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: April 18, 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 $\rightarrow$ 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

Chapter 1

Introduction

TODO: Very short intro to Spectroscopy, Polarisation, and Spectropolarisation 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.

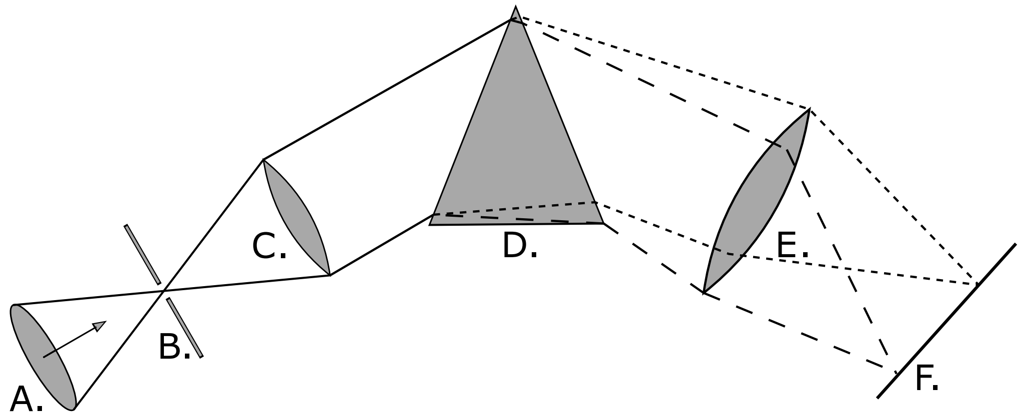

Figure 2.1: Layout depicting the light path through a spectrometer. Diagram adapted from Birney et al. (2006).

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 miniscule 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:

$$R = \frac{\lambda}{FWHM}, \tag{2.1}$$

where λ 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, λ. 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:

$$\frac{\partial\theta}{\partial\lambda} = \frac{B}{a}\frac{dn}{d\lambda}, \tag{2.2}$$

where θ is the angle at which the refracted light differs from the incident light, λ is the wavelength of the incident light, B is the longest distance the beam would travel through

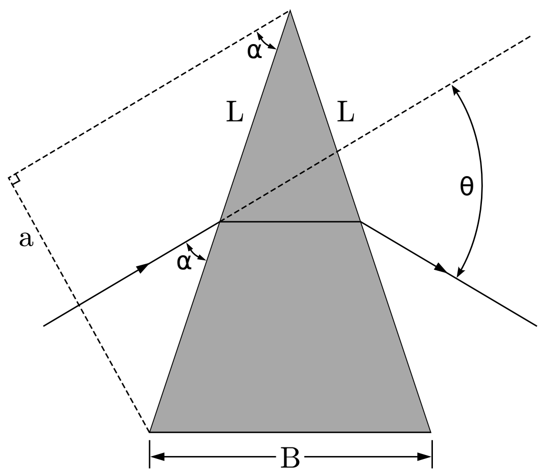

Figure 2.2: Geometry of a prism refracting an incident monochromatic beam at a minimum deviation angle. Diagram adapted from Birney et al. (2006).

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, α, 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:

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

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,

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

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, λ, 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.[1]

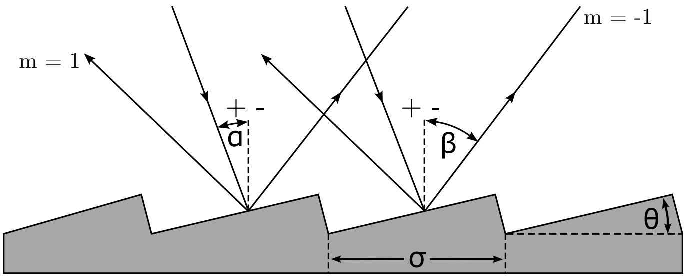

Figure 2.3: Geometry of a reflective blazed grating refracting an incident monochromatic beam. Diagram adapted from Birney et al. (2006).

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 $m \in Z$, 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 α from the grating normal. Due to the interference, a diffracted beam of wavelength λ is found at an angle of β from the grating normal. The relation between the incident and diffracted beams is given by the grating equation:

$$m\lambda = \sigma(\sin(\alpha) \pm \sin(\beta))\,, \tag{2.5}$$

where σ 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 α and β.

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 θ, 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, λ_b, which is determined by:

$$m\lambda_b = 2\sigma\sin(\theta)\cos(\alpha - \theta)\,, \tag{2.6}$$

where

$$2\theta = \alpha + \beta. \tag{2.7}$$

[1]See manufacturers technical specifications, THORLABS, or Edmund Optics for example.

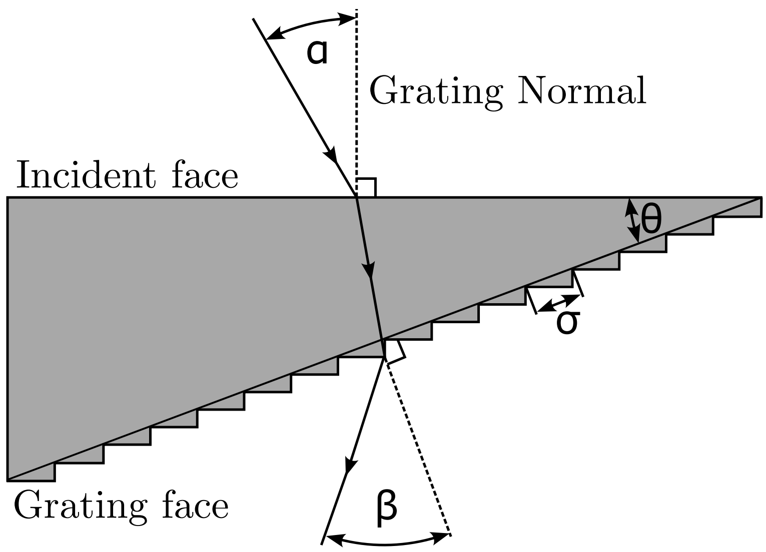

Figure 2.4: Diagram of a grism for an incident monochromatic beam of light and a diffracted beam of order $m = 1$. Diagram adapted from Birney et al. (2006).

Taking the derivative of Equation 2.5 with respect to λ while keeping α constant, allows us to determine the angular dispersion of a diffraction grating,

$$\frac{\partial \beta}{\partial \lambda} = \frac{m}{\sigma \cos(\beta)} \, . \tag{2.8}$$

Substituting m/σ with the grating equation results in

$$\frac{\partial \beta}{\partial \lambda} = \frac{\sin(\alpha) + \sin(\beta)}{\lambda \cos(\beta)} \propto \lambda^{-1} \, . \tag{2.9}$$

Similar to the dispersion of a prism, Equation 2.9 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

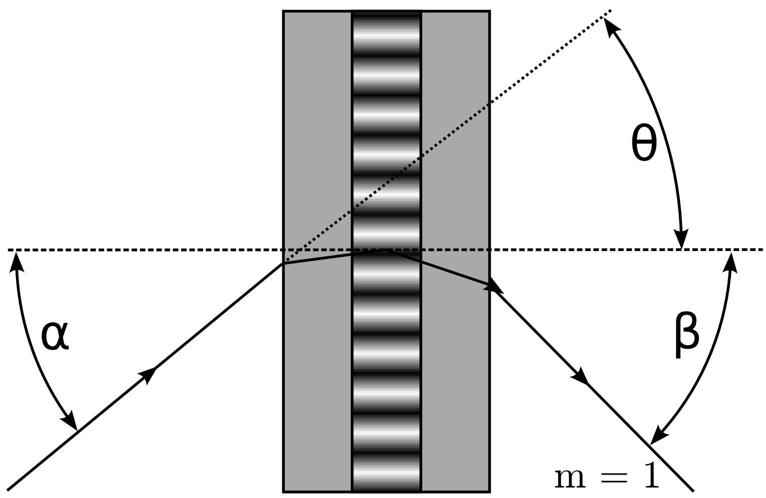

Figure 2.5: Diagram of a VPH grating for an incident monochromatic beam of light. Diagram adapted from Birney et al. (2006).

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 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

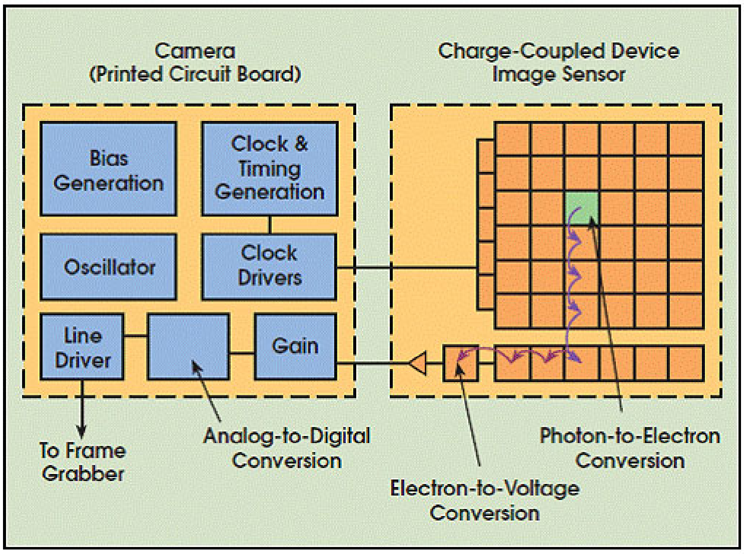

Figure 2.6: Diagram of the inner logic of a CCD. Figure adapted from Litwiller (2001).

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 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

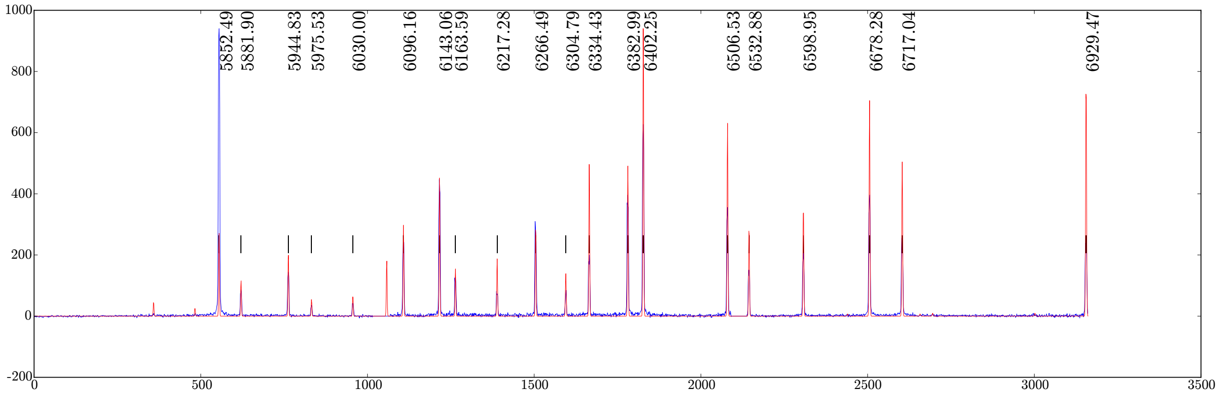

Figure 2.7: Example of an arc spectrum for NeAr taken with SALT's RSS using the PG1800 grating at a grating angle of 34.625°, an articulation angle of 69.258°, and covering a wavelength range of $\sim 5600 - 6900$ Å. Plot adapted from SALT's published Longslit Line Atlases, (2023).[2]

produced from images of the twilight (or dawn) sky. They are taken when the Sun has just set, in the opposite direction, at $\sim 20°$ from zenith and provide a better S/Ns at the cost of careful timing of the images.

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, $F^n_\lambda(x,y)$, can be calculated as:

$$F^n_\lambda(x,y) = \frac{F_\lambda(x,y) - B(x,y) - (\frac{t_S}{t_D})D(x,y)}{\mathrm{med}_{lp}(F_\lambda(x,y) - B(x,y) - (\frac{t_S}{t_D})D(x,y))}\,, \tag{2.10}$$

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. 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:

$$S^*_\lambda(x,y) = \frac{S_\lambda(x,y) - B(x,y) - (\frac{t_S}{t_D})D(x,y)}{F^n_\lambda(x,y)}\,. \tag{2.11}$$

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.

[2]NeAr plot sourced from `https://astronomers.salt.ac.za/data/salt-longslit-line-atlas/`

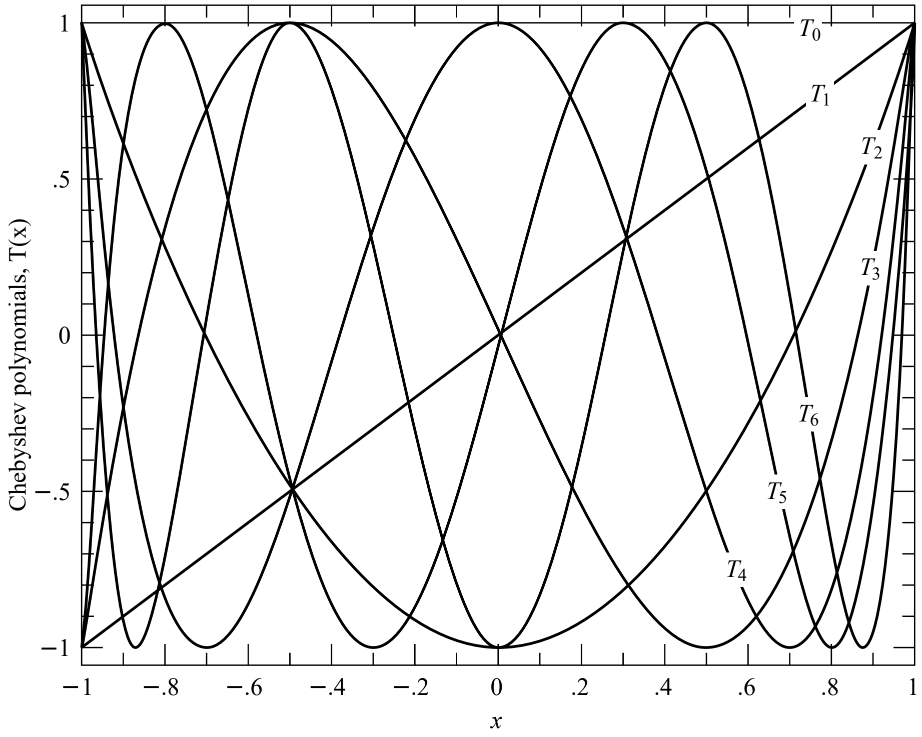

Figure 2.8: The first seven Chebyshev polynomials (T_0 through T_6) as defined by Equation 2.13 over the region $[-1, 1]$ for which they are orthogonal. Plot adapted from (Press et al., 2007) (2023)[3]

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 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:

$$T_n(x) = \cos(n \cos^{-1}(x)) \,, \tag{2.12}$$

[3]Excellent resources on Chebyshev and Legendre polynomials are available digitally at www.numerical.recipes/book.

or recursively as:

$$\begin{aligned} T_0(x) &= 1\,, \\ T_1(x) &= x\,, \quad \text{and} \\ T_n(x) &= 2xT_{n-1}(x) - T_{n-2}(x)\,, \quad \text{for } n > 1\,, \end{aligned} \tag{2.13}$$

where T is a Chebyshev polynomial of order n.[4] 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:

$$\int_{-1}^{1} T_i(x)T_j(x)\frac{1}{\sqrt{1-x^2}}\,dx = \begin{cases} 0, & i \neq j \\ \pi/2, & i = j \neq 0 \\ \pi, & i = j = 0 \end{cases} \tag{2.14}$$

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:

$$f(x) \approx \sum_{i=0}^{N} c_i T_i(u)\,, \tag{2.15}$$

or

$$F(x, y) \approx \sum_{i=0}^{N}\sum_{j=0}^{M} c_{ij} T_i(u) T_j(v)\,, \tag{2.16}$$

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:

$$(u, v) = \frac{2(x, y) - a - b}{b - a}\,. \tag{2.17}$$

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:

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2 - 1)^n\,, \tag{2.18}$$

[4] Chebyshev polynomials are denoted T as a hold-over from the alternate spelling of 'Tchebycheff'.

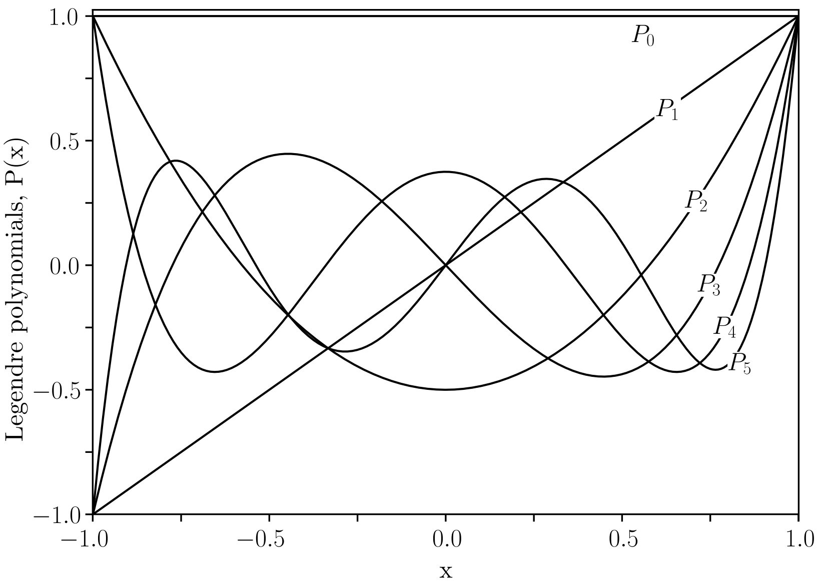

Figure 2.9: The first six Legendre polynomials (P_0 through P_5) as defined by Equation 2.21 over the region $[-1, 1]$ for which they are orthogonal. Plot adapted from Geek3, CC BY-SA 3.0, via Wikimedia Commons (2023).

or recursively as:

$$\begin{aligned} P_0(x) &= 1\,, \\ P_1(x) &= x\,, \quad \text{and} \\ P_n(x) &= \frac{2n+1}{n+1} x P_{n-1}(x) - \frac{n}{n+1} P_{n-2}(x)\,, \quad \text{for } n > 1\,, \end{aligned} \tag{2.19}$$

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:

$$\int_{-1}^{1} P_i(x) P_j(x)\, dx = \begin{cases} 0, & i \neq j \\ \frac{2}{2n+1}, & i = j \end{cases} \tag{2.20}$$

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:

$$f(x) \approx \sum_{n=0}^{N} a_n P_n(u)\,, \tag{2.21}$$

or

$$F(x, y) \approx \sum_{i=0}^{N} \sum_{j=0}^{M} a_{ij} P_i(u) P_j(v)\,, \tag{2.22}$$

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 2.17, 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 minimized function may then be used to convert the science frames from an (x-pixel, y-pixel) coordinate system to a (λ, 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:

$$\begin{aligned} R_s &= \left| \frac{Z_2 \cos\theta_i - Z_1 \cos\theta_t}{Z_2 \cos\theta_i + Z_1 \cos\theta_t} \right|^2 , \\ R_p &= \left| \frac{Z_2 \cos\theta_t - Z_1 \cos\theta_i}{Z_2 \cos\theta_t + Z_1 \cos\theta_i} \right|^2 , \\ T_s &= 1 - R_s\,, \quad \text{and} \\ T_p &= 1 - R_p, \end{aligned} \tag{2.23}$$

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 θ_i, θ_t, and θ_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

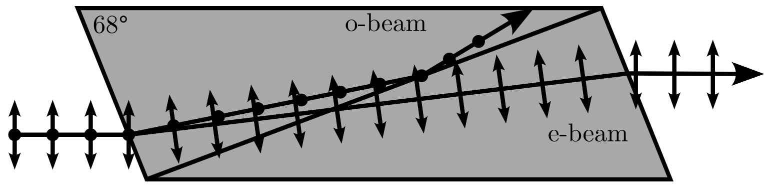

Figure 2.10: Diagram of a Nicol prism for incident non-polarized light. Diagram adapted from Fred the Oyster, CC BY-SA 4.0, via Wikimedia Commons (2023).

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:

$$\begin{aligned} \nabla \cdot \mathbf{E} &= 0\,, \\ \nabla \cdot \mathbf{B} &= 0\,, \\ \nabla \times \mathbf{E} &= -\frac{1}{c}\frac{\partial \mathbf{B}}{\partial t}\,, \quad \text{and} \\ \nabla \times \mathbf{B} &= \frac{1}{c}\frac{\partial \mathbf{E}}{\partial t}\,, \end{aligned} \tag{2.24}$$

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:

$$\begin{aligned} \mathbf{E} &= E_x \cos(kz - \omega t + \Phi_x)\hat{\boldsymbol{x}} + E_y \cos(kz - \omega t + \Phi_y)\hat{\boldsymbol{y}}\,, \quad \text{and} \\ \mathbf{B} &= \frac{1}{c}E_y \cos(kz - \omega t + \Phi_y)\hat{\boldsymbol{x}} + \frac{1}{c}E_x \cos(kz - \omega t + \Phi_x)\hat{\boldsymbol{y}}\,, \end{aligned} \tag{2.25}$$

where E_x, E_y, Φ_x, and Φ_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 2.25 using complex values allows us to simplify the form of the solution to:

$$\mathbf{E} = \Re(\mathbf{E}_0 e^{-i\omega t})\,, \tag{2.26}$$

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

$$\mathbf{E}_0 = E_x e^{i\Phi_x}\hat{\boldsymbol{x}} + E_y e^{i\Phi_y}\hat{\boldsymbol{y}}\,, \tag{2.27}$$

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

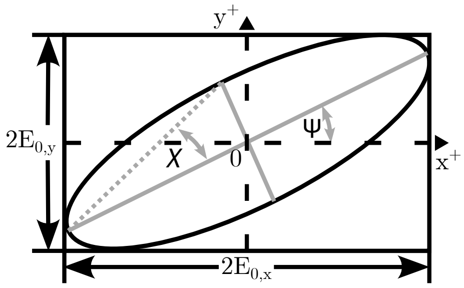

Figure 2.11: The polarization ellipse for an electric field vector propagating through free space. Diagram adapted from Inductiveload, PDM 1.0, via Wikimedia Commons (2023).

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:

$$\left(\frac{\mathbf{E}_x}{\mathbf{E}_{0,x}}\right)^2 + \left(\frac{\mathbf{E}_y}{\mathbf{E}_{0,y}}\right)^2 - \frac{2\mathbf{E}_x\mathbf{E}_y}{\mathbf{E}_{0,x}\mathbf{E}_{0,y}}\cos\Phi = \sin^2\Phi\,, \tag{2.28}$$

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 $\mathbf{E}_{0,x}$, $\mathbf{E}_{0,y}$, Φ_x, and Φ_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 2.28 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:

$$\langle\mathbf{E}_i\mathbf{E}_j\rangle = \lim_{dt\to\infty}\frac{1}{T}\int_0^T \mathbf{E}_i\mathbf{E}_j\,dt\,, \quad \text{for } i,j\in(x,y)\,, \tag{2.29}$$

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 2.28 and simplifying results in:

$$(E_{0x}^2 + E_{0y}^2)^2 - (E_{0x}^2 - E_{0y}^2)^2 - (2E_xE_y\cos\Phi)^2 = (2E_xE_y\sin\Phi)^2\,. \tag{2.30}$$

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

$$\mathbf{S} = \begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{pmatrix} = \begin{pmatrix} I \\ Q \\ U \\ V \end{pmatrix} = \begin{pmatrix} E_{0x}^2 + E_{0y}^2 \\ E_{0x}^2 - E_{0y}^2 \\ 2E_{0x}E_{0y}\cos\Phi \\ 2E_{0x}E_{0y}\sin\Phi \end{pmatrix} \tag{2.31}$$

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° Polarized (L+45°) to Linear −45° Polarized (L−45°) 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).

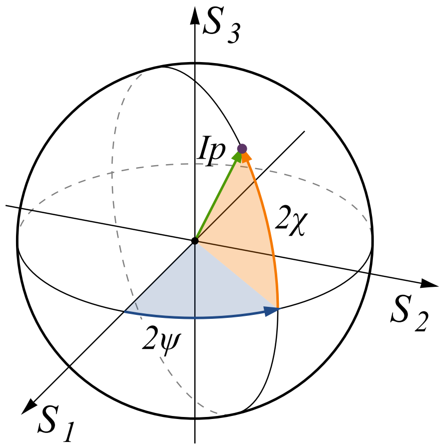

Figure 2.12: The Poincaré sphere describing the polarization properties of a wave-packet propagating through free space. Diagram adapted from Inductiveload, PDM 1.0, via Wikimedia Commons (2023).

From Equations 2.30, 2.31, the polarization parameters are related by:

$$I^2 = Q^2 + U^2 + V^2 \,, \tag{2.32}$$

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 for partially polarized light such that:

$$I^2 \geq Q^2 + U^2 + V^2 \,, \tag{2.33}$$

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

$$\bar{Q} = \frac{Q}{I} \,, \quad \bar{U} = \frac{U}{I} \,, \quad \text{and} \quad \bar{V} = \frac{V}{I} \,. \tag{2.34}$$

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:

$$\begin{aligned} I &= S_0 \,, \\ P &= \frac{\sqrt{S_1^2 + S_2^2 + S_3^2}}{S_0} \,, \quad \text{for } 0 \leq P \leq 1 \,, \\ 2\Psi &= \arctan \frac{S_3}{\sqrt{S_1^2 + S_2^2}} \,, \quad \text{and} \\ 2\chi &= \arctan \frac{S_2}{S_1} \,, \end{aligned} \tag{2.35}$$

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, χ denotes the polarization angle, and Ψ 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

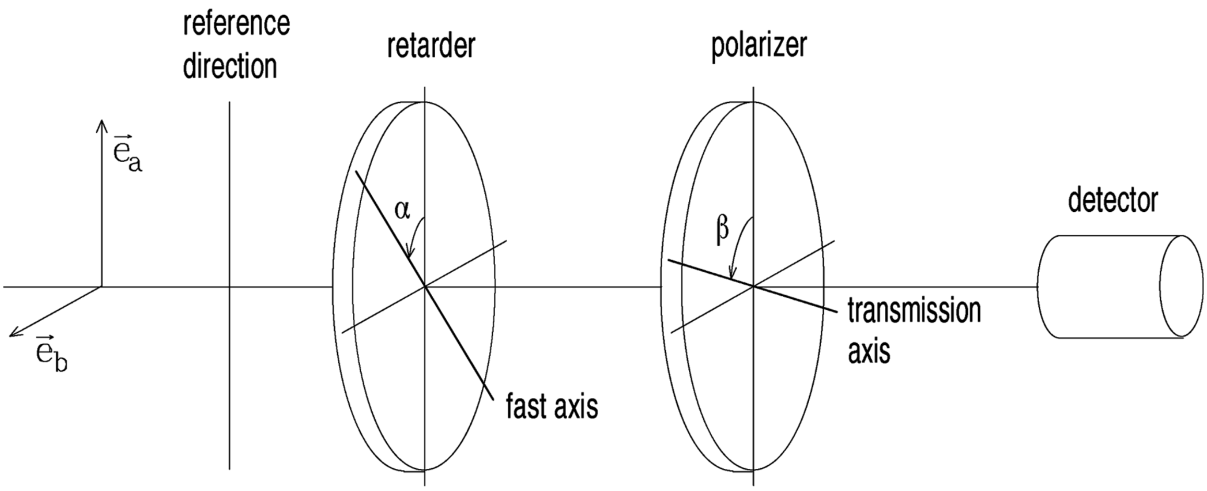

Figure 2.13: A diagram of an ideal polarimeter. Diagram adapted from Degl'Innocenti and Landolfi (2004).

the beam through the manipulation of the four parameters given in Equation 2.25. 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 $\mathbf{M}$ represents the 4×4 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 $(\mathbf{M}_0, \ldots, \mathbf{M}_N)$ results in an outgoing beam $\mathbf{S}'$ such that:

$$\mathbf{S}' = \mathbf{M}_N \ldots \mathbf{M}_0 \mathbf{S}\,. \tag{2.36}$$

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 θ,

$$\mathbf{R}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 2\theta & \sin 2\theta & 0 \\ 0 & -\sin 2\theta & \cos 2\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.37}$$

General linear retardance The Mueller matrix for retardance where α is the angle between the incoming vector and fast axis, and δ is the retardance introduced by the retarder,

$$\mathbf{W}(\alpha, \delta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos^2 2\alpha + \sin^2 2\alpha \cos\delta & \cos 2\alpha \sin 2\alpha (1 - \cos\delta) & \sin 2\alpha \sin\delta \\ 0 & \cos 2\alpha \sin 2\alpha (1 - \cos\delta) & \cos^2 2\alpha \cos\delta + \sin^2 2\alpha & -\cos 2\alpha \sin\delta \\ 0 & -\sin 2\alpha \sin\delta & \cos 2\alpha \sin\delta & \cos\delta \end{bmatrix}. \tag{2.38}$$

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 β is the angle between the incoming vector and transmission axis,

$$\mathbf{P}(\beta) = \frac{1}{2}\begin{bmatrix} 1 & \cos 2\beta & \sin 2\beta & 0 \\ \cos 2\beta & \cos^2 2\beta & \cos 2\beta \sin 2\beta & 0 \\ \sin 2\beta & \sin 2\beta \cos 2\beta & \sin^2 2\beta & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{2.39}$$

These matrices in combination with Equation 2.36 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:

$$\begin{aligned} S'(\alpha, \beta, \gamma) \propto \frac{1}{2}\{ I &+ [Q \cos 2\alpha + U \sin 2\alpha] \cos(2\beta - 2\alpha) \\ &- [Q \sin 2\alpha + U \cos 2\alpha] \sin(2\beta - 2\alpha) \cos \gamma \\ &+ V \sin(2\beta - 2\alpha) \sin \gamma \}\,, \end{aligned} \tag{2.40}$$

where the retandance angle, α, polarization angle, β, for a wave plate with a relative phase difference, γ, 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 2.40 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, α.

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, γ, which is given by:

$$\gamma = \frac{2\pi \Delta n L}{\lambda_0} \tag{2.41}$$

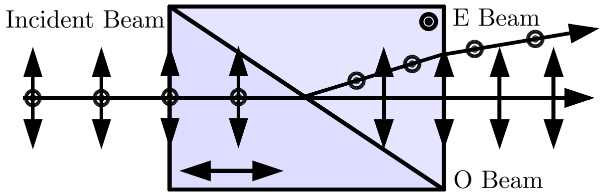

Figure 2.14: Diagram of a Rochon prism. Included in the diagram are the optical axes of the differing sections of the birefringent material as well as polarizing directions of the incident beam, denoted using the $\leftrightarrow$ and $\odot$ symbols, for the O- and E-beams, respectively. Figure adapted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

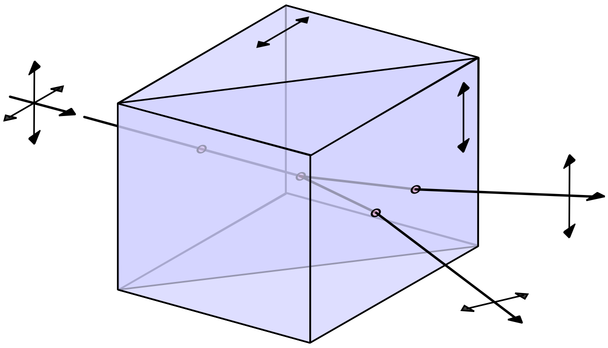

Figure 2.15: Diagram of a Wollaston prism. Included in the diagram are the optical axes of the differing sections of the birefringent material as well as polarizing directions of the incident beam, denoted using the $\leftrightarrow$ and $\updownarrow$ symbols, for the O- and E-beams, respectively. Diagram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

where Δn and L refer to the birefringence and thickness of the wave plate medium, respectively, and λ_0 refers to the vacuum wavelength of the beam (Hecht, 2017).

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 $m \in Z^+$, 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 Romaniello, 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, 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.

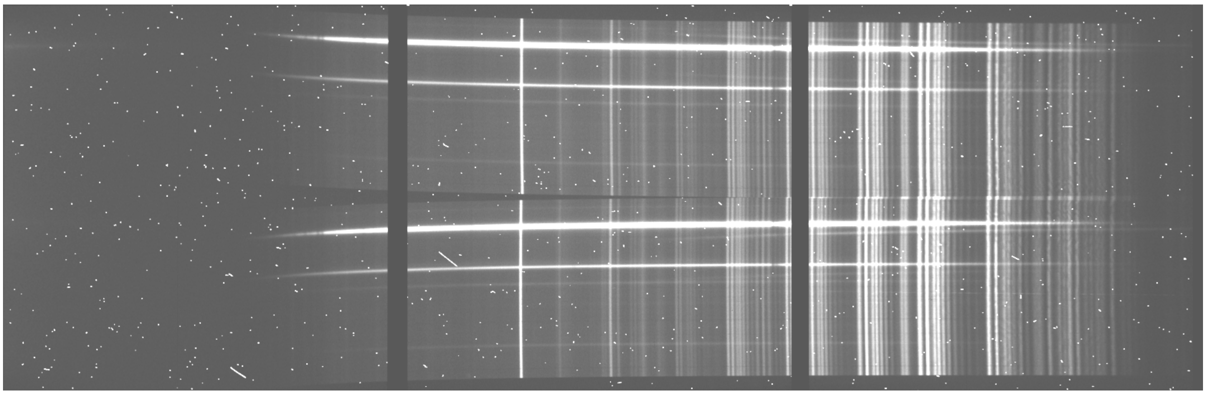

Figure 2.16: A spectropolarimetric target exposure as observed by the SALT RSS in spectropolarimetry mode.

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 2.29), 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 θ_i at angle i. These intensities thus relate to the wavelength dependent Stokes parameters as:

$$\begin{aligned} f_{O,i}(\lambda) &= \frac{1}{2}[I(\lambda) + Q(\lambda)\cos(4\theta_i) + U(\lambda)\sin(4\theta_i)] \,, \quad \text{and} \\ f_{E,i}(\lambda) &= \frac{1}{2}[I(\lambda) - Q(\lambda)\cos(4\theta_i) - U(\lambda)\sin(4\theta_i)] \,. \end{aligned} \tag{2.42}$$

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

$$I_i(\lambda) = f_{O,i}(\lambda) + f_{E,i}(\lambda) \,. \tag{2.43}$$

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:

$$F_i(\lambda) \equiv \frac{f_{O,i}(\lambda) - f_{E,i}(\lambda)}{f_{O,i}(\lambda) + f_{E,i}(\lambda)}\,, \tag{2.44}$$

which allows Equation 2.42 to be written, as

$$F_i(\lambda) = \bar{Q}(\lambda)\cos(4\theta_i) + \bar{U}(\lambda)\sin(4\theta_i) = P\cos(4\theta_i - 2\chi)\,, \tag{2.45}$$

in terms of the normalized Stokes parameters, or, alternatively, the degree of polarization, P, and polarization angle, χ (as described in Equations 2.34, and 2.35).

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:

$$\begin{aligned} \bar{Q}(\lambda) &= \frac{2}{N}\sum_{i=0}^{N-1} F_i(\lambda)\cos\left(\frac{\pi}{2}i\right)\,, \quad \text{and} \\ \bar{U}(\lambda) &= \frac{2}{N}\sum_{i=0}^{N-1} F_i(\lambda)\sin\left(\frac{\pi}{2}i\right)\,, \end{aligned} \tag{2.46}$$

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

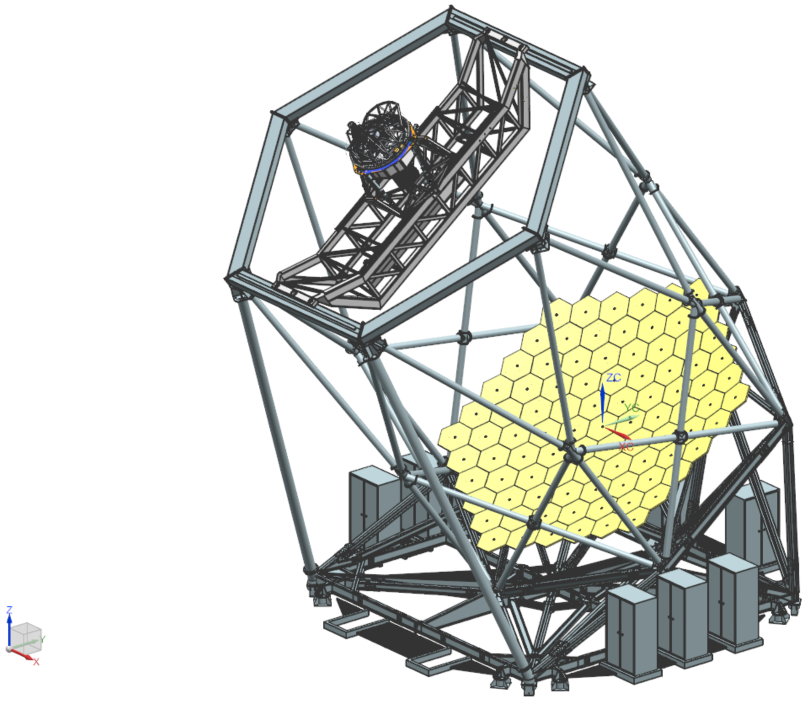

Figure 2.17: The tracker, supporting structure, and primary mirror of SALT. Figure adapted from the SALT call for proposals (2022).[5]

the pointing of the telescope's primary mirror to a fixed elevation (37° 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 μm and is capable of tracking $\pm 6°$ from the optimal central track position. Targets at declinations from 10.5° to -75.3°, 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

[5] http://pysalt.salt.ac.za/proposal_calls/current/ProposalCall.html

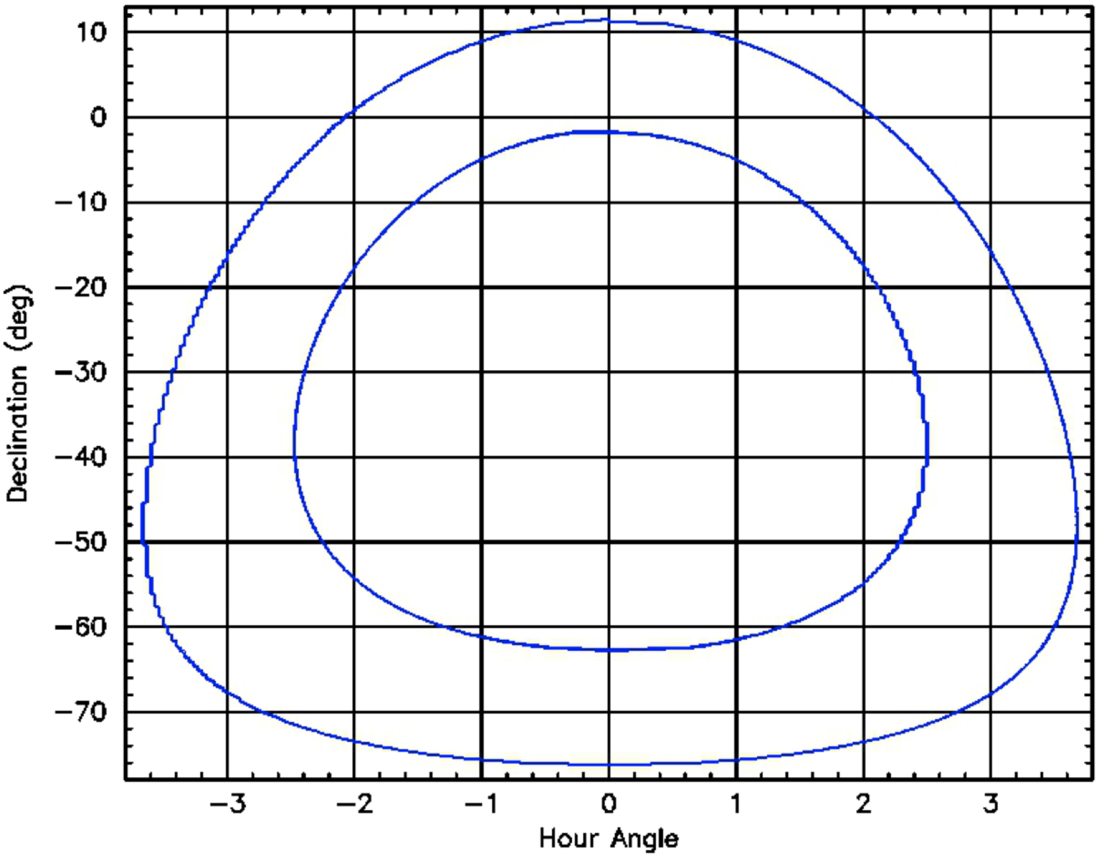

Figure 2.18: The visibility annulus of objects observable by SALT. Figure adapted from the SALT call for proposals (2013).[6]

of ~ 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.

[6] `https://pysalt.salt.ac.za/proposal_calls/2013-2/`

[7] `https://pysalt.salt.ac.za/proposal_calls/current/ProposalCall.html`

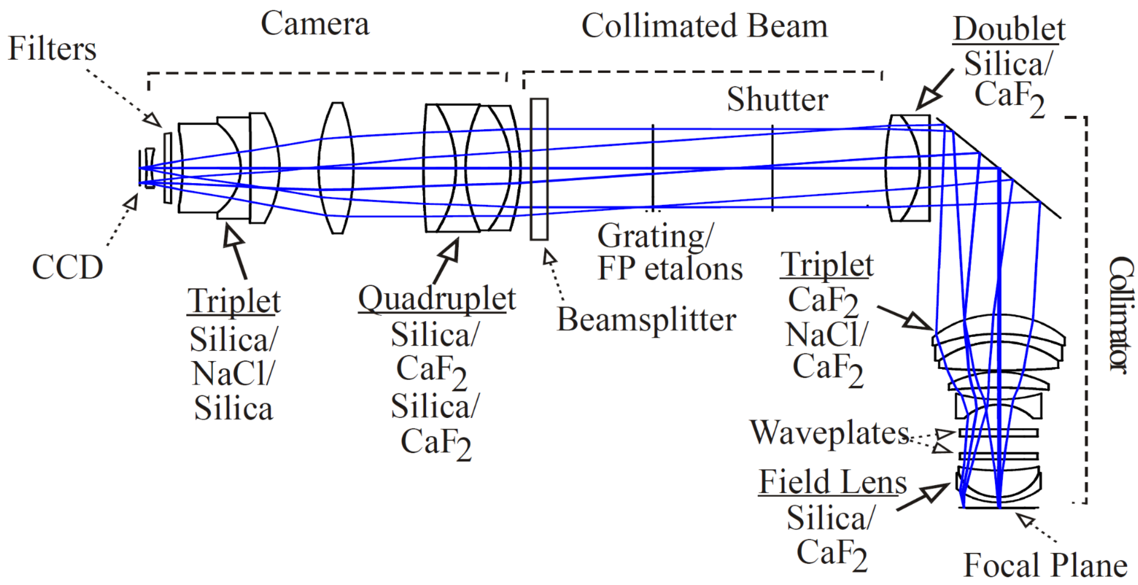

Figure 2.19: The optical path of the SALT RSS. Figure adapted from the SALT call for proposals (2023).[7]

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[8]	3700 – 9000		3900/4400	250 – 600
PG0700[8]	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.[9]

[8] 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.

[9] Commission status sighted from the latest 'Polarimetry Observers Guide' (2024).

Linear (°)		Linear-Hi (°)		Circular (°)		Circular-Hi (°)		All Stokes (°)	
$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$
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 POLSALT, IRAF, and STOPS

This chapter contains an overview of POLSALT (§ 3.1) and the limitations faced during POLSALT 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 POLSALT 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 POLSALT - Polarimetric reductions for SALT

POLSALT is the current reduction package being constantly developed and used within the SAAO/SALT research group as the official reduction pipeline for spectropolarimetric data taken using the SALT RSS.[1] Newer versions of the software, aptly named the 'beta version' (last updated 23 January 2020), include a GUI as well as limited interactivity during key steps in the reduction process and was the version adapted in this study.[2]

The steps that make up the POLSALT 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 (a detailed documentation for the reduction steps and purpose may be found at the GitHub wiki for POLSALT).[3]

[1]POLSALT is made freely available via the POLSALT GitHub repository, available at `https://github.com/saltastro/polsalt`. It is strongly advised to follow the wiki for installation instructions.

[2]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.

[3]The GitHub wiki for POLSALT is available at `https://github.com/saltastro/polsalt/wiki`.

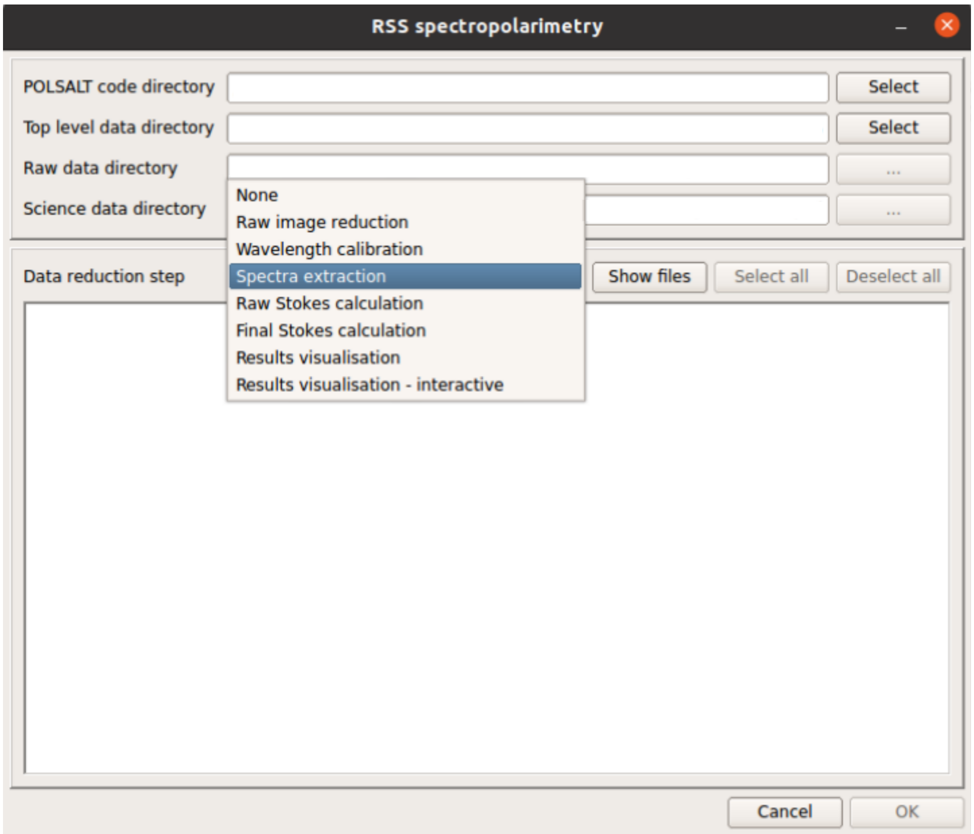

Figure 3.1: The layout of the POLSALT Graphical User Interface (GUI)

3.1.1 Basic CCD reductions

Basic CCD reductions are run via **imred.py** and apply basic reductions to the raw data necessary before any calibrations may be done. These corrections 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, the reduction step is automatically run for all RSS spectropolarimetric observations as part of the default SALT basic reduction pipeline that is run each day.

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.

3.1.3 Spectral extraction

Background subtraction and spectral extraction is run via **specpolextract_dev.py** which corrects for the beam-splitter distortion and tilt, performs sky subtraction, and

TODO: Include POLSALT spectra extraction GUI

Figure 3.2: The layout of the interactive POLSALT `spectra extraction` GUI

TODO: Include POLSALT visualisation GUI

Figure 3.3: The layout of the interactive POLSALT `visualisation` GUI

extracts a one dimensional wavelength dependent spectrum for each beam 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:

$$\begin{aligned} I &= \frac{1}{2}(O_1 + O_2 + E_1 + E_2)\,, \quad \text{and} \\ S &= \frac{1}{2}\left[\left(\frac{O_1 - O_2}{O_1 + O_2}\right) - \left(\frac{E_1 - E_2}{E_1 + E_2}\right)\right]\,, \end{aligned} \tag{3.1}$$

respectively. 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 used being included.

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 polarimetric zero-point and waveplate efficiency calibrations and calculates the final Stokes parameters. Before the final Stokes calculations are performed, data culling is applied to the raw Stokes to eliminate outlier results which may arise due to, for example, atmospheric conditions. Data culling compares observation cycles against one another, compares the deviation of the means which estimate the systematic polarization baseline fluctuations (due to imperfections in repeatability), and performs a chi-squared 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 (°) 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

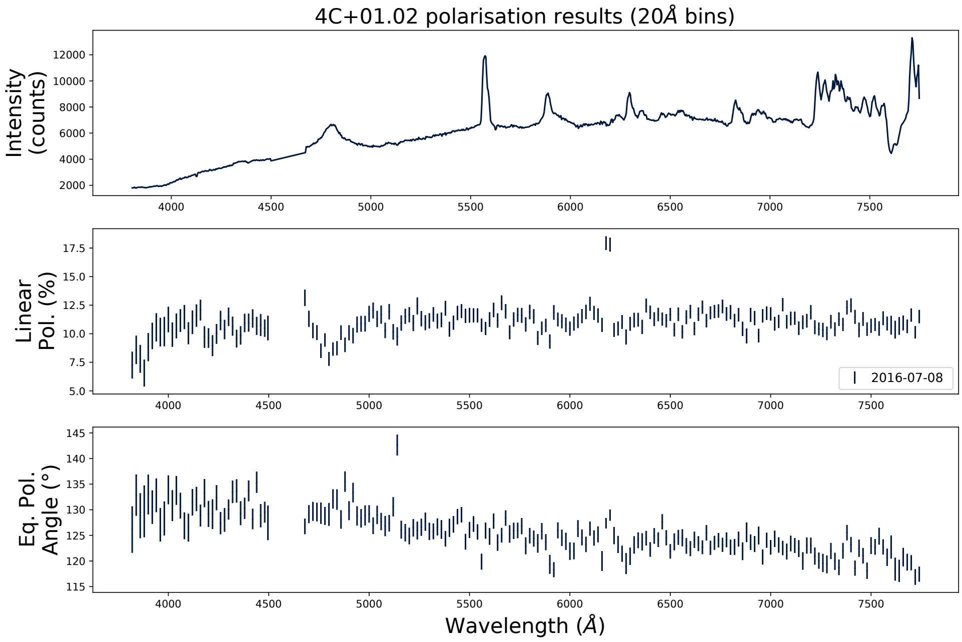

Figure 3.4: A typical plot resulting from the reduction process. Figure adapted from (Cooper et al., 2022)

exist which may be used after the polarization reductions have been completed, 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, the flux database file must exist for the standard observed and must be copied over to the working science directory.

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.

3.1.8 Limitations of POLSALT and the Need for Supplementary Tools

The creation of supplementary tools for POLSALT spectropolarimetric reductions stemmed, primarily, from the limitations of the wavelength calibration process and a need for a way to compare wavelength solutions across matching 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.

The prime motivation of finding an alternate method to wavelength calibrate SALT spectropolarimetric data stemmed from a large backlog of unused data taken using the PG0300. The only arc available for the PG0300 with a close enough articulation and grating angle ($\sim 10.68°$ and $\sim 5.38°$, respectively) was SALT's Argon lamp which dis-

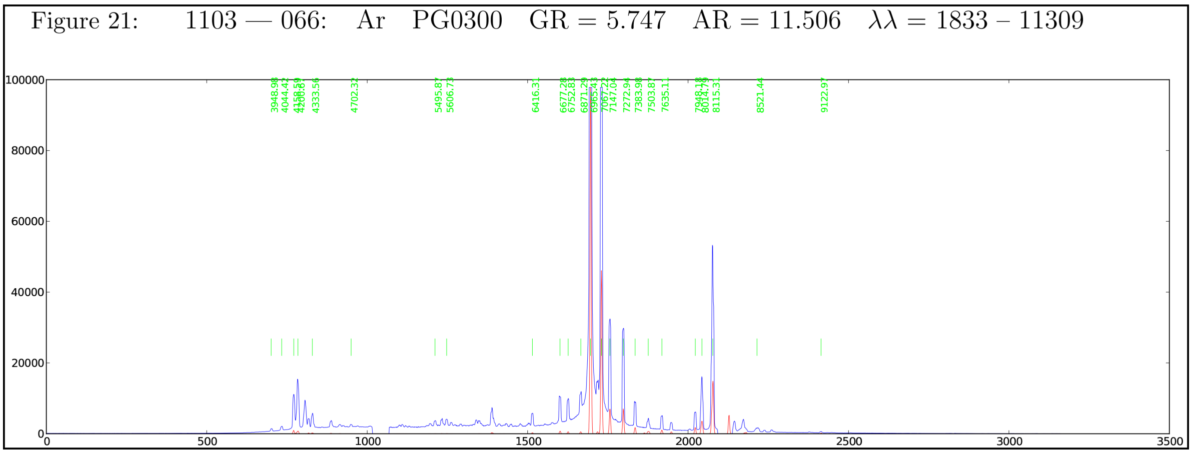

Figure 3.5: One of many Argon arc lamp spectra as provided by SALT for line identification. Plot adapted from SALT's published Longslit Line Atlases (as of 2024), resized to fit within the document margins but otherwise unchanged.[4]

played sparse spectral features with large gaps over the wavelength range at these grating and articulation angles (Figure 3.5). This often lead the POLSALT pipeline to create inconsistent wavelength solutions, or 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 and is thus not a purely free spectral range (§ 2.1.7, Eq. 2.5).

The chosen solution was to use a well established tool to perform the wavelength calibration - one which allows for rapid recalibrations as well as provides a familiar interface with which the user can analyze their wavelength solutions. IRAF provides this familiar environment and reliability, even when considering its age and limited community development.

Unfortunately, IRAF is unable to natively parse the file structure implemented by POLSALT 'as is' and formatting of the data structures are necessary for integration purposes. This restructuring works both ways as once the IRAF reductions are complete the format must be reformatted 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

IRAF is a collection of software designed specifically for the reduction and analysis of astronomical images and spectra. 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 as every researcher, university, and research group have their own preferred wavelength calibration procedures and often use specific parameters for the various IRAF tasks (e.g. the order and type of the polynomial used in `identify`, etc.).

[4]'low resolution' Ar plot sourced from `https://astronomers.salt.ac.za/data/salt-longslit-line-atlas/`

A useful IRAF task that will not be discussed but nevertheless deserves a mention is 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. It is instrumental as a scripting aid and is what allows IRAF its rapid recalibrations of the wavelength solutions.

For wavelength calibrations of spectropolarimetric observations 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 and must all be run twice to find the wavelength solutions for each of the two spectropolarimetric beams.

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. `identify` gives the first approximation of the wavelength solution, which is saved to a local database, and is built on in subsequent tasks. It is thus imperative that the initial fit is done well to minimize errors further down 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.

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. `reidentify` 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 defined conditions, most common of which is the loss of features as the task moves further from the row at which the user ran `identify`.

When running `reidentify` non-interactively, it is recommended to set the `verbose` parameter to '`yes`' as this will provide immediate confirmation of whether the task quit early or not. Regardless of where the task ended, the newly defined wavelength solutions are appended to the local database.

3.2.3 Fitcoords

The `fitcoords` task is used to combine the collection of one-dimensional wavelength solutions in the local database to a two-dimensional surface function. 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.

TODO: Include poor and good transformed image examples

Figure 3.6: Examples of a terrible-, poorly-, and well-fit wavelength solution as presented by the IRAF `transform` task.

The process of using `fitcoords`, follows closely 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 type and degree, the overall error of the fit decreases, matching more closely to what the 'true' wavelength solution is.

3.2.4 Transform

The `transform` task is an optional step in the IRAF wavelength calibration process but is good to perform since it is quick to run and easy to script. `transform` converts the (*pixel*, *pixel*) units stored in the exposure to (*wavelength*, *pixel*) units which allows for an immediate check of whether the wavelength solution was found correctly. Any error in the wavelength solution will be easily spotted in the transformed images and may range from minor, 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 severe, such as the wavelength solution completely readjusting the frame to an incoherent mess.

3.3 STOPS - Supplementary Tools for POLSALT Spectropolarimetry

STOPS allows an alternate method for wavelength calibrations, namely IRAF, to be used instead of the POLSALT `wavelength calibrations` method. 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* respectively, is performed by STOPS. In order to implement the alternate wavelength calibrations, methods to check the wavelength solution were implemented to verify the sky line positions across the frame and correlate the O- and E-beams, named `skyline` and `correlate`, respectively.

Before creating the supplementary tool's `split` and `join` methods used to perform wavelength calibrations in IRAF, it was deemed necessary to create a tool to allow for the comparison of the wavelength solutions between the extracted spectra of the O and E beams, referred to as `correlate`. The scope was later expanded to allow for the inspection of the cross-row and cross-column axes of the wavelength solutions as the IRAF wavelength calibration procedure provided much more flexibility.

3.3.1 Splitting

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 3.13), the variance of the image, and a map of the pixels to be masked out, respectively.

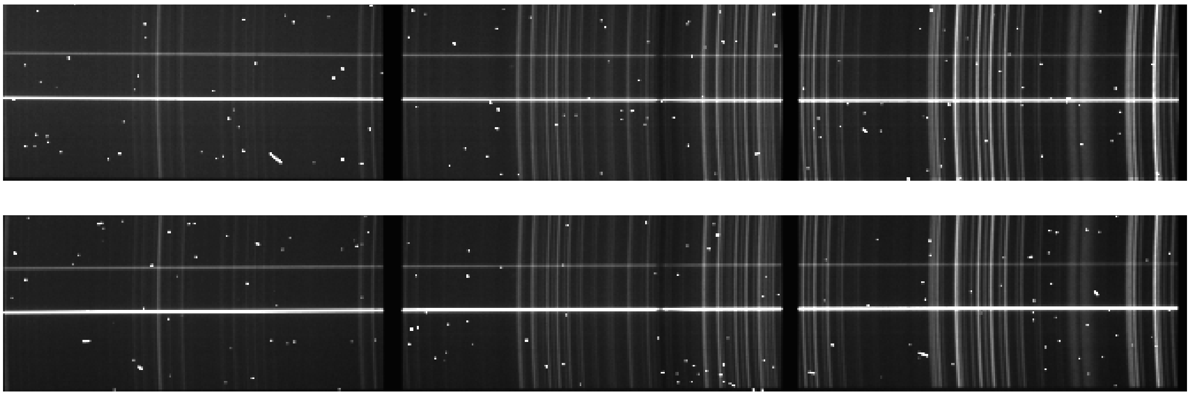

Figure 3.7: The split *O*- and *E*-beams as handed to IRAF.

IRAF is capable of dealing with multiple traces in an extension or lists of input files but is not as proficient 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.

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 light as possible. This is especially necessary when considering the amount of exposures that are taken for a single spectropolarimetric observation run, and then how the number of observations increases for long term studies.

To aid the IRAF wavelength calibrations, row cropping and file list creation were introduced into the `split` method to ignore the regions without a trace either side of the frame, and to list the *O*- and *E*-beam FITS files, respectively. The row cropping was decided on as IRAF does not handle the empty rows well, specifically when it comes to the `reidentify` task. 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

As mentioned previously, the format of the FITS file created by IRAF after wavelength calibrations and that expected by POLSALT for the `spectra extraction` are incompatible. A typical FITS file expected by the POLSALT `spectra extraction` contains a primary header along with the various image extensions, the most notable extension being the newly added wavelength extension. All images contained within the extensions have the trace for both polarimetry beams split, as seen in Figure 3.9 and the headers of each extension updated.

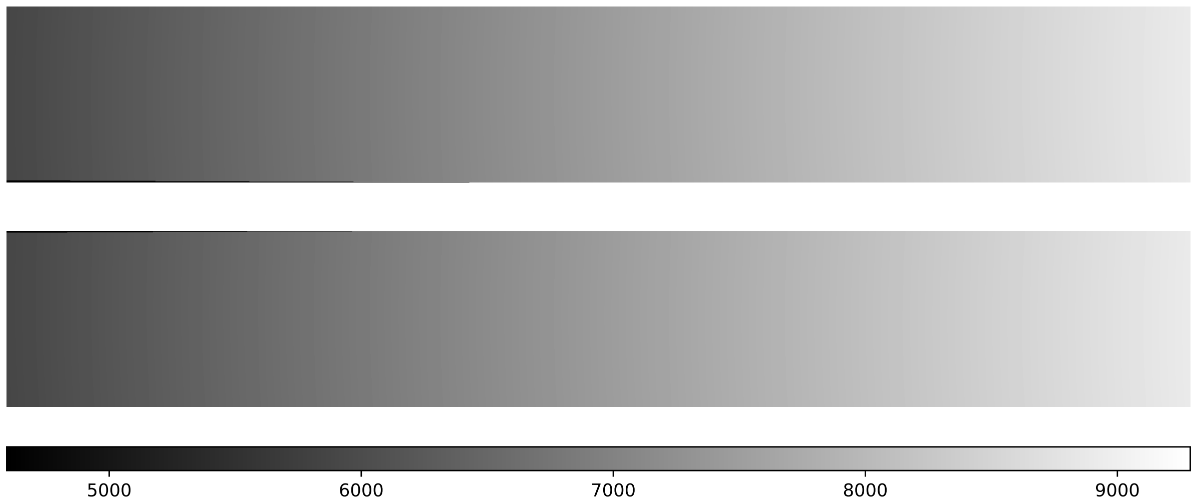

Figure 3.8: The wavelength extension of a FITS file ready to be handed back to the POLSALT pipeline.

All pieces necessary to recreate the POLSALT wavelength calibrated FITS files exist once the IRAF procedure to generate the database entry for the two-dimensional wavelength solution is complete. The `join` method of the supplementary tools is used at this point and, once run, automatically creates the desired files.

Running `join` finds all the relevant FITS and local database files necessary to run the POLSALT `spectra extraction`, creates an empty HDU structure for each pair of matching spectropolarimetric beams, copies over the extensions and their respective image and header information, checks and corrects the trace splitting to best match that of POLSALT, appends a new extension and parses the database wavelength solutions into the POLSALT intensity-wavelength format, cleans the science extension for cosmic rays, and does some house-cleaning to align the finalized FITS files to those created when using the 'pure' POLSALT pipeline.

The FITS files created by the `join` method and POLSALT pipeline's `wavelength calibration` methods are almost identical. The only difference between the FITS files is the shape of the images stored within them, reflected also through specifically the 'NAXIS2' header keyword, since `split` introduces a cropping. It was deemed unnecessary to reintroduce the cropped region as it is promptly discarded in the following POLSALT `spectra extraction` process and raises no issues when left out. Otherwise, both the `join` method and POLSALT `wavelength calibration` update the headers to reflect the new shape of the data and data type, through header keywords 'CTYPE3' and 'BITPIX', respectively.

The wavelength extension is created entirely by `join` by appending a blank extension to the HDU and filling the image pixels with their respective wavelength value. This is done entirely by `join` which parses the wavelength database file and creates a function which provides the corresponding wavelength when provided with a ($pixel$, $pixel$) position. This is used to fill the pixels of the wavelength extension with their respective wavelength, as seen in Figure 3.8. Note that regions that fall outside the trace are masked by setting the wavelength extensions corresponding pixel value to 0.

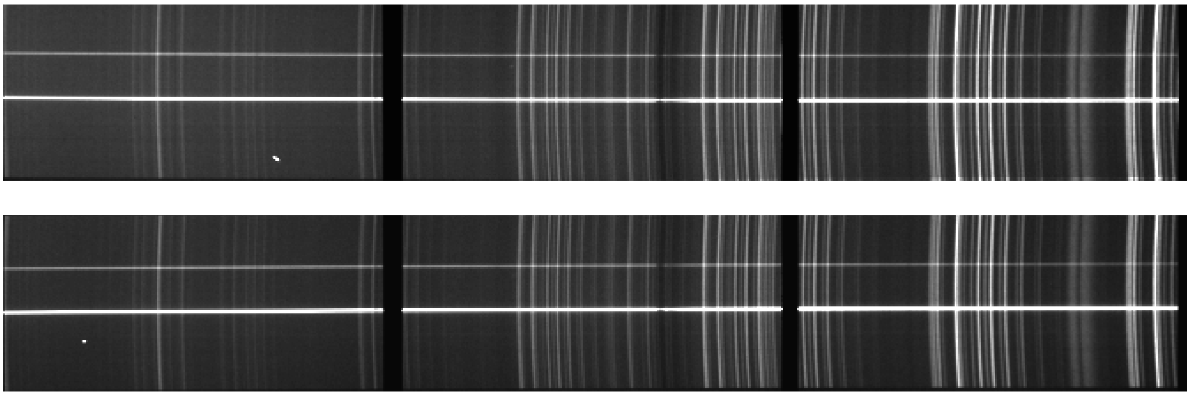

Figure 3.9: The science extension of a FITS file ready to be handed back to the POLSALT pipeline.

TODO: Include example of skyline result

Figure 3.10: The resultant output plot of the STOPS `skylines` method.

`join` cleans the science extension of cosmic rays using the `lacosmic` python package which was specifically designed for this purpose and uses the L.A. Cosmic algorithm, based on Laplacian edge detection. The parameters used for cosmic ray cleaning were chosen based on the properties of the RSS, specifically the read noise and gain, as well as a publication and suggestions by the algorithm's creator (van Dokkum, 2001). The chosen parameters work well for all but the worst of cosmic rays, as can be seen when comparing Figures 3.7 and 3.9.

The wavelength extension is masked to remove any impossible wavelengths and also corrected for the skewing of the trace introduced by the wollaston element. The skewing must be added to the wavelength extension since POLSALT introduces a wollaston correction in the `spectra extraction` process. Finally, the BPM extension is masked to reflect the valid wavelength calibrated regions for both spectropolarimetric beams and the files are saved with the POLSALT wavelength calibrated 'wmxgbp' prefix.

3.3.3 Sky line checks

Sky line comparisons serve two unique yet interconnected services. Firstly, they naively transform the wavelength calibrated frames, without conserving flux, allowing the user confirmation of the variation of sky lines across the columns of the frame, and secondly, they compare the wavelength position of the sky lines with the SALT sky lines,[5] allowing confirmation of the wavelength solution at positions across the rows of the frame. The file used for skyline comparisons may be the IRAF `transform` FITS file, which allows for flux conservation through the '`flux`' parameter.

The `skyline` method loads the wavelength calibrated files, transforms the frames (as described above) if the frame was not transformed by IRAF's `transform` method, divides out the continua, compares the cross-column sky lines to those of a single row, and compares the wavelength position of said sky lines to a list of sky lines known by SALT.

[5]The first iteration of a sky line atlas is available at `https://astronomers.salt.ac.za/data/salt-longslit-line-atlas/`

TODO: Include example of correlate result

Figure 3.11: The resultant output plot of the STOPS `correlate` method.

Determining if there is an 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 the sky lines. Any horizontal deviation of the sky lines after transformation reflects a poor fit of the wavelength solution. Any vertical variation may be found through a quick visual inspection of a transformed frame, as mentioned previously, but may be inspected more thoroughly using the `skyline` method. As mentioned, the sky lines are averaged and compared to sky lines of a typical row. A wavelength solution exhibiting a poor fit across the spatial axis will display broader averaged sky lines than that of a relatively good fit.

As no features, other than the trace of sources exposed across a frame, exist that uniformly cover the wavelength (x, horizontal) axis of a typical frame, determining if the horizontal fit of the wavelength solution is more challenging. Thankfully, SALT has published a sky line atlas which we may make use of. By first considering the spatial fit of the wavelength solution, it is ensured that the wavelength positions of all sky lines are well-defined. Comparisons may now be made to the wavelength positions measured by SALT. Minor variations in the comparison of the sky lines are expected, but any uniform trends indicate an underlying poor fit across the wavelength axis of the wavelength solution. A poor horizontal fit is difficult to spot without supplementary tools and may have drastic adverse effect on the final polarization results.

3.3.4 Cross correlation

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 one another. The difficulty arises in comparing the two spectra since variations between the two are expected and are what define the Stokes, and thus final polarization, results. The `correlate` method was created for this express purpose.

The `correlate` method loads the provided FITS files created by the POLSALT `spectra extraction`, removes the continuum and separates the CCD regions. The relevant, separated, CCD regions are then cross correlated and any offset between the spectra may be plotted.

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. Cross correlation of the two spectra for the different, O and E, polarization beams allows for a comparison of the features within the spectra as a function of the wavelength displacement.

Sources under spectropolarimetric observation are often expected to vary over time and as such as the ratio of polarized to unpolarized light varies. The accuracy of correlation may decrease as features with differences in the polarized component of the polar-

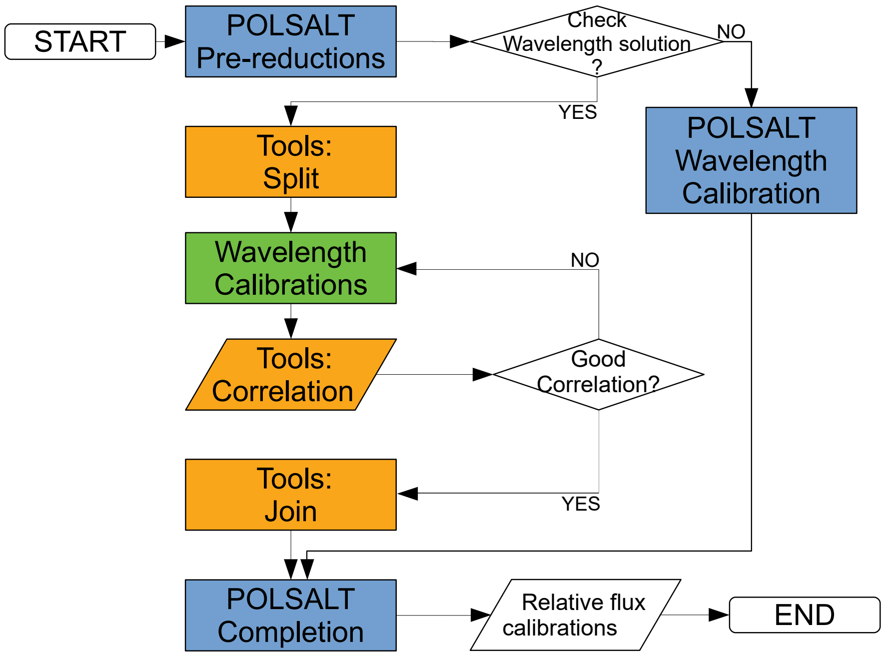

Figure 3.12: A general workflow for data reductions using a combination of POLSALT, IRAF, and the developed supplementary tools.

ization beams change. The differences in the features of the different spectra are often negligible when compared to the overall trend of the spectra and are generally only reflected in a change in the intensity of said features.

Cross correlation is useful when dealing with spectropolarimetric spectra as it allows a comparison of how well aligned the notable features of the spectra are wavelength-wise. Minor deviations between spectra weight the cross correlation less than the more prominent features, and therefore, cross correlation results acquired when using the `correlate` method more accurately reflect any general offset between polarization beams that may not necessarily be found when using the `skyline` method.

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 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.

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 drop-down menu, as seen in Figure 3.1. Reductions are possible, however, purely through the Command Line Interface (CLI) using the POLSALT ‘beta’ scripts.

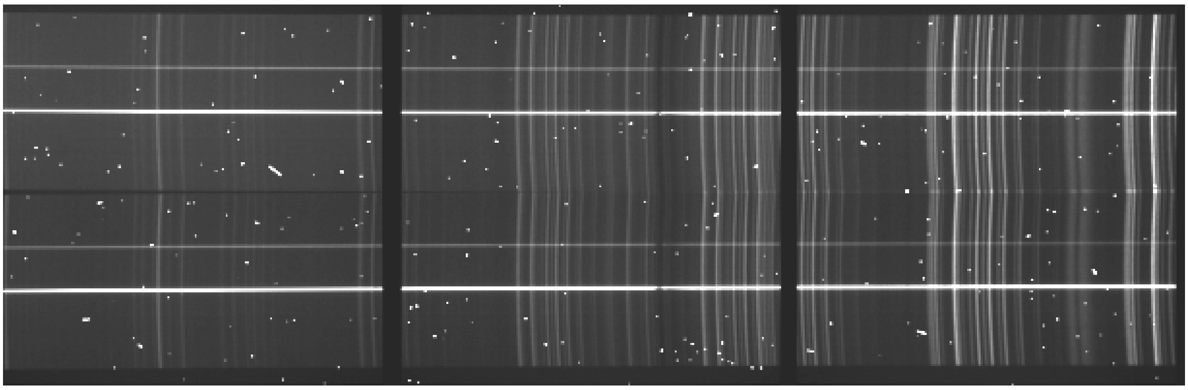

Figure 3.13: The science extension of a typical spectropolarimetric FITS file taken with the SALT RSS, after basic POLSALT CCD reductions have been completed.

Help documentation, primarily describing the possible arguments, is available in the CLI for STOPS using the `-h|-help` flag, invoked as:

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

and for IRAF through the `?|:.help` 'cursor commands' while running an interactive task. Help for POLSALT may be found at the POLSALT wiki.

3.4.1 POLSALT Pre-reductions

The POLSALT reduction process requires a file structure such that the raw data received from SALT is located in a folder labelled using the observing date with a sub-folder labelled raw, such as `YYYYMMDD/raw/`. This directory structure allows POLSALT to create a 'working' directory named `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 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. The 'raw image reduction' may then be selected from the dropdown and 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 attempt to run the entire reduction process. The script may be quit once the POLSALT `wavelength calibration` GUI opens and the rest of the reduction procedure followed.

3.4.2 Wavelength Calibration

The wavelength calibrations may now be completed in IRAF. This section concerns the procedure for parsing the FITS files to be read by 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 Listings I.2 - I.8, but it should be noted that the art of wavelength calibration consists of modifying the parameters to achieve a good calibration function. This process depends heavily and varies greatly based on the user and as such not all use cases can be discussed herein.

Preparing data for IRAF

Splitting the data is presented in Lisiting I.2. The STOPS `split` method may take multiple parameters, as seen in § 3.3, but default parameters should be used where ever 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`'. As an aside, the save prefix may be worth changing as, later in the reduction process, the POLSALT raw Stokes reductions indiscriminately selects files named `YYYYMMDD/sci/e*.fits`.

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:[6]

```
cl> identify images
cl> reidentify reference images
cl> fitcoords images fitname
cl> transform images output fitname
```

where 'images' refer to a list or file containing the FITS files relevant to the task, 'reference' refers to the FITS file previously identified, 'fitname' refers to the name to be used for the final two-dimensional wavelength solution, and 'output' refers to the new file name for the transformed input images.

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 Listings I.4 - 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, but this is not strictly necessary. The scripts are created using the `mkscript` task which interactively asks for a task to script and parameters to use. Multiple tasks may

[6] Please see the IRAF help docs, available at `https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/iraf.html`, on the relevant tasks for a comprehensive discussion of the parameters available.

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 data for POLSALT

The results of the wavelength calibrations may now be parsed back into the format expected by POLSALTJoining the separate beams with their respective wavelength solutions is once again performed in the CLIfollowing Listing I.8.

Similar to the `split` procedure mentioned before, the `join` procedure has the same defaults defined and so the responsibility falls on a user to keep track of which defaults were changed, and to keep the parameters consistent between the two tasks. Note that STOPS has logging implemented, see § 3.3, and so the onus of tracking the parameters may be passed on to a logging file.

Sky line checks of the wavelength solution

The optional IRAF `transform` task and STOPS `skylines` method are used to confirm the wavelength solution across the frame, as described in § 3.3.3, by comparing known and observed sky line wavelength positions.

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.

A final reminder is made here about the clash of default naming schemes and the wildcard file collection performed by POLSALT. A simple wildcard '`mv`' move or '`rm`' remove command may be run in the CLI to deal with the created split files used by IRAF. The remove command may be run using:

```
$ rm obeam* ebeam*
```

while moving the files to a new subfolder may be done following Listing I.10.

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.

3.4.3 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 a 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 GUIs dropdown menu, as explained in Appendix I (pages 57 onwards). As no commands are necessary, save for those to launch the GUI, not much can be said of the reduction process. Excellent resources, created by the SALT / SAAO team, are available online for any queries about the reduction process using any version of POLSALT, including the GUI.[7]

POLSALT beta in a CLI

The reduction script may be run using:

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

which will run the entire reduction process interactively without the need to select which process to run next. For the purposes of using the script alongside IRAF wavelength calibrations, a few changes must be made. The `imred` and `specpolwavmap` function calls before `specpolextract_sc` should be commented out, since the raw images have already been processed and the wavelength calibrations were dealt with using IRAF.

The POLSALT 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 POLSALT 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 POLSALT analysis scripts may be run using:

```
$ python script.py
```

followed by `specpolfinalstokes.py`, `specpolview.py`, `specpolflux.py`, or `specpolfilter.py`, respectively, for the different analysis modes. A description of the use for each mode of the analysis script is available from `https://github.com/saltastro/polsalt/wiki/Linear-Polarization-Reduction.--Beta-version` and is exhaustive enough for general use, with the source code also publically available for in depth queries.

TODO: From appendix → Discuss - salt/py3 env, add polsalt GUI spectra extract and visualisation windows as images

[7]See the official POLSALT wiki or alternative online resources such as SALT workshop slides.

Chapter 4

Testing

TODO: Add all tests done and comparisons.

- **3C 279**
- **4C+01.02**
- **David data (not in next section publications because still during pipeline development. Reductions done through polsalt, but after publication used as preliminary testing data)**

Chapter 5

Science Applications

TODO: short introduction to chapter contents

5.1 Application to Spectropolarimetric Standards

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

- **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 polarisation 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.

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 ‘`cl>`’ or ‘`>>>`’ tokens are used for IRAF’s xgterm single- and multi-line commands.

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 a ‘$<\ldots>$’ notation. They signify parameters that may vary from reduction to reduction, such as filenames, but which are necessary in each reduction process. Notable uses of this notation include the date of observation, <*OBSDATE*> (formatted ‘YYYYMMDD’), and the filenames for the science and arc FITS files, <*O-beam ARC*> and <*O-beam FILES*> (or <*E-beam ARC*> and <*E-beam FILES*> for the other polarimetric beam), respectively.

```
<OBSDATE>
 ├── raw
 │    └── P<OBSDATE><*>.fits
 └── sci
      └── mxgbpP<OBSDATE><*>.fits
```

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 does not form part of the minimal file structure necessary to begin the reduction process. If 'mxgbp' prefixed data is available, the reduction may be begun starting at Listing I.2.

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 <*OBSDATE*>
- 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

Listing I.2: Splitting data using STOPS

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

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 `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 along with the cursor commands, but may also be found using any FITS viewing software capable of displaying rows of image data.[1]

Listing I.4: The IRAF `identify` task

```
cl> mkscript 01_identify.cl
```

[1] See `https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/plot.implot.html` for documentation on the `implot` task.

```
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$idhenear.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="")
```

IRAF will launch an interactive window for the `identify` task. Cursor commands 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.[2] Repeat the `identify` procedure, replacing *<O-beam ARC>* with *<E-beam ARC>*.

Listing I.5: 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$idhenear.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".[3] Repeat the `reidentify` procedure, replacing *<O-beam ARC>* with *<E-beam ARC>* at both the 'reference' and 'image' parameter locations.

Listing I.6: The IRAF `fitcoords` task

[2]See `https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.onedspec.identify.html` for documentation on the `identify` task.

[3]See `https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.onedspec.reidentify.html` for documentation on the `reidentify` 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 ("<O-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="")
```

IRAF will launch an interactive window for the `fitcoords` task. The interactive window allows the parameters to be optimized without having to rerun the task. The x- and y-axis being plotted may be changed using the 'x' or 'y' cursor commands followed by the desired data axis (such as 'x', 'y', or 'r' for residuals).[4] Repeat the `fitcoords` procedure, replacing *<O-beam ARC>* with *<E-beam ARC>*.

Listing I.7: 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 ("@<O-beam FILES>",
>>> "t//@<O-beam FILES>", "<O-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, notably the arc images, using any FITS viewer as a cursory check that the wavelength calibrations were completed without error.[5]

The gain and read noise is now needed since part of the STOPS `join` method, the cosmic ray rejection, may need them as parameters. Determining these parameters may be done using the '*GAINSET*' and '*ROSPEED*' FITS keywords, where the cosmic ray rejection defaults to *GAINSET*='FAINT', and *ROSPEED*='SLOW'. If the values for the keywords differ the gain and read noise parameters should be updated.[6]

Listing I.8: Joining the data using STOPS

[4]See `https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.twodspec.longslit.fitcoords.html` for documentation on the `fitcoords` task.

[5]See `https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.twodspec.longslit.transform.html` for documentation on the `transform` task.

[6]The read noise and gain may be determined from `http://pysalt.salt.ac.za/proposal_calls/current/ProposalCall.html`, specifically Tables 6.1 and 6.2.

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

Listing I.9: The STOPS `skylines` method

```
$ cd <OBSDATE>/sci
$ conda activate stops
$ python ~/STOPS . skylines <O-beam SCI>
```

Listing I.10: File cleanup for POLSALT

```
$ cd <OBSDATE>/sci
$ mkdir split_files
$ mv *obeam* *ebeam* *oarc* *earc* 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'.

Listing I.11: The STOPS `correlate` method

```
$ cd <OBSDATE>/sci
$ conda activate stops
$ python ~/STOPS . correlate <O-beam SCI>
```

The POLSALT `raw Stokes calculation`, `final Stokes calculation`, and `results visualisation` can now be completed. For the last time, if the POLSALT GUI was closed it should now be reopened using 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.

```
<OBSDATE>
    raw
        P<OBSDATE><*>.fits
    sci
        database
            01_identify.cl
            02_reidentify.cl
            03_fitcoords.cl
            04_transform.cl
            deletions.db
            fcearc00<##>
            fcoarc00<##>
            idearc00<##>
            idoarc00<##>
            <*>.eps
        split_files
            oarc00<##>.fits
            earc00<##>.fits
            obeam<*>.fits
            ebeam<*>.fits
            toarc00<##>.fits
            tearc00<##>.fits
            tobeam<*>.fits
            tebeam<*>.fits
        <OBSDATE>_geom.txt
        <OBSDATE>_filtered.txt
        cwmxgbpP<OBSDATE><*>.fits
        ecwmxgbpP<OBSDATE><*>.fits
        mxgbpP<OBSDATE><*>.fits
        wmxgbpP<OBSDATE><*>.fits
        <*>.log
        <OBJ>_c0_h<*>_01.fits
        <OBJ>_c0_1_stokes.fits
        <OBJ>_c0_1_stokes_<BIN>_Ipt.txt
        <OBJ>_c0_1_stokes_<BIN>_Ipt.pdf
```

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

Appendix II

STOPS Source Code

This appendix entry 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**

```
"""Argument parser for STOPS."""

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

__version__ = "2024.04.13"
__author__ = "Justin Cooper"
__email__ = "justin.jb78+Masters@gmail.com"

# MARK: Imports
import os
import argparse
import logging

import split
import join
import cross_correlate
import skylines

from utils import ParserUtils as pu
from utils.Constants import SPLIT_ROW, PREFIX, PARSE, CORR_SAVENAME

# MARK: Constants
PROG = "STOPS"
DESCRIPTION = """
Supplementary TOols for Polsalt Spectropolarimetry (STOPS) is a
collection of supplementary tools created for SALT's POLSALT pipeline,
allowing for wavelength calibrations with IRAF. The tools provide
support for splitting and joining polsalt formatted data as well as
cross correlating complementary polarimetric beams.

DOI: 10.22323/1.401.0056
"""
```

```python


# MARK: Universal Parser
parser = argparse.ArgumentParser(
    prog=PROG,
    description=DESCRIPTION,
    formatter_class=argparse.RawDescriptionHelpFormatter,
)
parser.add_argument(
    "-V",
    "--version",
    action="version",
    version=f"%(prog)s as of {__version__}",
)
parser.add_argument(
    "-v",
    "--verbose",
    action="count",
    default=PARSE['VERBOSE'],
    help=(
        "Counter flag which enables and increases verbosity. "
        "Use -v or -vv for greater verbosity levels."
    ),
)
parser.add_argument(
    "-l",
    "--log",
    action="store",
    type=pu.parse_logfile,
    help=(
        "Filename to which logging is saved to. "
        "File is created if it does not exist. Defaults to None."
    ),
)
parser.add_argument(
    "data_dir",
    action="store",
    nargs="?",
    default=PARSE["DATA_DIR"],
    type=pu.parse_path,
    help=(
        "Path of the directory which contains the working data. "
        f"Defaults to the cwd -> `{PARSE["DATA_DIR"]}' (I.E. `.')."
    ),
)


# MARK: Split\Join Parent Args
split_join_args = argparse.ArgumentParser(add_help=False)
split_join_args.add_argument(
    "-n",
    "--no_arc",
    action="store_true",
    help="Flag to exclude arc files from processing.",
)
split_join_args.add_argument(
    "-s",
    "--split_row",
```

```
    default=SPLIT_ROW,
    type=int,
    help=(
        "Row along which the O and E beams are split. "
        f"Defaults to polsalt's default -> {SPLIT_ROW}."
    ),
)
split_join_args.add_argument(
    "-p",
    "--save_prefix",
    nargs=2,
    default=PREFIX,
    help=(
        "Prefix appended to the filenames, "
        "with which the O and E beams are saved. "
        f"Defaults to {PREFIX}."
    ),
)


# MARK: Create subparser modes
subparsers = parser.add_subparsers(
    dest="mode",
    help="Operational mode of supplementary tools",
)


# MARK: Split Subparser
split_parser = subparsers.add_parser(
    "split",
    aliases=["s"],
    help="Split mode",
    parents=[split_join_args],
)
# 'children' split args here
# Change defaults here
split_parser.set_defaults(
    mode="split",
    func=split.Split,
)


# MARK: Join Subparser
join_parser = subparsers.add_parser(
    "join",
    aliases=["j"],
    help="Join mode",
    parents=[split_join_args],
)
# 'children' join args here
join_parser.add_argument(
    "-c",
    "--coefficients",
    dest="solutions_list",
    nargs='*',
    type=pu.parse_coeff_file,
    help=(
        "Custom coefficients to use instead of the `IRAF` fitcoords "
```

```
        "database. Use as either '-c <o_solution> <e_solution>' or "
        "a regex descriptor '-c <*_solution*extention>'."
    ),
)
# Change defaults here
join_parser.set_defaults(
    mode="join",
    func=join.Join,
)


# MARK: Correlate Subparser
corr_parser = subparsers.add_parser(
    "correlate",
    aliases=["x"],
    help="Cross correlation mode",
)
# 'children' correlate args here
corr_parser.add_argument(
    "fits_list",
    action="store",
    nargs="+",
    type=pu.parse_corr_file,
    help=(
        "Filenames to be compared. "
        "Provide only one filename for O/E beam comparisons. "
        "Include relative path to working directory."
    ),
)
corr_parser.add_argument(
    "-ccd",
    "--split_ccd",
    action="store_false",
    help=(
        "Flag to NOT split CCD's. "
        "Recommended to leave off unless the chip gaps "
        "have been removed from the data."
    ),
)
corr_parser.add_argument(
    "-c",
    "--continuum_order",
    type=int,
    default=PARSE["CONT_ORD"],
    dest="cont_ord",
    help=(
        "Order of continuum to remove from spectra. "
        "Higher orders recommended to remove most variation, "
        "leaving only significant features."
    ),
)
corr_parser.add_argument(
    "-p",
    "--continuum_plot",
    action="store_true",
    dest="cont_plot",
    help=(
        "Flag to plot fitting of continuum. "
```

```
        "Used to confirm only notable features left in spectrum."
    ),
)
corr_parser.add_argument(
    "-o",
    "--offset",
    type=int,
    default=PARSE["OFFSET"],
    help=(
        "Introduces an offset when correcting for "
        "known offset in spectra or for testing purposes. "
        f"Defaults to {PARSE["OFFSET"]}"
        "(For testing, not used during regular operation.)"
    ),
)
corr_parser.add_argument(
    "-s",
    "--save_name",
    help=(
        "Name with which to save the output plot. "
        "If left undefined, plot will not be saved. "
        f"If a path is provided, {CORR_SAVENAME} will be used."
    ),
)
# Change defaults here
corr_parser.set_defaults(
    mode="correlate",
    func=cross_correlate.CrossCorrelate,
)


# MARK: Skyline Subparser
sky_parser = subparsers.add_parser(
    "skylines",
    aliases=["sky"],
    help="Sky line check mode",
)
# 'children' skyline args here
sky_parser.add_argument(
    "filenames",
    action="store",
    nargs="+",
    type=pu.parse_file,
    help=(
        "File name(s) of FITS file(s) to be checked "
        "using SALT's sky atlas. "
        "A minimum of one filename is required."
    ),
)
sky_parser.add_argument(
    "-s",
    "--save_prefix",
    action="store",
    nargs="?",
    const="sky",
    help=(
        "Prefix used when saving plot. "
        "Excluding flag does not save output plot, "
```

```python
            "flag usage of option uses 'sky' default prefix, "
            "and a provided prefix overwrites default prefix."
        ),
)
sky_parser.add_argument(
        "-b",
        "--beams",
        choices=["o", "e", "oe"],
        type=str.lower,
        default=PARSE["BEAMS"],
        help=(
            "Beam(s) for skyline checking. "
            "Defaults to both beams overplotted, but "
            "may be given either 'o', 'e', or 'oe' for "
            "separating and excluding beam plots."
        ),
)
sky_parser.add_argument(
        "-t",
        "--transform",
        action="store_true",
        help=(
            "Flag to NOT transform image. "
            "Defaults to true. "
            "Recommended to use only when input image(s) "
            "are already transformed."
        ),
)
sky_parser.set_defaults(
        mode="skyline",
        func=skylines.Skylines,
)


# MARK: Keyword Clean Up
args = parser.parse_args()
args.verbose = pu.parse_loglevel(args.verbose)
if 'log' in args and args.log not in ["", None]:
        args.log = args.data_dir / args.log
if "filenames" in args:
        args.filenames = pu.flatten(args.filenames)
if "solutions_list" in args:
        args.solutions_list = pu.flatten(args.solutions_list)

# MARK: Begin logging
logging.basicConfig(
        filename=args.log,
        format="%(asctime)s - %(module)s - %(levelname)s - %(message)s",
        datefmt="%Y-%m-%d %H:%M:%S",
        level=args.verbose,
)


# MARK: Call Relevant Class(Args)
logging.debug(f"Argparse namespace: {args}")
logging.info(f"Mode:{args.mode}")
args.func(**vars(args)).process()
```

```

# Confirm all processes completed and exit without error
logging.info("All done! Come again!\n")
```

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

```python
"""Module for splitting ``polsalt`` FITS files."""

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from __main__ import __author__, __email__, __version__

# MARK: Imports
import os
import sys
import logging
from copy import deepcopy

import numpy as np
from astropy.io import fits as pyfits

from utils.SharedUtils import find_files, find_arc
from utils.Constants import SAVE_PREFIX, CROP_DEFAULT, SPLIT_ROW


# MARK: Split Class
class Split:
    """
    The `Split` class allows for the splitting of `polsalt` FITS files
    based on the polarization beam. The FITS files must have basic
    `polsalt` pre-reductions already applied (`mxgbp...` FITS files).

    Parameters
    ----------
    data_dir : str
        The path to the data to be split
    fits_list : list[str], optional
        A list of pre-reduced `polsalt` FITS files to be split within `data_dir`.
        (The default is None, `Split` will search for `mxgbp*.fits` files)
    split_row : int, optional
        The row along which to split the data of each extension in the FITS file.
        (The default is SPLIT_ROW (See Notes), the SALT RSS CCD's middle row)
    no_arc : bool, optional
        Decides whether the arc frames should be recombined.
        (The default is False, `polsalt` has no use for the arc after wavelength calibrations)
    save_prefix : dict[str, list[str]], optional
        The prefix with which to save the  O & E beams.
        Setting `save_prefix` = ``None`` does not save the split O & E beams.
        (The default is SAVE_PREFIX (See Notes))

    Attributes
    ----------
    arc : str
        Name of arc FITS file within `data_dir`.
        `arc` = `""` if `no_arc` or not detected in `data_dir`.
```

```
    o_files, e_files : list[str]
        A list of the `O`- and `E`-beam FITS file names.
        The first entry is the arc file if `arc` defined.
    data_dir
    fits_list
    split_row
    save_prefix

    Methods
    -------
    split_file(file: os.PathLike)
        -> tuple[astropy.io.fits.HDUList]
        Handles creation and saving the separated FITS files
    split_ext(hdulist: astropy.io.fits.HDUList, ext: str = 'SCI')
        -> astropy.io.fits.HDUList
        Splits the data in the `ext` extension along the `split_row`
    crop_file(hdulist: astropy.io.fits.HDUList, crop: int =
    ↪ CROP_DEFAULT (See Notes))
        -> tuple[numpy.ndarray]
        Crops the data along the edge of the frame, that is,
        `O`-beam cropped as [crop:], and
        `E`-beam cropped as [:-crop].
    update_beam_lists(o_name: str, e_name: str)
        -> None
        Updates `o_files` and `e_files`.
    save_beam_lists(file_suffix: str = 'frames')
        -> None
        Creates (Overwrites if exists) and writes the `o_files` and
        ↪ `e_files` to files named
        `o_{file_suffix}` and `e_{file_suffix}`, respectively.
    process()
        -> None
        Calls `split_file` and `save_beam_lists` on each file in
        ↪ `fits_list` for automation.

    Other Parameters
    ----------------
    **kwargs : dict
        keyword arguments. Allows for passing unpacked dictionary to
        ↪ the class constructor.

    Notes
    -----
    Constants Imported (See utils.Constants):
        SAVE_PREFIX
        CROP_DEFAULT
        SPLIT_ROW

    """
    # MARK: Split init
    def __init__(
        self,
        data_dir: str,
        fits_list: list[str] = None,
        split_row: int = SPLIT_ROW,
        no_arc: bool = False,
        save_prefix=None,
        **kwargs
```

```
    ) -> None:
        self.data_dir = data_dir
        self.fits_list = find_files(
            data_dir=data_dir,
            filenames=fits_list,
            prefix="mxgbp",
            ext="fits"
        )
        self.split_row = split_row
        self.save_prefix = SAVE_PREFIX
        if type(save_prefix) == dict:
            self.save_prefix = save_prefix

        self.arc = "" if no_arc else find_arc(self.fits_list)
        self.o_files = []
        self.e_files = []

        return

    # MARK: Split Files
    def split_file(
        self,
        file: os.PathLike
    ) -> tuple[pyfits.HDUList]:
        """
        Split the single FITS file into separated `O`- and `E`- FITS
        ↪ files.

        Parameters
        ----------
        file : os.PathLike
            The name of the FITS file to be split.

        Returns
        -------
        tuple[astropy.io.fits.HDUList]
            Tuple containing the split O and E beam HDULists.

        """
        # Create empty HDUList
        O_beam = pyfits.HDUList()
        E_beam = pyfits.HDUList()

        # Open file and split O & E beams
        with pyfits.open(file) as hdul:
            O_beam.append(hdul["PRIMARY"].copy())
            E_beam.append(hdul["PRIMARY"].copy())

            # Split specific extention
            raw_split = self.split_ext(hdul, "SCI")

            # O_beam[0].data = raw_split['SCI'].data[1]
            # E_beam[0].data = raw_split['SCI'].data[0]
            O_beam[0].data, E_beam[0].data = self.crop_file(raw_split)

            # Handle prefix and names
            pref = "arc" if file == self.arc else "beam"
            o_name = self.save_prefix[pref][0] + file.name[-9:]
```

```
            e_name = self.save_prefix[pref][1] + file.name[-9:]

            # Add split data to O & E beam lists
            self.update_beam_lists(o_name, e_name, pref == "arc")

            # Handle don't save case
            if self.save_prefix == None:
                return O_beam, E_beam

            # Handle save case
            O_beam.writeto(o_name, overwrite=True)
            E_beam.writeto(e_name, overwrite=True)

            return O_beam, E_beam

    # MARK: Split extensions
    def split_ext(
        self,
        hdulist: pyfits.HDUList,
        ext: str = "SCI"
    ) -> pyfits.HDUList:
        """
        Split the data of the specified extension of `hdulist` into its `O`- and `E`- beams.

        Parameters
        ----------
        hdulist : astropy.io.fits.HDUList
            The FITS HDUList to be split.
        ext : str, optional
            The name of the extension to be split.
            (Defaults to 'SCI')

        Returns
        -------
        astropy.io.fits.HDUList
            The HDUList with the split applied.

        """
        hdu = deepcopy(hdulist)
        rows, cols = hdu[ext].data.shape

        # if odd number of rows, strip off the last one
        rows = int(rows / 2) * 2

        # how far split is from center of detector
        offset = int(self.split_row - rows / 2)

        # split arc into o/e images
        ind_rc = np.indices((rows, cols))[0]
        padbins = (ind_rc < offset) | (ind_rc > rows + offset)

        # Roll split_row to be centre row
        image_rc = np.roll(hdu[ext].data[:rows, :], -offset, axis=0)
        image_rc[padbins] = 0.0

        # Split columns equally
        hdu[ext].data = image_rc.reshape((2, int(rows / 2), cols))
```

```python

        return hdu

    # MARK: Crop files
    def crop_file(
        self,
        hdulist: pyfits.HDUList,
        crop: int = CROP_DEFAULT
    ) -> tuple[np.ndarray]:
        """
        Crop the data with respect to the `O`/`E` beam.

        Parameters
        ----------
        hdulist : astropy.io.fits.HDUList
            The HDUList containing the data to be cropped.
        crop : int, optional
            The number of rows to be cropped from the bottom and top
            of the `O` and `E` beam, respectively.
            (Defaults to 40)

        Returns
        -------
        tuple[numpy.ndarray]
            Tuple containing the cropped O and E beam data arrays.

        """
        o_data = hdulist["SCI"].data[1, 0:-crop]
        e_data = hdulist["SCI"].data[0, crop:]

        return o_data, e_data

    # MARK: Update beam lists
    def update_beam_lists(
        self,
        o_name,
        e_name,
        arc: bool = True
    ) -> None:
        """
        Update the `o_files` and `e_files` attributes.

        Parameters
        ----------
        o_name : str
            The filename of the O beam.
        e_name : str
            The filename of the E beam.
        arc : bool, optional
            Indicates whether the first entry should be the arc frame.
            (Defaults to True)

        """
        if arc:
            self.o_files.insert(0, o_name)
            self.e_files.insert(0, e_name)
        else:
            self.o_files.append(o_name)
```

```
                self.e_files.append(e_name)

        return

    # MARK: Save beam lists
    def save_beam_lists(self, file_suffix: str = 'frames') -> None:
        with open(f"o_{file_suffix}", "w+") as f_o, \
             open(f"e_{file_suffix}", "w+") as f_e:
            for i, j in zip(self.o_files, self.e_files):
                f_o.write(i + "\n")
                f_e.write(j + "\n")

        return

    # MARK: Process all Listed Images
    def process(self) -> None:
        """Process all FITS images stored in the `fits_list`
        ↪ attribute"""
        for target in self.fits_list:
            logging.debug(f"Processing {target}")
            self.split_file(target)

        self.save_beam_lists()

        return

# MARK: Main function
def main(argv) -> None:
    """Main function."""

    return


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

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

```python
"""Module for joining the split FITS files with an external wavelength
↪ solution."""

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from __main__ import __author__, __email__, __version__

# MARK: Imports
import os
import sys
import logging
import re

import numpy as np
from numpy.polynomial.chebyshev import chebgrid2d as chebgrid2d
from numpy.polynomial.legendre import leggrid2d as leggrid2d
from astropy.io import fits as pyfits

# from lacosmic import lacosmic # Deprecated: ccdproc is ~6x faster
from ccdproc import cosmicray_lacosmic as lacosmic

from utils.specpolpy3 import read_wollaston, split_sci
from utils.SharedUtils import find_files, find_arc
from utils.Constants import DATADIR, SAVE_PREFIX, SPLIT_ROW, CR_PARAMS


# MARK: Join Docstring
class Join:
    """
    The `Join` class allows for the joining of previously
    split files and the appending of an external wavelength
    solution to the `polsalt` FITS file format.

    Parameters
    ----------
    data_dir : str
        The path to the data to be joined
    database : str, optional
        The name of the `IRAF` database folder.
        (The default is "database")
    fits_list : list[str], optional
        A list of pre-reduced `polsalt` FITS files to be joined within
        ↪ `data_dir`.
        (The default is ``None``, `Join` will search for `mxgbp*.fits`
        ↪ files)
    solutions_list: list[str], optional
        A list of solution filenames from which the wavelength solution
        ↪ is created.
        (The default is ``None``, `Join` will search for `fc*` files
        ↪ within the `database` directory)
    split_row : int, optional
        The row along which the data of each extension in the FITS file
        ↪ was split.
        Necessary when Joining cropped files.
        (The default is 517, the SALT RSS CCD's middle row)
```

```
    save_prefix : dict[str, list[str]], optional
        The prefix with which the previously split `O`- & `E`-beams were saved.
        Used for detecting if cropping was applied during the splitting procedure.
        (The default is SAVE_PREFIX (See Notes))
    verbose : int, optional
        The level of verbosity to use for the Cosmic ray rejection
        (The default is 30, I.E. logging.INFO)

    Attributes
    ----------
    fc_files : list[str]
        Valid solutions found from `solutions_list`.
    custom : bool
        Internal flag for whether `solutions_list` uses the `IRAF` or a custom format.
        See Notes for custom solution formatting.
        (Default (inherited from `solutions_list`) is False)
    arc : str
        Deprecated. Name of arc FITS file within `data_dir`.
    data_dir
    database
    fits_list
    split_row
    save_prefix


    Methods
    -------
    get_solutions(wavlist: list | None, prefix: str = "fc")
        -> (fc_files, custom): tuple[list[str], bool]
        Parse `solutions_list` and return valid solution files and if they are non-`IRAF` solutions.
    parse_solution(fc_file: str, xshape: int, yshape: int)
        -> tuple[dict[str, int], np.ndarray]
        Loads the wavelength solution file and parses keywords necessary for creating the wavelength extension.
    join_file(file: os.PathLike)
        -> None
        Joins the files,
        attaches the wavelength solutions,
        performs cosmic ray cleaning,
        masks the extension,
        and checks cropping performed in `Split`.
        Writes the FITS file in a `polsalt` valid format.
    check_crop(hdu: pyfits.HDUList, o_file: str, e_file: str)
        -> int
        Opens the split `O`- and `E`-beam FITS files and returns the amount of cropping that was performed.
    process()
        -> None
        Calls `join_file` on each file in `fits_list` for automation.


    Other Parameters
    ----------------
    no_arc : bool, optional
```

```
            Deprecated. Decides whether the arc frames should be processed.
            (The default is False, `polsalt` has no use for the arc after wavelength calibrations)
        **kwargs : dict
            keyword arguments. Allows for passing unpacked dictionary to the class constructor.

        Notes
        -----
        Constants Imported (See utils.Constants):
            DATADIR
            SAVE_PREFIX
            SPLIT_ROW
            CR_PARAMS

        Custom wavelength solutions must be formatted as:
            `x`,
            `y`,
            *coefficients...
        where the solutions are of order (`x` by `y`) and contain x*y coefficients.
        The name of the custom wavelength solution file must contain either "cheb" or "leg"
        for Chebychev or Legendre wavelength solutions, respectively.

        Cosmic ray rejection is performed using lacosmic [1]_ implemented in ccdproc via astroscrappy [2]_.

        References
        ----------
        .. [1] van Dokkum 2001, PASP, 113, 789, 1420 (article : http://adsabs.harvard.edu/abs/2001PASP..113.1420V)
        .. [2] https://zenodo.org/records/1482019

        """
        # MARK: Join init
        def __init__(
            self,
            data_dir: str,
            database: str = "database",
            fits_list: list[str] = None,
            solutions_list: list[str] = None,
            split_row: int = SPLIT_ROW,
            no_arc: bool = True,
            save_prefix=None,
            verbose: int = 30,
            **kwargs,
        ) -> None:
            self.data_dir = data_dir
            self.database = database
            self.fits_list = find_files(
                data_dir=self.data_dir,
                filenames=fits_list,
                prefix="mxgbp",
                ext="fits",
            )
            self.fc_files, self.custom = self.get_solutions(solutions_list)
            self.split_row = split_row
```

```
        self.save_prefix = SAVE_PREFIX
        if type(save_prefix) == dict:
            self.save_prefix = save_prefix

        self.no_arc = no_arc
        self.arc = find_arc(self.fits_list)

        self.verbose = verbose < 30
        return

    # MARK: Find 2D WAV Functions
    def get_solutions(
        self,
        wavlist: list[str] | None,
        prefix: str = "fc"
    ) -> tuple[list[str], bool]:
        """
        Get the list of wavelength solution files.

        Parameters
        ----------
        wavlist : list[str] | None
            A list of custom wavelength solutions files.
            If ``None``, `Join` will search for wavelength solutions in
            ↪ the `database` directory.
        prefix : str, optional
            The prefix of the wavelength solution files.
            (Defaults to "fc")

        Returns
        -------
        tuple[list[str], bool]
            A tuple containing the list of wavelength solutions files
            ↪ and
            a boolean indicating whether custom solutions were provided.

        """
        # No custom solutions
        if not wavlist:
            # Handle finding solutions
            ws = []
            for fl in os.listdir(
                os.path.join(self.data_dir, self.database)
            ):
                if os.path.isfile(
                    os.path.join(self.data_dir, self.database, fl)
                ) and (prefix == fl[0:2]):
                    ws.append(fl)

            if len(ws) != 2:
                # Handle incorrect number of solutions found
                msg = (
                    f"Incorrect amount of wavelength solutions "
                    f"({len(ws)} fc... files) found in the solution "
                    f"dir.: {os.path.join(self.data_dir,
                    ↪ self.database)}"
                )
                logging.error(msg)
```

```python
                raise FileNotFoundError(msg)

            return (sorted(ws, reverse=True), False)

        # Custom solution
        if len(wavlist) >= 2:
            if len(wavlist) > 2:
                logging.warning(f" Too many solutions: {wavlist}")
                wavlist = wavlist[:2]

            for fl in wavlist:
                if not os.path.isfile(os.path.join(self.data_dir, fl)):
                    msg = (
                        f"{fl} not found in the "
                        f"data directory {self.data_dir}"
                    )
                    logging.error(msg)
                    raise FileNotFoundError(msg)

            return (sorted(wavlist, reverse=True), True)

    # MARK: Parse 2D WAV Function
    def parse_solution(
        self,
        fc_file: str,
        xshape: int,
        yshape: int
    ) -> tuple[dict[str, int], np.ndarray]:
        """
        Parse the 2D wavelength solution function from `fc_file`.

        Parameters
        ----------
        fc_file : str
            The filename of the wavelength solutions file.
        xshape : int
            The x-order of the 2D solution.
        yshape : int
            The y-order of the 2D solution.

        Returns
        -------
        tuple[dict[str, int], np.ndarray]
            A tuple containing a dictionary of the parameters of the solution function
            and the function coefficients.

        """
        fit_params = {}
        coeff = []

        if self.custom:
            # Load coefficients
            coeff = np.loadtxt(fc_file)

            fit_params["xorder"] = coeff[0].astype(int)
            fit_params["yorder"] = coeff[1].astype(int)
            coeff = coeff[2:]
```

```python
            f_type = 3
            if "cheb" in str(fc_file): f_type = 1
            elif "leg" in str(fc_file): f_type = 2
            fit_params["function"] = f_type

            fit_params["xmin"], fit_params["xmax"] = 1, xshape
            fit_params["ymin"], fit_params["ymax"] = 1, yshape

        else:
            # Parse IRAF fc database files
            file_contents = []
            with open(self.database + "/" + fc_file) as fcfile:
                for i in fcfile:
                    file_contents.append(re.sub(r"[\n\t\s]*", "", i))

            if file_contents[9] != "1.":  # xterms - Cross-term type
                msg=(
                    "Cross-term not recognised (always 1 for "
                    "FITCOORDS), redo FITCOORDS or change manually."
                )
                raise Exception(msg)

            fit_params["function"] = int(file_contents[6][:-1])

            fit_params["xorder"] = int(file_contents[7][:-1])
            fit_params["yorder"] = int(file_contents[8][:-1])

            fit_params["xmin"] = int(file_contents[10][:-1])
            fit_params["xmax"] = xshape
            # int(file_contents[11][:-1])# stretch fit over x
            fit_params["ymin"] = int(file_contents[12][:-1])
            fit_params["ymax"] = yshape
            # int(file_contents[13][:-1])# stretch fit over y

            coeff = np.array(file_contents[14:], dtype=float)

        coeff = np.reshape(
            coeff,
            (fit_params["xorder"], fit_params["yorder"])
        )

        return (fit_params, coeff)

    # MARK: Join Files
    def join_file(self, file: os.PathLike) -> None:
        """
        Join the ‘O‘- and ‘E‘-beams, attach the wavelength solutions,
        perform cosmic ray cleaning, mask the extensions,
        and checks cropping performed by ‘Split‘.
        Write the FITS file in a ‘polsalt‘ valid format.

        Parameters
        ----------
        file : os.PathLike
            The path of the FITS file to be joined.

        See Also
```

```
        --------
        IRAF - ‘fitcoords‘ task
            https://iraf.net/irafdocs/formats/fitcoords.php,
        numpy.polynomial.chebyshev.chebgrid2d
            https://numpy.org/doc/stable/reference/generated/numpy.polynomial.chebys
        numpy.polynomial.legendre.leggrid2d
            https://numpy.org/doc/stable/reference/generated/numpy.polynomial.legend

        """
        # Create empty wavelength appended hdu list
        whdu = pyfits.HDUList()
        primary_ext = ""

        # Handle prefix and names
        pref = "arc" if file == self.arc else "beam"
        o_file = self.save_prefix[pref][0] + file.name[-9:]
        e_file = self.save_prefix[pref][1] + file.name[-9:]

        # Open file
        with pyfits.open(file) as hdu:
            # Check if file has been cropped
            cropsize = self.check_crop(hdu, o_file, e_file)

            y_shape = int(hdu["SCI"].data.shape[0] / 2) - cropsize
            x_shape = hdu["SCI"].data.shape[1]

            # No differences in "PRIMARY" extention header
            primary_ext = hdu["PRIMARY"]
            whdu.append(primary_ext)

            for ext in ["SCI", "VAR", "BPM"]:
                whdu.append(pyfits.ImageHDU(name=ext))
                whdu[ext].header = hdu[ext].header.copy()
                whdu[ext].header["CTYPE3"] = "O,E"

                # Create empty extentions with correct order and format
                if ext == "BPM":
                    whdu[ext].data = np.zeros(
                        (2, y_shape, x_shape),
                        dtype="uint8"
                    )
                    whdu[ext].header["BITPIX"] = "-uint8"
                else:
                    whdu[ext].data = np.zeros(
                        (2, y_shape, x_shape),
                        dtype=">f4"
                    )
                    whdu[ext].header["BITPIX"] = "-32"

                # Fill in empty extentions
                if cropsize:
                    temp_split = split_sci(
                        hdu,
                        self.split_row,
                        ext=ext
                    )[ext].data
                    whdu[ext].data[0] = temp_split[0, cropsize:]
                    whdu[ext].data[1] = temp_split[1, 0:-cropsize]
```

```
                else:
                    whdu[ext].data = split_sci(
                        hdu,
                        self.split_row,
                        ext=ext
                    )[ext].data
        # End of hdu calls, close hdu

        # MARK: Join (Wav. Ext.)
        whdu.append(pyfits.ImageHDU(name="WAV"))
        wav_header = whdu["SCI"].header.copy()
        wav_header["EXTNAME"] = "WAV"
        wav_header["CTYPE3"] = "O,E"
        whdu["WAV"].header = wav_header

        whdu["WAV"].data = np.zeros(
            whdu["SCI"].data.shape,
            dtype=">f4"
        )

        for num, fname in enumerate(self.fc_files):
            pars, chebvals = self.parse_solution(
                fname,
                x_shape,
                y_shape
            )

            if pars["function"] == 1:  # Function type (1 = chebyshev)
                # Set wavelength extention values to function
                whdu["WAV"].data[num] = chebgrid2d(
                    x=np.linspace(-1, 1, pars["ymax"]),
                    y=np.linspace(-1, 1, pars["xmax"]),
                    c=chebvals,
                )

            elif pars["function"] == 2:  # Function type (2 = legendre)
                # Set wavelength extention values to function
                whdu["WAV"].data[num] = leggrid2d(
                    x=np.linspace(-1, 1, pars["ymax"]),
                    y=np.linspace(-1, 1, pars["xmax"]),
                    c=chebvals,
                )

            else:
                msg = (
                    "Function type not recognised, please wavelength "
                    "calibrate using either chebychev or legendre."
                )
                raise Exception(msg)

            # MARK: Cosmic Ray Cleaning
            # See utils.Constants for `CR_PARAMS` discussion
            whdu["SCI"].data[num] = lacosmic(
                whdu["SCI"].data[num],
                # contrast=CR_PARAMS['CR_CONTRAST'],
                # threshold=CR_PARAMS['CR_THRESHOLD'],
                #
```

```
                ↪ neighbor_threshold=CR_PARAMS['CR_NEIGHBOUR_THRESHOLD'],
                # effective_gain=CR_PARAMS['GAIN'],
                # background=CR_PARAMS['BACKGROUND'],
                readnoise=CR_PARAMS['READNOISE'],
                gain=CR_PARAMS['GAIN'],
                verbose=self.verbose,
            )[0]

        # MARK: WAV masking
        # Left & Right Crop
        whdu["WAV"].data[whdu["WAV"].data[:] < 3_000] = 0.0
        whdu["WAV"].data[whdu["WAV"].data[:] >= 10_000] = 0.0

        # Top & Bottom Crop (shift\tilt)
        rpix_oc, cols, rbin, lam_c = read_wollaston(
            whdu,
            DATADIR + "wollaston.txt"
        )

        drow_oc = (rpix_oc - rpix_oc[:, int(cols / 2)][:, None]) / rbin

        ## Cropping as suggested
        for c, col in enumerate(drow_oc[0]):
            if np.isnan(col):
                continue

            if int(col) < 0:
                whdu["WAV"].data[0, int(col) :, c] = 0.0
            elif int(col) > cropsize:
                whdu["WAV"].data[0, 0 : int(col) - cropsize, c] = 0.0

        for c, col in enumerate(drow_oc[1]):
            if np.isnan(col):
                continue

            if int(col) > 0:
                whdu["WAV"].data[1, 0 : int(col), c] = 0.0
            elif (int(col) < 0) & (abs(int(col)) > cropsize):
                whdu["WAV"].data[1, int(col) + cropsize :, c] = 0.0

        # MARK: BPM masking
        whdu["BPM"].data[0] = np.where(
            whdu["WAV"].data[0] == 0,
            1,
            whdu["BPM"].data[0]
        )
        whdu["BPM"].data[1] = np.where(
            whdu["WAV"].data[1] == 0,
            1,
            whdu["BPM"].data[1]
        )

        whdu.writeto(f"w{os.path.basename(file)}", overwrite="True")

        return

    # MARK: Check Crop
    def check_crop(
```

```python
        self,
        hdu: pyfits.HDUList,
        o_file: str,
        e_file: str
    ) -> int:
        """
        Check if cropping is necessary when joining `O`- and `E`-beams.

        Parameters
        ----------
        hdu : astropy.io.fits.HDUList
            The HDUList to check for cropping.
        o_file : str
            The name of the previously split `O`-beam FITS file.
        e_file : str
            The name of the previously split `E`-beam FITS file.

        Returns
        -------
        int
            The number of rows which were cropped by `Split`.

        """
        cropsize = 0
        o_y = 0
        e_y = 0

        with pyfits.open(o_file) as o:
            o_y = o[0].data.shape[0]

        with pyfits.open(e_file) as e:
            e_y = e[0].data.shape[0]

        if hdu["SCI"].data.shape[0] != (o_y + e_y):
            # Get crop size, assuming crop same on both sides
            cropsize = int((hdu["SCI"].data.shape[0] - o_y - e_y) / 2)

        return cropsize

    # MARK: Process all Listed Images
    def process(self) -> None:
        """Process all FITS images stored in the `fits_list`
        ↪ attribute"""
        for target in self.fits_list:
            logging.debug(f"Processing {target}")
            self.join_file(target)

        return


def main(argv) -> None:
    """Main function."""

    return


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

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

```
"""Module for cross correlating polarization beams."""

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from __main__ import __author__, __email__, __version__

# MARK: Imports
import os
import sys
import logging
import itertools as iters

import numpy as np
from numpy.polynomial import chebyshev
import matplotlib.pyplot as plt
from astropy.io import fits as pyfits
from scipy import signal

from utils.SharedUtils import continuum
from utils.Constants import CORR_SAVENAME

# TODO@JustinotherGitter: Update correlate to use relevant args:
    # fits_list <- in1/in2,
    # cont_ord <- cont,


class CrossCorrelate:
    """
    Cross correlate allows for comparing the extensions of multiple
    FITS files, or comparing the O and E beams of a single FITS file.

    Parameters
    ----------
    data_dir : str
        The path to the data to be cross correlated
    fits_list : list[str]
        The ecwmxgbp*.fits files to be cross correlated.
        If only one filename is defined, correlation is done against the two polarization beams.
    split_ccd : bool, optional
        Decides whether the CCD regions should each be individually cross correlated.
        (The default is True, which splits the spectrum up into its seperate CCD regions)
    cont_ord : int, optional
        The degree of a chebyshev to fit to the continuum.
        (The default is 11)
    cont_plot : bool, optional
        Decides whether or not the continuum fitting should be plotted
        (The default is False, so no continua plots are displayed)
    save_name : str, optional
        The name or directory to save the figure produced to.
        "." saves a default name to the current working. A default name is also used when save_name is a directory.
        (The default is None, I.E. The figure is not saved, only
```

```
        displayed)

    Other Parameters
    ----------------
    offset : int, optional
        The amount the spectrum is shifted, mainly to test the effect of the cross correlation
        (The default is 0, I.E. no offset introduced)
    **kwargs : dict
        keyword arguments. Allows for passing unpacked dictionary to the class constructor.

    See Also
    --------
    scipy.signal.correlate
        https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.correlate.

    """

    def __init__(
        self,
        data_dir: os.PathLike,
        fits_list: list[os.PathLike],
        split_ccd: bool = True,
        cont_ord: int = 11,
        cont_plot: bool = False,
        offset: int = 0,
        save_name: str = None,
        **kwargs
    ) -> None:
        # Defined from parameters
        self.data_dir = data_dir
        self.fits_list = fits_list
        self.split_ccd = split_ccd
        self.cont_ord = cont_ord
        self.cont_plot = cont_plot
        self.offset = offset
        self.save_name = save_name

        # Defined when processed
        self.spec = None # spec1, spec2
        self.wav = None # wav1, wav2
        self.bpm = None # bpm1, bpm2

        self.exts = 0
        self.ccds = 1
        self.bounds = None # bounds1, bounds2

        self.wavUnits = "$\AA$"

        # self.invert = False
        # self.wav1, self.spec1, self.bpm1 = self.checkLoad(in1)
        # self.wav2, self.spec2, self.bpm2 = self.checkLoad(in2, in1)

        # Move to process
        for target in self.fits_list:
            self.spec, self.wav, self.bpm = self.loadFile(target)

```

```
            self.exts = self.spec[0].shape[0]

            # Bounds shape [extensions, ccds, lower / upper bound]
            self.bounds = self.setBounds()

            if split_ccd:
                self.splitCCD()

        # self.bounds.append(np.array(
        #     [[[0, self.spec1[0].shape[-1]]], [[0, self.spec1[1].shape[-1]]]], dtype=int
        # ))
        # self.bounds.append(np.array(
        #     [[[0, self.spec2[0].shape[-1]]], [[0, self.spec2[1].shape[-1]]]], dtype=int
        # ))


        # self.exts = self.spec1.shape[0]
        # self.ccds = 1
        # Bounds shape [extensions, ccds, lower / upper bound]
        # self.bounds1 = np.array(
        #     [[[0, self.spec1[0].shape[-1]]], [[0, self.spec1[1].shape[-1]]]], dtype=int
        # )
        # self.bounds2 = np.array(
        #     [[[0, self.spec2[0].shape[-1]]], [[0, self.spec2[1].shape[-1]]]], dtype=int
        # )
        if split_ccd:
            self.splitCCD()

        if cont_ord > 0:
            self.rmvCont(cont_plot)

        # Add an offset to the spectra to test cross correlation
        self.spec1 = np.insert(
            self.spec1, [0] * offset, self.spec1[:, :offset], axis=-1
        )[:, : self.spec1.shape[-1]]

        self.corrdb = []
        self.lagsdb = []
        # self.correlate()
        # self.checkPlot()

        return

    def loadFile(self, filename: os.PathLike) -> tuple[list, list, list]:
        spec, wav, bpm = None, None, None

        # Open HDU
        with pyfits.open(self.data_dir / filename) as hdu:
            #Load spec, wav, and bpm data - indexing [wav, intensity, beam]
            spec = hdu["SCI"].data.sum(axis=1)
            wav  = (
                np.arange(spec.shape[-1]) * hdu["SCI"].header["CDELT1"]
```

```
                + hdu["SCI"].header["CRVAL1"]
            )
            bpm = hdu["BPM"].data.sum(axis=1)

            # Check wavelength units unchanged
            if "Angstroms" not in hdu["SCI"].header["CTYPE1"]:
                self.wavUnits = hdu["SCI"].header["CTYPE1"]

        # TODO@JustinotherGitter: Recheck return of o and e beams.
        return ([spec, spec[::-1]], [wav, wav], [bpm, bpm[::-1]])

    def setBounds(self) -> list[np.ndarray, np.ndarray]:
        bounds = []
        bounds.append(np.array(
            [[[0, self.spec[0].shape[-1]]], [[0, self.spec[1].shape[-1]]]], dtype=int
        ))
        bounds.append(np.array(
            [[[0, self.spec[0].shape[-1]]], [[0, self.spec[1].shape[-1]]]], dtype=int
        ))

        return bounds

    # def checkLoad(self, path1: str, path2: str = None) -> np.ndarray:

    #     # If the first path is invalid
    #     if (path1 == None) or (not os.path.isfile(os.path.expanduser(path1))):
    #         # And the second path is not defined, raise an error
    #         if path2 == None:
    #             raise FileNotFoundError(f"{path1} is invalid")

    #         # Use the second path but swap the O and E beams
    #         path1 = path2
    #         self.invert = True

    #     # Load data
    #     with pyfits.open(os.path.expanduser(path1)) as hdu:
    #         spec = hdu["SCI"].data.sum(axis=1)
    #         wav = (
    #             np.arange(spec.shape[-1]) * hdu["SCI"].header["CDELT1"]
    #             + hdu["SCI"].header["CRVAL1"]
    #         )
    #         bpm = hdu["BPM"].data.sum(axis=1)

    #         if "Angstroms" not in hdu["SCI"].header["CTYPE1"]:
    #             self.wavUnits = hdu["SCI"].header["CTYPE1"]

    #     # Return data and implement swap if necessary
    #     return (wav, spec[::-1], bpm[::-1]) if self.invert else (wav, spec, bpm)

    def splitCCD(self) -> None:
        # Assumed BPM has a value of 2 near the center of each CCD (i.e. sum(bpm == 2) = count(ccd))
        self.ccds = sum(self.bpm1[0] == 2)
```

```python

        # update bounds to reflect ccds
        self.bounds1 = np.zeros([self.exts, self.ccds, 2], dtype=int)
        self.bounds2 = np.zeros([self.exts, self.ccds, 2], dtype=int)

        # Get lower and upper bound for each ccd, save to bounds
        for ext, ccd in iters.product(range(self.exts),
        ↪ range(self.ccds)):
            mid1 = np.where(self.bpm1[ext] == 2)[0][ccd]
            mid2 = np.where(self.bpm2[ext] == 2)[0][ccd]

            # Lower bound, min non-zero
            lowb1 = max(mid1 - self.bpm1.shape[-1] // (self.ccds * 2),
            ↪ 0)
            uppb1 = min(
                mid1 + self.bpm1.shape[-1] // (self.ccds * 2),
                ↪ self.bpm1.shape[-1]
            )

            # Upper bound, max bpm length
            lowb2 = max(mid2 - self.bpm2.shape[-1] // (self.ccds * 2),
            ↪ 0)
            uppb2 = min(
                mid2 + self.bpm2.shape[-1] // (self.ccds * 2),
                ↪ self.bpm2.shape[-1]
            )

            self.bounds1[ext, ccd] = (lowb1, uppb1)
            self.bounds2[ext, ccd] = (lowb2, uppb2)

    def rmvCont(self, plotCont) -> None:
        for ext, ccd in iters.product(range(self.exts),
        ↪ range(self.ccds)):
            # Get the range for current extension, ccd combination
            ccdBound1 = range(*self.bounds1[ext][ccd])
            ccdBound2 = range(*self.bounds2[ext][ccd])

            # Mask out the bad pixels for fitting continua
            okwav1 = np.where(self.bpm1[ext][ccdBound1] != 1)
            okwav2 = np.where(self.bpm2[ext][ccdBound2] != 1)

            # Define continua
            ctm1 = continuum(
                self.wav1[ccdBound1][okwav1],
                self.spec1[ext][ccdBound1][okwav1],
                deg=self.cont_ord,
                plot=plotCont,
            )
            ctm2 = continuum(
                self.wav2[ccdBound2][okwav2],
                self.spec2[ext][ccdBound2][okwav2],
                deg=self.cont_ord,
                plot=plotCont,
            )

            # Normalise spectra
            self.spec1[ext][ccdBound1] /=
            ↪ chebyshev.chebval(self.wav1[ccdBound1], ctm1)
```

```
            self.spec1[ext][ccdBound1] -= 1

            self.spec2[ext][ccdBound2] /= chebyshev.chebval(self.wav2[ccdBound2], ctm2)
            self.spec2[ext][ccdBound2] -= 1

        return

    def correlate(self) -> None:
        for ext, ccd in iters.product(range(self.exts), range(self.ccds)):
            # Get the range for current extension, ccd combination
            ccdBound1 = range(*self.bounds1[ext][ccd])
            ccdBound2 = range(*self.bounds2[ext][ccd])

            # Add rows/cols for correlation and lags data
            if len(self.corrdb) <= ext:
                self.corrdb.append([])
                self.lagsdb.append([])
            if len(self.corrdb[ext]) <= ccd:
                self.corrdb[ext].append([])
                self.lagsdb[ext].append([])

            # Invert BPM (and account for 2 in BPM) to zero bad pixels
            sig1 = self.spec1[ext][ccdBound1] * abs(self.bpm1[ext][ccdBound1] * -1 + 1)
            sig2 = self.spec2[ext][ccdBound2] * abs(self.bpm2[ext][ccdBound2] * -1 + 1)
            print(self.wav1[ccdBound1][0], self.wav2[ccdBound2][0])

            # Finally(!!!) cross correlate signals
            corr = signal.correlate(sig1, sig2)
            corr /= np.max(corr)  # Scales array so that the maximum correlation is at 1
            lags = signal.correlation_lags(sig1.shape[-1], sig2.shape[-1])

            self.corrdb[ext][ccd] = corr
            self.lagsdb[ext][ccd] = lags

        return

    def checkPlot(self, default_name: str = CORR_SAVENAME) -> None:
        # Plot
        fig, axs = plt.subplots(3, 3, sharey="row")

        for ext, ccd in iters.product(range(self.exts), range(self.ccds)):
            # Add cross correlation to plots
            axs[0, ccd].plot(
                self.lagsdb[ext][ccd],
                self.corrdb[ext][ccd] * 100,
                label=f"Ext: {ext + 1}, max lag @ {self.lagsdb[ext][ccd][self.corrdb[ext][ccd].argmax()]}",
            )

            ccdBound1 = range(*self.bounds1[ext][ccd])
            ccdBound2 = range(*self.bounds2[ext][ccd])
```

```

            axs[ext + 1, ccd].plot(
                self.wav2[ccdBound2],
                self.spec2[ext][ccdBound2] *
                ↪ abs(self.bpm2[ext][ccdBound2] * -1 + 1),
                label="sig2",
            )
            axs[ext + 1, ccd].plot(
                self.wav1[ccdBound1],
                self.spec1[ext][ccdBound1] *
                ↪ abs(self.bpm1[ext][ccdBound1] * -1 + 1),
                label="sig1",
            )

        axs[0, 0].set_ylabel("Normalised Correlation\n(%)")
        for ax in axs[0, :]:
            ax.set_xlabel("Signal Lag")
        for i, ax in enumerate(axs[1:, 0]):
            ax.set_ylabel(f"Ext. {i + 1} - Norm. Intensity\n(Counts)")
        for ax in axs[-1, :]:
            ax.set_xlabel(f"Wavelength ({self.wavUnits})")
        for ax in axs.flatten():
            ax.legend()

        plt.tight_layout()
        plt.show()

        # Handle do not save
        if not self.save_name:
            return

        # Handle lazy save_name
        if self.save_name == ".":
            self.save_name = os.getcwd()

        # Handle save name directory, use a default name (overwrite
        ↪ with warning)
        if self.save_name[-1] == "/" or os.path.isdir(self.save_name):
            self.save_name += default_name
            print(
                f"Save name is a directory. Saving cross correlation
                ↪ results as {default_name}"
            )

        # Check save location valid
        save_dir =
        ↪ os.path.expanduser("/".join(self.save_name.split("/")[:-1]))
        if not os.path.isdir(save_dir):
            raise FileNotFoundError(f"The path ({save_dir}) does not
            ↪ exist")

        # Save
        if self.save_name != None:
            fig.savefig(fname=self.save_name)

        return

    def process(self) -> None:
```

```
        for target in self.fits_list:
            logging.debug(f"Processing {target}")
            # self.correlate(target)
            # self.checkPlot()

        return


def main(argv) -> None: # TODO@JustinotherGitter: Handle
↪ cross_correlate.py called directly
    return

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

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

```python
"""Module for analyzing the sky lines of a wavelength calibrated
↪ image."""

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from __main__ import __author__, __email__, __version__

import os
import sys

import numpy as np
import matplotlib.pyplot as plt
from astropy.io import fits as pyfits
from scipy import signal

# plt.rcParams['figure.figsize'] = (20, 4)
plt.rcParams['image.origin'] = 'lower'


class Skylines:
    """
        Skylines class takes a

        Parameters
        ----------


        Returns
        -------


        Raises
        ------

    """
    def __init__(self,
        in1 : str,
        **kwargs,
    ) -> None:
        self.rawWav, self.rawSpec, self.rawBpm = self.checkLoad(in1)
        self.corrWav, self.corrSpec = self.transform(
            self.rawWav,
            self.rawSpec
        )
        self.spec = np.median(self.corrSpec, axis=1)
        self.normSpec = self.rmvCont(self.spec)

    def checkLoad(self, path1 : str) -> np.ndarray:
        # If the path is invalid
        if not os.path.isfile(os.path.expanduser(path1)):
            # Raise File Not Found Error
            raise FileNotFoundError(f"{path1} is invalid")

        # Load data
        with pyfits.open(os.path.expanduser(path1)) as hdul:
```

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

            # Return data
            return spec2D, wav2D, bpm2D

        def transform(wav_sol: np.ndarray, spec: np.ndarray) -> np.ndarray:
            # Create arrays to return
            cw = np.zeros_like(wav_sol)
            cs = np.zeros_like(wav_sol)

            exts = cw.shape[0]
            rows = cw.shape[1]

            for ext in range(exts):
                # Get middle row (to interpolate the rest of the rows to)
                avg_max = [np.where(
                    spec[ext][:, col] == spec[ext][:, col].max()
                )[0][0] for col in range(spec[ext].shape[1])]
                avg_max = np.sum(avg_max) // spec[ext].shape[1]

                # Get wavelength values at row with most trace
                wav = wav_sol[ext][avg_max, :]

                # Correct extensions based on wavelength
                # Wavelength ext
                cw[ext][:, :] = wav

                for row in range(rows):
                    # Spec extension
                    cs[ext][row, :] = np.interp(
                        wav,
                        wav_sol[ext][row, :],
                        spec[ext][row, :]
                    )

            return cw, cs

        def rmvCont(self):

            return self.spec / self.cont - 1

        def skylines(self,) -> None:
            pass


def main(argv) -> None:
    return

if __name__ == "__main__":
    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.

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, Darragh 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.

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.

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.

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. 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.

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=GnAFAAAAQAAJ.

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=1aAOdzK3FegC.

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.

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.

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. Bershady, 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
POLSALT	Polarimetric reductions for SALT
STOPS	Supplementary Tools for POLSALT Spectropolarimetry
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
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