SALT is located in a folder named using the observing date with a sub-folder named `raw`, following the format `YYYYMMDD/raw/`. This directory structure allows POLSALT to create a ‘working’ directory following the format `YYYYMMDD/sci/` which contains all the files modified during the reduction process. Multiple reduction procedures using the same data may therefore be separated by simply renaming the `sci/` sub-folder.

The POLSALT GUI may be launched by opening a CLI and running the commands given in Listing I.1. Once the window, depicted in Figure 3.1, has launched, ensure that the first two paths at the top of the window point to the POLSALT and working directories, as seen in Figure 3.1. The ‘raw image reduction’ entry may then be selected from the dropdown menu and the pre-reductions run.

Alternatively, if the data already includes ‘mxgbp’ FITS files in the `YYYYMMDD/sci/` working directory, a CLI may be used to complete the initial pre-reductions using

---

```
$ cd <OBSDATE>/sci
$ conda activate salt
$ python ~/polsalt/scripts/reducepoldata_sc.py <OBSDATE>
```

---

which will start the full POLSALT reduction process. This process is quit once the POLSALT `wavelength calibration` GUI opens, and the alternate wavelength calibration procedure is followed.

### 3.4.3 Wavelength Calibration

The wavelength calibrations may now be completed in IRAF. This section concerns the procedure for parsing the FITS files to and from both IRAF and POLSALT, as well as the relevant task names and methods to be run to complete the calibrations. A base working case of each of the tasks and methods are presented in Listing I.2 to I.8, but it should be noted that the art of wavelength calibration consists of modifying the parameters to achieve a well-fit calibration function. This process depends heavily, and varies greatly, based on the user and as such not all use cases can be discussed.

## Preparing the Data for IRAF

Splitting the data is presented in Listing I.2. The STOPS `split` method may take multiple parameters, as seen in § 3.3, but default parameters should be used wherever possible. The most notable parameters are the directory, which defaults to the current working directory of the CLI, the split row, which defaults to POLSALT’s default center row, and the save prefix, which defaults to ‘`obeam`’ and ‘`ebeam`’.

## IRAF Wavelength Calibrations

The IRAF wavelength calibrations are performed using the tasks described in § 3.2, namely `identify`, `reidentify`, `fitcoords`, and optionally `transform`. In general, these tasks are run directly in the IRAF terminal using:<sup>18</sup>

```
cl> identify arc_files
cl> reidentify arc_ref arc_files
cl> fitcoords arc_files fit_2d
cl> transform files tr_file fit_2d
```

where ‘`arc_files`’ refers to a list or file containing the FITS files relevant to the task, ‘`arc_ref`’ refers to the FITS file previously identified, ‘`fit_2d`’ refers to the name to be used for the final two-dimensional wavelength solution, and ‘`tr_file`’ refers to the new name for the transformed input ‘`files`’.

The interactive tasks take up the bulk of the reduction time as this is where the fine-tuning of the reduction is done, through the use of cursor (or colon) commands, which allow modification of the parameters mid-reduction. Task parameters may, however, be edited beforehand within the IRAF terminal using the `eparam` task, and optionally saved, and quit or run using a combination of `:w`, and `:q` or `:go` cursor commands, respectively.

The reduction process in Appendix I, namely Listing I.4 to I.7, describes how to script the tasks for posterity. It is recommended to create an IRAF Command Language (cl) script for each task to keep track of which parameters were used and for simple recalibrations. The scripts are created using the `mkscript` task which interactively asks for a task to script and parameters to use. Multiple tasks may be appended to an IRAF script, allowing for the parameters of both beams to be tracked. Running an IRAF script may be done by running:

```
cl> cl < script_name.cl
```

but is not suggested for interactive scripts, which run best when simply copied from the `<...>/sci/script_name.cl` file to the IRAF terminal.

## Preparing the Data for POLSALT

After the wavelength calibrations have been completed, the wavelength solution is parsed back into the format expected by POLSALT. Joining the separate beams with their respective wavelength solutions is performed in the CLI following Listing I.8.

---

<sup>18</sup>Please see the IRAF help docs, available at [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/iraf.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/iraf.html), on the relevant tasks for a comprehensive discussion of the parameters available.

Similar to the `split` procedure, the `join` procedure has the same defaults defined. The onus of keeping track of any previously changed default parameters falls to the user, but logging is implemented in STOPS see § 3.3, which allows for later reference of any changed parameters.

### Sky Line Checks

The optional IRAF `transform` task and STOPS `skylines` method are used to confirm the wavelength solution across the frame (see § 3.3.3) by transforming and comparing known and observed sky line wavelength positions, respectively.

The `skyline` method is run in the CLI following Listing I.9. The difference in the flux conservation when `skyline` transforms the frames is discussed in § 3.3.3. Otherwise, as with the rest of STOPS, default parameters describe the overplotting behavior for the *O*- and *E*-beams, the skylines provided by SALT, and the calculated variation of the wavelength axis of a frame.

### Cross Correlation Checks

The `correlate` method is run in the CLI following Listing I.11. The input of the `correlate` method takes the output of the POLSALT `spectra extraction` and is thus only run thereafter, but is mentioned here as the completion of the POLSALT reductions is not discussed in much depth. If the user wishes to compare the *O*- and *E*-beams of a single file then only that image name is to be provided, otherwise it is assumed that the user wishes to compare the same polarization beam across each file provided.

### Cleaning Up the IRAF and STOPS Output

Before the final POLSALT reductions, it is recommended that the user ‘clean up’ the `sci/` directory of all IRAF and STOPS files since the ‘wmxgbp’ FITS files are all that is expected by POLSALT. The POLSALT methods use wildcard file collection and as such any errant detections of files added by the user will result in unexpected crashes. It is suggested to move any additional files to a new subfolder following Listing I.10, but a remove command may also be used, for example, following:

---

```
$ rm beamO* beamE* arcO* arcE*
```

---

## 3.4.4 POLSALT Reduction Completion

Reductions may now be completed using POLSALT. The reduction process consists of correcting for the wollaston tilt, extracting the spectra, creating the Stokes files, and displaying the results. The ‘beta’ version of POLSALT provides access to a GUI but may also be handled entirely through a CLI as scripts.

### POLSALT Beta in the GUI

The reduction process using the POLSALT GUI is completed by selecting and, when applicable, interactively modifying the reduction step through the interactive windows, one-by-one, from the GUI’s dropdown menu, as explained in Appendix I (p. 67 onwards).<sup>19</sup>

---

<sup>19</sup>See the official POLSALT wiki or alternative online resources such as the SALT workshop slides.

### POLSLT Beta through a CLI

Both GUI and CLI implementations of the POLSLT beta pipeline access the same script files. Although the GUI is more user-friendly, the CLI offers a more streamlined approach to the reduction process, allowing the reduction process to be automated once the IRAF wavelength solution is known and parsed into the ‘wmxgbp’ FITS file format. A modified version of the POLSLT beta `reducepoldata_sc.py` script (see Listing 3.7) is used to run the entire reduction process without needing to select which process to run next, using:

---

```
$ python reducepoldata_sc.py YYYYMMDD
```

---

where the only modification made to the `reducepoldata_sc.py` script file is the removal of a call to the `specpolwavmap` method.

**Listing 3.7:** The modified `reducepoldata_sc.py` script file.

```
import os, sys, glob, shutil
poldir = '/home/justin/polsalt-beta/'                                     # Will differ according to user
reddir=poldir+'polsalt/'
scrdir=poldir+'scripts/'
sys.path.append((reddir,scrdir,poldir))

datadir = reddir+'data/'
import numpy as np
from astropy.io import fits as pyfits
from specpolview import printstokes
from imred import imred
from specpolwavmap import specpolwavmap
from specpolextract_sc import specpolextract_sc
from specpolrawstokes import specpolrawstokes
from specpolfinalstokes import specpolfinalstokes

print sys.argv
obsdate = sys.argv[1]
print obsdate
os.chdir(obsdate)
if not os.path.isdir('sci'): os.mkdir('sci')
shutil.copy(scrdir+'script.py','sci')
os.chdir('sci')

#basic image reductions
infilelist = sorted(glob.glob('../raw/P*fits'))
imred(infilelist, '.', datadir+'bpm_rss_11.fits', crthresh=False, cleanup=True)

#basic polarimetric reductions
logfile='specpol'+obsdate+'.log'                                         # The following lines may be removed or commented out as below

# wavelength map
# infilelist = sorted(glob.glob('m*fits'))
# linelistlib=""
# specpolwavmap(infilelist, linelistlib=linelistlib, logfile=logfile)

#background subtraction and extraction
infilelist = sorted(glob.glob('wmxfits'))
extract = 10. # star +/-5, bkg= +/-(25-35)arcsec: 2nd order is 9-20 arcsec away
locate = (-120.,120.) # science target is brightest target in whole slit
#locate = (-20.,20.)

specpolextract_sc(infilelist,logfile=logfile,locate=locate,extract=extract)
#specpolextract_sc(infilelist,logfile=logfile,locate=locate,extract=extract,doconv=True,useoldc=True)

#raw stokes
infilelist = sorted(glob.glob('e*fits'))
specpolrawstokes(infilelist, logfile=logfile)

#final stokes
infilelist = sorted(glob.glob('*_h*.fits'))
specpolfinalstokes(infilelist, logfile=logfile)
```

---

The POLSLT beta `reducepoldata_sc.py` copies a `script.py` file into the science working directory, ‘YYYYMMDD/sci/’, which provides analysis scripts for analysis and modification of the POLSLT beta results. These tools consist of data culling for the final Stokes calculations, text and plot output, relative flux calibration corrections, and synthetic filtering of polarization results. The POLSLT analysis scripts may be run using:

---

```
$ python script.py
```

---

followed by `specpolfinalstokes.py`, `specpolview.py`, `specpolflux.py`, or `specpolfilter.py`, for the different analysis modes, respectively.<sup>20</sup>

---

<sup>20</sup>Please see <https://github.com/saltastro/polsalt/wiki/Linear-Polarization-Reduction---Beta-version> for a comprehensive discussion of the POLSALT beta analysis scripts.



# Chapter 4

## Testing and Application

Short intro to chapter contents.

- No POLSALT or data tests
  - POLSALT is trusted to be accurate. See ... (reference tests of POLSALT)
- Testing `split`
  - POLSALT to IRAF file structure conversion.
  - Show changes to data files are intended (I.E. only splitting the data, no changes to the data itself)
  - Tested over multiple grating/articulation angles to ensure robustness.
  - Mention any header updates skipped specific to POLSALT (if any)
  - Figure showing split fits file contents difference (cropped rows shown, etc.)
- Testing IRAF wavelength solution
  - IRAF is trusted to be accurate. See ... (reference tests of IRAF)
  - The `skylines` and `correlate` outputs are tests of the wavelength calibration.
    - \* Testing correlate functionality using ‘offset’, comparisons of arcs, FSRQ’s and BLLac’s.
    - \* Any figures showing correlation tests?
    - \* Testing skylines using known spectral sky lines.
    - \* Any figures showing skyline tests?
- Testing `join`
  - IRAF to POLSALT file structure conversion
  - Show changes to data files are intended (I.E. only joining the data, and ‘WAV’ appended. No changes to the data itself)
  - Wollaston correction of wavelength and bpm extensions
  - update to python 3 of ‘polsalt’ functions
  - Mention any header differences (if any)
  - Figure showing joined fits file contents difference (cropped rows shown once again, bpm differences due to CRR and NO wollaston bpm differences, etc.)
- Testing reduction results not negatively impacted by STOPS.
  - General discussion of testing (I.E. not this test was done specifically this source, more along the lines of these tests were done to check this issue, seen here using this source for example.)

- Wavelength solution validation from correlate and skylines results.
  - \* Figures showing Correlate and Skyline results.
  - \* RMS comparisons between POLSALT and IRAF to quantify differences.
  - \* Any Figures for RMS comparisons or wavelength validation?
- Polarization parameters validation from known sources.
  - \* Polarization tested using 3C 279, 4C+01.02, and preliminary testing data provided by David.
  - \* Polarization tested using spectropolarimetric standards (4 highly polarized, 2 non-polarized).
  - \* Tabulate sources used, with their properties.
  - \* Figures showing comparison plots of polarization parameters from POLSALT and STOPS.
- Background information on each object.
- Detailed reductions steps performed on each object.
- Comparison of POLSALT results to those obtained using the STOPS pipeline.
- Application to Spectropolarimetric Standards
  - Science results, what the results can tell us and why it is useful, also comparison of results to FORS1/2 published data, focus on the polarization results.
- Application in Publications
  - Summarize the results of the publications appended to appendix.

# Chapter 5

## Science Applications

**TODO:** Short introduction to chapter contents

### 5.1 Application to Spectropolarimetric Standards

**TODO:** Spectropolarimetric standards (4 highly polarized, 2 non-polarized)

- (Same as ch04 with science results)
- Background on objects
- Reductions
- Actual results - comparison of polsalt results to supplementary pipeline results
- Science results, what the results can tell us and why it is useful, also comparison of results to FORS1/2 published data, focus on the polarization results

### 5.2 Application in Publications

**TODO:** Summary of results from papers in appendix.

- Hester paper(s)
- Joleen proceedings and work
- My proceedings

**TODO:** 3C 279 and 4C+01.02

- Give Background on objects, Reduction steps, and Science results (what the results can tell us and why it is useful)
- (comparison of polsalt results to supplementary pipeline results will be in testing)



# Chapter 6

## Conclusions

**TODO:** A summary of the dissertation, main focus on the results and that the supplementary pipeline is a success since it allows an alternate method using IRAF to wavelength calibrate the polsalt data.

### 6.1 Future Work

**TODO:** Edit paragraph below to mention python wavelength solutions implemented to ‘future-proof’ the pipeline.

Another option to perform the wavelength calibration is Python which allows for a more modern and flexible approach, but is not discussed here. What will be discussed, however, is the structure of the wavelength solutions created through Python to be later reintroduced to the POLSALT pipeline. The solutions must be stored such that the ‘ $x$ ’ and ‘ $y$ ’ orders of the solution, as well as all the coefficients ( $C_{00}$  to  $C_{xy}$ ) making up the solution, separated by new lines, are included. The only limitations to the names of the solution files is that they must make mention of the specific  $O$ - or  $E$ -beam as well as the wavelength solution type (e.g. ‘Chebyshev’, ‘Legendre’, etc.).



# Appendix I

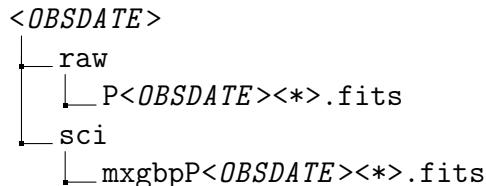
## The Modified Reduction Process

This section of the Appendix aims to provide a minimum working example of the commands necessary to reduce POLSALT data using STOPS and IRAF. It contains the commands necessary to activate all software and run through the reduction process but makes no attempt at discussion.

Both POLSALT and IRAF are launched from the default CLI but use independent interfaces during the reduction process. To distinguish which window is in focus, the ‘\$’ token is used for default CLI commands while the ‘c1>’ and ‘>>>’ tokens are used for IRAF’s ‘xgterm’ single- and multi-line commands, respectively.

General instructions for the reduction process which might not necessarily be line-fed commands passed to a CLI may either be discussed outside a ‘Listing’ environment or included as part of the ‘Listing’ environment with a preceding ‘#’ token. Finally, POLSALT implements a GUI and thus takes no line-fed commands. As such, the instructions when using the POLSALT GUI follow those of the general instructions with the added exception that they relate to the GUI.

As a final note, some parameters are distinguished using an ‘<angle brackets>’ notation. They signify necessary parameters that may vary from reduction to reduction. Notable uses of this notation include the date of observation,  $\langle OBSDATE \rangle$  (formatted ‘YYYYMMDD’), the split science FITS files,  $\langle O\text{-beam FILES} \rangle$  or  $\langle E\text{-beam FILES} \rangle$ , the split arc FITS files,  $\langle O\text{-beam ARC} \rangle$  or  $\langle E\text{-beam ARC} \rangle$ , and a wildcard symbol,  $\langle * \rangle$ .



**Figure I.1:** The typical minimal file structure of data provided by SALT.

Ensure the data is formatted in a file structure similar to that in Figure I.1. Data located in the ‘sci’ folder is often provided by SALT but is not strictly necessary to begin the reduction process. If ‘mxgbp’ prefixed data is available, the reductions may be begun starting at Listing I.2. The POLSALT GUI is launched from the default CLI running the commands in Listing I.1.

**Listing I.1:** Launching the POLSALT GUI

---

```
$ cd ~/polsalt
$ conda activate salt
$ python -W ignore reducepoldataGUI.py &
```

---

Refer to Figure 3.1 for a depiction of the POLSALT GUI. To complete the POLSALT pre-calibrations, and with the GUI in focus:

- Ensure that the ‘POLSALT code directory’ is correct.
- Set the ‘Top level data directory’ to  $\langle OBSDATE \rangle$ .
- Ensure ‘Raw data directory’ is correct.
- Ensure ‘Science data directory’ is correct.
- Select ‘Raw image reduction’ from the ‘Data reduction step’ drop down menu.
- Check the tick boxes of all raw images to be processed (include the arc) in the display box covering the lower half of the GUI.
- Proceed with the reductions by clicking the ‘OK’ button.

The pre-calibrated ‘mxgbp’ FITS files are now available in the ‘sci’ folder. The files may be split using STOPS by running the commands in Listing I.2.

**Listing I.2:** Splitting data using STOPS

---

```
$ cd <OBSDATE>/sci
$ conda activate stops
$ python ~/STOPS . split
```

---

The split *O*- and *E*-beam FITS files are now available. The IRAF wavelength calibrations are now run. The IRAF xgterm CLI is launched using Listing I.3.

**Listing I.3:** Launching IRAF in xgterm

---

```
$ cd ~/iraf
$ xgterm -sb &
cl> conda activate salt
cl> cl
cl> noao
cl> twodspec
cl> longslit
cl> unlearn longslit
cl> longslit.dispaxis=1
```

---

The IRAF `identify` task requires an average feature width, ‘fwidth’, as a parameter. The width of a feature may be found in IRAF using the `implot` task<sup>1</sup> along with cursor

---

<sup>1</sup>See [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/plot.implot.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/plot.implot.html) for documentation on the `implot` task.

commands, but may also be found using FITS viewing software, such as `ds9`.<sup>2</sup> The `identify` task may be run using the commands in Listing I.4.

**Listing I.4:** Running the IRAF `identify` task

---

```
cl> mkscript 01_identify.cl
cl> # Add identify to 01_identify.cl twice, for both beams
cl> # Edit the parameters of 01_identify.cl in a text editor
cl> # Paste an identify script into the CLI, resulting in:
cl>
cl> identify ("<O-beam ARC>",
>>> "", "", section="middle line", database="database",
>>> coordlist="linelists$idheneare.dat", units="", nsum="10", match=-3.,
>>> maxfeatures=50, zwidth=100., ftype="emission", fwidth=8.,
>>> cradius=5., threshold=0., minsep=2., function="spline3", order=2,
>>> sample="*", niterate=0, low_reject=3., high_reject=3., grow=0.,
>>> autowrite=no, graphics="stdgraph", cursor="", aidpars="")
```

---

The `identify` task will launch an interactive window. Cursor commands refer to keys that provide unique functionality to the interactive IRAF tasks. The cursor commands for `identify` allow the arc lines to be identified using ‘m’ (and typing the relevant wavelength), while ‘d’ and ‘i’ will delete a single and all identified arc lines, respectively. The ‘f’ cursor command will perform a preliminary fit which can be quit using the ‘q’ cursor command. The ‘l’ cursor command will attempt to identify any unidentified arc lines. Once complete, a figure of the identified lines may be saved using ‘:labels coord’ and ‘:.snap eps’, and the task safely quit with the ‘q’ cursor command.<sup>3</sup> The `identify` procedure is repeated, replacing `<O-beam ARC>` with `<E-beam ARC>`.

The `reidentify` task may be run using the commands in Listing I.5.

**Listing I.5:** Running the IRAF `reidentify` task

---

```
cl> mkscript 02_reidentify.cl
cl> # Add reidentify to 02_reidentify.cl twice, for both beams
cl> # Edit the parameters of 02_reidentify.cl in a text editor
cl> # Paste a reidentify script into the CLI, resulting in:
cl>
cl> reidentify ("<O-beam ARC>",
>>> "<O-beam ARC>", "yes", "", "", interactive="no", section="middle
>>> line", newaps=yes, override=no, refit=yes, trace=yes, step="10",
>>> nsum="10", shift="0.", search=0., nlost=0, cradius=5.,
>>> threshold=0., addfeatures=no, coordlist="linelists$idheneare.dat",
>>> match=-3., maxfeatures=50, minsep=2., database="database",
>>> logfiles="logfile", plotfile="", verbose=yes, graphics="stdgraph",
>>> cursor="", aidpars="")
```

---

The `reidentify` task will run autonomously so long as the `interactive` parameter is set to “no”.<sup>4</sup> Repeat the `reidentify` procedure, replacing `<O-beam ARC>` with `<E-beam ARC>` at both the ‘reference’ and ‘image’ parameter locations.

<sup>2</sup>See <https://sites.google.com/cfa.harvard.edu/saoimageds9> for documentation on the `ds9` software.

<sup>3</sup>See [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/noao.onedspec.identify.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.onedspec.identify.html) for documentation on the `identify` task.

<sup>4</sup>See [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/noao.onedspec.reidentify.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.onedspec.reidentify.html) for documentation on the `reidentify` task.

The `fitcoords` task may be run using the commands in Listing I.6.

**Listing I.6:** Running the `IRAF fitcoords` task

---

```
cl> mkscript 03_fitcoords.cl
cl> # Add fitcoords to 03_fitcoords.cl twice, for both beams
cl> # Edit the parameters of 03_fitcoords.cl in a text editor
cl> # Paste a fitcoords script into the CLI, resulting in:
cl>
cl> fitcoords ("<0-beam ARC> (exclude the file extension)",
>>> fitname="", interactive=yes, combine=no, database="database",
>>> deletions="deletions.db", function="chebyshev", xorder=6, yorder=6,
>>> logfiles="STDOUT,logfile", plotfile="plotfile",
>>> graphics="stdgraph", cursor="")
```

---

The `fitcoords` task will launch an interactive window. The x- and y-axis being plotted may be changed using the ‘x’ or ‘y’ cursor commands followed by the desired data axis (‘x’ for the x-axis, ‘y’ for the y-axis, or ‘r’ for the residuals).<sup>5</sup> Repeat the `fitcoords` procedure, replacing  $<O\text{-beam } ARC>$  with  $<E\text{-beam } ARC>$ .

The `transform` task may be run using the commands in Listing I.7.

**Listing I.7:** Running the `IRAF transform` task

---

```
cl> mkscript 04_transform.cl
cl> # Add transform to 04_transform.cl twice, for both beams
cl> # Edit the parameters of 04_transform.cl in a text editor
cl> # Paste a transform script into the CLI, resulting in:
cl>
cl> transform ("@<0-beam FILES>",
>>> "t//@<0-beam FILES>", "<0-beam ARC> (exclude the file extension)",
>>> minput="", moutput="", database="database", interptype="linear",
>>> x1="INDEF", x2="INDEF", dx="INDEF", nx="INDEF", xlog="no",
>>> y1="INDEF", y2="INDEF", dy="INDEF", ny="INDEF", ylog="no",
>>> flux="yes", blank="INDEF", logfiles="STDOUT,logfile")
```

---

Inspect the transformed images, most notably the arc images, using any FITS viewer as a cursory check that the wavelength calibrations were completed without error.<sup>6</sup>

The ‘gain’ and ‘read noise’ are now needed as the cosmic-ray rejection of the STOPS `join` method accepts them as parameters. These parameters may be found using the ‘*GAINSET*’ and ‘*ROSPEED*’ keywords in the FITS headers. The cosmic ray rejection defaults to *GAINSET*=‘FAINT’, and *ROSPEED*=‘SLOW’. If the gain and read noise values differ from the defaults, the parameters should be updated when running `join`.<sup>7</sup>

<sup>5</sup>See [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/noao.twodspec.longslit.fitcoords.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.twodspec.longslit.fitcoords.html) for documentation on the `fitcoords` task.

<sup>6</sup>See [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/noao.twodspec.longslit.transform.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.twodspec.longslit.transform.html) for documentation on the `transform` task.

<sup>7</sup>The read noise and gain may be determined from [http://pysalt.salt.ac.za/proposal\\_calls/current/ProposalCall.html](http://pysalt.salt.ac.za/proposal_calls/current/ProposalCall.html), specifically Table 6.1 and Table 6.2.

The STOPS `join` method may be run using the commands in Listing I.8.

**Listing I.8:** Joining the data using STOPS

---

```
$ cd <OBSDATE>/sci
$ conda activate stops
$ python ~/STOPS . join
```

---

The STOPS `skylines` method may be run on any ‘joined’ or transformed FITS files,  $\langle FILE(S) \rangle$ , using the commands in Listing I.9.

**Listing I.9:** Running the STOPS `skylines` method

---

```
$ cd <OBSDATE>/sci
$ conda activate stops
$ python ~/STOPS . skylines <FILE(S)>
```

---

The ‘`sci/`’ directory may now be slightly organized by running the commands in Listing I.10, moving all the files relevant to the wavelength calibrations into either the ‘`database`’ or ‘`split_files`’ directories.

**Listing I.10:** Directory cleanup for POLSALT

---

```
$ cd <OBSDATE>/sci
$ mkdir split_files
$ mv *beam0* *beamE* *arc0* *arcE* split_files/
$ mv *.eps *.cl *.db database/
```

---

The POLSALT `spectra extraction` is now run. If the POLSALT GUI was closed it should now be reopened using Listing I.1. With the GUI in focus:

- Ensure all directories are still correct.
- Select ‘Spectra extraction’ from the ‘Data reduction step’ drop down menu.
- Check the tick boxes of all wavelength calibrated images to be processed (exclude the arc) in the display box covering the lower half of the GUI.
- Proceed with the reductions by clicking ‘OK’.

The POLSALT `spectra extraction` is interactive and will launch a separate GUI for the background subtraction and spectral extraction (see Figure 3.2). The background and spectral regions to be extracted may be adjusted, noting that adjustments affect both *O*- and *E*-beams. Once both background regions contain no trace and the spectral region fully contains only the science trace, the reduction may be completed by clicking ‘OK’.

The STOPS `correlate` method may now be run on any ‘joined’ FITS files by running the commands in Listing I.11.

**Listing I.11:** Running the STOPS `correlate` method

---

```
$ cd <OBSDATE>/sci
$ conda activate stops
$ python ~/STOPS . correlate <FILE(S)>
```

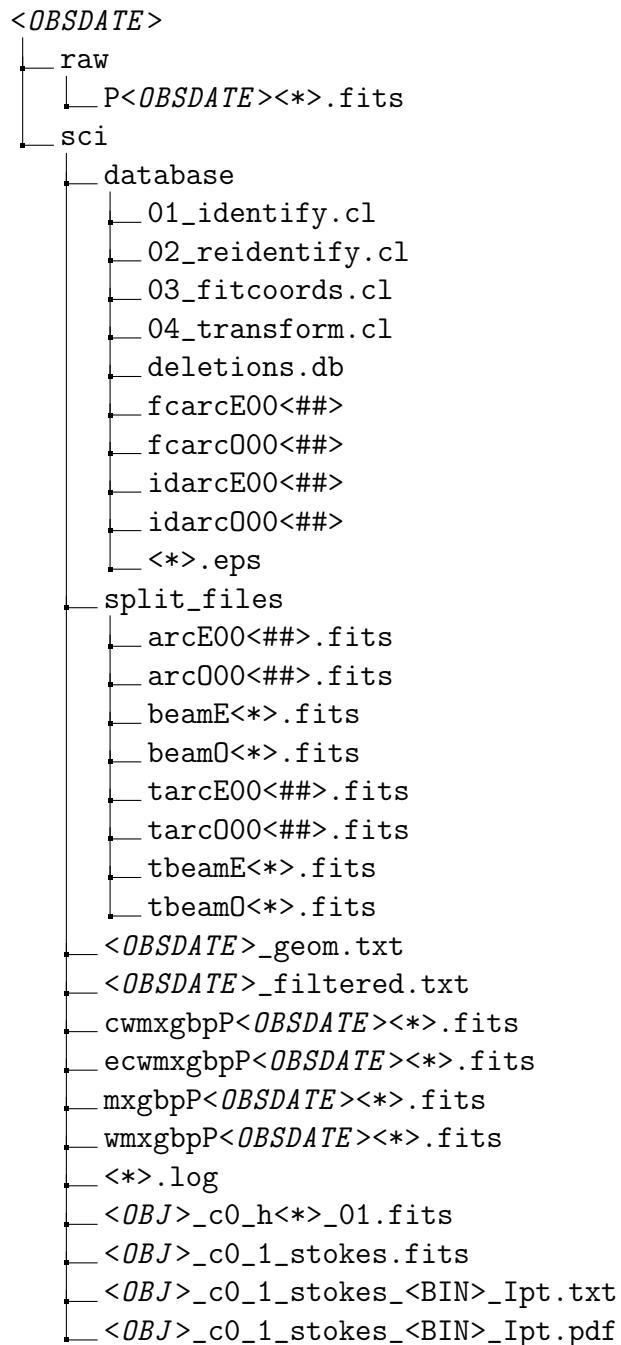
---

The POLSALT **raw Stokes calculation**, **final Stokes calculation**, and **results visualisation** may now be completed. For the last time, if the POLSALT GUI was closed it should now be reopened using Listing I.1. With the GUI in focus:

- Ensure all directories are still correct.
- Select ‘Raw Stokes calculation’ from the ‘Data reduction step’ drop down menu.
- Check the tick boxes of all the extracted spectra images to be processed in the display box covering the lower half of the GUI.
- Proceed with the **raw Stokes calculation** by clicking ‘OK’.
- Select ‘Final Stokes calculation’ from the ‘Data reduction step’ drop down menu.
- Check the tick boxes of all the “raw Stokes” images to be processed in the display box covering the lower half of the GUI.
- Proceed with the **Final Stokes calculation** by clicking ‘OK’.
- Select ‘Results visualisation - interactive’ from the ‘Data reduction step’ drop down menu.
- Check the tick boxes of the “final Stokes” image to be visualized in the display box covering the lower half of the GUI.
- Proceed with the **visualisation** by clicking ‘OK’.

The POLSALT **visualisation** is interactive and will launch a separate GUI (See Figure 3.3). The GUI may be used to change the binning and parameters of the plot before saving the plot to a PDF file.

This concludes the minimum working example of the POLSALT reduction process when substituting the POLSALT **wavelength calibrations** with those done in IRAF. Aside from the final results, the file structure after reductions should resemble something akin to that provided in Figure I.2.



**Figure I.2:** The typical file structure after completing the reduction process.



# Appendix II

## STOPS Source Code

This section of Appendix includes all the major STOPS source code files related to the reduction process. Files such as those related to python initialization, testing directories, and other non-essential modules have been excluded for brevity and clarity.

**Listing II.1:** The source code for `__main__.py`

```
1 """Argument parser for STOPS."""
2
3 #!/usr/bin/env python3
4 # -*- coding: utf-8 -*-
5
6 from __init__ import __version__, __author__, __email__
7
8 # MARK: Imports
9 import os
10 import sys
11 import argparse
12 import logging
13 from pathlib import Path
14
15 from split import Split
16 from join import Join
17 from cross_correlate import CrossCorrelate
18 from skylines import Skylines
19
20 from utils import ParserUtils as pu
21 from utils.Constants import SPLIT_ROW, PREFIX, PARSE, SAVE_CORR,
22   ↪ SAVE_SKY
23
24 # MARK: Constants
25 PROG = "STOPS"
26 DESCRIPTION = """
27 Supplementary Tools for Polsalt Spectropolarimetry (STOPS) is a
28 collection of supplementary tools created for SALT's POLSALT pipeline,
29 allowing for wavelength calibrations with IRAF. The tools provide
30 support for splitting and joining polsalt formatted data as well as
31 cross correlating complementary polarimetric beams.
32 DOI: 10.22323/1.401.0056
```

```

33 """
34
35
36 # MARK: Universal Parser
37 parser = argparse.ArgumentParser(
38     prog=PROG,
39     description=DESCRIPTION,
40     formatter_class=argparse.RawDescriptionHelpFormatter,
41 )
42 parser.add_argument(
43     "-V",
44     "--version",
45     action="version",
46     version=f"%(prog)s as of {__version__}",
47 )
48 parser.add_argument(
49     "-v",
50     "--verbose",
51     action="count",
52     default=PARSE['VERBOSE'],
53     help=(
54         "Counter flag which enables and increases verbosity. "
55         "Use -v or -vv for greater verbosity levels."
56     ),
57 )
58 parser.add_argument(
59     "-l",
60     "--log",
61     action="store",
62     type=pu.parse_logfile,
63     help=(
64         "Filename of the logging file. "
65         "File is created if it does not exist. Defaults to None."
66     ),
67 )
68 parser.add_argument(
69     "data_dir",
70     action="store",
71     nargs="?",
72     default=PARSE['DATA_DIR'],
73     type=pu.parse_path,
74     help=(
75         "Path of the directory which contains the working data. "
76         f"Defaults to the cwd -> '{PARSE['DATA_DIR']}' (I.E. '.')."
77     ),
78 )
79
80
81 # MARK: Split\Join Parent
82 split_join_args = argparse.ArgumentParser(add_help=False)
83 split_join_args.add_argument(
84     "-n",
85     "--no_arc",
86     action="store_true",
87     help="Flag to exclude arc files from processing.",
88 )
89 split_join_args.add_argument(
90     "-s",

```

```

91     "--split_row",
92     default=SPLIT_ROW,
93     type=int,
94     help=(
95         "Row along which the O and E beams are split. "
96         f"Defaults to polsalt's default -> {SPLIT_ROW}.""
97     ),
98 )
99 split_join_args.add_argument(
100    "-p",
101    "--save_prefix",
102    nargs=2,
103    default=PREFIX,
104    help=(
105        "Prefix appended to the filenames, "
106        "with which the O and E beams are saved. "
107        f"Defaults to {PREFIX}.""
108    ),
109 )
110
111
112 # MARK: Correlate\Skylines Parent
113 corr_sky_args = argparse.ArgumentParser(add_help=False)
114 corr_sky_args.add_argument(
115    "filenames",
116    action="store",
117    nargs="+",
118    type=pu.parse_file,
119    help=(
120        "File name(s) of FITS file(s) to be processed."
121        "A minimum of one filename is required."
122    ),
123 )
124 corr_sky_args.add_argument(
125    "-b",
126    "--beams",
127    choices=["O", "E", "OE"],
128    type=str.upper,
129    default=PARSE['BEAMS'],
130    help=(
131        "Beams to process. "
132        f"Defaults to {PARSE['BEAMS']}, but "
133        "may be given 'O', 'E', or 'OE' to "
134        "determine which beams are processed."
135    ),
136 )
137 corr_sky_args.add_argument(
138    "-ccd",
139    "--split_ccd",
140    action="store_false",
141    help=(
142        "Flag to NOT split CCD's. "
143        "Recommended to leave off unless the chip gaps "
144        "have been removed from the data."
145    ),
146 )
147 corr_sky_args.add_argument(
148    "-c",

```

```

149     "--continuum_order",
150     type=int,
151     default=PARSE['CONT_ORD'],
152     dest="cont_ord",
153     help=(
154         "Order of continuum to remove from spectra. "
155         "Higher orders recommended to remove most variation, "
156         "leaving only significant features."
157     ),
158 )
159 corr_sky_args.add_argument(
160     "-p",
161     "--plot",
162     action="store_true",
163     help="Flag for additional plot outputs.",
164 )
165 corr_sky_args.add_argument(
166     "-s",
167     "--save_prefix",
168     action="store",
169     nargs="?",
170     type=lambda path: Path(path).expanduser().resolve(),
171     const=SAVE_CORR,
172     help=(
173         "Prefix used when saving plot. "
174         "Excluding flag does not save output plot, "
175         f"flag usage of option uses default prefix, "
176         "and a provided prefix overwrites default prefix."
177     ),
178 )
179
180
181 # MARK: Create subparser modes
182 subparsers = parser.add_subparsers(
183     dest="mode",
184     help="Operational mode of supplementary tools",
185 )
186
187
188 # MARK: Split Subparser
189 split_parser = subparsers.add_parser(
190     "split",
191     aliases=["s"],
192     help="Split mode",
193     parents=[split_join_args],
194 )
195 # 'children' split args here
196 # Change defaults here
197 split_parser.set_defaults(
198     mode="split",
199     func=Split,
200 )
201
202
203 # MARK: Join Subparser
204 join_parser = subparsers.add_parser(
205     "join",
206     aliases=["j"],

```

```

207     help="Join mode",
208     parents=[split_join_args],
209 )
210 # 'children' join args here
211 join_parser.add_argument(
212     "-c",
213     "--coefficients",
214     dest="solutions_list",
215     nargs='*',
216     type=pu.parse_file,
217     help=(
218         "Custom coefficients to use instead of the `IRAF` fitcoords "
219         "database. Use as either '-c <o_solution> <e_solution>' or "
220         "a regex descriptor '-c <*solution*extention>'."
221     ),
222 )
223 # Change defaults here
224 join_parser.set_defaults(
225     mode="join",
226     func=Join,
227 )
228
229
230 # MARK: Correlate Subparser
231 corr_parser = subparsers.add_parser(
232     "correlate",
233     aliases=["x"],
234     help="Cross correlation mode",
235     parents=[corr_sky_args],
236 )
237 # 'children' join args here
238 corr_parser.add_argument(
239     "-o",
240     "--offset",
241     type=int,
242     default=PARSE['OFFSET'],
243     help=(
244         "Introduces an offset when correcting for "
245         "known offset in spectra or for testing purposes. "
246         f"Defaults to {PARSE['OFFSET']}. "
247         "(For testing, not used during regular operation.)"
248     ),
249 )
250 # Change defaults here
251 corr_parser.set_defaults(
252     mode="correlate",
253     func=CrossCorrelate,
254 )
255
256
257 # MARK: Skyline Subparser
258 sky_parser = subparsers.add_parser(
259     "skylines",
260     aliases=["sky"],
261     help="Sky line check mode",
262     parents=[corr_sky_args],
263 )
264 # 'children' skyline args here

```

```

265 sky_parser.add_argument(
266     "-t",
267     "--transform",
268     action="store_false",
269     help=(
270         "Flag to force transform images. "
271         "Recommended to use only when input image(s) "
272         "are prefixed 't' but are not yet transformed."
273     ),
274 )
275 # Change defaults here
276 sky_parser.set_defaults(
277     mode="skyline",
278     func=Skylines,
279 )
280
281
282 # MARK: Keyword Clean Up
283 args = parser.parse_args()
284
285 if len(sys.argv) == 1:
286     parser.print_help(sys.stderr)
287     sys.exit(2)
288
289 args.verbose = pu.parse_loglevel(args.verbose)
290
291 if 'log' in args and args.log not in [None]:
292     args.log = args.data_dir / args.log
293
294 if "filenames" in args:
295     args.filenames = pu.flatten(args.filenames)
296
297 if "solutions_list" in args and type(args.solutions_list) == list:
298     args.solutions_list = pu.flatten(args.solutions_list)
299
300 # MARK: Begin logging
301 logging.basicConfig(
302     filename=args.log,
303     format"%(asctime)s - %(module)s - %(levelname)s - %(message)s",
304     datefmt="%Y-%m-%d %H:%M:%S",
305     level=args.verbose,
306 )
307
308 # MARK: Call Relevant Class(Args)
309 logging.debug(f"Argparse namespace: {args}")
310 logging.info(f"Mode:{args.mode}")
311 args.func(**vars(args)).process()
312
313
314 # Confirm all processes completed and exit without error
315 logging.info("All done! Come again!\n")

```

---

**Listing II.2:** The source code for `split.py`

```

51     -----
52     arc : str
53         Name of arc FITS file within `data_dir`.
54         `arc` = `""` if `no_arc` or not detected in `data_dir`.
55     o_files, e_files : list[str]
56         A list of the `O`- and `E`-beam FITS file names.
57         The first entry is the arc file if `arc` defined.
58     data_dir
59     fits_list
60     split_row
61     save_prefix
62
63     Methods
64     -----
65     split_file(file: os.PathLike)
66         -> tuple[astropy.io.fits.HDUList]
67             Handles creation and saving the separated FITS files
68     split_ext(hdulist: astropy.io.fits.HDUList, ext: str = 'SCI')
69         -> astropy.io.fits.HDUList
70             Splits the data in the `ext` extension along the `split_row`.
71     crop_file(hdulist: astropy.io.fits.HDUList, crop: int =
72         ↪ CROP_DEFAULT (See Notes))
73         -> tuple[numpy.ndarray]
74             Crops the data along the edge of the frame, that is,
75             `O`-beam cropped as [crop:], and
76             `E`-beam cropped as [: - crop].
77     update_beam_lists(o_name: str, e_name: str)
78         -> None
79             Updates `o_files` and `e_files`.
80     save_beam_lists(file_suffix: str = 'frames')
81         -> None
82             Creates (Overwrites if exists) and writes the `o_files` and `e_files` to files named
83             `o_{file_suffix}` and `e_{file_suffix}`, respectively.
84     process()
85         -> None
86             Calls `split_file` and `save_beam_lists` on each file in `fits_list` for automation.
87
88     Other Parameters
89     -----
90     **kwargs : dict
91         keyword arguments. Allows for passing unpacked dictionary to
92             ↪ the class constructor.
93
94     Notes
95     -----
96     Constants Imported (See utils.Constants):
97         SAVE_PREFIX
98         CROP_DEFAULT
99         SPLIT_ROW
100
101     """
102     # -----split1-----
103     # MARK: Split init
104     def __init__(


```

```

105     self,
106     data_dir: Path,
107     fits_list: list[str] = None,
108     split_row: int = SPLIT_ROW,
109     no_arc: bool = False,
110     save_prefix: Path | None = None,
111     **kwargs
112 ) -> None:
113     self.data_dir = data_dir
114     self.fits_list = find_files(
115         data_dir=data_dir,
116         filenames=fits_list,
117         prefix="mxgbp",
118         ext="fits"
119     )
120     self.split_row = split_row
121     self.save_prefix = SAVE_PREFIX
122     if type(save_prefix) == dict:
123         self.save_prefix = save_prefix
124
125     self.arc = "" if no_arc else find_arc(self.fits_list)
126     self.o_files = []
127     self.e_files = []
128
129     logging.debug(f"__init__ - \n{self.__dict__}")
130     return
131
132 # MARK: Split Files
133 def split_file(
134     self,
135     file: os.PathLike
136 ) -> tuple[pyfits.HDUList]:
137     """
138         Split the single FITS file into separated `O`- and `E`- FITS
139         files.
140
141     Parameters
142     -----
143     file : os.PathLike
144         The name of the FITS file to be split.
145
146     Returns
147     -----
148     tuple[astropy.io.fits.HDUList]
149         Tuple containing the split O and E beam HDULists.
150
151     """
152     # Create empty HDUList
153     O_beam = pyfits.HDUList()
154     E_beam = pyfits.HDUList()
155
156     # Open file and split O & E beams
157     with pyfits.open(file) as hdul:
158         O_beam.append(hdul["PRIMARY"].copy())
159         E_beam.append(hdul["PRIMARY"].copy())
160
161         # Split specific extention
162         raw_split = self.split_ext(hdul, "SCI")

```

```

162
163     # O_beam[0].data = raw_split['SCI'].data[1]
164     # E_beam[0].data = raw_split['SCI'].data[0]
165     O_beam[0].data, E_beam[0].data = self.crop_file(raw_split)
166
167     # Handle prefix and names
168     pref = "arc" if file == self.arc else "beam"
169     o_name = self.save_prefix[pref][0] + file.name[-9:]
170     e_name = self.save_prefix[pref][1] + file.name[-9:]
171
172     # Add split data to O & E beam lists
173     self.update_beam_lists(o_name, e_name, pref == "arc")
174
175     # Handle don't save case
176     if self.save_prefix == None:
177         return O_beam, E_beam
178
179     # Handle save case
180     O_beam.writeto(o_name, overwrite=True)
181     E_beam.writeto(e_name, overwrite=True)
182
183     return O_beam, E_beam
184
185     # MARK: Split extensions
186 def split_ext(
187     self,
188     hdulist: pyfits.HDUList,
189     ext: str = "SCI"
190 ) -> pyfits.HDUList:
191     """
192         Split the data of the specified extension of `hdulist` into its
193         → `O`- and `E`- beams.
194
195         Parameters
196         -----
197         hdulist : astropy.io.fits.HDUList
198             The FITS HDUList to be split.
199         ext : str, optional
200             The name of the extension to be split.
201             (Defaults to 'SCI')
202
203         Returns
204         -----
205         astropy.io.fits.HDUList
206             The HDUList with the split applied.
207
208         """
209         hdu = deepcopy(hdulist)
210         rows, cols = hdu[ext].data.shape
211
212         # if odd number of rows, strip off the last one
213         rows = int(rows / 2) * 2
214
215         # how far split is from center of detector
216         offset = int(self.split_row - rows / 2)
217
218         # split arc into o/e images
219         ind_rc = np.indices((rows, cols))[0]

```

```

219     padbins = (ind_rc < offset) | (ind_rc > rows + offset)
220
221     # Roll split_row to be centre row
222     image_rc = np.roll(hdu[ext].data[:rows, :], -offset, axis=0)
223     image_rc[padbins] = 0.0
224
225     # Split columns equally
226     hdu[ext].data = image_rc.reshape((2, int(rows / 2), cols))
227
228     return hdu
229
230 # MARK: Crop files
231 def crop_file(
232     self,
233     hdulist: pyfits.HDUList,
234     crop: int = CROP_DEFAULT
235 ) -> tuple[np.ndarray]:
236     """
237         Crop the data with respect to the `O`/`E` beam.
238
239     Parameters
240     -----
241     hdulist : astropy.io.fits.HDUList
242         The HDUList containing the data to be cropped.
243     crop : int, optional
244         The number of rows to be cropped from the bottom and top
245         of the `O` and `E` beam, respectively.
246         (Defaults to 40)
247
248     Returns
249     -----
250     tuple[numpy.ndarray]
251         Tuple containing the cropped O and E beam data arrays.
252
253     """
254     o_data = hdulist["SCI"].data[1, 0:-crop]
255     e_data = hdulist["SCI"].data[0, crop:]
256
257     return o_data, e_data
258
259 # MARK: Update beam lists
260 def update_beam_lists(
261     self,
262     o_name,
263     e_name,
264     arc: bool = True
265 ) -> None:
266     """
267         Update the `o_files` and `e_files` attributes.
268
269     Parameters
270     -----
271     o_name : str
272         The filename of the O beam.
273     e_name : str
274         The filename of the E beam.
275     arc : bool, optional
276         Indicates whether the first entry should be the arc frame.

```

```

277             (Defaults to True)
278
279         Returns
280         -----
281         None
282
283         """
284         if arc:
285             self.o_files.insert(0, o_name)
286             self.e_files.insert(0, e_name)
287         else:
288             self.o_files.append(o_name)
289             self.e_files.append(e_name)
290
291     return
292
293 # MARK: Save beam lists
294 def save_beam_lists(self, file_suffix: str = 'frames') -> None:
295     with open(f"o_{file_suffix}", "w+") as f_o, \
296         open(f"e_{file_suffix}", "w+") as f_e:
297         for i, j in zip(self.o_files, self.e_files):
298             f_o.write(i + "\n")
299             f_e.write(j + "\n")
300
301     return
302
303 # MARK: Process all Listed Images
304 def process(self) -> None:
305     """
306         Process all FITS images stored in the `fits_list` attribute
307
308     Returns
309     -----
310     None
311
312     """
313     for target in self.fits_list:
314         logging.debug(f"Processing {target}")
315         self.split_file(target)
316
317     self.save_beam_lists()
318
319     return
320
321 # MARK: Main function
322 def main(argv) -> None:
323     """Main function."""
324
325     return
326
327
328 if __name__ == "__main__":
329     main(sys.argv[1:])

```

**Listing II.3:** The source code for `join.py`

```

1 """Module for joining the split FITS files with an external wavelength
2     solution."""
3
4 #!/usr/bin/env python3
5 # -*- coding: utf-8 -*-
6
7 from __init__ import __author__, __email__, __version__
8
9 # MARK: Imports
10 import os
11 import sys
12 import logging
13 import re
14 from pathlib import Path
15
16 import numpy as np
17 from numpy.polynomial.chebyshev import chebgrid2d as chebgrid2d
18 from numpy.polynomial.legendre import leggrid2d as leggrid2d
19 from astropy.io import fits as pyfits
20
21 # from lacosmic import lacosmic # Replaced: ccdproc is ~6x faster
22 from ccdproc import cosmicray_lacosmic as lacosmic
23
24 from utils.specpolpy3 import read_wollaston, split_sci
25 from utils.SharedUtils import find_files, find_arc
26 from utils.Constants import DATADIR, SAVE_PREFIX, SPLIT_ROW, CR_PARAMS
27
28 # MARK: Join Class
29 class Join:
30
31     #-----join0-----
32
33     """
34     The `Join` class allows for the joining of previously
35     split files and the appending of an external wavelength
36     solution in the `polsalt` FITS file format.
37
38     Parameters
39     -----
40     data_dir : str
41         The path to the data to be joined
42     database : str, optional
43         The name of the `IRAF` database folder.
44         (The default is "database")
45     fits_list : list[str], optional
46         A list of pre-reduced `polsalt` FITS files to be joined within `data_dir`.
47         (The default is ``None``, `Join` will search for `mxbp*.fits` files)
48     solutions_list: list[str], optional
49         A list of solution filenames from which the wavelength solution
50         is created.
51         (The default is ``None``, `Join` will search for `fc*` files
52         within the `database` directory)
53     split_row : int, optional

```

```

52     The row along which the data of each extension in the FITS file
53     ↪ was split.
54     Necessary when Joining cropped files.
55     (The default is 517, the SALT RSS CCD's middle row)
56     save_prefix : dict[str, list[str]], optional
57     The prefix with which the previously split `O`- & `E`-beams
58     ↪ were saved.
59     Used for detecting if cropping was applied during the splitting
60     ↪ procedure.
61     (The default is SAVE_PREFIX (See Notes))
62     verbose : int, optional
63     The level of verbosity to use for the Cosmic ray rejection
64     (The default is 30, I.E. logging.INFO)
65
66     Attributes
67     -----
68     fc_files : list[str]
69     Valid solutions found from `solutions_list`.
70     custom : bool
71     Internal flag for whether `solutions_list` uses the `IRAF` or a
72     ↪ custom format.
73     See Notes for custom solution formatting.
74     (Default (inherited from `solutions_list`) is False)
75     arc : str
76     Deprecated. Name of arc FITS file within `data_dir`.
77     data_dir
78     database
79     fits_list
80     Methods
81     -----
82     get_solutions(wavlist: list / None, prefix: str = "fc")
83     -> (fc_files, custom): tuple[list[str], bool]
84     Parse `solutions_list` and return valid solution files and if
85     ↪ they are non-`IRAF` solutions.
86     parse_solution(fc_file: str, xshape: int, yshape: int)
87     -> tuple[dict[str, int], np.ndarray]
88     Loads the wavelength solution file and parses keywords
89     ↪ necessary for creating the wavelength extension.
90     join_file(file: os.PathLike)
91     -> None
92     Joins the files,
93     attaches the wavelength solutions,
94     performs cosmic ray cleaning,
95     masks the extension,
96     and checks cropping performed in `Split`.
97     Writes the FITS file in a `polsalt` valid format.
98     check_crop(hdu: pyfits.HDUList, o_file: str, e_file: str)
99     -> int
100    Opens the split `O`- and `E`-beam FITS files and returns the
101    ↪ amount of cropping that was performed.
102    process()
103    -> None
104    Calls `join_file` on each file in `fits_list` for automation.

```

```

103
104     Other Parameters
105     -----
106     no_arc : bool, optional
107         Deprecated. Decides whether the arc frames should be processed.
108         (The default is False, `polsalt` has no use for the arc after
109         ↪ wavelength calibrations)
110     **kwargs : dict
111         keyword arguments. Allows for passing unpacked dictionary to
112         ↪ the class constructor.
113
114     Notes
115     -----
116     Constants Imported (See utils.Constants):
117         DATADIR
118         SAVE_PREFIX
119         SPLIT_ROW
120         CR_PARAMS
121
122     Custom wavelength solutions must be formatted as:
123         `x`,
124         `y`,
125         *coefficients...
126     where the solutions are of order (`x` by `y`) and contain  $x*y$ 
127     ↪ coefficients.
128     The name of the custom wavelength solution file must contain either
129     ↪ "cheb" or "leg"
130     for Chebychev or Legendre wavelength solutions, respectively.
131
132     Cosmic ray rejection is performed using lacosmic [1]_
133     ↪ in ccdproc via astroscrappy [2]_.
134
135     References
136     -----
137     .. [1] van Dokkum 2001, PASP, 113, 789, 1420 (article :
138         ↪ http://adsabs.harvard.edu/abs/2001PASP..113.1420V)
139     .. [2] https://zenodo.org/records/1482019
140
141     """
142
143     # -----join1-----
144
145     # MARK: Join init
146     def __init__(
147         self,
148         data_dir: Path,
149         database: str = "database",
150         fits_list: list[str] = None,
151         solutions_list: list[Path] = None,
152         split_row: int = SPLIT_ROW,
153         no_arc: bool = True,
154         save_prefix: Path | None = None,
155         verbose: int = 30,
156         **kwargs,
157     ) -> None:
158         self.data_dir = data_dir
159         self.database = Path(data_dir) / database
160         self.fits_list = find_files(

```

```

155         data_dir=self.data_dir,
156         filenames=fits_list,
157         prefix="mxgbp",
158         ext="fits",
159     )
160     self.fc_files, self.custom = self.get_solutions(solutions_list)
161     self.split_row = split_row
162     self.save_prefix = SAVE_PREFIX
163     if type(save_prefix) == dict:
164         self.save_prefix = save_prefix
165
166     self.no_arc = no_arc
167     self.arc = find_arc(self.fits_list)
168
169     self.verbose = verbose < 30
170
171     logging.debug("__init__ - \n", self.__dict__)
172     return
173
174 # MARK: Find 2D WAV Functions
175 def get_solutions(
176     self,
177     wavlist: list[str] | None,
178     prefix: str = "fc"
179 ) -> tuple[list[str], bool]:
180     """
181         Get the list of wavelength solution files.
182
183     Parameters
184     -----
185     wavlist : list[str] / None
186         A list of custom wavelength solutions files.
187         If ``None``, `Join` will search for wavelength solutions in
188         → the `database` directory.
189     prefix : str, optional
190         The prefix of the wavelength solution files.
191         (Defaults to "fc")
192
193     Returns
194     -----
195     tuple[list[str], bool]
196         A tuple containing the list of wavelength solutions files
197         → and
198         a boolean indicating whether custom solutions were provided.
199
200     """
201     # No custom solutions
202     if not wavlist:
203         # Handle finding solutions
204         ws = []
205         for fl in os.listdir(self.database):
206             if os.path.isfile(self.database / fl) and (prefix ==
207                 fl[0:len(prefix)]):
208                 ws.append(fl)
209
210         if len(ws) != 2:
211             # Handle incorrect number of solutions found
212             msg = (

```

```

210             f"\"Incorrect amount of wavelength solutions \""
211             f"\"{len(ws)} fc... files) found in the solution \""
212             f"\"dir.: {self.database}\""
213         )
214         logging.error(msg)
215         raise FileNotFoundError(msg)
216
217     return (sorted(ws, reverse=True), False)
218
219 # Custom solution
220 if len(wavlist) >= 2:
221     if len(wavlist) > 2:
222         logging.warning(f" Too many solutions, only
223                         ↪ {wavlist[:2]} are considered")
223         wavlist = wavlist[:2]
224
225     for fl in wavlist:
226         if not os.path.isfile(os.path.join(self.data_dir, fl)):
227             msg = (
228                 f"{fl} not found in the "
229                 f"data directory {self.data_dir}"
230             )
231             logging.error(msg)
232             raise FileNotFoundError(msg)
233
234     return (sorted(wavlist, reverse=True), True)
235
236 # MARK: Parse 2D WAV Function
237 def parse_solution(
238     self,
239     fc_file: str,
240     xshape: int,
241     yshape: int
242 ) -> tuple[dict[str, int], np.ndarray]:
243     """
244         Parse the 2D wavelength solution function from `fc_file`.
245
246     Parameters
247     -----
248     fc_file : str
249         The filename of the wavelength solutions file.
250     xshape : int
251         The x-order of the 2D solution.
252     yshape : int
253         The y-order of the 2D solution.
254
255     Returns
256     -----
257     tuple[dict[str, int], np.ndarray]
258         A tuple containing a dictionary of the parameters of the
259             ↪ solution function
260             and the function coefficients.
261
262     """
263     fit_params = {}
264     coeff = []
265
266     if self.custom:

```

```

266     # Load coefficients
267     coeff = np.loadtxt(fc_file)
268
269     fit_params["xorder"] = coeff[0].astype(int)
270     fit_params["yorder"] = coeff[1].astype(int)
271     coeff = coeff[2:]
272
273     f_type = 3
274     if "cheb" in str(fc_file): f_type = 1
275     elif "leg" in str(fc_file): f_type = 2
276     fit_params["function"] = f_type
277
278     fit_params["xmin"], fit_params["xmax"] = 1, xshape
279     fit_params["ymin"], fit_params["ymax"] = 1, yshape
280
281 else:
282     # Parse IRAF fc database files
283     file_contents = []
284     with open(self.database / fc_file) as fcfile:
285         for i in fcfile:
286             file_contents.append(re.sub(r"\n\t\s*", "", i))
287
288     if file_contents[9] != "1.": # xterms - Cross-term type
289         msg = (
290             "Cross-term not recognised (always 1 for "
291             "'FITCOORDS'), redo FITCOORDS or change manually."
292         )
293         raise Exception(msg)
294
295     fit_params["function"] = int(file_contents[6][-1])
296
297     fit_params["xorder"] = int(file_contents[7][-1])
298     fit_params["yorder"] = int(file_contents[8][-1])
299
300     fit_params["xmin"] = int(file_contents[10][-1])
301     fit_params["xmax"] = xshape
302     # int(file_contents[11][-1])# stretch fit over x
303     fit_params["ymin"] = int(file_contents[12][-1])
304     fit_params["ymax"] = yshape
305     # int(file_contents[13][-1])# stretch fit over y
306
307     coeff = np.array(file_contents[14:], dtype=float)
308
309     coeff = np.reshape(
310         coeff,
311         (fit_params["xorder"], fit_params["yorder"]))
312
313
314     return (fit_params, coeff)
315
316 # MARK: Join Files
317 def join_file(self, file: os.PathLike) -> None:
318     """
319     Join the `O`- and `E`-beams, attach the wavelength solutions,
320     perform cosmic ray cleaning, mask the extensions,
321     and checks cropping performed by `Split`.
322     Write the FITS file in a `polsalt` valid format.
323

```

```

324     Parameters
325     -----
326     file : os.PathLike
327         The path of the FITS file to be joined.
328
329     See Also
330     -----
331     IRAF - `fitcoords` task
332         https://iraf.net/irafdocs/formats/fitcoords.php,
333         numpy 2D grid functions
334             https://numpy.org/doc/stable/reference/generated/numpy
335             Chebyshev: + '.polynomial.chebyshev.chebgrid2d.html'">''.polynomial.chebyshev.chebgrid2d.html'
336             Legendre: + '.polynomial.legendre.leggrid2d.html'">''.polynomial.legendre.leggrid2d.html'
337
338     """
339     # Create empty wavelength appended hdu list
340     whdu = pyfits.HDUList()
341     primary_ext = ""
342
343     # Handle prefix and names
344     pref = "arc" if file == self.arc else "beam"
345     o_file = self.save_prefix[pref][0] + file.name[-9:]
346     e_file = self.save_prefix[pref][1] + file.name[-9:]
347
348     # Open file
349     with pyfits.open(file) as hdu:
350         # Check if file has been cropped
351         cropsize = self.check_crop(hdu, o_file, e_file)
352
353         y_shape = int(hdu["SCI"].data.shape[0] / 2) - cropsize
354         x_shape = hdu["SCI"].data.shape[1]
355
356         # No differences in "PRIMARY" extention header
357         primary_ext = hdu["PRIMARY"]
358         whdu.append(primary_ext)
359
360         for ext in ["SCI", "VAR", "BPM"]:
361             whdu.append(pyfits.ImageHDU(name=ext))
362             whdu[ext].header = hdu[ext].header.copy()
363             whdu[ext].header["CTYPE3"] = "O,E"
364
365             # Create empty extention with correct order and format
366             if ext == "BPM":
367                 whdu[ext].data = np.zeros(
368                     (2, y_shape, x_shape),
369                     dtype="uint8"
370                 )
371                 whdu[ext].header["BITPIX"] = "-uint8"
372             else:
373                 whdu[ext].data = np.zeros(
374                     (2, y_shape, x_shape),
375                     dtype=">f4"
376                 )
377                 whdu[ext].header["BITPIX"] = "-32"
378
379             # Fill in empty extentions
380             if cropsize:
381                 temp_split = split_sci(

```

```

382             hdu ,
383             self.split_row ,
384             ext=ext
385         )[ext].data
386         whdu[ext].data[0] = temp_split[0, cropsize:]
387         whdu[ext].data[1] = temp_split[1, 0:-cropsize]
388
389     else:
390         whdu[ext].data = split_sci(
391             hdu ,
392             self.split_row ,
393             ext=ext
394         )[ext].data
395     # End of hdu calls, close hdu
396
397     # MARK: Join (Wav. Ext.)
398     whdu.append(pyfits.ImageHDU(name="WAV"))
399     wav_header = whdu["SCI"].header.copy()
400     wav_header["EXTNAME"] = "WAV"
401     wav_header["CTYPE3"] = "O,E"
402     whdu["WAV"].header = wav_header
403
404     whdu["WAV"].data = np.zeros(
405         whdu["SCI"].data.shape ,
406         dtype='>f4'
407     )
408
409     for num, fname in enumerate(self.fc_files):
410         pars, chebvals = self.parse_solution(
411             fname ,
412             x_shape ,
413             y_shape
414         )
415
416         if pars["function"] == 1: # Function type (1 = chebyshev)
417             # Set wavelength extention values to function
418             whdu["WAV"].data[num] = chebgrid2d(
419                 x=np.linspace(-1, 1, pars["ymax"]),
420                 y=np.linspace(-1, 1, pars["xmax"]),
421                 c=chebvals,
422             )
423
424         elif pars["function"] == 2: # Function type (2 = legendre)
425             # Set wavelength extention values to function
426             whdu["WAV"].data[num] = leggrid2d(
427                 x=np.linspace(-1, 1, pars["ymax"]),
428                 y=np.linspace(-1, 1, pars["xmax"]),
429                 c=chebvals,
430             )
431
432     else:
433         msg = (
434             "Function type not recognised, please wavelength "
435             "calibrate using either chebychev or legendre."
436         )
437         raise Exception(msg)
438
439     # MARK: Cosmic Ray Cleaning

```

```

440     # See utils.Constants for `CR_PARAMS` discussion
441     whdu["SCI"].data[num] = lacosmic(
442         whdu["SCI"].data[num],
443         # contrast=CR_PARAMS['CR_CONTRAST'],
444         # threshold=CR_PARAMS['CR_THRESHOLD'],
445         # neighbor_threshold=CR_PARAMS['CR_NEIGH_THRESH'],
446         # effective_gain=CR_PARAMS['GAIN'],
447         # background=CR_PARAMS['BACKGROUND'],
448         readnoise=CR_PARAMS['READNOISE'],
449         gain=CR_PARAMS['GAIN'],
450         verbose=self.verbose,
451     )[0]
452
453     # MARK: WAV masking
454     # Left & Right Crop
455     whdu["WAV"].data[whdu["WAV"].data[:] < 3_000] = 0.0
456     whdu["WAV"].data[whdu["WAV"].data[:] >= 10_000] = 0.0
457
458     # Top & Bottom Crop (shift\tilt)
459     rpix_oc, cols, rbin, lam_c = read_wollaston(
460         whdu,
461         DATADIR + "wollaston.txt"
462     )
463
464     drow_oc = (rpix_oc - rpix_oc[:, int(cols / 2)][:, None]) / rbin
465
466     ## Cropping as suggested
467     for c, col in enumerate(drow_oc[0]):
468         if np.isnan(col):
469             continue
470
471         if int(col) < 0:
472             whdu["WAV"].data[0, int(col) :, c] = 0.0
473         elif int(col) > cropsize:
474             whdu["WAV"].data[0, 0 : int(col) - cropsize, c] = 0.0
475
476     for c, col in enumerate(drow_oc[1]):
477         if np.isnan(col):
478             continue
479
480         if int(col) > 0:
481             whdu["WAV"].data[1, 0 : int(col), c] = 0.0
482         elif (int(col) < 0) & (abs(int(col)) > cropsize):
483             whdu["WAV"].data[1, int(col) + cropsize :, c] = 0.0
484
485     # MARK: BPM masking
486     whdu["BPM"].data[0] = np.where(
487         whdu["WAV"].data[0] == 0,
488         1,
489         whdu["BPM"].data[0]
490     )
491     whdu["BPM"].data[1] = np.where(
492         whdu["WAV"].data[1] == 0,
493         1,
494         whdu["BPM"].data[1]
495     )
496
497     whdu.writeto(f"w{os.path.basename(file)}", overwrite="True")

```

```

498     return
499
500
501 # MARK: Check Crop
502 def check_crop(
503     self,
504     hdu: pyfits.HDUList,
505     o_file: str,
506     e_file: str
507 ) -> int:
508     """
509         Check if cropping is necessary when joining `O`- and `E`-beams.
510
511     Parameters
512     -----
513     hdu : astropy.io.fits.HDUList
514         The HDUList to check for cropping.
515     o_file : str
516         The name of the previously split `O`-beam FITS file.
517     e_file : str
518         The name of the previously split `E`-beam FITS file.
519
520     Returns
521     -----
522     int
523         The number of rows which were cropped by `Split`.
524
525     """
526     cropsize = 0
527     o_y = 0
528     e_y = 0
529
530     with pyfits.open(o_file) as o, \
531         pyfits.open(e_file) as e:
532         o_y = o[0].data.shape[0]
533         e_y = e[0].data.shape[0]
534
535     if hdu["SCI"].data.shape[0] != (o_y + e_y):
536         # Get crop size, assuming crop same on both sides
537         cropsize = int((hdu["SCI"].data.shape[0] - o_y - e_y) / 2)
538
539     return cropsize
540
541 # MARK: Process all Listed Images
542 def process(self) -> None:
543     """Process all FITS images stored in the `fits_list`  

544     ↳ attribute"""
545     for target in self.fits_list:
546         logging.debug(f"Processing {target}")
547         self.join_file(target)
548
549     return
550
551 def main(argv) -> None:
552     """Main function."""
553
554     return

```

```
555
556
557 if __name__ == "__main__":
558     main(sys.argv[1:])
```

---

Listing II.4: The source code for `cross_correlate.py`

```

1 """Module for cross correlating polarization beams."""
2
3 #!/usr/bin/env python3
4 # -*- coding: utf-8 -*-
5
6 from __init__ import __author__, __email__, __version__
7
8 # MARK: Imports
9 import os
10 import sys
11 import logging
12 import itertools as iters
13 from pathlib import Path
14 from typing import Callable
15
16 import numpy as np
17 from numpy.polynomial import chebyshev
18 import matplotlib.pyplot as plt
19 from astropy.io import fits as pyfits
20 from scipy import signal
21
22 from utils.SharedUtils import find_files, continuum
23 from utils.Constants import SAVE_CORR
24
25 OFFSET = 0.3
26
27 mpl_logger = logging.getLogger('matplotlib')
28 mpl_logger.setLevel(logging.INFO)
29
30 # MARK: Correlate class
31 class CrossCorrelate:
32
33     #-----corr0-----
34
35     """
36     Cross correlate allows for comparing the extensions of multiple
37     FITS files, or comparing the O and E beams of a single FITS file.
38
39     Parameters
40     -----
41     data_dir : str
42         The path to the data to be cross correlated
43     filenames : list[str]
44         The ecwmxgbp*.fits files to be cross correlated.
45         If only one filename is defined, correlation is done against
46         ↪ the two polarization beams.
47     split_ccd : bool, optional
48         Decides whether the CCD regions should each be individually
49         ↪ cross correlated.
50         (The default is True, which splits the spectrum up into its
51         ↪ separate CCD regions)
52     cont_ord : int, optional
53         The degree of a chebyshev to fit to the continuum.
54         (The default is 11)
55     plot : bool, optional
56         Decides whether or not the continuum fitting should be plotted

```

```

54     (The default is False, so no continua plots are displayed)
55 save_prefix : str, optional
56     The name or directory to save the figure produced to.
57     "." saves a default name to the current working. A default name
58     ↪ is also used when save_prefix is a directory.
59     (The default is None, I.E. The figure is not saved, only
60     ↪ displayed)

61 Attributes
62 -----
63 data_dir
64 fits_list
65 beams : str
66     The mode of correlation.
67     'OE' for same file, and 'O' or 'E' for different files but same
68     ↪ ext's.
69 ccds : int
70     The number of CCD's in the data. Used to split the CCD's if
71     ↪ split_ccd is True.
72 cont_ord : int
73     The degree of the chebyshev to fit to the continuum.
74 can_plot : bool
75     Decides whether or not the continuum fitting should be plotted
76 offset : int
77     The amount the spectrum is shifted, mainly to test the effect
78     ↪ of the cross correlation
79     (The default is 0, I.E. no offset introduced)
80 save_prefix
81 wav_unit : str
82     The units of the wavelength axis.
83     (The default is Angstroms)
84 wav_cdelt : int
85     The wavelength increment.
86     (The default is 1)
87 alt : Callable
88     An alternate method of cross correlating the data.
89     (The default is None)

90 Methods
91 -----
92 load_file(filename: Path)
93     -> tuple[np.ndarray, np.ndarray, np.ndarray]
94     Loads the data from a FITS file.
95 get_bounds(bpm: np.ndarray)
96     -> np.ndarray
97     Finds the bounds for the CCD regions.
98 remove_cont(spec: list, wav: list, bpm: list, plotCont: bool)
99     -> None
100    Removes the continuum from the data.
101 correlate(filename1: Path, filename2: Path / None = None)
102    -> None
103    Cross correlates the data.
104 FTCS(filename1: Path, filename2: Path / None = None)
105    -> None
106    Cross correlates the data using the Fourier Transform.
107 plot(spec, wav, bpm, corrrdb, lagsdb)
108    -> None
109    Plots the data.
110

```

```

107     process()
108         -> None
109         Processes the data.
110
111     Other Parameters
112     -----
113     offset : int, optional
114         The amount the spectrum is shifted, mainly to test the effect
115         ↵ of the cross correlation
116         (The default is 0, I.E. no offset introduced)
117     **kwargs : dict
118         keyword arguments. Allows for passing unpacked dictionary to
119         ↵ the class constructor.
120     FTCS : bool, optional
121         Decides whether the Fourier Transform should be used for
122         ↵ cross correlation.
123
124     See Also
125     -----
126     scipy
127         https://docs.scipy.org/doc/scipy/reference/generated/
128         correlation: scipy.signal.correlate.html
129
130     Notes
131     -----
132     Constants Imported (See utils.Constants):
133         SAVE_CORR
134
135     #-----corr1-----
136
137     # MARK: Correlate init
138     def __init__(self,
139         data_dir: Path,
140         filenames: list[str],
141         beams: str = "OE",
142         split_ccd: bool = True,
143         cont_ord: int = 11,
144         plot: bool = False,
145         offset: int = 0,
146         save_prefix: Path | None = None,
147         **kwargs
148     ) -> None:
149         self.data_dir = data_dir
150         self.fits_list = find_files(
151             data_dir=self.data_dir,
152             filenames=filenames,
153             prefix="ecwmxgbp",
154             ext="fits",
155         )
156         self._beams = None
157         self.beams = beams
158         self.ccds = 1
159         if split_ccd:
160             # BPM == 2 near center of CCD if CCD count varies

```

```

162         with pyfits.open(self.fits_list[0]) as hdu:
163             self.ccds = sum(hdu["BPM"].data.sum(axis=1)[0] == 2)
164
165             self.cont_ord = cont_ord
166             self.can_plot = plot
167             self.offset = offset
168             if offset != 0:
169                 logging.warning("'offset' is only for testing.")
170                 # Add an offset to the spectra to test cross correlation
171                 # self.spec1 = np.insert(
172                 #     self.spec1, [0] * offset, self.spec1[:, :offset],
173                 #     axis=-1
174                 # )[ :, : self.spec1.shape[-1]]
175
176             self.save_prefix = save_prefix
177             # Handle directory save name
178             if self.save_prefix and self.save_prefix.is_dir():
179                 self.save_prefix /= SAVE_CORR
180                 logging.warning((
181                     f"Correlation save name resolves to a directory. "
182                     f"Saving under {self.save_prefix}"
183                 ))
184
185             self.wav_unit = "$\AA$"
186             self.wav_cdelt = 1
187
188             self.alt = self.FTCS if kwargs.get("FTCS") else None
189
190             logging.debug("__init__ - \n", self.__dict__)
191             return
192
193             # MARK: Beams property
194             @property
195             def beams(self) -> str:
196                 return self._beams
197
198             @beams.setter
199             def beams(self, mode: str) -> None:
200                 if mode not in ['O', 'E', 'OE']:
201                     errMsg = f"Correlation mode '{mode}' not recognized."
202                     logging.error(errMsg)
203                     raise ValueError(errMsg)
204
205                 self._beams = mode
206
207             return
208
209             # MARK: Load file
210             def load_file(
211                 self,
212                 filename: Path
213             ) -> tuple[np.ndarray, np.ndarray, np.ndarray]:
214                 """
215
216                 Load the data from a FITS file.
217
218                 Parameters
219                 -----
220                 filename : Path

```

```

219     The name of the FITS file to load.
220
221     Returns
222     -----
223     tuple[np.ndarray, np.ndarray, np.ndarray]
224         The spectrum, wavelength, and bad pixel mask.
225
226     """
227     spec, wav, bpm = None, None, None
228
229     # Open HDU
230     with pyfits.open(filename) as hdul:
231         spec = hdul["SCI"].data.sum(axis=1)
232         wav = (
233             np.arange(spec.shape[-1])
234             * hdul["SCI"].header["CDELT1"]
235             + hdul["SCI"].header["CRVAL1"]
236         )
237         wav = np.array((wav, wav))
238         bpm = hdul["BPM"].data.sum(axis=1)
239
240         self.wav_cdel = float(hdul["SCI"].header["CDELT1"])
241
242         if hdul["SCI"].header["CTYPE1"] != 'Angstroms':
243             self.wav_unit = hdul["SCI"].header["CTYPE1"]
244
245     return spec, wav, bpm
246
247     # MARK: Get bounds
248     def get_bounds(self, bpm: np.ndarray) -> np.ndarray:
249         """
250             Find the bounds for a file based on the CCD count.
251
252             Parameters
253             -----
254             bpm : np.ndarray
255                 The bad pixel mask.
256
257             Returns
258             -----
259             np.ndarray
260                 The bounds for the CCD regions.
261
262         """
263         # bounds.shape -> (0/E, CCD's, low./up. bound)
264         if self.ccds == 1:
265             return np.array(
266                 [(0, bpm[0].shape[-1]), (0, bpm[1].shape[-1])]
267             ).astype(int)
268
269         bounds = np.zeros((2, self.ccds, 2))
270
271         # Get lower and upper bound for each ccd, save to bounds
272         # Lower -> min is zero, Upper -> max is bpm length
273         for ext, ccd in iters.product(range(2), range(self.ccds)):
274             mid = np.where(bpm[ext] == 2)[0][ccd]
275             CCDs = self.ccds * 2
276             bounds[ext, ccd] = (

```

```

277             max(mid - bpm.shape[-1] // CCDs, 0),
278             min(mid + bpm.shape[-1] // CCDs, bpm.shape[-1])
279         )
280
281     return bounds.astype(int)
282
283 # MARK: Remove Continua
284 def remove_cont(
285     self,
286     spec: list,
287     wav: list,
288     bpm: list,
289     plotCont: bool
290 ) -> None:
291     """
292         Remove the continuum from the data.
293
294     Parameters
295     -----
296     spec : list
297         The spectrum to remove the continuum from.
298     wav : list
299         The wavelength of the spectrum.
300     bpm : list
301         The bad pixel mask.
302     plotCont : bool
303         Decides whether or not the continuum fitting should be
304         ↪ plotted
305
306     Returns
307     -----
308     None
309
310     """
311     # Mask out the bad pixels for fitting continua
312     okwav = np.where(bpm != 1)
313
314     # Define continua
315     ctm = continuum(
316         wav[okwav],
317         spec[okwav],
318         deg=self.cont_ord,
319         plot=plotCont,
320     )
321
322     # Normalise spectra
323     spec /= chebyshev.chebval(wav, ctm)
324     spec -= 1
325
326     return spec
327
328 # MARK: Correlate
329 def correlate(
330     self,
331     filename1: Path,
332     filename2: Path | None = None,
333     alt: Callable = None
334 ) -> None:

```

```

334     """
335     Cross correlates the data.
336
337     Parameters
338     -----
339     filename1 : Path
340         The name of the first FITS file to cross correlate.
341     filename2 : Path, optional
342         The name of the second FITS file to cross correlate.
343         (Defaults to None)
344     alt : Callable, optional
345         An alternate method of cross correlating the data.
346         (Defaults to None)
347
348     Returns
349     -----
350     None
351
352     """
353     # mode: 0E -> '01' & 'E1', 0 -> '01' & '02', E -> 'E1' & 'E2'
354     # Load data
355     spec, wav, bpm = self.load_file(filename1)
356     if filename2 and self.beams != '0E':
357         unpack = lambda ext, *args: [arr[ext] for arr in args]
358
359         if self.beams == '0':
360             spec[-1], wav[-1], bpm[-1] = unpack(
361                 0, *self.load_file(filename2))
362         )
363
364     else:
365         spec[0], wav[0], bpm[0] = spec[-1], wav[-1], bpm[-1]
366         spec[-1], wav[-1], bpm[-1] = unpack(
367             -1, *self.load_file(filename2))
368         )
369
370     bounds = self.get_bounds(bpm)
371
372     logging.debug(
373         f"correlate - data shape:\n\tspec/wav/bpm: {spec.shape}"
374     )
375
376     corrdb = [[] for _ in range(self.ccdb)]
377     lagsdb = [[] for _ in range(self.ccdb)]
378     for ccd in range(self.ccdb):
379         sig = []
380         for ext in range(2):
381             lb, ub = bounds[ext, ccd]
382
383             if self.cont_ord > 0:
384                 spec[ext, lb:ub] = self.remove_cont(
385                     spec[ext, lb:ub],
386                     wav[ext, lb:ub],
387                     bpm[ext, lb:ub],
388                     self.can_plot
389                 )
390
391             # Invert BPM (and account for 2); zero bad pixels

```

```

392         sig.append((
393             spec[ext, lb:ub]
394             * abs(bpm[ext, lb:ub] * -1 + 1)
395         ))
396
397     # Finally(!!!) cross correlate signals and scale max -> 1
398     corrdb[ccd] = signal.correlate(*sig) if not alt else
399     ↪ alt(*sig)
400     corrdb[ccd] /= np.max(corrdb[ccd])
401     lagsdb[ccd] = signal.correlation_lags(
402         sig[0].shape[-1],
403         sig[1].shape[-1]
404     ) * self.wav_cdel
405
406     return (spec, wav, bpm), (corrdb, lagsdb)
407
408 # MARK: FTCS alternate
409 def FTCS(
410     self,
411     signal1: np.ndarray,
412     signal2: np.ndarray
413 ) -> None:
414     """
415     Cross correlates the data using the Fourier Transform.
416
417     Parameters
418     -----
419     signal1 : np.ndarray
420         The first signal to cross correlate.
421     signal2 : np.ndarray
422         The second signal to cross correlate.
423
424     Returns
425     -----
426     np.ndarray
427         The correlation data using the Fourier Transform.
428
429     """
430     logging.debug(
431         f"FTCS - data shape:\n{tspec/wav/bpm: {signal1.shape}}"
432     )
433
434     # Invert BPM (and account for 2); zero bad pixels
435     ft_spec1 = np.fft.fft(signal1)
436     ft_spec2 = np.fft.fft(signal2)
437
438     if self.can_plot:
439         plt.plot(ft_spec1)
440         plt.plot(ft_spec2)
441         plt.show()
442
443     # Cross correlate signals
444     # ft_spectrum1 * np.conj(ft_spectrum2)
445     corr_entry = signal.correlate(ft_spec1, ft_spec2)
446
447     return np.fft.ifft(corr_entry)
448
449 # MARK: Plot

```

```

449     def plot(self, spec, wav, bpm, corrdb, lagsdb) -> None:
450         """
451             Plot the data.
452
453             Parameters
454             -----
455             spec : np.ndarray
456                 The spectrum.
457             wav : np.ndarray
458                 The wavelength.
459             bpm : np.ndarray
460                 The bad pixel mask.
461             corrdb : np.ndarray
462                 The cross correlation data.
463             lagsdb : np.ndarray
464                 The lags data.
465
466             Returns
467             -----
468             None
469
470             """
471     plt.style.use(Path(__file__).parent.resolve() /
472                   'utils/STOPS.mplstyle')
472     bounds = self.get_bounds(bpm)
473
474     fig, axs = plt.subplots(2, self.ccds, sharey="row")
475
476     if self.ccds == 1:
477         # Convert axes to a 2D array
478         axs = np.swapaxes(np.atleast_2d(axs), 0, 1)
479
480     # for ext, ccd in iters.product(range(2), range(self.ccds)):
481
482         for ccd in range(self.ccds):
483             axs[0, ccd].plot(
484                 lagsdb[ccd],
485                 corrdb[ccd] * 100,
486                 color='C4',
487                 label=f"max lag @ {lagsdb[ccd][corrdb[ccd].argmax()]} - "
488                       f"({bounds[1, ccd, 0] - bounds[0, ccd, 0]})",
489             )
490
491         for ext in range(2):
492             lb, ub = bounds[ext, ccd]
493             logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
494                           f"1]}")
495
496             axs[1, ccd].plot(
497                 wav[ext, lb:ub],
498                 spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
499                 OFFSET * ext,
500                 label=
501                     f"${self.beams if self.beams != 'OE' else "
502                      f"self.beams[{ext}]}"
503                     f"_{{ext + 1 if self.beams != 'OE' else 1}}$"
504                     f"{{{('+' + str(OFFSET * ext)) + ')')}} if ext > 0 "
505                         f"else ''}"

```

```

501             ),
502         )
503
504     axs[0, 0].set_ylabel("Normalised Correlation\n(\%)")
505     for ax in axs[0, :]:
506         ax.set_xlabel("Signal Lag")
507     for ax in axs[1:, 0]:
508         ax.set_ylabel(f"Norm. Intensity\n(Counts)")
509     for ax in axs[-1, :]:
510         ax.set_xlabel(f"Wavelength ({self.wav_unit})")
511     for ax in axs.flatten():
512         ax.legend()
513
514     # plt.tight_layout()
515     # fig1 = plt.gcf()
516     # DPI = fig1.get_dpi()
517     # fig1.set_size_inches(700.0/float(DPI), 250.0/float(DPI))
518     plt.show()
519
520     # Handle do not save
521     if not self.save_prefix:
522         return
523
524     # Handle save
525     fig.savefig(fname=self.save_prefix)
526
527     return
528
529     # MARK: Process all listed images
530 def process(self) -> None:
531     """
532     Process the data.
533
534     Returns
535     -----
536     None
537
538     """
539     if self.beams != 'OE' and len(self.fits_list) == 1:
540         # change mode to OE with warning
541         logging.warning((
542             f"`{self.beams}` correlation not possible for "
543             "a single file. correlation `mode` changed to 'OE'."
544         ))
545         self.beams = 'OE'
546
547     # OE `mode` (same file, diff. ext.)
548     if self.beams == 'OE':
549         for fl in self.fits_list:
550             logging.info(f'"OE' correlation of {fl}.')
551             (spec, wav, bpm), (corr, lags) = self.correlate(fl,
552                 ↪ alt=self.alt)
553             self.plot(spec, wav, bpm, corr, lags)
554
555     return
556
557     # O/E `mode` (diff. files, same ext.)
558     for fl1, fl2 in iters.combinations(self.fits_list, 2):

```

```
558     logging.info(f"{self.beams} correlation of {f11} vs {f12}.")  
559     (spec, wav, bpm), (corr, lags) = self.correlate(f11, f12,  
560      ↪ alt=self.alt)  
561     self.plot(spec, wav, bpm, corr, lags)  
562  
563  
564  
565 # MARK: Main function  
566 def main(argv) -> None:  
567     return  
568  
569 if __name__ == "__main__":  
570     main(sys.argv[1:])
```

Listing II.5: The source code for `skylines.py`

```

1 """Module for analyzing the sky lines of a wavelength calibrated
2    ↪ image."""
3
4 #!/usr/bin/env python3
5 # -*- coding: utf-8 -*-
6
7 from __init__ import __author__, __email__, __version__
8
9 # MARK: Imports
10 import os
11 import sys
12 import logging
13 from pathlib import Path
14
15 import numpy as np
16 import matplotlib.pyplot as plt
17 from astropy.io import fits as pyfits
18 from scipy import signal, stats, interpolate
19
20 from utils.SharedUtils import find_files, continuum
21 from utils.Constants import SAVE_SKY
22
23 mpl_logger = logging.getLogger('matplotlib')
24 mpl_logger.setLevel(logging.INFO)
25 # plt.rcParams['figure.figsize'] = (20, 4)
26
27 # MARK: Skylines Class
28 class Skylines:
29
30     """sky0"""
31
32     """
33         Class representing the Skylines object.
34
35     Parameters
36     -----
37     data_dir : Path
38         The directory containing the data files.
39     filenames : list[str]
40         The list of filenames to be processed.
41     beam : str, optional
42         The beam mode, by default "OE".
43     plot : bool, optional
44         Flag indicating whether to plot the continuum, by default False.
45     save_prefix : Path / None, optional
46         The prefix for saving the data, by default None.
47     **kwargs
48         Additional keyword arguments.
49
50     Attributes
51     -----
52     data_dir : Path
53         The directory containing the data files.
54     fits_list : list[str]
55         The list of fits file paths.
      beam : str

```

```

56     The beam mode.
57     can_plot : bool
58         Flag indicating whether to plot the continuum.
59     save_prefix : Path | None
60         The prefix for saving the data.
61     wav_unit : str
62         The unit of wavelength.
63     rawWav : np.ndarray
64         The raw wavelength data.
65     rawSpec : np.ndarray
66         The raw spectral data.
67     rawBpm : np.ndarray
68         The raw bad pixel mask data.
69     corrWav : np.ndarray
70         The corrected wavelength data.
71     corrSpec : np.ndarray
72         The corrected spectral data.
73     spec : np.ndarray
74         The median spectrum.
75     normSpec : np.ndarray
76         The normalized spectrum.

77
78     Methods
79     -----
80     checkLoad(self, path1: str) -> np.ndarray:
81         Checks and loads the data from the given path.
82     transform(self, wav_sol: np.ndarray, spec: np.ndarray) ->
83         np.ndarray:
84         Transforms the input wavelength and spectral data based on the
85         given wavelength solution.
86     rmvCont(self) -> np.ndarray:
87         Removes the continuum from the spectrum.
88     skylines(self) -> None:
89         Placeholder method for processing skylines.
90     process(self) -> None:
91         Placeholder method for processing the data.
92     """
93
94     # -----sky1-----
95
96     # MARK: Skylines init
97     def __init__(
98         self,
99         data_dir: Path,
100        filenames : list[str],
101        beams: str = "OE",
102        split_ccd: bool = False,
103        cont_ord: int = 11,
104        plot: bool = False,
105        transform: bool = True,
106        save_prefix: Path | None = None,
107        **kwargs,
108    ) -> None:
109         self.data_dir = data_dir
110         self.fits_list = find_files(
111             data_dir=self.data_dir,
112             filenames=filenames,
113             prefix="wmxgbp", # t[o/e]beam

```

```

112         ext="fits",
113     )
114     self._beams = None
115     self.beams = beams
116
117     self.split_ccd = split_ccd
118     self.cont_ord = cont_ord
119     self.can_plot = plot
120     self.must_transform = transform
121
122     self.save_prefix = save_prefix
123     # Handle directory save name
124     if self.save_prefix and self.save_prefix.is_dir():
125         self.save_prefix /= SAVE_SKY
126         logging.warning((
127             f"Skylines save name resolves to a directory. "
128             f"Saving under {self.save_prefix}"
129         ))
130
131     self.wav_unit = "$\AA$"
132
133     # self.rawWav, self.rawSpec, self.rawBpm = self.checkLoad(
134     #     self.fits_list
135     # )
136     # self.corrWav, self.corrSpec = self.transform(
137     #     self.rawWav,
138     #     self.rawSpec
139     # )
140     # self.spec = np.median(self.corrSpec, axis=1)
141     # self.normSpec = self.rmvCont(self.spec)
142
143     logging.debug("__init__ - \n", self.__dict__)
144     return
145
146     # MARK: Beams property
147     @property
148     def beams(self) -> str:
149         return self._beams
150
151     @beams.setter
152     def beams(self, mode: str) -> None:
153         if mode not in ['0', 'E', 'OE']:
154             errMsg = f"Correlation mode '{mode}' not recognized."
155             logging.error(errMsg)
156             raise ValueError(errMsg)
157
158         self._beams = mode
159
160     return
161
162     # MARK: Load Sky lines
163     def load_sky_lines(self, filename: Path | None = None) ->
164     np.ndarray:
165         """
166             Loads the sky lines from the given file.
167
168             Parameters
169             -----

```

```

169     filename : Path / None, optional
170         The path to the file to be loaded.
171         Defaults to loading the skylines from utils/sky.salt
172
173     Returns
174     -----
175     sky_lines : np.ndarray
176         The sky lines from the file.
177
178     """
179     if not filename:
180         filename = Path(__file__).parent.resolve() /
181             'utils/sky.salt'
182
183     dtype = [('wav', float), ('flux', float)]
184     skylines = np.genfromtxt(filename, dtype=dtype, skip_header=3,
185                             skip_footer=1)
186
187     if self.can_plot:
188         plt.plot(skylines['wav'], skylines['flux'], 'x',
189                   label="Model peaks")
190         plt.xlabel(f'Wavelength {self.wav_unit}')
191         plt.ylabel('Relative intensities')
192         plt.title('Known sky lines')
193         plt.legend()
194         plt.show()
195
196     return skylines
197
198 # MARK: Transform spectra
199 @staticmethod
200 def transform(wav_sol: np.ndarray, spec: np.ndarray, resPlot: bool
201               = False) -> np.ndarray:
202     """
203         Transforms the input wavelength and spectral data based on the
204         given wavelength solution.
205
206     Parameters
207     -----
208     wav_sol : np.ndarray
209         The wavelength solution.
210     spec : np.ndarray
211         The spectral data.
212
213     Returns
214     -----
215     wav, spec : np.ndarray
216         The transformed wavelength and spectral data.
217
218     """
219     # Create arrays to return
220     cw = np.zeros_like(wav_sol)
221     cs = np.zeros_like(wav_sol)
222
223     exts = cw.shape[0]
224     rows = cw.shape[1]
225
226     for ext in range(exts):
227
228         for row in range(rows):
229
230             if resPlot:
231                 if ext == 0:
232                     if row == 0:
233                         cw[ext][row] = 1
234
235                 else:
236                     cw[ext][row] = 1
237
238             else:
239                 if ext == 0:
240                     if row == 0:
241                         cw[ext][row] = 1
242
243                     else:
244                         cw[ext][row] = 1
245
246                 else:
247                     if row == 0:
248                         cw[ext][row] = 1
249
250                     else:
251                         cw[ext][row] = 1
252
253             if ext == 0:
254                 if row == 0:
255                     cs[ext][row] = 1
256
257                 else:
258                     cs[ext][row] = 1
259
260             else:
261                 if row == 0:
262                     cs[ext][row] = 1
263
264                 else:
265                     cs[ext][row] = 1
266
267     return cw, cs
268
269
270     """
271     # Create arrays to return
272     cw = np.zeros_like(wav_sol)
273     cs = np.zeros_like(wav_sol)
274
275     exts = cw.shape[0]
276     rows = cw.shape[1]
277
278     for ext in range(exts):
279
280         for row in range(rows):
281
282             if resPlot:
283                 if ext == 0:
284                     if row == 0:
285                         cw[ext][row] = 1
286
287                 else:
288                     cw[ext][row] = 1
289
290             else:
291                 if ext == 0:
292                     if row == 0:
293                         cw[ext][row] = 1
294
295                     else:
296                         cw[ext][row] = 1
297
298                 else:
299                     if row == 0:
300                         cw[ext][row] = 1
301
302                     else:
303                         cw[ext][row] = 1
304
305             if ext == 0:
306                 if row == 0:
307                     cs[ext][row] = 1
308
309                 else:
310                     cs[ext][row] = 1
311
312             else:
313                 if row == 0:
314                     cs[ext][row] = 1
315
316                 else:
317                     cs[ext][row] = 1
318
319     return cw, cs
320
321
322     """
323     # Create arrays to return
324     cw = np.zeros_like(wav_sol)
325     cs = np.zeros_like(wav_sol)
326
327     exts = cw.shape[0]
328     rows = cw.shape[1]
329
330     for ext in range(exts):
331
332         for row in range(rows):
333
334             if resPlot:
335                 if ext == 0:
336                     if row == 0:
337                         cw[ext][row] = 1
338
339                 else:
340                     cw[ext][row] = 1
341
342             else:
343                 if ext == 0:
344                     if row == 0:
345                         cw[ext][row] = 1
346
347                     else:
348                         cw[ext][row] = 1
349
350                 else:
351                     if row == 0:
352                         cw[ext][row] = 1
353
354                     else:
355                         cw[ext][row] = 1
356
357             if ext == 0:
358                 if row == 0:
359                     cs[ext][row] = 1
360
361                 else:
362                     cs[ext][row] = 1
363
364             else:
365                 if row == 0:
366                     cs[ext][row] = 1
367
368                 else:
369                     cs[ext][row] = 1
370
371     return cw, cs
372
373
374     """
375     # Create arrays to return
376     cw = np.zeros_like(wav_sol)
377     cs = np.zeros_like(wav_sol)
378
379     exts = cw.shape[0]
380     rows = cw.shape[1]
381
382     for ext in range(exts):
383
384         for row in range(rows):
385
386             if resPlot:
387                 if ext == 0:
388                     if row == 0:
389                         cw[ext][row] = 1
390
391                 else:
392                     cw[ext][row] = 1
393
394             else:
395                 if ext == 0:
396                     if row == 0:
397                         cw[ext][row] = 1
398
399                     else:
400                         cw[ext][row] = 1
401
402                 else:
403                     if row == 0:
404                         cw[ext][row] = 1
405
406                     else:
407                         cw[ext][row] = 1
408
409             if ext == 0:
410                 if row == 0:
411                     cs[ext][row] = 1
412
413                 else:
414                     cs[ext][row] = 1
415
416             else:
417                 if row == 0:
418                     cs[ext][row] = 1
419
420                 else:
421                     cs[ext][row] = 1
422
423     return cw, cs
424
425
426     """
427     # Create arrays to return
428     cw = np.zeros_like(wav_sol)
429     cs = np.zeros_like(wav_sol)
430
431     exts = cw.shape[0]
432     rows = cw.shape[1]
433
434     for ext in range(exts):
435
436         for row in range(rows):
437
438             if resPlot:
439                 if ext == 0:
440                     if row == 0:
441                         cw[ext][row] = 1
442
443                 else:
444                     cw[ext][row] = 1
445
446             else:
447                 if ext == 0:
448                     if row == 0:
449                         cw[ext][row] = 1
450
451                     else:
452                         cw[ext][row] = 1
453
454                 else:
455                     if row == 0:
456                         cw[ext][row] = 1
457
458                     else:
459                         cw[ext][row] = 1
460
461             if ext == 0:
462                 if row == 0:
463                     cs[ext][row] = 1
464
465                 else:
466                     cs[ext][row] = 1
467
468             else:
469                 if row == 0:
470                     cs[ext][row] = 1
471
472                 else:
473                     cs[ext][row] = 1
474
475     return cw, cs
476
477
478     """
479     # Create arrays to return
480     cw = np.zeros_like(wav_sol)
481     cs = np.zeros_like(wav_sol)
482
483     exts = cw.shape[0]
484     rows = cw.shape[1]
485
486     for ext in range(exts):
487
488         for row in range(rows):
489
490             if resPlot:
491                 if ext == 0:
492                     if row == 0:
493                         cw[ext][row] = 1
494
495                 else:
496                     cw[ext][row] = 1
497
498             else:
499                 if ext == 0:
500                     if row == 0:
501                         cw[ext][row] = 1
502
503                     else:
504                         cw[ext][row] = 1
505
506                 else:
507                     if row == 0:
508                         cw[ext][row] = 1
509
510                     else:
511                         cw[ext][row] = 1
512
513             if ext == 0:
514                 if row == 0:
515                     cs[ext][row] = 1
516
517                 else:
518                     cs[ext][row] = 1
519
520             else:
521                 if row == 0:
522                     cs[ext][row] = 1
523
524                 else:
525                     cs[ext][row] = 1
526
527     return cw, cs
528
529
530     """
531     # Create arrays to return
532     cw = np.zeros_like(wav_sol)
533     cs = np.zeros_like(wav_sol)
534
535     exts = cw.shape[0]
536     rows = cw.shape[1]
537
538     for ext in range(exts):
539
540         for row in range(rows):
541
542             if resPlot:
543                 if ext == 0:
544                     if row == 0:
545                         cw[ext][row] = 1
546
547                 else:
548                     cw[ext][row] = 1
549
550             else:
551                 if ext == 0:
552                     if row == 0:
553                         cw[ext][row] = 1
554
555                     else:
556                         cw[ext][row] = 1
557
558                 else:
559                     if row == 0:
560                         cw[ext][row] = 1
561
562                     else:
563                         cw[ext][row] = 1
564
565             if ext == 0:
566                 if row == 0:
567                     cs[ext][row] = 1
568
569                 else:
570                     cs[ext][row] = 1
571
572             else:
573                 if row == 0:
574                     cs[ext][row] = 1
575
576                 else:
577                     cs[ext][row] = 1
578
579     return cw, cs
580
581
582     """
583     # Create arrays to return
584     cw = np.zeros_like(wav_sol)
585     cs = np.zeros_like(wav_sol)
586
587     exts = cw.shape[0]
588     rows = cw.shape[1]
589
590     for ext in range(exts):
591
592         for row in range(rows):
593
594             if resPlot:
595                 if ext == 0:
596                     if row == 0:
597                         cw[ext][row] = 1
598
599                 else:
600                     cw[ext][row] = 1
601
602             else:
603                 if ext == 0:
604                     if row == 0:
605                         cw[ext][row] = 1
606
607                     else:
608                         cw[ext][row] = 1
609
610                 else:
611                     if row == 0:
612                         cw[ext][row] = 1
613
614                     else:
615                         cw[ext][row] = 1
616
617             if ext == 0:
618                 if row == 0:
619                     cs[ext][row] = 1
620
621                 else:
622                     cs[ext][row] = 1
623
624             else:
625                 if row == 0:
626                     cs[ext][row] = 1
627
628                 else:
629                     cs[ext][row] = 1
630
631     return cw, cs
632
633
634     """
635     # Create arrays to return
636     cw = np.zeros_like(wav_sol)
637     cs = np.zeros_like(wav_sol)
638
639     exts = cw.shape[0]
640     rows = cw.shape[1]
641
642     for ext in range(exts):
643
644         for row in range(rows):
645
646             if resPlot:
647                 if ext == 0:
648                     if row == 0:
649                         cw[ext][row] = 1
650
651                 else:
652                     cw[ext][row] = 1
653
654             else:
655                 if ext == 0:
656                     if row == 0:
657                         cw[ext][row] = 1
658
659                     else:
660                         cw[ext][row] = 1
661
662                 else:
663                     if row == 0:
664                         cw[ext][row] = 1
665
666                     else:
667                         cw[ext][row] = 1
668
669             if ext == 0:
670                 if row == 0:
671                     cs[ext][row] = 1
672
673                 else:
674                     cs[ext][row] = 1
675
676             else:
677                 if row == 0:
678                     cs[ext][row] = 1
679
680                 else:
681                     cs[ext][row] = 1
682
683     return cw, cs
684
685
686     """
687     # Create arrays to return
688     cw = np.zeros_like(wav_sol)
689     cs = np.zeros_like(wav_sol)
690
691     exts = cw.shape[0]
692     rows = cw.shape[1]
693
694     for ext in range(exts):
695
696         for row in range(rows):
697
698             if resPlot:
699                 if ext == 0:
700                     if row == 0:
701                         cw[ext][row] = 1
702
703                 else:
704                     cw[ext][row] = 1
705
706             else:
707                 if ext == 0:
708                     if row == 0:
709                         cw[ext][row] = 1
710
711                     else:
712                         cw[ext][row] = 1
713
714                 else:
715                     if row == 0:
716                         cw[ext][row] = 1
717
718                     else:
719                         cw[ext][row] = 1
720
721             if ext == 0:
722                 if row == 0:
723                     cs[ext][row] = 1
724
725                 else:
726                     cs[ext][row] = 1
727
728             else:
729                 if row == 0:
730                     cs[ext][row] = 1
731
732                 else:
733                     cs[ext][row] = 1
734
735     return cw, cs
736
737
738     """
739     # Create arrays to return
740     cw = np.zeros_like(wav_sol)
741     cs = np.zeros_like(wav_sol)
742
743     exts = cw.shape[0]
744     rows = cw.shape[1]
745
746     for ext in range(exts):
747
748         for row in range(rows):
749
750             if resPlot:
751                 if ext == 0:
752                     if row == 0:
753                         cw[ext][row] = 1
754
755                 else:
756                     cw[ext][row] = 1
757
758             else:
759                 if ext == 0:
760                     if row == 0:
761                         cw[ext][row] = 1
762
763                     else:
764                         cw[ext][row] = 1
765
766                 else:
767                     if row == 0:
768                         cw[ext][row] = 1
769
770                     else:
771                         cw[ext][row] = 1
772
773             if ext == 0:
774                 if row == 0:
775                     cs[ext][row] = 1
776
777                 else:
778                     cs[ext][row] = 1
779
780             else:
781                 if row == 0:
782                     cs[ext][row] = 1
783
784                 else:
785                     cs[ext][row] = 1
786
787     return cw, cs
788
789
790     """
791     # Create arrays to return
792     cw = np.zeros_like(wav_sol)
793     cs = np.zeros_like(wav_sol)
794
795     exts = cw.shape[0]
796     rows = cw.shape[1]
797
798     for ext in range(exts):
799
800         for row in range(rows):
801
802             if resPlot:
803                 if ext == 0:
804                     if row == 0:
805                         cw[ext][row] = 1
806
807                 else:
808                     cw[ext][row] = 1
809
810             else:
811                 if ext == 0:
812                     if row == 0:
813                         cw[ext][row] = 1
814
815                     else:
816                         cw[ext][row] = 1
817
818                 else:
819                     if row == 0:
820                         cw[ext][row] = 1
821
822                     else:
823                         cw[ext][row] = 1
824
825             if ext == 0:
826                 if row == 0:
827                     cs[ext][row] = 1
828
829                 else:
830                     cs[ext][row] = 1
831
832             else:
833                 if row == 0:
834                     cs[ext][row] = 1
835
836                 else:
837                     cs[ext][row] = 1
838
839     return cw, cs
840
841
842     """
843     # Create arrays to return
844     cw = np.zeros_like(wav_sol)
845     cs = np.zeros_like(wav_sol)
846
847     exts = cw.shape[0]
848     rows = cw.shape[1]
849
850     for ext in range(exts):
851
852         for row in range(rows):
853
854             if resPlot:
855                 if ext == 0:
856                     if row == 0:
857                         cw[ext][row] = 1
858
859                 else:
860                     cw[ext][row] = 1
861
862             else:
863                 if ext == 0:
864                     if row == 0:
865                         cw[ext][row] = 1
866
867                     else:
868                         cw[ext][row] = 1
869
870                 else:
871                     if row == 0:
872                         cw[ext][row] = 1
873
874                     else:
875                         cw[ext][row] = 1
876
877             if ext == 0:
878                 if row == 0:
879                     cs[ext][row] = 1
880
881                 else:
882                     cs[ext][row] = 1
883
884             else:
885                 if row == 0:
886                     cs[ext][row] = 1
887
888                 else:
889                     cs[ext][row] = 1
890
891     return cw, cs
892
893
894     """
895     # Create arrays to return
896     cw = np.zeros_like(wav_sol)
897     cs = np.zeros_like(wav_sol)
898
899     exts = cw.shape[0]
900     rows = cw.shape[1]
901
902     for ext in range(exts):
903
904         for row in range(rows):
905
906             if resPlot:
907                 if ext == 0:
908                     if row == 0:
909                         cw[ext][row] = 1
910
911                 else:
912                     cw[ext][row] = 1
913
914             else:
915                 if ext == 0:
916                     if row == 0:
917                         cw[ext][row] = 1
918
919                     else:
920                         cw[ext][row] = 1
921
922                 else:
923                     if row == 0:
924                         cw[ext][row] = 1
925
926                     else:
927                         cw[ext][row] = 1
928
929             if ext == 0:
930                 if row == 0:
931                     cs[ext][row] = 1
932
933                 else:
934                     cs[ext][row] = 1
935
936             else:
937                 if row == 0:
938                     cs[ext][row] = 1
939
940                 else:
941                     cs[ext][row] = 1
942
943     return cw, cs
944
945
946     """
947     # Create arrays to return
948     cw = np.zeros_like(wav_sol)
949     cs = np.zeros_like(wav_sol)
950
951     exts = cw.shape[0]
952     rows = cw.shape[1]
953
954     for ext in range(exts):
955
956         for row in range(rows):
957
958             if resPlot:
959                 if ext == 0:
960                     if row == 0:
961                         cw[ext][row] = 1
962
963                 else:
964                     cw[ext][row] = 1
965
966             else:
967                 if ext == 0:
968                     if row == 0:
969                         cw[ext][row] = 1
970
971                     else:
972                         cw[ext][row] = 1
973
974                 else:
975                     if row == 0:
976                         cw[ext][row] = 1
977
978                     else:
979                         cw[ext][row] = 1
980
981             if ext == 0:
982                 if row == 0:
983                     cs[ext][row] = 1
984
985                 else:
986                     cs[ext][row] = 1
987
988             else:
989                 if row == 0:
990                     cs[ext][row] = 1
991
992                 else:
993                     cs[ext][row] = 1
994
995     return cw, cs
996
997
998     """
999     # Create arrays to return
1000    cw = np.zeros_like(wav_sol)
1001    cs = np.zeros_like(wav_sol)
1002
1003    exts = cw.shape[0]
1004    rows = cw.shape[1]
1005
1006    for ext in range(exts):
1007
1008        for row in range(rows):
1009
1010            if resPlot:
1011                if ext == 0:
1012                    if row == 0:
1013                        cw[ext][row] = 1
1014
1015                else:
1016                    cw[ext][row] = 1
1017
1018            else:
1019                if ext == 0:
1020                    if row == 0:
1021                        cw[ext][row] = 1
1022
1023                    else:
1024                        cw[ext][row] = 1
1025
1026                else:
1027                    if row == 0:
1028                        cw[ext][row] = 1
1029
1030                    else:
1031                        cw[ext][row] = 1
1032
1033            if ext == 0:
1034                if row == 0:
1035                    cs[ext][row] = 1
1036
1037                else:
1038                    cs[ext][row] = 1
1039
1040            else:
1041                if row == 0:
1042                    cs[ext][row] = 1
1043
1044                else:
1045                    cs[ext][row] = 1
1046
1047    return cw, cs
1048
1049
1050    """
1051    # Create arrays to return
1052    cw = np.zeros_like(wav_sol)
1053    cs = np.zeros_like(wav_sol)
1054
1055    exts = cw.shape[0]
1056    rows = cw.shape[1]
1057
1058    for ext in range(exts):
1059
1060        for row in range(rows):
1061
1062            if resPlot:
1063                if ext == 0:
1064                    if row == 0:
1065                        cw[ext][row] = 1
1066
1067                else:
1068                    cw[ext][row] = 1
1069
1070            else:
1071                if ext == 0:
1072                    if row == 0:
1073                        cw[ext][row] = 1
1074
1075                    else:
1076                        cw[ext][row] = 1
1077
1078                else:
1079                    if row == 0:
1080                        cw[ext][row] = 1
1081
1082                    else:
1083                        cw[ext][row] = 1
1084
1085            if ext == 0:
1086                if row == 0:
1087                    cs[ext][row] = 1
1088
1089                else:
1090                    cs[ext][row] = 1
1091
1092            else:
1093                if row == 0:
1094                    cs[ext][row] = 1
1095
1096                else:
1097                    cs[ext][row] = 1
1098
1099    return cw, cs
1100
1101
1102    """
1103    # Create arrays to return
1104    cw = np.zeros_like(wav_sol)
1105    cs = np.zeros_like(wav_sol)
1106
1107    exts = cw.shape[0]
1108    rows = cw.shape[1]
1109
1110    for ext in range(exts):
1111
1112        for row in range(rows):
1113
1114            if resPlot:
1115                if ext == 0:
1116                    if row == 0:
1117                        cw[ext][row] = 1
1118
1119                else:
1120                    cw[ext][row] = 1
1121
1122            else:
1123                if ext == 0:
1124                    if row == 0:
1125                        cw[ext][row] = 1
1126
1127                    else:
1128                        cw[ext][row] = 1
1129
1130                else:
1131                    if row == 0:
1132                        cw[ext][row] = 1
1133
1134                    else:
1135                        cw[ext][row] = 1
1136
1137            if ext == 0:
1138                if row == 0:
1139                    cs[ext][row] = 1
1140
1141                else:
1142                    cs[ext][row] = 1
1143
1144            else:
1145                if row == 0:
1146                    cs[ext][row] = 1
1147
1148                else:
1149                    cs[ext][row] = 1
1150
1151    return cw, cs
1152
1153
1154    """
1155    # Create arrays to return
1156    cw = np.zeros_like(wav_sol)
1157    cs = np.zeros_like(wav_sol)
1158
1159    exts = cw.shape[0]
1160    rows = cw.shape[1]
1161
1162    for ext in range(exts):
1163
1164        for row in range(rows):
1165
1166            if resPlot:
1167                if ext == 0:
1168                    if row == 0:
1169                        cw[ext][row] = 1
1170
1171                else:
1172                    cw[ext][row] = 1
1173
1174            else:
1175                if ext == 0:
1176                    if row == 0:
1177                        cw[ext][row] = 1
1178
1179                    else:
1180                        cw[ext][row] = 1
1181
1182                else:
1183                    if row == 0:
1184                        cw[ext][row] = 1
1185
1186                    else:
1187                        cw[ext][row] = 1
1188
1189            if ext == 0:
1190                if row == 0:
1191                    cs[ext][row] = 1
1192
1193                else:
1194                    cs[ext][row] = 1
1195
1196            else:
1197                if row == 0:
1198                    cs[ext][row] = 1
1199
1200                else:
1201                    cs[ext][row] = 1
1202
1203    return cw, cs
1204
1205
1206    """
1207    # Create arrays to return
1208    cw = np.zeros_like(wav_sol)
1209    cs = np.zeros_like(wav_sol)
1210
1211    exts = cw.shape[0]
1212    rows = cw.shape[1]
1213
1214    for ext in range(exts):
1215
1216        for row in range(rows):
1217
1218            if resPlot:
1219                if ext == 0:
1220                    if row == 0:
1221                        cw[ext][row] = 1
1222
1223                else:
1224                    cw[ext][row] = 1
1225
1226            else:
1227                if ext == 0:
1228                    if row == 0:
1229                        cw[ext][row] = 1
1230
1231                    else:
1232                        cw[ext][row] = 1
1233
1234                else:
1235                    if row == 0:
1236                        cw[ext][row] = 1
1237
1238                    else:
1239                        cw[ext][row] = 1
1240
1241            if ext == 0:
1242                if row == 0:
1243                    cs[ext][row] = 1
1244
1245                else:
1246                    cs[ext][row] = 1
1247
1248            else:
1249                if row == 0:
1250                    cs[ext][row] = 1
1251
1252                else:
1253                    cs[ext][row] = 1
1254
1255    return cw, cs
1256
1257
1258    """
1259    # Create arrays to return
1260    cw = np.zeros_like(wav_sol)
1261    cs = np.zeros_like(wav_sol)
1262
1263    exts = cw.shape[0]
1264    rows = cw.shape[1]
1265
1266    for ext in range(exts):
1267
1268        for row in range(rows):
1269
1270            if resPlot:
1271                if ext == 0:
1272                    if row == 0:
1273                        cw[ext][row] = 1
1274
1275                else:
1276                    cw[ext][row] = 1
1277
1278            else:
1279                if ext == 0:
1280                    if row == 0:
1281                        cw[ext][row] = 1
1282
1283                    else:
1284                        cw[ext][row] = 1
1285
1286                else:
1287                    if row == 0:
1288                        cw[ext][row] = 1
1289
1290                    else:
1291                        cw[ext][row] = 1
1292
1293            if ext == 0:
1294                if row == 0:
1295                    cs[ext][row] = 1
1296
1297                else:
1298                    cs[ext][row] = 1
1299
1300            else:
1301                if row == 0:
1302                    cs[ext][row] = 1
1303
1304                else:
1305                    cs[ext][row] = 1
1306
1307    return cw, cs
1308
1309
1310    """
1311    # Create arrays to return
1312    cw = np.zeros_like(wav_sol)
1313    cs = np.zeros_like(wav_sol)
1314
1315    exts = cw.shape[0]
1316    rows = cw.shape[1]
1317
1318    for ext in range(exts):
1319
1320        for row in range(rows):
1321
1322            if resPlot:
1323                if ext == 0:
1324                    if row == 0:
1325                        cw[ext][row] = 1
1326
1327                else:
1328                    cw[ext][row] = 1
1329
1330            else:
1331                if ext == 0:
1332                    if row == 0:
1333                        cw[ext][row] = 1
1334
1335                    else:
1336                        cw[ext][row] = 1
1337
1338                else:
1339                    if row == 0:
1340                        cw[ext][row] = 1
1341
1342                    else:
1343                        cw[ext][row] = 1
1344
1345            if ext == 0:
1346                if row == 0:
1347                    cs[ext][row] = 1
1348
1349                else:
1350                    cs[ext][row] = 1
1351
1352            else:
1353                if row == 0:
1354                    cs[ext][row] = 1
1355
1356                else:
1357                    cs[ext][row] = 1
1358
1359    return cw, cs
1360
1361
1362    """
1363    # Create arrays to return
1364    cw = np.zeros_like(wav_sol)
1365    cs = np.zeros_like(wav_sol)
1366
1367    exts = cw.shape[0]
1368    rows = cw.shape[1]
1369
1370    for ext in range(exts):
1371
1372        for row in range(rows):
1373
1374            if resPlot:
1375                if ext == 0:
1376                    if row == 0:
1377                        cw[ext][row] = 1
1378
1379                else:
1380                    cw[ext][row] = 1
1381
1382            else:
1383                if ext == 0:
1384                    if row == 0:
1385                        cw[ext][row] = 1
1386
1387                    else:
1388                        cw[ext][row] = 1
1389
1390                else:
1391                    if row == 0:
1392                        cw[ext][row] = 1
1393
1394                    else:
1395                        cw[ext][row] = 1
1396
1397            if ext == 0:
1398                if row == 0:
1399                    cs[ext][row] = 1
1400
1401                else:
1402                    cs[ext][row] = 1
1403
1404            else:
1405                if row == 0:
1406                    cs[ext][row] = 1
1407
1408                else:
1409                    cs[ext][row] = 1
1410
1411    return cw, cs
1412
1413
1414    """
1415    # Create arrays to return
1416    cw = np.zeros_like(wav_sol)
1417    cs = np.zeros_like(wav_sol)
1418
1419    exts = cw.shape[0]
1420    rows = cw.shape[1]
1421
1422    for ext in range(exts):
1423
1424        for row in range(rows):
1425
1426            if resPlot:
1427                if ext == 0:
1428                    if row == 0:
1429                        cw[ext][row] = 1
1430
1431                else:
1432                    cw[ext][row] = 1
1433
1434            else:
1435                if ext == 0:
1436                    if row == 0:
1437                        cw[ext][row] = 1
1438
1439                    else:
1440                        cw[ext][row] = 1
1441
1442                else:
1443                    if row == 0:
1444                        cw[ext][row] = 1
1445
1446                    else:
1447                        cw[ext][row] = 1
1448
1449            if ext == 0:
1450                if row == 0:
1451                    cs[ext][row] = 1
1452
1453                else:
1454                    cs[ext][row] = 1
1455
1456            else:
1457                if row == 0:
1458                    cs[ext][row] = 1
1459
1460                else:
1461                    cs[ext][row] = 1
1462
1463    return cw, cs
1464
1465
1466    """
1467    # Create arrays to return
1468    cw = np.zeros_like(wav_sol)
1469    cs = np.zeros_like(wav_sol)
1470
1471    exts = cw.shape[0]
1472    rows = cw.shape[1]
1473
1474    for ext in range(exts):
1475
1476        for row in range(rows):
1477
1478            if resPlot:
1479                if ext == 0:
1480                    if row == 0:
1481                        cw[ext][row] = 1
1482
1483                else:
1484                    cw[ext][row] = 1
1485
1486            else:
1487                if ext == 0:
1488                    if row == 0:
1489                        cw[ext][row] = 1
1490
1491                    else:
1492                        cw[ext][row] = 1
1493
1494                else:
1495                    if row == 0:
1496                        cw[ext][row] = 1
1497
1498                    else:
1499                        cw[ext][row] = 1
1500
1501            if ext == 0:
1502                if row == 0:
1503                    cs[ext][row] = 1
1504
1505                else:
1506                    cs[ext][row] = 1
1507
1508            else:
1509                if row == 0:
1510                    cs[ext][row] = 1
1511
1512                else:
1513                    cs[ext][row] = 1
1514
1515    return cw, cs
1516
1517
1518    """
1519    # Create arrays to return
1520    cw = np.zeros_like(wav_sol)
1521    cs = np.zeros_like(wav_sol)
1522
1523    exts = cw.shape[0]
1524    rows = cw.shape[1]
1525
1526    for ext in range(exts):
1527
1528        for row in range(rows):
1529
1530            if resPlot:
1531                if ext == 0:
1532                    if row == 0:
1533                        cw[ext][row] = 1
1534
1535                else:
1536                    cw[ext][row] = 1
1537
1538            else:
1539                if ext == 0:
1540                    if row == 0:
1541                        cw[ext][row] = 1
1542
1543                    else:
1544                        cw[ext][row] = 1
1545
1546                else:
1547                    if row == 0:
1548                        cw[ext][row] = 1
1549
1550                    else:
1551                        cw[ext][row] = 1
1552
1553            if ext == 0:
1554                if row == 0:
1555                    cs[ext][row] = 1
1556
1557                else:
1558                    cs[ext][row] = 1
1559
1560            else:
1561                if row == 0:
1562                    cs[ext][row] = 1
1563
1564                else:
1565                    cs[ext][row] = 1
1566
1567    return cw, cs
1568
1569
1570    """
1571    # Create arrays to return
1572    cw = np.zeros_like(wav_sol)
1573    cs = np.zeros_like(wav_sol)
15
```

```

222     # Get middle row (to interpolate the rest of the rows to)
223     avg_max = [np.where(spec[ext, :, col] == spec[ext, :, 
224         ↪ col].max())[0][0] for col in range(spec[ext].shape[1])]
225     avg_max = np.sum(avg_max) // spec[ext].shape[1]
226
227     # Get wavelength values at row with most trace
228     wav = wav_sol[ext, avg_max, :]
229
230     # Correct extensions based on wavelength
231     cw[ext, :, :] = wav
232
233     # Spec ext
234     # for row in range(rows):
235     #     f_2d = interpolate.interp2d(
236     #         wav_sol[ext, row],
237     #         np.arange(rows),
238     #         spec[ext],
239     #         )
240     #     cs[ext] = f_2d(cw[ext, row], np.arange(rows))
241     for row in range(rows):
242         cs[ext][row, :] = np.interp(
243             wav,
244             wav_sol[ext][row, :],
245             spec[ext][row, :]
246         )
247
248     # Plot results
249     if resPlot:
250         fig, ax1 = plt.subplots(figsize=[20, 4])
251         ax1.imshow(cs[ext],
252                     vmax=cs[0].mean() + 2*cs[0].std(),
253                     vmin=cs[0].mean() - 2*cs[0].std()
254                 )
255         print(f"Average continuum of {'E' if ext else 'O'} at
256             ↪ {np.median(np.median(cs[ext], axis=0)):4.3f}")
257         ax2 = ax1.twinx()
258         ax2.hlines(np.median(np.median(cs[ext], axis=0)), 0,
259             ↪ cs[ext].shape[-1], colors='black')
260         ax2.plot(cs[ext].mean(axis=0), "k", label=f"mean {'E',
261             ↪ if ext else 'O'}")
262         ax2.plot(np.median(cs[ext], axis=0), "r",
263             ↪ label=f"median {'E' if ext else 'O'}")
264         ax2.legend()
265         plt.show()
266
267     return cw, cs
268
269     # MARK: Remove continuum
270     def remove_cont(self, spec: np.ndarray, wav: np.ndarray) ->
271         ↪ np.ndarray:
272         ctm = continuum(wav, spec, deg=self.cont_ord,
273             ↪ plot=self.can_plot)
274
275         return self.spec / ctm - 1
276
277     # MARK: Skyline
278     def skyline(self, filename) -> None:

```

```

# raise error if arc image
with pyfits.open(filename) as hdul:
    if hdul[0].header['OBSTYPE'] == 'ARC':
        logging.warning(f"ARC images, {filename}, contain no
                         sky lines. File skipped.")
        return

    # Load data
    spec2D = hdul["SCI"].data
    wav2D = hdul["WAV"].data
    bpm2D = hdul["BPM"].data

    spec2D *= ~bpm2D

    logging.debug(f"skylines - {filename.name} - spec:
                  {spec2D.shape}")

    # Mask trace
    # TODO@JustinotherGitter: Add trace masking if median is
    # insufficient

    # Save initial feature widths
    # Mean to not filter out skewed features
    spec1D_init = np.mean(spec2D, axis=1)
    # Median to sort for most common wavelength
    wav1D_init = np.median(wav2D, axis=1)

    peaks = [[] for _ in range(2)]
    properties = [[] for _ in range(2)]

if self.can_plot: fig, axs = plt.subplots(2, 1)
for ext in range(2):
    peaks[ext], properties[ext] = signal.find_peaks(
        spec1D_init[ext],
        prominence=0.5 * np.std(spec1D_init[ext]),
        width=[1, 1000],
        rel_height=0.3
    )
    peak_width = [properties[ext]['widths'],
                  properties[ext]['width_heights'],
                  properties[ext]['left_ips'],
                  properties[ext]['right_ips']]

    logging.debug(f"skylines - initial features {'E' if ext
                  else 'O'}: {len(peaks[ext])}")
    logging.debug(f"skylines - props: {properties[ext]}")

if self.can_plot:
    axs[ext].plot(spec1D_init[ext], label=f'{0 if ext
                                              else 1} initial')
    axs[ext].vlines(peaks[ext], 0,
                    np.mean(spec1D_init[ext]), color='r', label='Feature
                    positions')
    axs[ext].hlines(*peak_width[1:], color='g',
                   label='Initial widths')
    axs[ext].errorbar(
        peaks[ext],
        properties[ext]['prominences'],
        yerr=peak_width[0])

```

```

320             xerr=np.array([
321                 peaks[ext] - properties[ext]['left_ips'],
322                 properties[ext]['right_ips'] - peaks[ext]
323             ]),
324             fmt='x',
325             label='Prominences'
326         )
327
328     if self.can_plot:
329         for ax in axs: ax.legend()
330         plt.show()
331
332     # Transform data, skip if filename starts with 't'
333     if self.must_transform or not filename.name.startswith('t'):
334         wav2D, spec2D = self.transform(wav2D, spec2D, self.can_plot)
335
336     # Convert to 1D spectra
337     wav1D = np.median(wav2D, axis=1)
338     spec1D = np.median(spec2D, axis=1)
339
340     # Remove continuum
341     if self.cont_ord > 0:
342         for ext in range(2):
343             # spec1D_init[ext] = self.remove_cont(spec1D_init[ext],
344             #                                     ↪ wav1D[ext])
345             # spec1D[ext] = self.remove_cont(spec1D[ext],
346             #                                     ↪ wav1D[ext])
347             pass
348
349     if self.can_plot:
350         # Plot transformed & normalized feature widths
351         plt.plot(spec1D_init[0], label="O, initial")
352         plt.plot(spec1D_init[1], label="E, initial")
353         plt.plot(spec1D[0], label="O spec")
354         plt.plot(spec1D[1], label="E spec")
355         plt.legend()
356         plt.show()
357
358     # Find observed skylines
359     # skyline_cols, properties = find_peaks(sky_norm,
360     #                                         ↪ prominence=0.2) # 1000
361     # prominence is basically the ylimit above which peaks should
362     # be found
363     skyline_cols, skyline_wavs, properties = [], [], []
364     for ext in range(2):
365         cols, prop = signal.find_peaks(wav1D[ext], spec1D[ext],
366                                         ↪ prominence=1)
367         skyline_cols.append(cols)
368         properties.append(prop)
369         skyline_wavs.append(wav1D[ext, cols]) # col_to_wav(coeff2,
370         ↪ skyline_cols)
371
372     plt.plot(wav1D[ext, cols], cols, label='Observed')
373     plt.xlabel('Wavelength ($\AA$)')
374     plt.ylabel('Counts')
375     plt.legend()
376     plt.show()
377

```

```

372     for ext in range(2):
373         for i in range(len(skyline_cols[ext])):
374             logging.debug(("skylines - found features:\n"
375                         f"{properties[ext]['right_bases'][i] - "
376                         f"properties[ext]['left_bases'][i]:4d}\t"
377                         f"{skyline_cols[ext][i]:4d} - "
378                         f" {skyline_wavs[ext][i]:.1f} - "
379                         f" {properties[ext]['prominences'][i]:.2f}"
380                     ))
381
382
383     # Return results
384     return
385
386
387     # Plot results
388
389     # Find deviation of observed skyliners from known skyliners
390
391     # Load known skyliners
392     skyliners = self.load_sky_lines()
393
394     # Save results
395     raise NotImplementedError
396
397
398     def show_frame(self, frame: np.ndarray, title: str = None, label:
399     ↪ str = None, std: int = 1) -> None:
400         if not self.can_plot:
401             return
402
403         fig, axs = plt.subplots(2, 1)
404         axs[0].set_title(title)
405         axs[0].imshow(
406             frame[0],
407             label=f"O beam - {label}",
408             vmin=frame[0].mean() - std * frame[0].std(),
409             vmax=frame[0].mean() + std * frame[0].std()
410         )
411         axs[1].imshow(
412             frame[1],
413             label=f"E beam - {label}",
414             vmin=frame[1].mean() - std * frame[0].std(),
415             vmax=frame[1].mean() + std * frame[0].std()
416         )
417         for ax in axs: ax.legend()
418         plt.show()
419
420
421
422     # MARK: Process all listed images
423     def process(self,) -> None:
424         if self.beams == 'OE':

```

```
425     for fl in self.fits_list:
426         logging.info(f'{fl} skylines of {fl}.')
427         self.skylines(fl)
428         self.plot()
429
430     if self.beams in ['O', 'E']:
431         for fl in self.fits_list:
432             logging.info(f'{self.beams} skylines of {fl}.')
433             self.skylines(fl)
434             self.plot()
435
436     return
437
438
439 # MARK: Main function
440 def main(argv) -> None:
441     return
442
443 if __name__ == "__main__":
444     main(sys.argv[1:])
```



# Bibliography

R. R. J. Antonucci and J. S. Miller. Spectropolarimetry and the nature of NGC 1068. *ApJ*, 297:621–632, October 1985. doi: 10.1086/163559.

George B. Arfken and Hans J. Weber. Mathematical methods for physicists, 1999.

Astropy Collaboration, T. P. Robitaille, E. J. Tollerud, P. Greenfield, M. Droettboom, E. Bray, T. Aldcroft, M. Davis, A. Ginsburg, A. M. Price-Whelan, W. E. Kerzendorf, A. Conley, N. Crighton, K. Barbary, D. Muna, H. Ferguson, F. Grollier, M. M. Parikh, P. H. Nair, H. M. Unther, C. Deil, J. Woillez, S. Conseil, R. Kramer, J. E. H. Turner, L. Singer, R. Fox, B. A. Weaver, V. Zabalza, Z. I. Edwards, K. Azalee Bostroem, D. J. Burke, A. R. Casey, S. M. Crawford, N. Dencheva, J. Ely, T. Jenness, K. Labrie, P. L. Lim, F. Pierfederici, A. Pontzen, A. Ptak, B. Refsdal, M. Servillat, and O. Streicher. Astropy: A community Python package for astronomy. *A&A*, 558:A33, October 2013. doi: 10.1051/0004-6361/201322068.

Astropy Collaboration, A. M. Price-Whelan, B. M. Sipőcz, H. M. Günther, P. L. Lim, S. M. Crawford, S. Conseil, D. L. Shupe, M. W. Craig, N. Dencheva, A. Ginsburg, J. T. VanderPlas, L. D. Bradley, D. Pérez-Suárez, M. de Val-Borro, T. L. Aldcroft, K. L. Cruz, T. P. Robitaille, E. J. Tollerud, C. Ardelean, T. Babej, Y. P. Bach, M. Bachetti, A. V. Bakanov, S. P. Bamford, G. Barentsen, P. Barmby, A. Baumbach, K. L. Berry, F. Biscani, M. Boquien, K. A. Bostroem, L. G. Bouma, G. B. Brammer, E. M. Bray, H. Breytenbach, H. Buddelmeijer, D. J. Burke, G. Calderone, J. L. Cano Rodríguez, M. Cara, J. V. M. Cardoso, S. Cheedella, Y. Copin, L. Corrales, D. Crichton, D. D’Avella, C. Deil, É. Depagne, J. P. Dietrich, A. Donath, M. Droettboom, N. Earl, T. Erben, S. Fabbro, L. A. Ferreira, T. Finethy, R. T. Fox, L. H. Garrison, S. L. J. Gibbons, D. A. Goldstein, R. Gommers, J. P. Greco, P. Greenfield, A. M. Groener, F. Grollier, A. Hagen, P. Hirst, D. Homeier, A. J. Horton, G. Hosseinzadeh, L. Hu, J. S. Hunkeler, Ž. Ivezić, A. Jain, T. Jenness, G. Kanarek, S. Kendrew, N. S. Kern, W. E. Kerzendorf, A. Khvalko, J. King, D. Kirkby, A. M. Kulkarni, A. Kumar, A. Lee, D. Lenz, S. P. Littlefair, Z. Ma, D. M. Macleod, M. Mastropietro, C. McCully, S. Montagnac, B. M. Morris, M. Mueller, S. J. Mumford, D. Muna, N. A. Murphy, S. Nelson, G. H. Nguyen, J. P. Ninan, M. Nöthe, S. Ogaz, S. Oh, J. K. Parejko, N. Parley, S. Pasqual, R. Patil, A. A. Patil, A. L. Plunkett, J. X. Prochaska, T. Rastogi, V. Reddy Janga, J. Sabater, P. Sakurikar, M. Seifert, L. E. Sherbert, H. Sherwood-Taylor, A. Y. Shih, J. Sick, M. T. Silbiger, S. Singanamalla, L. P. Singer, P. H. Sladen, K. A. Sooley, S. Sornarajah, O. Streicher, P. Teuben, S. W. Thomas, G. R. Tremblay, J. E. H. Turner, V. Terrón, M. H. van Kerkwijk, A. de la Vega, L. L. Watkins, B. A. Weaver, J. B.

- Whitmore, J. Woillez, V. Zabalza, and Astropy Contributors. The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package. AJ, 156(3):123, September 2018. doi: 10.3847/1538-3881/aabc4f.
- Astropy Collaboration, Adrian M. Price-Whelan, Pey Lian Lim, Nicholas Earl, Nathaniel Starkman, Larry Bradley, David L. Shupe, Aarya A. Patil, Lia Corrales, C. E. Brasseur, Maximilian N"othe, Axel Donath, Erik Tollerud, Brett M. Morris, Adam Ginsburg, Eero Vaher, Benjamin A. Weaver, James Tocknell, William Jamieson, Marten H. van Kerkwijk, Thomas P. Robitaille, Bruce Merry, Matteo Bachetti, H. Moritz G"unther, Thomas L. Aldcroft, Jaime A. Alvarado-Montes, Anne M. Archibald, Attila B'odi, Shreyas Bapat, Geert Barentsen, Juanjo Baz'an, Manish Biswas, M'ed'eric Boquien, D. J. Burke, Daria Cara, Mihai Cara, Kyle E. Conroy, Simon Conseil, Matthew W. Craig, Robert M. Cross, Kelle L. Cruz, Francesco D'Eugenio, Nadia Dencheva, Hadrien A. R. Devillepoix, J"org P. Dietrich, Arthur Davis Eigenbrot, Thomas Erben, Leonardo Ferreira, Daniel Foreman-Mackey, Ryan Fox, Nabil Freij, Suyog Garg, Robel Geda, Lauren Glattly, Yash Gondhalekar, Karl D. Gordon, David Grant, Perry Greenfield, Austen M. Groener, Steve Guest, Sebastian Gurovich, Rasmus Handberg, Akeem Hart, Zac Hatfield-Dodds, Derek Homeier, Griffin Hosseinzadeh, Tim Jenness, Craig K. Jones, Prajwel Joseph, J. Bryce Kalmbach, Emir Karamehmetoglu, Mikolaj Kaluszy'nski, Michael S. P. Kelley, Nicholas Kern, Wolfgang E. Kerzendorf, Eric W. Koch, Shankar Kulumani, Antony Lee, Chun Ly, Zhiyuan Ma, Conor MacBride, Jakob M. Maljaars, Demitri Muna, N. A. Murphy, Henrik Norman, Richard O'Steen, Kyle A. Oman, Camilla Pacifici, Sergio Pascual, J. Pascual-Granado, Rohit R. Patil, Gabriel I. Perren, Timothy E. Pickering, Tanuj Rastogi, Benjamin R. Roulston, Daniel F. Ryan, Eli S. Rykoff, Jose Sabater, Parikshit Sakurikar, Jes'us Salgado, Aniket Sanghi, Nicholas Saunders, Volodymyr Savchenko, Ludwig Schwardt, Michael Seifert-Eckert, Albert Y. Shih, Anany Shrey Jain, Gyanendra Shukla, Jonathan Sick, Chris Simpson, Sudheesh Singanamalla, Leo P. Singer, Jaladh Singhal, Manodeep Sinha, Brigitta M. SipHocz, Lee R. Spitler, David Stansby, Ole Streicher, Jani Sumak, John D. Swinbank, Dan S. Taranu, Nikita Tewary, Grant R. Tremblay, Miguel de Val-Borro, Samuel J. Van Kooten, Zlatan Vasovi'c, Shresth Verma, Jos'e Vin'icius de Miranda Cardoso, Peter K. G. Williams, Tom J. Wilson, Benjamin Winkel, W. M. Wood-Vasey, Rui Xue, Peter Yoachim, Chen Zhang, Andrea Zonca, and Astropy Project Contributors. The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package. ApJ, 935(2):167, August 2022. doi: 10.3847/1538-4357/ac7c74.
- S. Bagnulo, M. Landolfi, J. D. Landstreet, E. Landi Degl'Innocenti, L. Fossati, and M. Sterzik. Stellar spectropolarimetry with retarder waveplate and beam splitter devices. Publications of the Astronomical Society of the Pacific, 121(883):993, aug 2009. doi: 10.1086/605654. URL <https://dx.doi.org/10.1086/605654>.
- Erasmus Bartholinus. Experimenta crystalli islandici dis-diaclastici, quibus mira et insolita refractio detegitur (copenhagen, 1670). Edinburgh Philosophical Journal, 1:271, 1670.

D. Scott Birney, Guillermo Gonzalez, and David Oesper.

Observational Astronomy - 2nd Edition. Cambridge University Press, 2006. doi: 10.2277/0521853702.

Janus D. Brink, Moses K. Mogotsi, Melanie Saayman, Nicolaas M. Van der Merwe, Jonathan Love, and Alrin Christians. Preparing the SALT for near-infrared observations. In Heather K. Marshall, Jason Spyromilio, and Tomonori Usuda, editors, Ground-based and Airborne Telescopes IX, volume 12182 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, page 121822E, August 2022. doi: 10.1117/12.2627328.

David A. H. Buckley, Gerhard P. Swart, and Jacobus G. Meiring. Completion and commissioning of the Southern African Large Telescope. In Larry M. Stepp, editor, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, volume 6267 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, page 62670Z, June 2006. doi: 10.1117/12.673750.

Christian Buil. CCD astronomy : construction and use of an astronomical CCD camera / Christian Buil ; translated and adapted from the French by Emmanuel and Barbara Davoust. Willmann-Bell, Richmond, Va, 1st english ed. edition, 1991. ISBN 0943396298.

Eric B. Burgh, Kenneth H. Nordsieck, Henry A. Kobulnicky, Ted B. Williams, Daragh O'Donoghue, Michael P. Smith, and Jeffrey W. Percival. Prime Focus Imaging Spectrograph for the Southern African Large Telescope: optical design. In Masanori Iye and Alan F. M. Moorwood, editors, Instrument Design and Performance for Optical/Infrared Ground-based Telescopes, volume 4841 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, pages 1463–1471, March 2003. doi: 10.1117/12.460312.

Subrahmanyan Chandrasekhar. Radiative transfer, 1950.

Marshall H. Cohen. Genesis of the 1000-foot Arecibo dish. Journal of Astronomical History and Heritage, 12(2):141–152, July 2009.

E. Collett. Field Guide to Polarization. Field Guides. SPIE Press, 2005. ISBN 9780819458681. URL <https://books.google.co.za/books?id=5lJwcCsLbLsC>.

J. Cooper, B. van Soelen, and R. Britto. Development of tools for SALT/RSS spectropolarimetry reductions: application to the blazar 3C279. In High Energy Astrophysics in Southern Africa 2021, page 56, May 2022. doi: 10.22323/1.401.0056.

M. Craig, S. Crawford, M. Seifert, T. Robitaille, B. Sipőcz, J. Walawender, Z. Vinícius, J. P. Ninan, M. Droettboom, J. Youn, E. Tollerud, E. Bray, N. Walker, Janga V. R., C. Stotts, H. M. Günther, E. Rol, Yoonsoo P. Bach, L. Bradley, C. Deil, A. Price-Whelan, K. Barbary, A. Horton, W. Schoenell, N. Heidt, F. Gasdia, S. Nelson, and O. Streicher. astropy/ccdproc: v1.3.0.post1, December 2017. URL <https://doi.org/10.5281/zenodo.1069648>.

- G. Dahlquist and Å. Björck. Numerical Methods. Dover Books on Mathematics. Dover Publications, 2003. ISBN 9780486428079. URL <https://books.google.co.ls/books?id=armfeHpJIwAC>.
- E. Landi Degl’Innocenti, S. Bagnulo, and L. Fossati. Polarimetric standardization, 2006.
- Egidio Landi Degl’Innocenti. The physics of polarization. Proceedings of the International Astronomical Union, 10(S305):1–1, 2014.
- Egidio Landi Degl’Innocenti and M. Landolfi. Polarization in Spectral Lines, volume 307. Springer Dordrecht, 2004. doi: 10.1007/978-1-4020-2415-3.
- Königlich Bayerische Akademie der Wissenschaften. Denkschriften der Königlichen Akademie der Wissenschaften zu München für das Jahre 1820 und 1821, volume 8. Die Akademie, 1824. URL <https://books.google.co.za/books?id=k-EAAAAAYAAJ>.
- J. F. Donati, M. Semel, B. D. Carter, D. E. Rees, and A. Collier Cameron. Spectropolarimetric observations of active stars. MNRAS, 291(4):658–682, November 1997. doi: 10.1093/mnras/291.4.658.
- I. V. Florinsky and A. N. Pankratov. Digital terrain modeling with the chebyshev polynomials. Machine Learning and Data Analysis, 1(12):1647 – 1659, 2015. doi: 10.48550/ARXIV.1507.03960. URL <https://arxiv.org/abs/1507.03960>.
- Augustin Fresnel. Oeuvres completes d’Augustin Fresnel: 3. Imprimerie impériale, 1870.
- L. M. Freyhammer, M. I. Andersen, T. Arentoft, C. Sterken, and P. Nørregaard. On Cross-talk Correction of Images from Multiple-port CCDs. Experimental Astronomy, 12(3):147–162, January 2001. doi: 10.1023/A:1021820418263.
- David J Griffiths. Introduction to electrodynamics, 2005.
- George E. Hale. The Zeeman Effect in the Sun. PASP, 20(123):287, December 1908. doi: 10.1086/121847.
- George E. Hale. 16. On the Probable Existence of a Magnetic Field in Sun-Spots, pages 96–105. Harvard University Press, Cambridge, MA and London, England, 1979. ISBN 9780674366688. doi: doi:10.4159/harvard.9780674366688.c19. URL <https://doi.org/10.4159/harvard.9780674366688.c19>.
- P. D. Hale and G. W. Day. Stability of birefringent linear retarders(waveplates). Appl. Opt., 27(24):5146–5153, Dec 1988. doi: 10.1364/AO.27.005146. URL <https://opg.optica.org/ao/abstract.cfm?URI=ao-27-24-5146>.
- C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant.

- Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL <https://doi.org/10.1038/s41586-020-2649-2>.
- E. Hecht. *Optics*. Pearson Education, Incorporated, 2017. ISBN 9780133977226. URL <https://books.google.co.za/books?id=ZarLoQEACAAJ>.
- Steve B. Howell. *Handbook of CCD Astronomy*, volume 5. Cambridge University Press, 2006.
- J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.
- Christian Huygens. Treatise on light, 1690. translated by Thompson, s. p., 1690. URL <https://www.gutenberg.org/files/14725/14725-h/14725-h.htm>.
- Mourad E. H. Ismail. *Classical and Quantum Orthogonal Polynomials in One Variable*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2005. doi: 10.1017/CBO9781107325982.
- James Janesick, James T. Andrews, and Tom Elliott. Fundamental performance differences between CMOS and CCD imagers: Part 1. In David A. Dorn and Andrew D. Holland, editors, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 6276 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, page 62760M, June 2006. doi: 10.1117/12.678867.
- F.A. Jenkins and H.E. White. *Fundamentals of Optics*. International student edition. McGraw-Hill, 1976. ISBN 9780070323308. URL <https://books.google.co.za/books?id=dCdRAAAAMAAJ>.
- Christoph U. Keller. Instrumentation for astrophysical spectropolarimetry. *Astrophysical Spectropolarimetry*, 1:303–354, 2002.
- G. Kirchhoff and R. Bunsen. Chemische Analyse durch Spectralbeobachtungen. *Annalen der Physik*, 189(7):337–381, January 1861. doi: 10.1002/andp.18611890702.
- Henry A. Kobulnicky, Kenneth H. Nordsieck, Eric B. Burgh, Michael P. Smith, Jeffrey W. Percival, Ted B. Williams, and Darragh O’Donoghue. Prime focus imaging spectrograph for the Southern African large telescope: operational modes. In Masanori Iye and Alan F. M. Moorwood, editors, *Instrument Design and Performance for Optical/Infrared Ground-based Telescopes*, volume 4841 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 1634–1644, March 2003. doi: 10.1117/12.460315.
- Gerard Leng. Compression of aircraft aerodynamic database using multivariable chebyshev polynomials. *Advances in Engineering Software*, 28(2):133–141, 1997. ISSN 0965-9978. doi: [https://doi.org/10.1016/S0965-9978\(96\)00043-9](https://doi.org/10.1016/S0965-9978(96)00043-9). URL <https://www.sciencedirect.com/science/article/pii/S0965997896000439>.

- Dave Litwiller. Ccd vs. cmos. *Photonics spectra*, 35(1):154–158, 2001.
- Dongyue Liu and Bryan M. Hennelly. Improved wavelength calibration by modeling the spectrometer. *Applied Spectroscopy*, 76(11):1283–1299, 2022. doi: 10.1177/00037028221111796. URL <https://doi.org/10.1177/00037028221111796>. PMID: 35726593.
- Etienne L. Malus. Sur une propriété de la lumière réfléchie. *Mém. Phys. Chim. Soc. d'Arcueil*, 2:143–158, 1809.
- Curtis McCully, Steve Crawford, Gabor Kovacs, Erik Tollerud, Edward Betts, Larry Bradley, Matt Craig, James Turner, Ole Streicher, Brigitta Sipocz, Thomas Robitaille, and Christoph Deil. astropy/astroscrappy: v1.0.5 zenodo release, November 2018. URL <https://doi.org/10.5281/zenodo.1482019>.
- I. Newton and W. Innys. *Opticks:: Or, A Treatise of the Reflections, Refractions, Inflections and Colours of Light*. Opticks:: Or, A Treatise of the Reflections, Refractions, Inflections and Colours of Light. William Innys at the West-End of St. Paul's., 1730. URL <https://books.google.co.za/books?id=GnAFAAAQAAJ>.
- Kenneth H. Nordsieck, Kurt P. Jaehnig, Eric B. Burgh, Henry A. Kobulnicky, Jeffrey W. Percival, and Michael P. Smith. Instrumentation for high-resolution spectropolarimetry in the visible and far-ultraviolet. In Silvano Fineschi, editor, *Polarimetry in Astronomy*, volume 4843 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 170–179, February 2003. doi: 10.1117/12.459288.
- D. O'Donoghue, D. A. H. Buckley, L. A. Balona, D. Bester, L. Botha, J. Brink, D. B. Carter, P. A. Charles, A. Christians, F. Ebrahim, R. Emmerich, W. Esterhuyse, G. P. Evans, C. Fourie, P. Fourie, H. Gajjar, M. Gordon, C. Gumede, M. de Kock, A. Koeslag, W. P. Koorts, H. Kriel, F. Marang, J. G. Meiring, J. W. Menzies, P. Menzies, D. Metcalfe, B. Meyer, L. Nel, J. O'Connor, F. Osman, C. Du Plessis, H. Rall, A. Riddick, E. Romero-Colmenero, S. B. Potter, C. Sass, H. Schalekamp, N. Sessions, S. Siyengo, V. Sopela, H. Steyn, J. Stoffels, J. Scholtz, G. Swart, A. Swat, J. Swiegers, T. Tiheli, P. Vaisanen, W. Whittaker, and F. van Wyk. First science with the Southern African Large Telescope: peering at the accreting polar caps of the eclipsing polar SDSS J015543.40+002807.2. *MNRAS*, 372(1):151–162, October 2006. doi: 10.1111/j.1365-2966.2006.10834.x.
- Darragh O'Donoghue. Correction of spherical aberration in the Southern African Large Telescope (SALT). In Philippe Dierickx, editor, *Optical Design, Materials, Fabrication, and Maintenance*, volume 4003 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 363–372, July 2000. doi: 10.1117/12.391526.
- Darragh O'Donoghue. Atmospheric dispersion corrector for the Southern African Large Telescope (SALT). In Richard G. Bingham and David D. Walker, editors, *Large Lenses and Prisms*, volume 4411 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 79–84, February 2002. doi: 10.1117/12.454874.

- Ferdinando Patat and Martino Romaniello. Error Analysis for Dual-Beam Optical Linear Polarimetry. *PASP*, 118(839):146–161, January 2006. doi: 10.1086/497581.
- Alba Peinado, Angel Lizana, Josep Vidal, Claudio Iemmi, and Juan Campos. Optimization and performance criteria of a stokes polarimeter based on two variable retarders. *Opt. Express*, 18(10):9815–9830, May 2010. doi: 10.1364/OE.18.009815. URL <https://opg.optica.org/oe/abstract.cfm?URI=oe-18-10-9815>.
- W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes* 3rd Edition: The Art of Scientific Computing. Cambridge University Press, 2007. ISBN 9780521880688. URL <https://books.google.co.za/books?id=1aA0dzK3FegC>.
- J. R. Priebe. Operational form of the mueller matrices. *J. Opt. Soc. Am.*, 59(2):176–180, Feb 1969. doi: 10.1364/JOSA.59.000176. URL <https://opg.optica.org/abstract.cfm?URI=josa-59-2-176>.
- Lawrence W. Ramsey, M. T. Adams, Thomas G. Barnes, John A. Booth, Mark E. Cornell, James R. Fowler, Niall I. Gaffney, John W. Glaspey, John M. Good, Gary J. Hill, Philip W. Kelton, Victor L. Krabbendam, L. Long, Phillip J. MacQueen, Frank B. Ray, Randall L. Ricklefs, J. Sage, Thomas A. Sebring, W. J. Spiesman, and M. Steiner. Early performance and present status of the Hobby-Eberly Telescope. In Larry M. Stepp, editor, *Advanced Technology Optical/IR Telescopes VI*, volume 3352 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 34–42, August 1998. doi: 10.1117/12.319287.
- Maria C. Simon. Wollaston prism with large split angle. *Appl. Opt.*, 25(3):369–376, Feb 1986. doi: 10.1364/AO.25.000369. URL <https://opg.optica.org/ao/abstract.cfm?URI=ao-25-3-369>.
- G. G. Stokes. On the Composition and Resolution of Streams of Polarized Light from different Sources. *Transactions of the Cambridge Philosophical Society*, 9:399, January 1852.
- Doug Tody. The IRAF Data Reduction and Analysis System. In David L. Crawford, editor, *Instrumentation in astronomy VI*, volume 627 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, page 733, January 1986. doi: 10.1117/12.968154.
- Doug Tody. IRAF in the Nineties. In R. J. Hanisch, R. J. V. Brissenden, and J. Barnes, editors, *Astronomical Data Analysis Software and Systems II*, volume 52 of *Astronomical Society of the Pacific Conference Series*, page 173, January 1993.
- Stephen F. Tonkin. *Practical Amateur Spectroscopy*. The Patrick Moore Practical Astronomy Series. Springer London, 2013. ISBN 9781447101277. URL <https://books.google.fr/books?id=b2fgBwAAQBAJ>.
- Pieter G. van Dokkum. Cosmic-Ray Rejection by Laplacian Edge Detection. *PASP*, 113(789):1420–1427, November 2001. doi: 10.1086/323894.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

L. Wang and J. C. Wheeler. Spectropolarimetry of supernovae. *ARA&A*, 46:433–474, September 2008. doi: 10.1146/annurev.astro.46.060407.145139.

Marsha J. Wolf, Matthew A. Bershadsky, Michael P. Smith, Kurt P. Jaehnig, Jeffrey W. Percival, Joshua E. Oppor, Mark P. Mulligan, and Ron J. Koch. Laboratory performance and commissioning status of the SALT NIR integral field spectrograph. In Christopher J. Evans, Julia J. Bryant, and Kentaro Motohara, editors, *Ground-based and Airborne Instrumentation for Astronomy IX*, volume 12184 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, page 1218407, August 2022. doi: 10.1117/12.2630242.

William H. Wollaston. XII. A Method of Examining Refractive and Dispersive Powers, by Prismatic Reflection. *Philosophical Transactions of the Royal Society of London Series I*, 92:365–380, January 1802. doi: 10.1098/rstl.1802.0013.

# List of Acronyms

IRAF	Image Reduction and Analysis Facility
POLSLAT	Polarimetric reductions for SALT
STOPS	Supplementary Tools for POLSLAT Spectro-polarimetry
ADC	Analog-to-Digital Converter
BPM	Bad Pixel Map
CCD	Charged-Coupled Device
CLI	Command Line Interface
CMOS	Complementary Metal-Oxide-Semiconductor
FITS	Flexible Image Transport System
FWHM	Full Width at Half Maximum
GUI	Graphical User Interface
HDU	Header Data Unit
HET	Hobby-Eberly Telescope
HRS	High Resolution Spectrograph
L+45°	Linear +45° Polarized
L-45°	Linear -45° Polarized
LCP	Left Circularly Polarized
LHP	Linear Horizontally Polarized
LVP	Linear Vertically Polarized
NIR	Near Infra-Red
NIRWALS	Near Infra-Red Washburn Labs Spectrograph
RCP	Right Circularly Polarized
RMS	Root Mean Square
RSS	Robert Stobie Spectrograph
S/N	Signal-to-Noise Ratio
SAAO	South African Astronomical Observatory
SALT	Southern African Large Telescope
SALTICAM	SALT Imaging Camera
UV	Ultraviolet
VPH	Volume Phase Holographic