Supplementary wavelength calibration methods for SALT/RSS spectropolarimetric observations

Justin Cooper

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: July 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: STOPS, POLSALT, IRAF, SALT, RSS, Development: Python, Pipeline, Calibration: wavelength, Polarization: optical, galaxies: AGN, Blazars, Spectropolarimetry, Astrophysics, Astronomy,

TODO:

- **Add Keywords $\rightarrow$ look up the astronomy journal keywords**
- **Look up keywords for pipeline development and data reduction.**

Acknowledgements

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

- **TODO: Add acknowledgements!**

Contents

List of Acronyms and Symbols

A-DC	Analog-to-Digital Converter
ADC	Atmospheric Dispersion Compensator
Ar	Argon
BPM	Bad Pixel Map
CCD	Charged-Coupled Device
CLI	Command Line Interface
CMOS	Complementary Metal-Oxide-Semiconductor
CuAr	Copper-Argon
FITS	Flexible Image Transport System
FWHM	Full Width at Half Maximum
GUI	Graphical User Interface
HDU	Header Data Unit
HET	Hobby-Eberly Telescope
HgAr	Mercury-Argon
HRS	High Resolution Spectrograph
IRAF	*Image Reduction and Analysis Facility*
L+45°	Linear +45° Polarized
L−45°	Linear −45° Polarized
LCP	Left Circularly Polarized
LHP	Linear Horizontally Polarized
LVP	Linear Vertically Polarized
Ne	Neon-Argon
NIR	Near Infra-Red
NIRWALS	Near Infra-Red Washburn Labs Spectrograph
POLSALT	*Polarimetric reductions for SALT*
RCP	Right Circularly Polarized
RMS	Root Mean Square
RSS	Robert Stobie Spectrograph
S/N	Signal-to-Noise Ratio
SAAO	South African Astronomical Observatory
SAC	Spherical Aberration Corrector
SALT	Southern African Large Telescope
SALTICAM	SALT Imaging Camera
STOPS	*Supplementary Tools for POLSALT Spectropolarimetry*
ThAr	Thorium-Argon
UV	Ultraviolet

VPH	Volume Phase Holographic
Xe	Xenon

Chapter 1

Introduction

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

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

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

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

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

TODO: General layout of Dissertation

Chapter 2

Spectropolarimetry and the SALT RSS

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

2.1 Spectroscopy

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

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

2.1.1 Telescope Optics

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

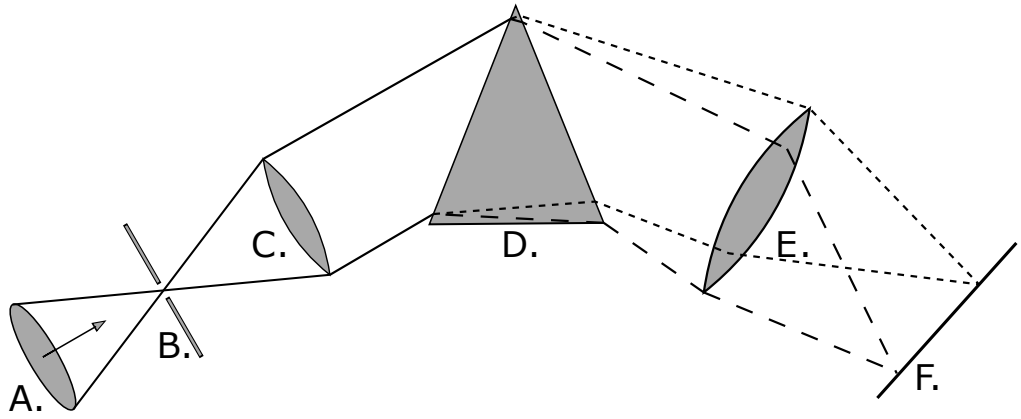


Figure 2.1: Layout depicting the light path through a spectrometer. Diagram adapted from Birney et al. (2006).

2.1.2 Slit

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

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

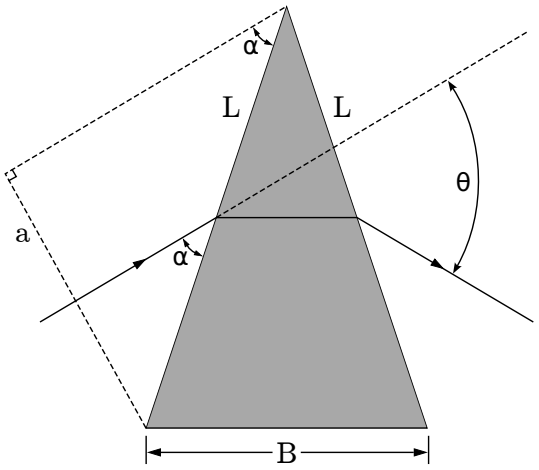


Figure 2.2: Geometry of a prism refracting an incident monochromatic beam at a minimum deviation angle. Diagram adapted from Birney et al. (2006).

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,

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

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]

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

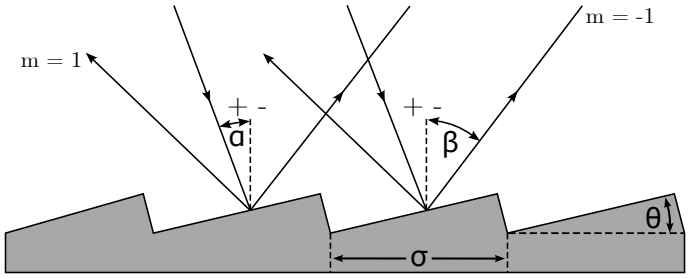


Figure 2.3: Geometry of a reflective blazed grating refracting an incident monochromatic beam. Diagram adapted from Birney et al. (2006).

Diffraction Grating

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

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

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,

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

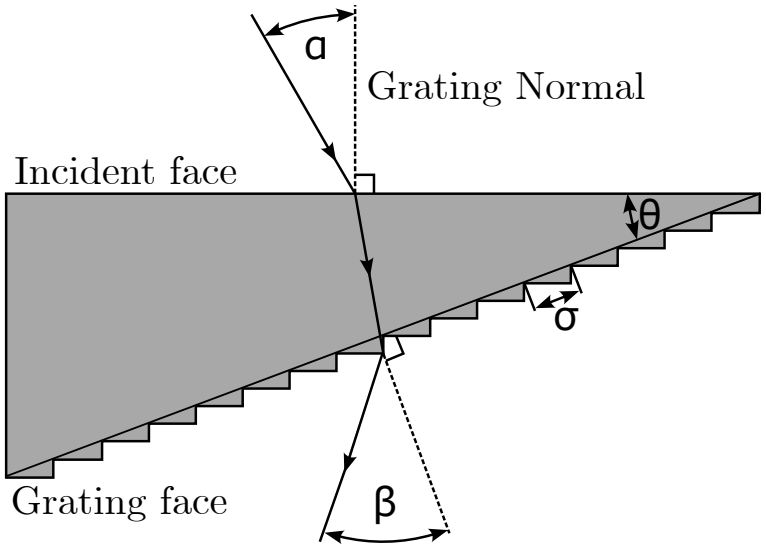


Figure 2.4: Diagram of a grism for an incident monochromatic beam of light and a diffracted beam of order $m = 1$. Diagram adapted from Birney et al. (2006).

Substituting m/σ with the grating equation results in

$$\begin{aligned} \frac{\partial \beta}{\partial \lambda} &= \frac{\sin(\alpha) + \sin(\beta)}{\lambda \cos(\beta)} \\ &\propto \lambda^{-1} . \end{aligned} \tag{2.8}$$

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

Alternate Diffraction Elements

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

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

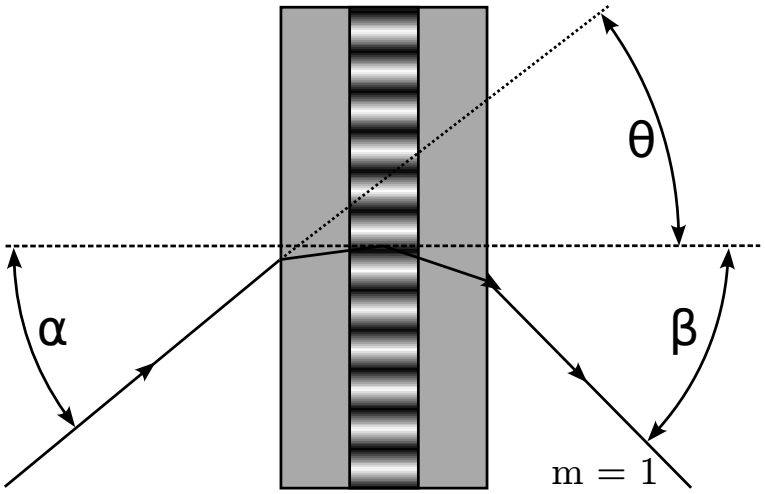


Figure 2.5: Diagram of a VPH grating for an incident monochromatic beam of light. Diagram adapted from Birney et al. (2006).

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

2.1.8 Detector and Spectroscopic Calibrations

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

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

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

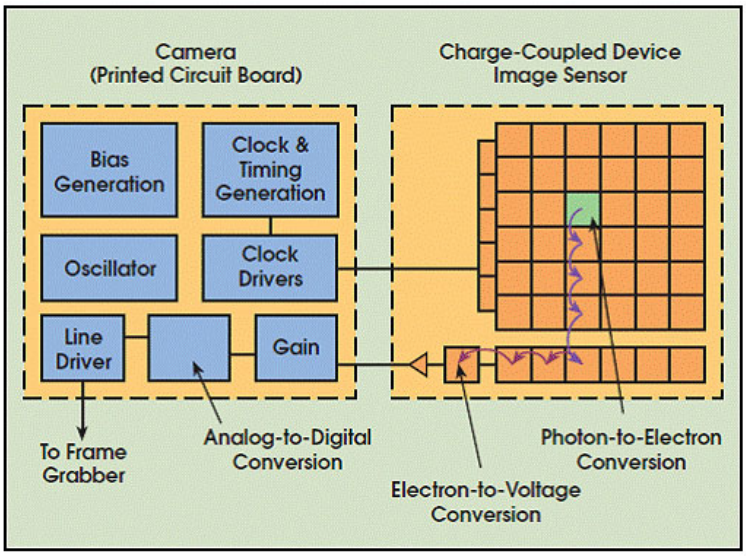


Figure 2.6: Diagram of the inner logic of a CCD. Figure adapted from Litwiller (2001).

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 of 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.

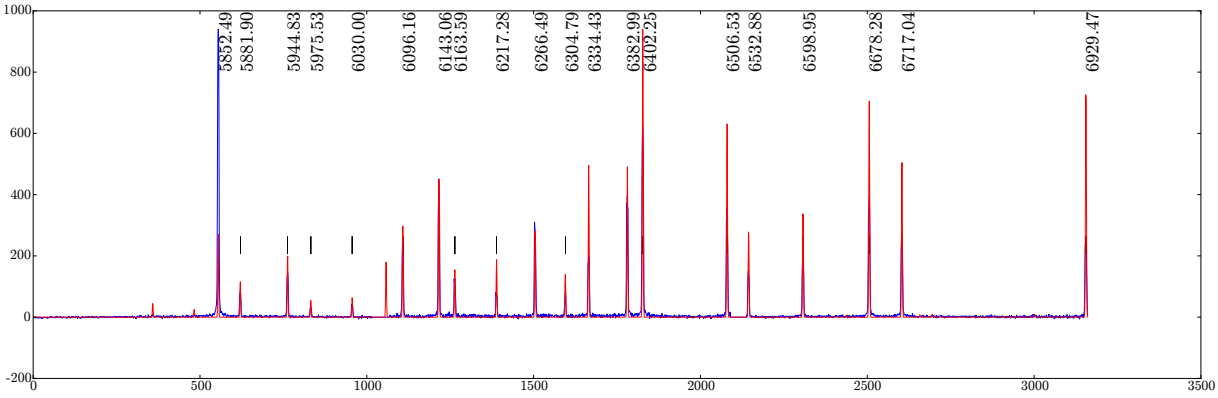


Figure 2.7: Example of an arc spectrum for NeAr taken with SALT's RSS using the PG1800 grating at a grating angle of 34.625°, an articulation angle of 69.258°, and covering a wavelength range of $\sim 5600 - 6900$ Å. Plot adapted from SALT's published Longslit Line Atlases, (2023).[2]

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

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

where $F_\lambda(x, y)$ is the non-corrected flat image, $B(x, y)$ is the bias image, $D(x, y)$ is the dark image which is scaled by the exposure time of the science image, t_S, and the dark image, t_D. 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:

$$S^*_\lambda(x,y) = \frac{S_\lambda(x,y) - B(x,y) - (\frac{t_S}{t_D})D(x,y)}{F^n_\lambda(x,y)}\,. \tag{2.10}$$

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

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

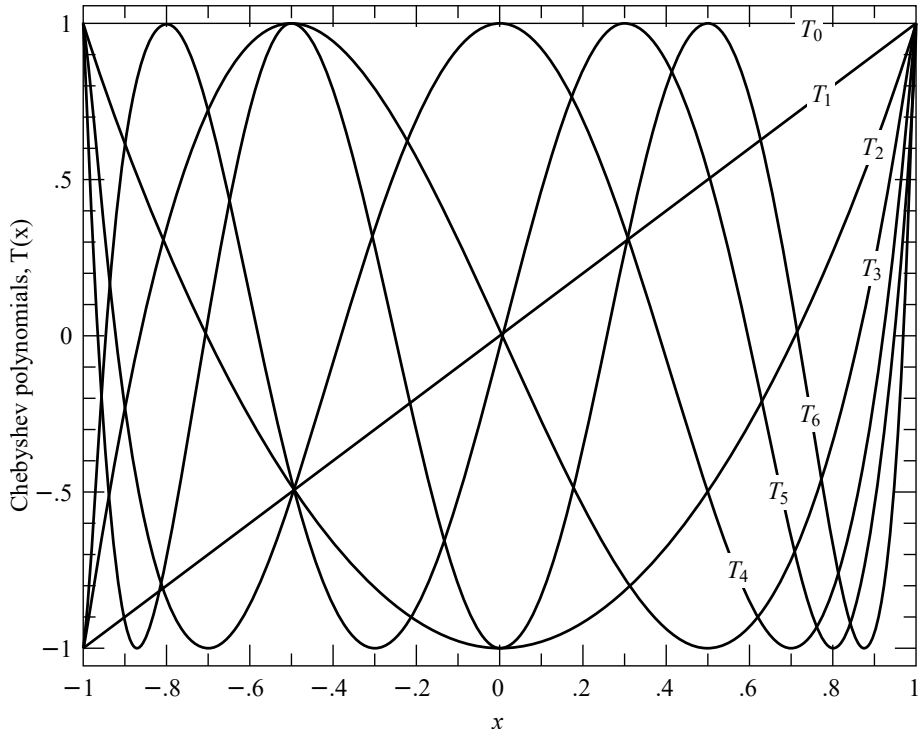


Figure 2.8: The first seven Chebyshev polynomials (T_0 through T_6) as defined by Equation 2.12 over the region $[-1, 1]$ for which they are orthogonal. Plot adapted from (Press et al., 2007) (2023)[3]

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.

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

Chebyshev Polynomials The Chebyshev polynomials are defined explicitly as:

$$T_n(x) = \cos(n \cos^{-1}(x)) \,, \tag{2.11}$$

or recursively as:

$$\begin{aligned} T_0(x) &= 1 \,, \\ T_1(x) &= x \,, \text{ and} \\ T_{n+1}(x) &= 2xT_n(x) - T_{n-1}(x) \,, \text{ for } n \geq 1 \,, \end{aligned} \tag{2.12}$$

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:

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

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

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

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

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

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

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

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

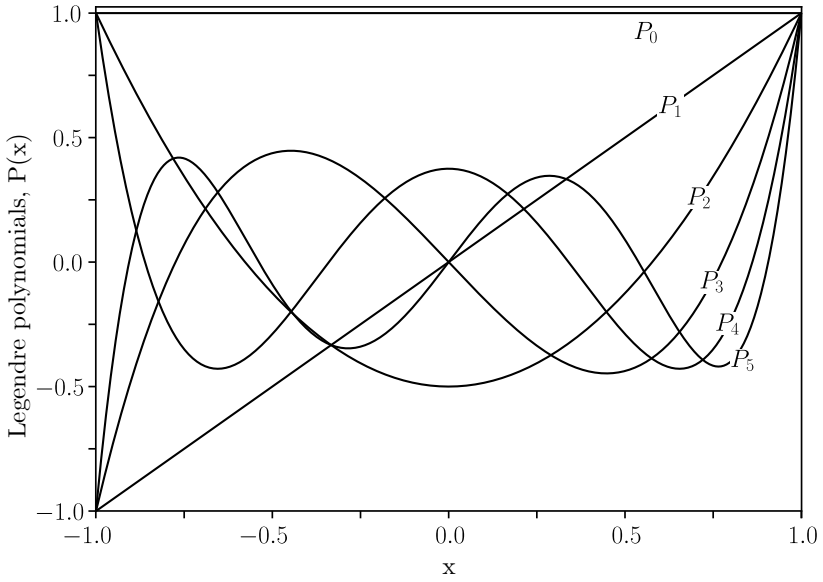


Figure 2.9: The first six Legendre polynomials (P_0 through P_5) as defined by Equation 2.20 over the region $[-1, 1]$ for which they are orthogonal. Plot adapted from Geek3, CC BY-SA 3.0, via Wikimedia Commons (2023).

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

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2 - 1)^n \,, \tag{2.17}$$

or recursively as:

$$\begin{aligned} P_0(x) &= 1 \,, \\ P_1(x) &= x \,, \text{ and} \\ (n+1)P_{n+1}(x) &= (2n+1)xP_n(x) - nP_{n-1}(x) \,, \text{ for } n \geq 1 \,, \end{aligned} \tag{2.18}$$

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:

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

where a weight of 1 is the weighting factor for Legendre polynomials (Dahlquist and Björck, 2003; Press et al., 2007). A Legendre approximation of an unknown wavelength calibration function is given by:

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

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

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

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

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

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

where s and p are the two polarized components of light perpendicular to one another, Z_1 and Z_2 are the impedance of the two media, and 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.

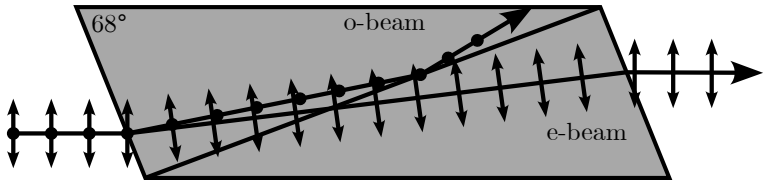


Figure 2.10: Diagram of a Nicol prism for incident non-polarized light. Diagram adapted from Fred the Oyster, CC BY-SA 4.0, via Wikimedia Commons (2023).

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

2.2.1 Polarization

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

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

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

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

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 2.24 using complex values allows us to simplify the form of the solution to:

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

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

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

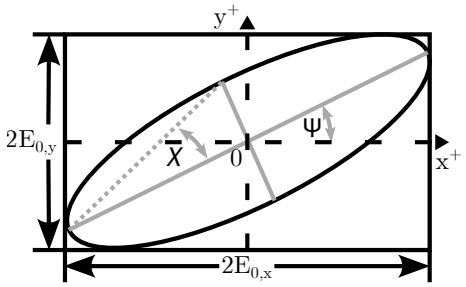


Figure 2.11: The polarization ellipse for an electric field vector propagating through free space. Diagram adapted from Inductiveload, PDM 1.0, via Wikimedia Commons (2023).

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

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

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

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 2.27 over the exposure time is necessary. Given the periodical nature and high frequencies of the fields, the time averaging may be found over a single oscillation using:

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

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

$$(E_{0x}^2 + E_{0y}^2)^2 - (E_{0x}^2 - E_{0y}^2)^2 - (2E_xE_y\cos\Phi)^2 = (2E_xE_y\sin\Phi)^2\,. \tag{2.29}$$

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

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

where S_0 to S_3 are referred to as the Stokes (polarization) parameters. The parameters describe the: S_0, total intensity (often normalized to 1); S_1, ratio of the Linear Horizontally Polarized (LHP) to Linear Vertically Polarized (LVP) light; S_2, ratio of the Linear $+45°$ 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)

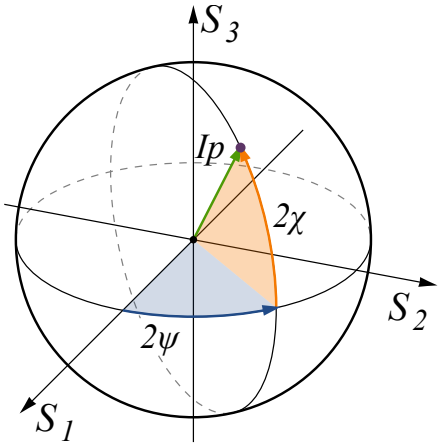


Figure 2.12: The Poincaré sphere describing the polarization properties of a wave-packet propagating through free space. Diagram adapted from Inductiveload, PDM 1.0, via Wikimedia Commons (2023).

(counter-clockwise) light. When the intensity is normalized, the Stokes parameters range from 1 to -1, based on the dominating component of the parameter (Chandrasekhar, 1950; Stokes, 1852).

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

$$I^2 = Q^2 + U^2 + V^2 \,, \tag{2.31}$$

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

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

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

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

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

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

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

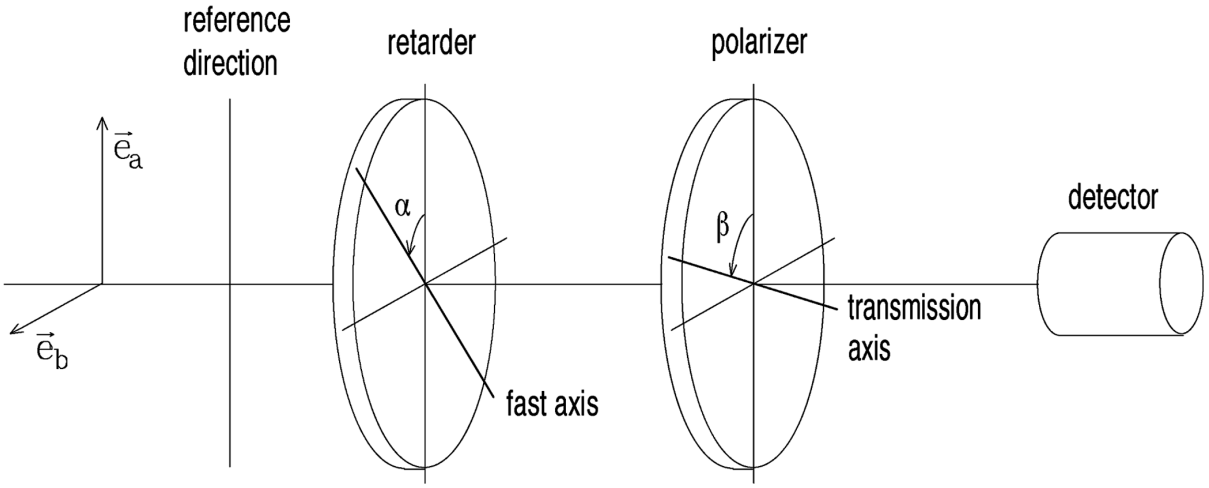


Figure 2.13: A diagram of an ideal polarimeter. Diagram adapted from Degl'Innocenti and Landolfi (2004).

2.2.2 Polarization Measurement

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

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

$$\mathbf{S}' = \mathbf{M}_N \ldots \mathbf{M}_0 \mathbf{S} \,. \tag{2.35}$$

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

General Rotation The Mueller matrix for coordinate space rotations about the origin by an angle θ,

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

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,

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

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

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

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

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

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

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

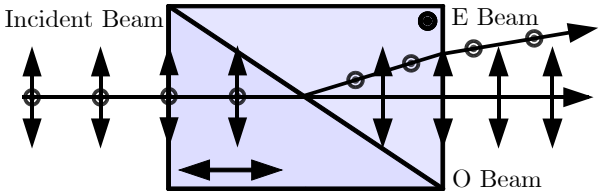


Figure 2.14: Diagram of a Rochon prism. Included in the diagram are the optical axes of the differing sections of the birefringent material as well as polarizing directions of the incident beam, denoted using the ↔ and ⊙ symbols, for the *O*- and *E*-beams, respectively. Figure adapted from ChrisHodgesUK,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

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:

$$\gamma = \frac{2\pi\Delta n L}{\lambda_0} \tag{2.40}$$

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

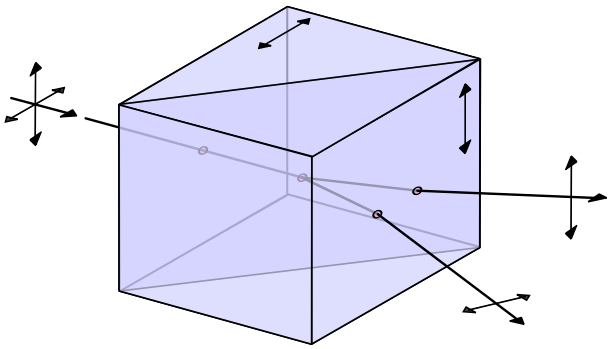


Figure 2.15: Diagram of a Wollaston prism. Included in the diagram are the optical axes of the differing sections of the birefringent material as well as polarizing directions of the incident beam, denoted using the ↔ and ↕ symbols, for the *O*- and *E*-beams, respectively. Diagram adapted from fgalore,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

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. 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 and § 2.2).

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

Along with common points of consideration when developing any instrumentation for observational astronomy, such as resolution and sensitivity, spectropolarimeters need also consider the spectral response of the polarimetric components as well as the polarization

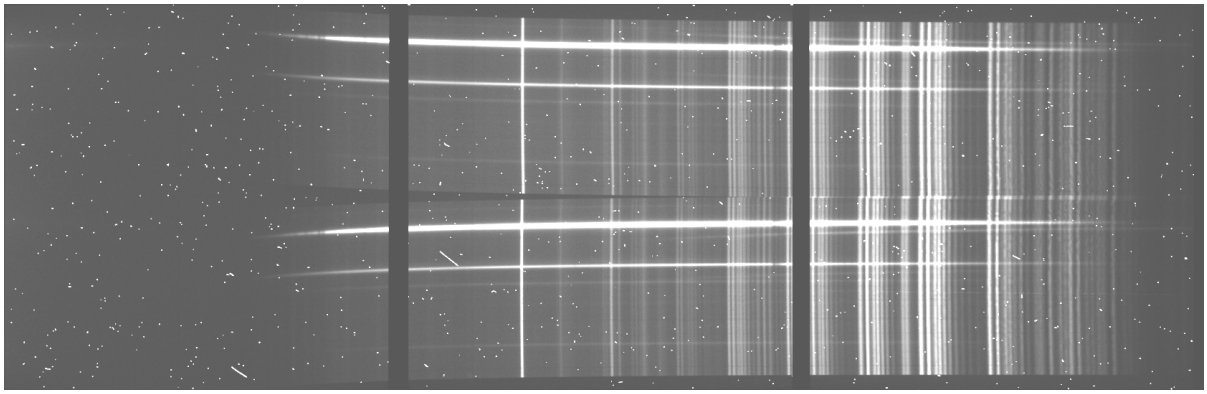


Figure 2.16: A spectropolarimetric target exposure as observed by the SALT RSS in spectropolarimetry mode.

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

2.3.1 Spectropolarimetric Measurement

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

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

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

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

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

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

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

Next, the $Q(\lambda)$ and $U(\lambda)$ Stokes parameters are found by first defining the normalized difference in relative intensities, $F_i(\lambda)$, as:

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

which allows Equation 2.41 to be written, as

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

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

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

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

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

2.3.2 Spectropolarimetric Calibrations

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

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

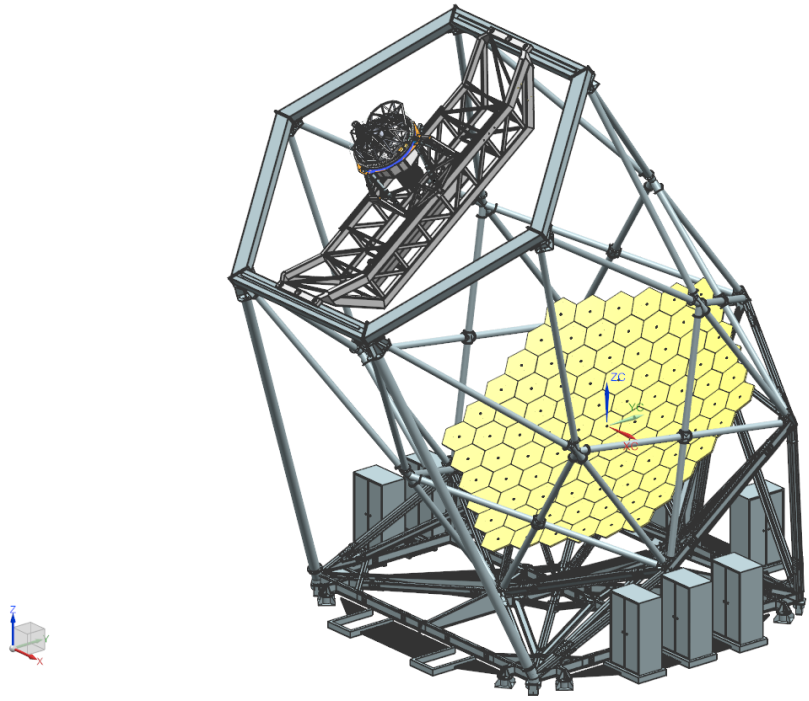


Figure 2.17: The tracker, supporting structure, and primary mirror of SALT. Figure adapted from the SALT call for proposals (2022).[5]

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 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).

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

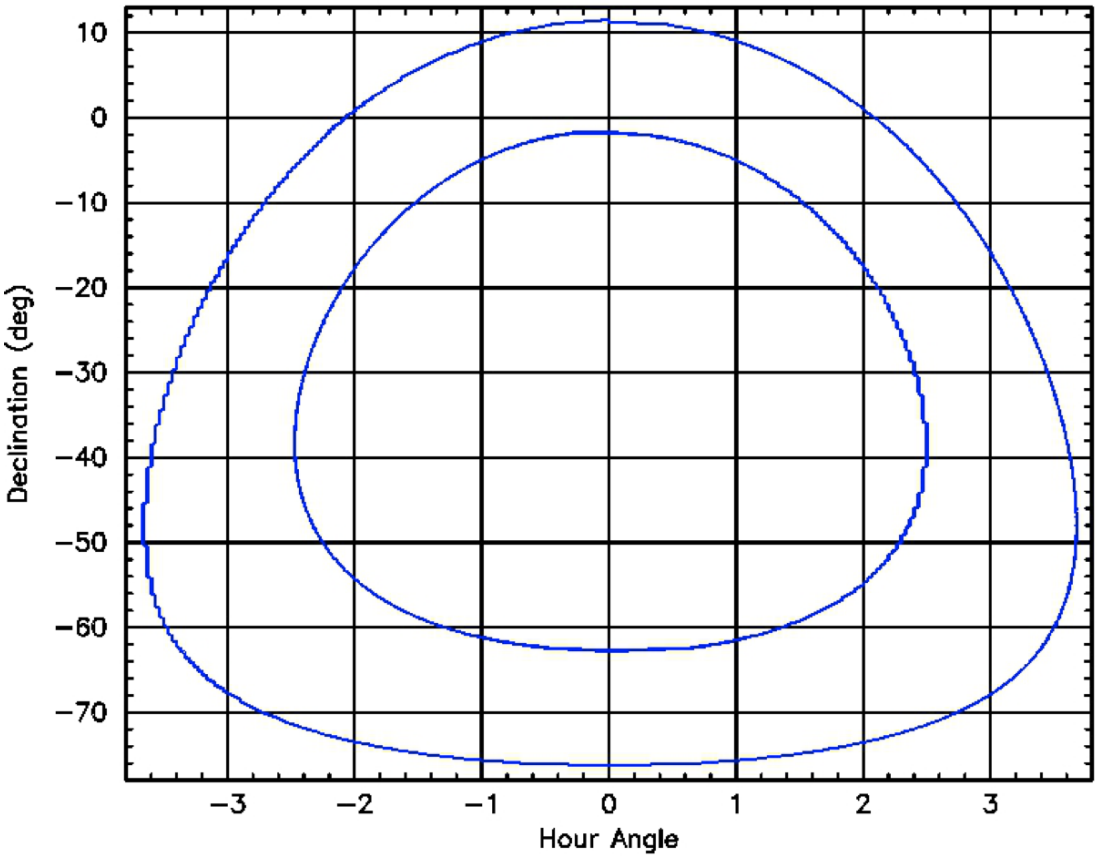


Figure 2.18: The visibility annulus of objects observable by SALT. Figure adapted from the SALT call for proposals (2013).[6]

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

The tracker is equipped with a Spherical Aberration Corrector (SAC) (O'Donoghue, 2000), and an Atmospheric Dispersion Compensator (ADC) (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. The tracker also houses the calibration system which contains the Ar, CuAr, HgAr, Ne, ThAr, and Xe wavelength calibration lamps (Buckley et al., 2008).

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

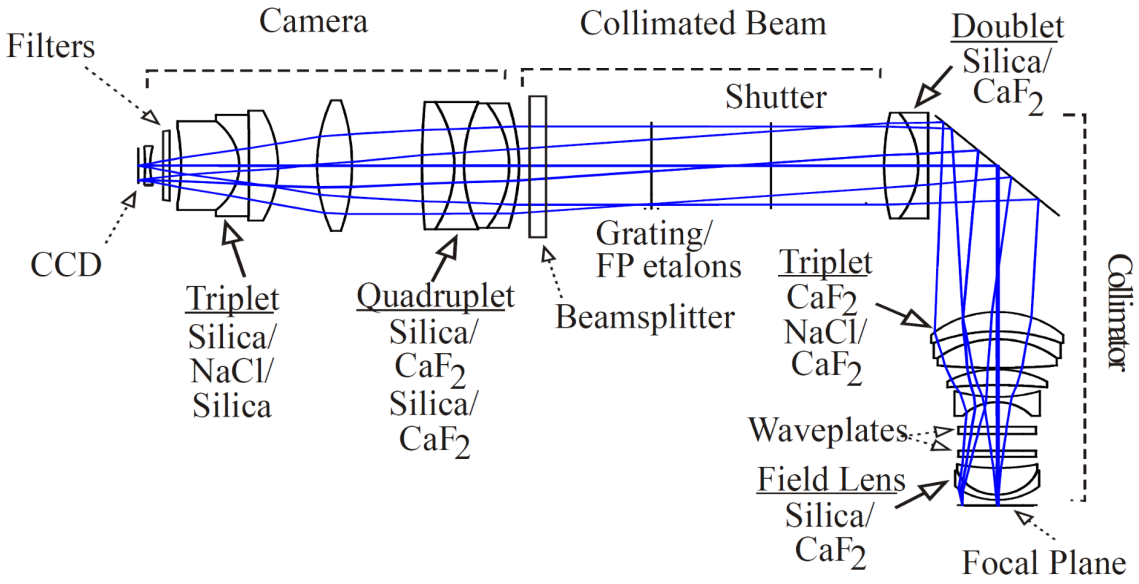


Figure 2.19: The optical path of the SALT RSS. Figure adapted from the SALT call for proposals (2023).[7]

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.

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.

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

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).

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 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 sourced 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 *Polarimetric reductions for SALT* (POLSALT) and the limitations faced during POLSALT wavelength calibrations (§ 3.1), a brief overview of the *Image Reduction and Analysis Facility* (IRAF) tasks relevant for spectropolarimetric wavelength calibrations (§ 3.2), and an overview of *Supplementary Tools for POLSALT Spectropolarimetry* (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 A, is included (§ 3.4).

3.1 POLSALT - *Polarimetric reductions for SALT*

The POLSALT (*Polarimetric reductions for SALT*) pipeline is the official reduction pipeline for spectropolarimetric data taken using the SALT RSS.[1] The newest version of the software, aptly named the 'beta version' ('version' 23 January 2020), was the version adapted in this study. It includes a GUI, depicted in Figure 3.1, which allows for limited interactivity during key steps in the reduction process.[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 at each step are crucial for accurate results and are thus briefly discussed below. Further details for the reduction process may be found at the POLSALT GitHub wiki.[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`.

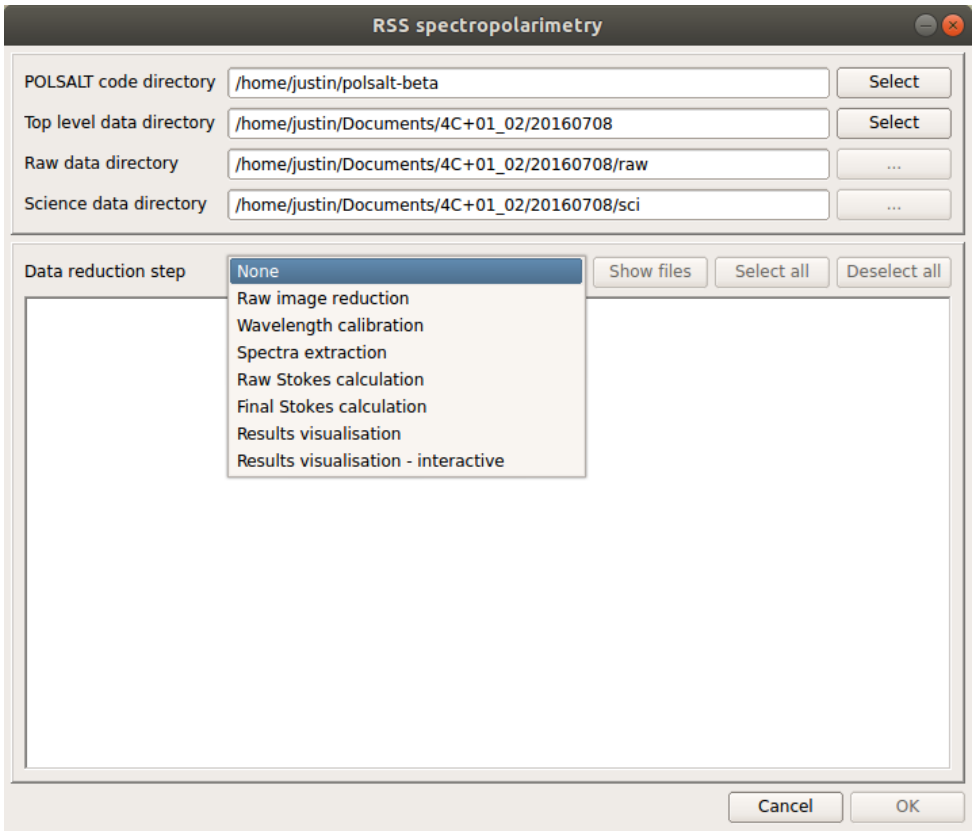


Figure 3.1: The layout of the POLSALT Graphical User Interface (GUI), including the contents of the reduction steps accessible via the dropdown menu. Note that there is no trailing forward slash after the 'Top level data directory'. Figure created from a local instance of the POLSALT GUI.

3.1.1 Basic CCD Reductions

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

3.1.2 Wavelength Calibrations

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

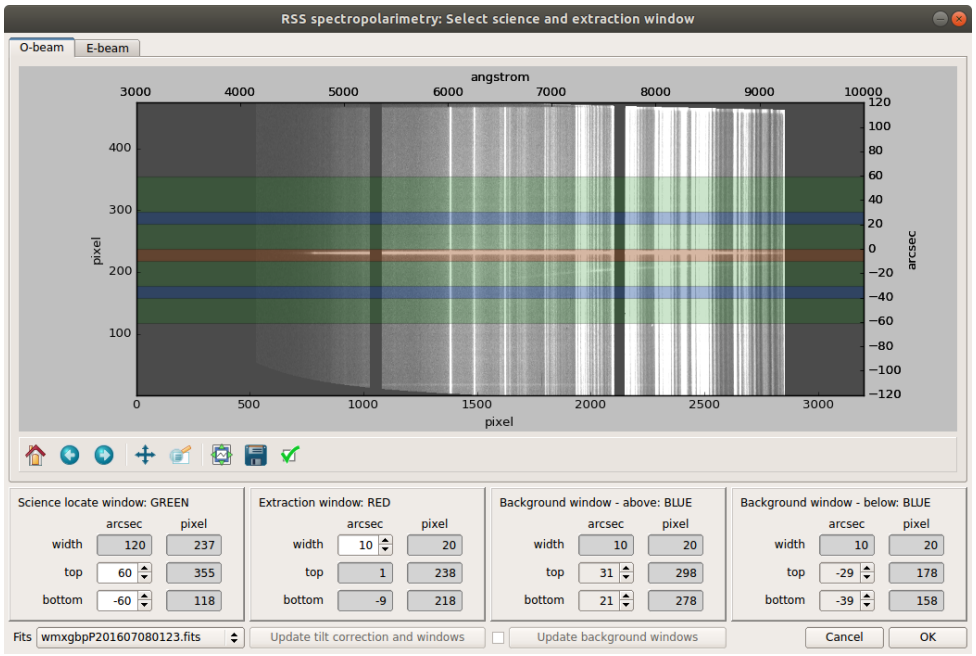


Figure 3.2: The layout of the interactive POLSALT `spectra extraction` GUI after selecting the 'update tilt correction and windows' button along the bottom border of the window. Figure created from a local instance of the POLSALT GUI.

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 extracts a one dimensional wavelength dependent spectrum for each beam sub-extension. This step is interactive with Figure 3.2 showing the interactive window used for spectral extraction. The user, using the brightest trace in the image as a reference, defines regions which span the wavelength axis which define the background and trace regions for the sky subtraction and spectral extraction. Files with background and geometric 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

The raw Stokes calculations are performed via **specpolrawstokes_dev.py** and identify waveplate pairs for which the intensity, I, and a 'raw Stokes' signal, S, are calculated as:

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

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

The raw Stokes signal is calculated as the normalized difference of the O- and E-beams, for a waveplate pair, taken perpendicular to one another. The created files contain the raw Stokes information and use a very specific naming style; most notably the indexes of the related waveplate pairs, from Table 2.2, are included in the file names.

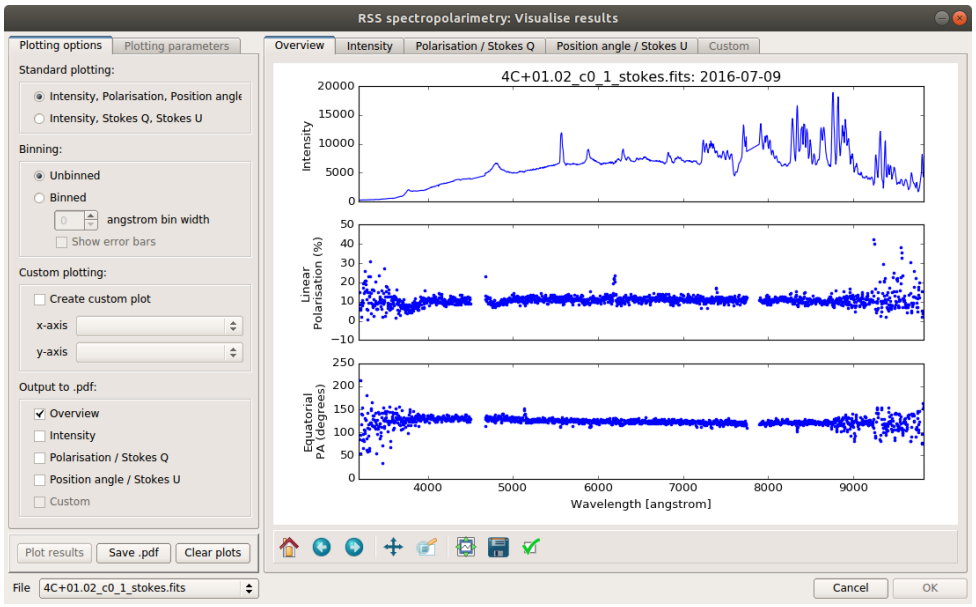


Figure 3.3: The layout of the interactive POLSALT `visualization` GUI after selecting the 'Plot results' button along the bottom border of the window. Figure created from a local instance of the POLSALT GUI.

3.1.5 Final Stokes Calculations

The Final Stokes calculations are performed via **specpolfinalstokes.py** and, using the waveplate pattern along with the raw Stokes signals, calibrates for the polarimetric zero-point and waveplate efficiency, and calculates the final Stokes parameters. Before the final Stokes calculations are performed, and if a sufficient number of redundant exposures were taken, the raw Stokes signals are culled to eliminate outlier signals which may arise from, for example, temporary atmospheric conditions affecting the signal. The culling is performed by comparing observation cycles against one another, comparing the deviation of the signal means which estimate the baseline systematic polarization fluctuations (due to imperfections in repeatability), and performing a χ^2 analysis to eliminate any statistical outliers.

3.1.6 Visualization

Plotting the results of the spectropolarimetric reduction process is done using **specpolview.py**, which 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 allowing the user control over various options, such as the wavelength range, binning, etc., with the GUI shown in Figure 3.3.

3.1.7 Post-Processing Analysis

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

Flux-calibrations are performed via **specpolflux.py** and are only intended for shape corrections of the spectrum. Additionally, a flux database file must exist for the observed

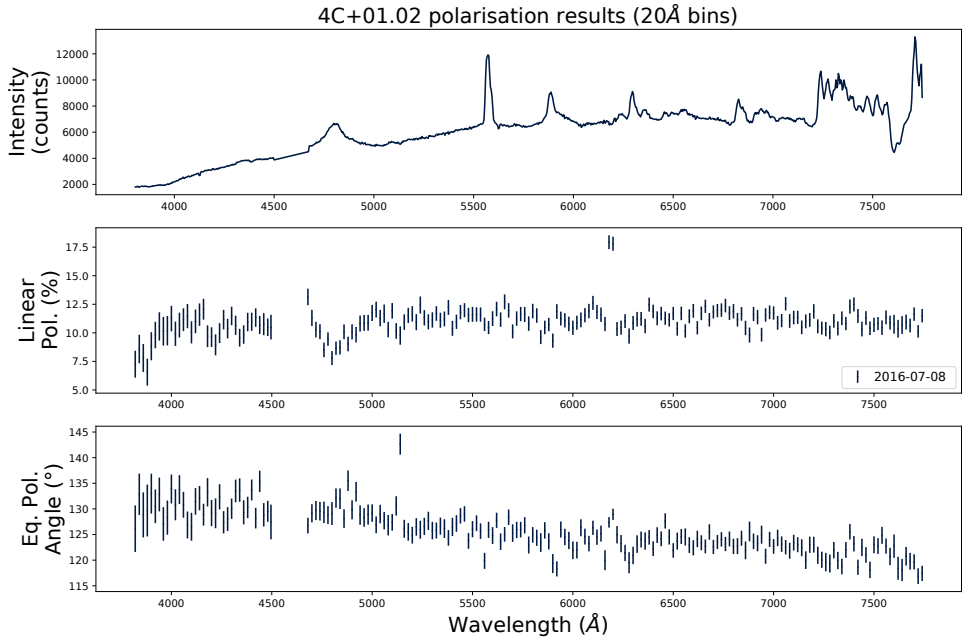


Figure 3.4: A typical plot resulting from the reduction process. Figure adapted from (Cooper et al., 2022).

standard and must be included in the working science directory.

Synthetic filtering is calculated via **specpolfilter.py** and computes the synthetically filtered polarization results. Any wavelength dependent throughput filter curve may be synthesized when defined by the user, but a few pre-defined filter curves are available, namely: the SALTICAMs U, B, V, R, and I Johnson-Cousins filter curves.

3.1.8 POLSALT Limitations and the Need for Supplementary Tools

The creation of supplementary tools for POLSALT spectropolarimetric reductions stemmed from the limitations of the wavelength calibration process and a need to compare wavelength solutions across the perpendicular O and E polarization beams. The process of calibrating wavelength solutions using the POLSALT pipeline is time-consuming for the average user, and often results in unexpected program 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 anything larger than a handful of observations. This is particularly true for observations performed with the SALT PG0300 grating as the sparse spectral features of the Ar arc lamp are not handled well by the POLSALT pipeline.

Since PG0300 provided the widest wavelength range and highest throughput, it was almost exclusively used for observations of flaring blazars, resulting in a large backlog of unanalyzed data. The only arc available for the PG0300 grating with a close enough articulation and grating angle ($\sim 10.68°$ and $\sim 5.38°$, respectively), was the Ar arc lamp which displays sparse spectral features with large gaps over the wavelength range at these grating and articulation angles (Figure 3.5). This often led the POLSALT pipeline to create inconsistent wavelength solutions, or to fail to create a wavelength solution altogether, since minor deviations of identified spectral features resulted in large deviations in regions with no spectral features. To only further compound the difficulty of the wavelength

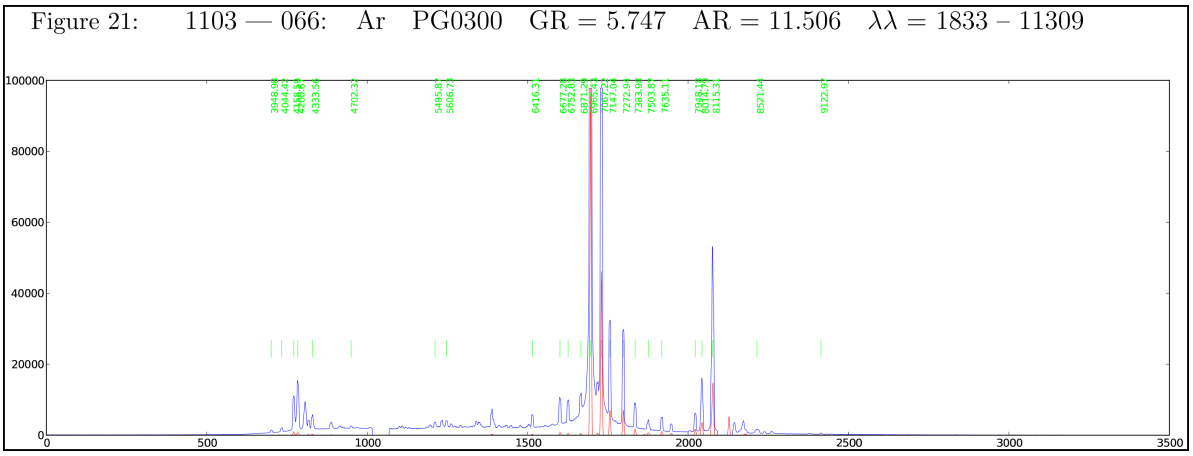


Figure 3.5: One of many Ar arc lamp spectra as provided by SALT for line identification. Plot adapted from SALT's published Longslit Line Atlases (as of 2024), resized to fit within the document margins but otherwise unchanged.[4]

calibrations, the spectrum of the Ar arc lamp contains a partial overlap of a higher order at longer wavelengths (§ 2.1.7, Equation 2.5).

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

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

3.2 IRAF - *Image Reduction and Analysis Facility*

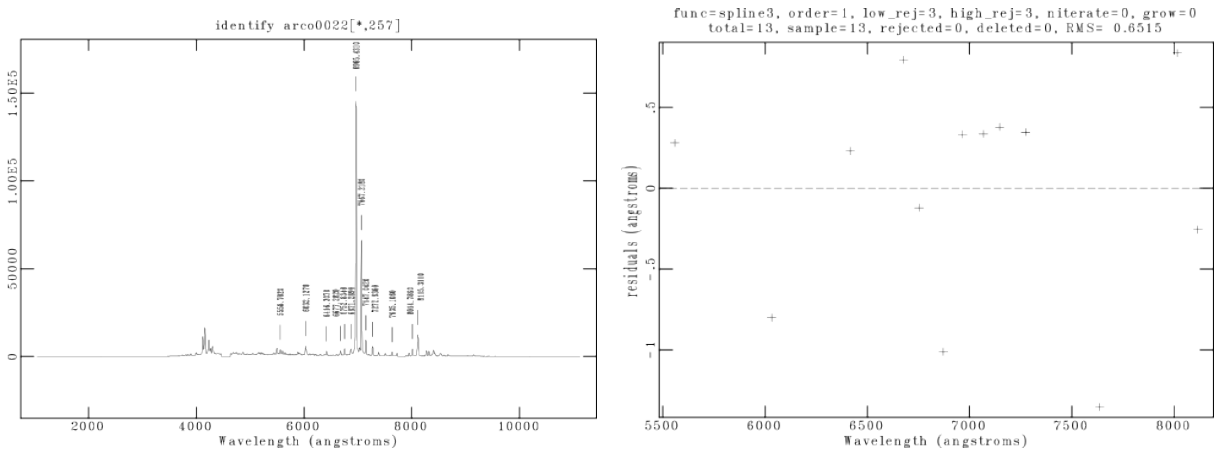
The IRAF (*Image Reduction and Analysis Facility*) software is a collection of software designed specifically for the reduction and analysis of astronomical images and spectra (Tody, 1986, 1993). The software consists of many tasks which perform specific operations and which are grouped into relevant packages. Only a brief overview of the tasks will be provided here. Help documentation for any of the IRAF tasks may be found online or through the IRAF Command Line Interface (CLI) through the `?` or `:.help` 'cursor commands' when running interactive tasks, with more specific help documentation provided in the relevant section.[5]

Useful IRAF tasks that deserve a brief mention are: the `mkscript` task in the `system` package which allows a user to create and save a task along with the defined parameters as a file which can later be called as a script,[6] the `implot` task in the `plot` package which

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

[5]Online help for IRAF is available at `https://iraf.net/irafdocs/`

[6]Help documentation for the `mkscript` task may be found at `https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/system.mkscript.html`.



(a) Identified features.

(b) Root Mean Square (RMS) of the identified features.

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

allows the rows or columns of an image to be interactively displayed,[7] and the `eparam` task in the `language` package which allows the parameters of a task to be edited within the IRAF CLI.[8]

For the wavelength calibration of SALT spectropolarimetric data, the relevant tasks are the `identify` and `reidentify` tasks located in the `noao.onedspec` package, and the `fitcoords` and the (optional) `transform` tasks located under the `noao.twodspec.longslit` package. These tasks produce a two-dimensional wavelength solution which must be obtained separately for the *O*- and *E*-beam.

3.2.1 Identify

The `identify` task is used to interactively determine a one-dimensional wavelength function across a chosen row of an arc exposure by identifying features in the spectrum with known wavelengths.[9] The task creates the first approximation of the wavelength solution (see Figure 3.6) as well as a local database in which the solution is saved (see Listing 3.1). The initial solution is built on in subsequent tasks, and it is, therefore, imperative that the initial solution is well-fit to minimize errors further along the calibration process.

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

[7]Help documentation for the `implot` task may be found at `https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/plot.implot.html`.

[8]Help documentation for the `eparam` task may be found at `https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/language.eparam.html`.

[9]Help documentation for the `identify` task may be found at `https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.onedspec.identify.html`.

[10]See also `https://iraf.net/irafdocs/formats/identify.php` for an explanation of the database contents.

Listing 3.1: An example of the `identify` database contents.[10]

```
# Thu 15:19:16 13-May-2021
begin   identify arc00057[*,237]
    id  arc00057
    task        identify
    image       arc00057[*,237]
    units       Angstroms
    features    35
              53.61 5944.74989   5944.834  13.0 1 1 15257
             140.19 6029.9793    6029.997  13.0 1 1  4652
             185.30 6074.34644   6074.338  13.0 1 1 13396
             207.49 6096.15873   6096.161  13.0 1 1 21700
             255.23 6143.0493    6143.063  13.0 1 1 33330
             276.13 6163.56995   6163.594  13.0 1 1 11344
             330.89 6217.29293   6217.281  13.0 1 1 13705
             381.10 6266.51524   6266.495  13.0 1 1 21747
             420.21 6304.8113    6304.789  13.0 1 1 10226
             450.49 6334.45415   6334.428  13.0 1 1 36235
             500.18 6383.04826   6382.991  13.0 1 1 35824
             519.85 6402.26802   6402.248  13.0 1 1 70163
             626.70 6506.56147   6506.528  13.0 1 1 46165
             653.73 6532.91083   6532.882  13.0 1 1 21413
             721.60 6598.98642   6598.953  13.0 1 1 26396
             803.21 6678.31069   6678.277  13.0 1 1 51338
             843.15 6717.0732    6717.043  13.0 1 1 36780
            1099.95 6965.36335   6965.431  13.0 1 1  5618.4  ar
            1169.57 7032.38598   7032.413  13.0 1 1 100000
            1317.05 7173.89814   7173.938  13.0 1 1  5000    decrease
            1391.52 7245.11148   7245.167  13.0 1 1 73545
            1537.20 7383.93022   7383.981  13.0 1 1  5557.5  ar
            1595.02 7438.83545   7438.898  13.0 1 1 15000    decrease
            1663.64 7503.86263   7503.869  13.0 1 1 30000    ar; increase
            1697.46 7535.84584   7535.774  13.0 1 1  8000    increase
            1802.64 7635.07335   7635.106  13.0 1 1 20000    ar; decrease
            2209.19 8014.79559   8014.786  13.0 1 1  3000    ar; decrease
            2604.58 8377.66137   8377.607  13.0 1 1 14543
            2734.54 8495.41423   8495.359  13.0 1 1  8765
            2763.48 8521.52355   8521.442  13.0 1 1  4537.5  ar
            2840.92 8591.20799   8591.258  13.0 1 1  2000    decrease
            2889.39 8634.67334   8634.647  13.0 1 1  3059
            2911.42 8654.39264   8654.383  13.0 1 1  3000    decrease
            2926.56 8667.93501   8667.944  13.0 1 1   702.5  ar
            3135.72 8853.77575   8853.867  13.0 1 1  1820
    function legendre
    order 4
    sample *
    naverage 1
    niterate 0
    low_reject 3.
    high_reject 3.
    grow 0.
    coefficients        8
        2.
        4.
        53.60757446289061
        3135.715576171875
        7425.420339270724
        1457.513831286474
        -26.15751926622308
        -3.000903509842187
```

3.2.2 Reidentify

The `reidentify` task is used to run the `identify` task autonomously and repeatedly across the entirety of the arc frame at defined (row) intervals.[11] The task uses the one-dimensional wavelength solution stored in the database created by the initial `identify` call and refits the positions of the relevant spectral features. The task may fail based on a number of conditions, most common of which is the loss of features as the task moves further from the row at which the user manually ran `identify`.

When running `reidentify` non-interactively, it is recommended to set the `verbose` parameter to '`yes`' as this will provide immediate confirmation if the task quit early. Regardless of whether the task quit successfully, the newly defined wavelength solutions are appended to the local database following the `identify` task database format, an example of which is given in Listing 3.1.

3.2.3 Fitcoords

The `fitcoords` task is used to find a two-dimensional surface function from the one-dimensional wavelength solutions found for specific rows in the previous steps.[12] The usage of `fitcoords` is similar to that of `identify` and consists of examining the distribution of identified points and eliminating any points that `reidentify` may have misidentified (see Figure 3.7).

By eliminating outliers with bad residuals and modifying the two-dimensional surface function's type and degree, the overall error of the fit is decreased, aligning more closely to what the 'true' wavelength solution is. This surface function is the final two-dimensional wavelength solution for each two-dimensional spectrum. It is saved using the `fitcoords` database format, an example of which is given in Listing 3.2, as the list of parameters and function coefficients required to recreate the closest two-dimensional model. The IRAF wavelength solution is used by the STOPS `join` method to create the wavelength extension required by POLSALT, further described in § 3.3.2.

3.2.4 Transform

The `transform` task is the optional final step in the IRAF wavelength calibration process.[14] Simply put, `transform` converts the (x_p, y_p) pixel units of an exposure to (λ, y_p) wavelength units which allows for an immediate check of whether the wavelength solution is consistent across the frame. Any general error in the wavelength solution may be spotted in the transformed images; ranging from minor errors, such as the arc lines or sky lines not being purely vertical across the frame, to more major errors, such as an incorrect wavelength solution skewing the exposure beyond recognition.

[11] Help documentation for the `reidentify` task may be found at `https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.onedspec.reidentify.html`.

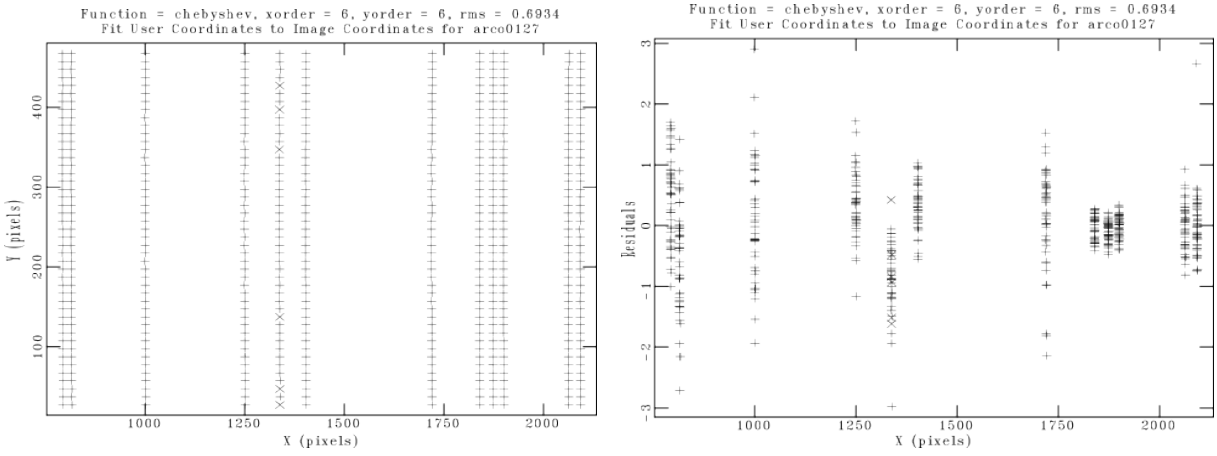
[12] Help documentation for the `fitcoords` task may be found at `https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.twodspec.longslit.fitcoords.html`.

[13] See also `https://iraf.net/irafdocs/formats/fitcoords.php` for an explanation of the database contents.

[14] Help documentation for the `transform` task may be found at `https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.twodspec.longslit.transform.html`.

Listing 3.2: An example of the `fitcoords` database contents.[13]

```
# Thu 15:26:55 13-May-2021
begin   arc00057
        task    fitcoords
        axis    1
        units   angstroms
        surface 33
                1.
                5.
                5.
                1.
                1.
                3199.
                1.
                474.
                7419.096745914063
                1510.03933621895
                -21.10886852752348
                -2.079553916887794
                0.06772631420528228
                0.7720164913117386
                -1.506773900054024
                0.1341878190232142
                -0.01659697703758917
                0.0251087019569153
                -3.318493303995171
                -0.3612632489821799
                0.003270665801371641
                -0.0157962041414068
                -0.0030736908715892 42
                0.007533453962924031
                0.02839687304474069
                -0.003233465769521899
                0.00174111456659807
                0.00645177595090841
                0.0105080093855621
                -0.01157827440314294
                -0.007789479002470706
                -0.006562085282926231
                -0.002321476801926803
```

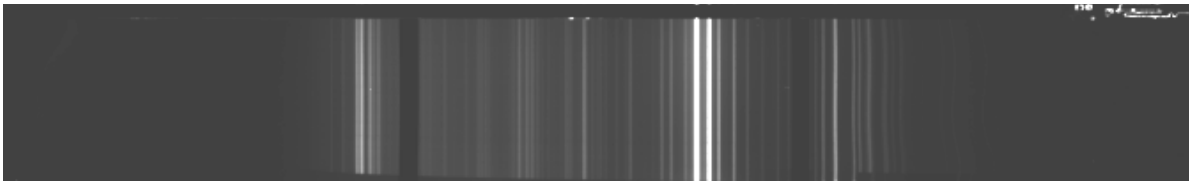


(a) Features identified across an exposure.

(b) RMS of the identified features.

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

(a) A well-fit wavelength solution.



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

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

For example, Figure 3.8a shows a good fit to the wavelength solution, as after transformation all the sky lines run exactly vertical. Figure 3.8b, on the other hand, shows a seemingly good fit, but closer inspection reveals that the sky lines (especially towards the right of the frame) deviate from the vertical, indicating a poor fit to the wavelength solution.

3.3 STOPS - *Supplementary Tools for POLSALT Spectropolarimetry*

Supplementary Tools for POLSALT Spectropolarimetry (STOPS) provides supplementary tools which convert the POLSALT and IRAF formats back and forth, allowing IRAF to be used for wavelength calibrations of SALT spectropolarimetric data. It also provides additional tools to check the accuracy of the wavelength calibration. STOPS is written in, and requires, Python 3 (3.11+) to run, as well as `Astropy` (6.0.0+) (Astropy Collaboration et al., 2013, 2018, 2022), `ccdproc` (2.4.1+) (Craig et al., 2017), `Matplotlib` (3.5.2+) (Hunter, 2007), `NumPy` (1.26.4+) (Harris et al., 2020), and `SciPy` (1.13.0+) (Virtanen et al., 2020).

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

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

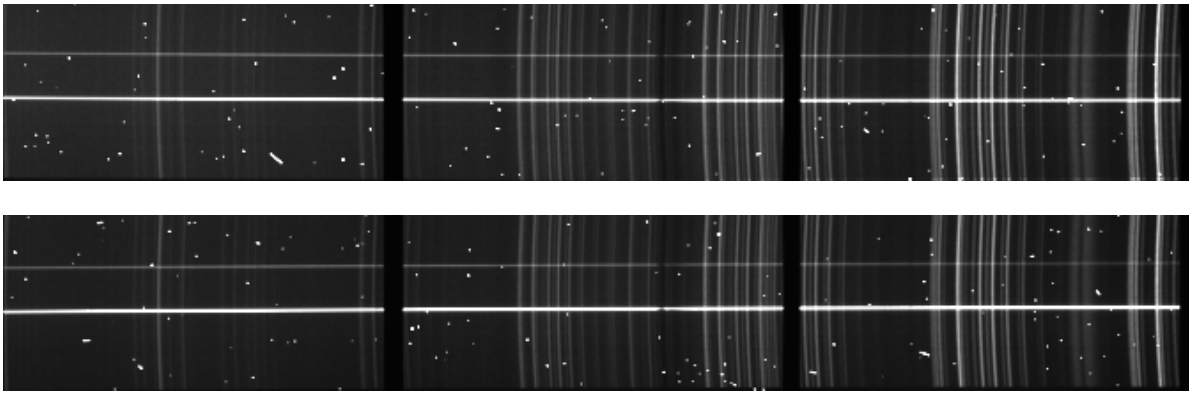


Figure 3.9: The split *O*- and *E*-beams as handed to IRAF. Figure created from the STOPS `split` method output.

Help on the usage of STOPS in a CLI can be viewed by running:

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

which retrieves and prints the help documentation to the CLI from Listing B.1 (in Appendix B), such as how to enable logging or increase the verbosity, or change default parameters of the various methods. Finally, help documentation for the specific STOPS methods may be found within this section (Listing 3.3 to 3.6) or in Appendix B.

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 (see Figure 3.9), the variance of the image, and a map of the pixels to be masked out, split into sub-extensions for both polarimetry beams, respectively.

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

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 FITS file (i.e. for both the *O*- and *E*-beam), and appends all header and science data necessary for wavelength calibration to the relevant HDU structure. Otherwise, defaults, such as which row to split the image along to separate the beams, were kept as close to the POLSALT pipeline as possible.

Listing 3.3: The 'docstring' for **split.py**

```
26
27  """
28  The `Split` class allows for the splitting of `polsalt` FITS files
29  based on the polarization beam. The FITS files must have basic
30  `polsalt` pre-reductions already applied (`mxgbp...` FITS files).
31
32  Parameters
33  ----------
34  data_dir : str
35      The path to the data to be split
36  fits_list : list[str], optional
37      A list of pre-reduced `polsalt` FITS files to be split within `data_dir`.
38      (The default is None, `Split` will search for `mxgbp*.fits` files)
39  split_row : int, optional
40      The row along which to split the data of each extension in the FITS file.
41      (The default is SPLIT_ROW (See Notes), the SALT RSS CCD's middle row)
42  no_arc : bool, optional
43      Decides whether the arc frames should be recombined.
44      (The default is False, `polsalt` has no use for the arc after wavelength calibrations)
45  save_prefix : dict[str, list[str]], optional
46      The prefix with which to save the  O & E beams.
47      Setting `save_prefix` = ``None`` does not save the split O & E beams.
48      (The default is SAVE_PREFIX (See Notes))
49
50  Attributes
51  ----------
52  arc : str
53      Name of arc FITS file within `data_dir`.
54      `arc` = `""` if `no_arc` or not detected in `data_dir`.
55  o_files, e_files : list[str]
56      A list of the `O`- and `E`-beam FITS file names.
57      The first entry is the arc file if `arc` defined.
58  data_dir
59  fits_list
60  split_row
61  save_prefix
62
63  Methods
64  -------
65  split_file(file: os.PathLike)
66      -> tuple[astropy.io.fits.HDUList]
67      Handles creation and saving the separated FITS files
68  split_ext(hdulist: astropy.io.fits.HDUList, ext: str = 'SCI')
69      -> astropy.io.fits.HDUList
70      Splits the data in the `ext` extension along the `split_row`
71  crop_file(hdulist: astropy.io.fits.HDUList, crop: int = CROP_DEFAULT (See Notes))
72      -> tuple[numpy.ndarray]
73      Crops the data along the edge of the frame, that is,
74      `O`-beam cropped as [crop:], and
75      `E`-beam cropped as [:-crop].
76  update_beam_lists(o_name: str, e_name: str)
77      -> None
78      Updates `o_files` and `e_files`.
79  save_beam_lists(file_suffix: str = 'frames')
80      -> None
81      Creates (Overwrites if exists) and writes the `o_files` and `e_files` to files named
82      `o_{file_suffix}` and `e_{file_suffix}`, respectively.
83  process()
84      -> None
85      Calls `split_file` and `save_beam_lists` on each file in `fits_list` for automation.
86
87  Other Parameters
88  ----------------
89  **kwargs : dict
90      keyword arguments. Allows for passing unpacked dictionary to the class constructor.
91
92  Notes
93  -----
94  Constants Imported (See utils.Constants):
95      SAVE_PREFIX
96      CROP_DEFAULT
97      SPLIT_ROW
98
99  """
```

As the intent was always to parse the wavelength function back into POLSALT it was decided to keep these temporary FITS files as small as possible by only including the header and science extension. To aid the scripting of the IRAF wavelength calibration process, the `split` method also performs row cropping to exclude CCD regions which are not exposed to light, and creates files listing the split *O*- and *E*-beam FITS files which may be passed to the IRAF task inputs. Row cropping was decided on as IRAF does not handle rows with no exposure well, specifically when it comes to the autonomous `reidentify` task. The full STOPS `split` class docstring is given in Listing 3.3.

3.3.2 Joining

After the IRAF `fitcoords` task has been successfully run for both the *O*- and *E*-beams, the STOPS `join` method is used to extract and parse the wavelength solution from the IRAF database, and to create the wavelength calibrated FITS files required by the POLSALT pipeline. More specifically, the `join` method performs the following steps:

First, `join` parses the wavelength database file, described in § 3.2.3, for either a 'Chebyshev' or 'Legendre' solution, and calculates the wavelength at each (x_p, y_p) pixel position. This new image containing the corresponding wavelength values, seen in Figure 3.10, is appended to the wavelength calibrated FITS file as the 'WAV' extension.

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

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

Finally, the wavelength extension is masked to remove any uncalibrated wavelength regions as well as masked for the skewing of the trace introduced by the wollaston element. The masking of the wollaston skewing is necessary since POLSALT introduces a wollaston correction in the `spectra extraction` process. The BPM extension is masked to reflect the valid wavelength calibrated region, and the files are saved with the POLSALT wavelength calibrated 'wmxgbp' prefix. The full STOPS `join` class docstring is given in Listing 3.4.

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

Listing 3.4: The 'docstring' for **join.py**

```
"""
The `Join` class allows for the joining of previously split files and the
appending of an external wavelength solution in 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 containing:
    `x`, `y`, *coefficients...
where a solution are of order (`x` by `y`) and must contain x*y coefficients,
all separated by newlines. 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 : https://adsabs.harvard.edu/abs/2001PASP..113.1420V)
.. [2] https://zenodo.org/records/1482019

"""
```

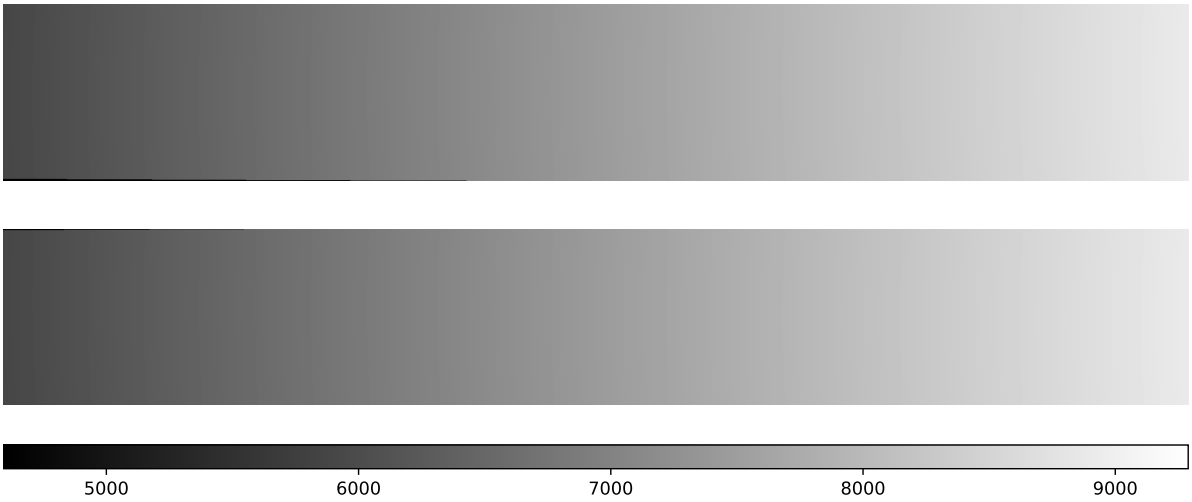


Figure 3.10: A representative wavelength extension of a FITS file, for the *O*- (top) and *E*-beam (bottom), ready to be processed by the POLSALT pipeline. The color bar shows the wavelength in Å. Note that regions which fall outside the exposed region are masked by setting the corresponding pixel values of the wavelength and BPM extensions to 0. Figure created from the STOPS `join` method output.

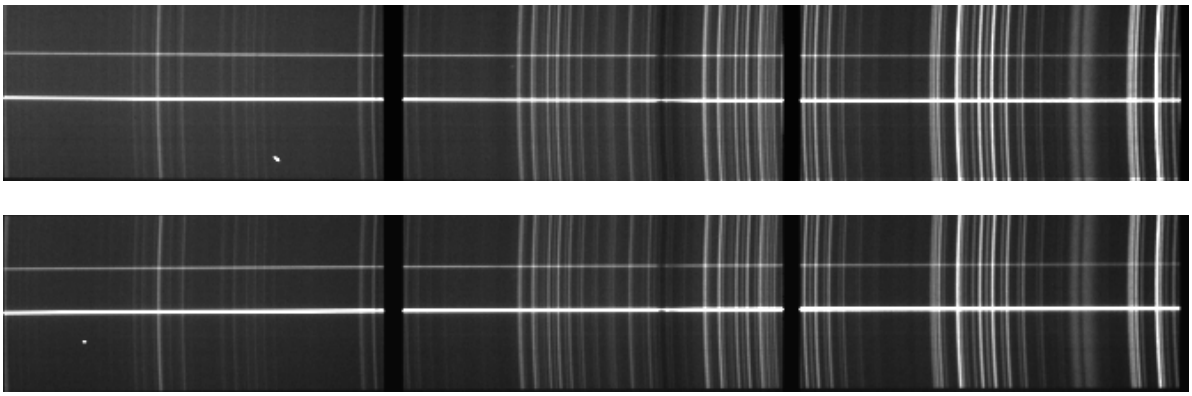


Figure 3.11: A representative science extension of a FITS file, for the *O*- (top) and *E*-beam (bottom), ready to be processed by the POLSALT pipeline. The observed intensity is displayed via the grayscale value at each pixel. Figure created from the STOPSjoin method output.

3.3.3 Sky Line Checks

The `skyline` method has been implemented to compare the position of the sky lines on the science extension, or arc lines in the arc exposure, to the known positions of the sky lines, or arc lines, as measured by SALT, respectively.[16] This provides an additional check of the accuracy of the wavelength solution across the frame. This method accepts both the IRAF `transform` FITS file or the 'wmxgbp' FITS files created by the `join` method as the input.

[16]Both sky and arc lines are available at `https://astronomers.salt.ac.za/data/salt-longslit-line-atlas/` .

Listing 3.5: The 'docstring' for **skylines.py**

```
37
38  """
39  Class representing the Skylines object.
40
41  Parameters
42  ----------
43  data_dir : Path
44      The directory containing the data files.
45  filenames : list[str]
46      The list of filenames to be processed.
47  beam : str, optional
48      The beam mode, by default "OE".
49  plot : bool, optional
50      Flag indicating whether to plot the continuum, by default False.
51  save_prefix : Path | None, optional
52      The prefix for saving the data, by default None.
53  **kwargs
54      Additional keyword arguments.
55
56  Attributes
57  ----------
58  data_dir : Path
59      The directory containing the data files.
60  fits_list : list[str]
61      The list of fits file paths.
62  beam : str
63      The beam mode.
64  can_plot : bool
65      Flag indicating whether to plot the continuum.
66  save_prefix : Path | None
67      The prefix for saving the data.
68  wav_unit : str
69      The unit of wavelength.
70  rawWav : np.ndarray
71      The raw wavelength data.
72  rawSpec : np.ndarray
73      The raw spectral data.
74  rawBpm : np.ndarray
75      The raw bad pixel mask data.
76  corrWav : np.ndarray
77      The corrected wavelength data.
78  corrSpec : np.ndarray
79      The corrected spectral data.
80  spec : np.ndarray
81      The median spectrum.
82  normSpec : np.ndarray
83      The normalized spectrum.
84
85  Methods
86  -------
87  checkLoad(self, path1: str) -> np.ndarray:
88      Checks and loads the data from the given path.
89  transform(self, wav_sol: np.ndarray, spec: np.ndarray) -> np.ndarray:
90      Transforms the input wavelength and spectral data based on
91      the given wavelength solution.
92  rmvCont(self) -> np.ndarray:
93      Removes the continuum from the spectrum.
94  process(self) -> None:
95      Placeholder method for processing the data.
96  """
```

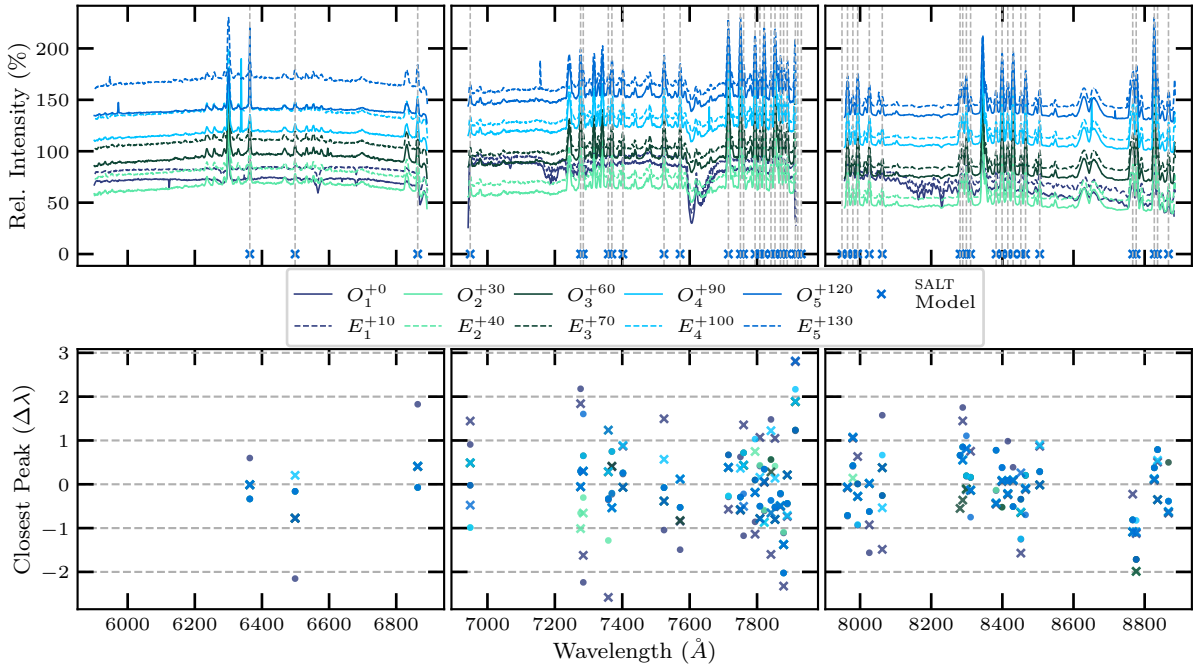


Figure 3.12: An example of a well calibrated wavelength solution, specifically shown for science images. Figure created via the STOPS `skyline` method.

The `skyline` method loads the wavelength calibrated files, masks the traces present in the frames, transforms the frames from (x_p, y_p) pixel to (Å, y_p) wavelength units if the frame was not transformed by `transform`,[17] and compares the peak wavelength position of the sky lines to the reference sky lines as measured by SALT.

Minor variations in the comparison of the sky lines are expected, such as those seen in Figure 3.12, but any uniform trends, such as those in Figure 3.13 (bottom left), indicate an underlying poor fit across the horizontal axis of the wavelength solution. The full STOPS `skyline` class docstring is given in Listing 3.5.

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 each other. As the Stokes results, and thus final polarization results, are determined by the difference between the *O*- and *E*-beams, a direct comparison is not appropriate. Any observed unpolarized light, however, will reflect equally in both polarization beams and so the general trend of the two spectra may reasonably be expected to follow one another. The `correlate` method was created to allow for comparisons between the wavelength solutions of the *O*- and *E*-beams of a single exposure or the *O*- or *E*-beams of differing exposures by cross correlating the spectra.

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

[17] The transformation applied by the `skyline` method uses linear interpolation and is thus less accurate at flux conservation than the transformation applied by the `transform` method.

Listing 3.6: The 'docstring' for **cross_correlate.py**

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

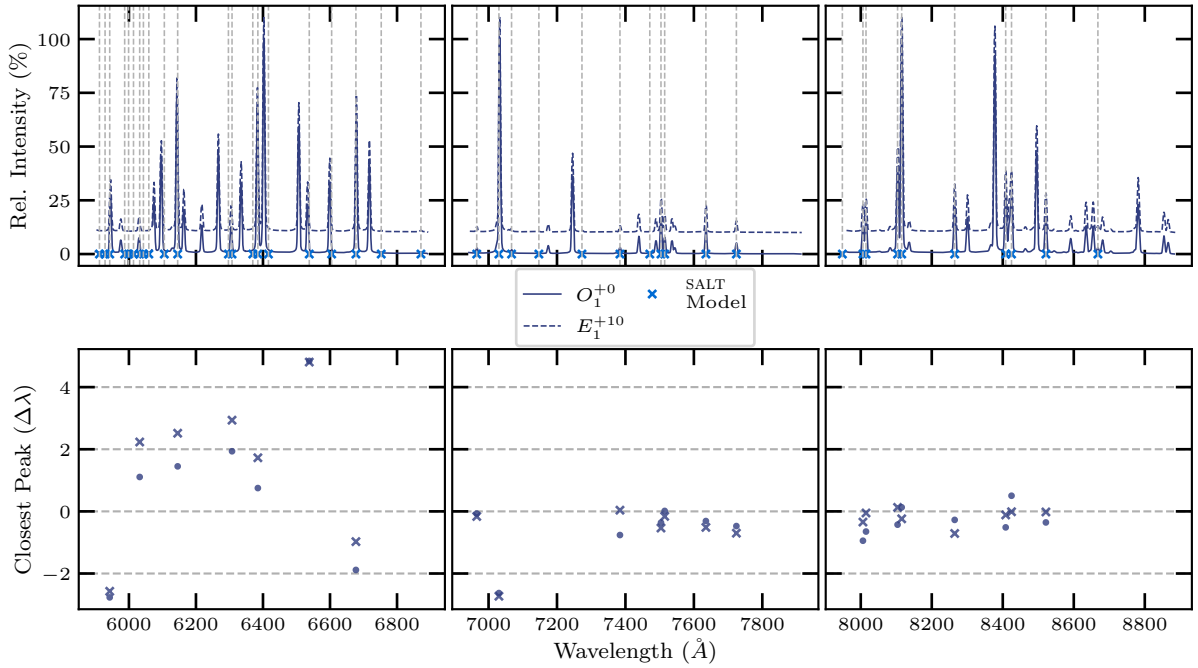


Figure 3.13: An example of a wavelength solution with a poor fit at shorter wavelengths (bottom left), specifically shown for an arc image. Arc lines are Figure created via the STOPS `skyline` method.

Figure 3.14.

Sources under spectropolarimetric observation are generally expected to vary over time and, as such, the ratio of polarized to unpolarized light is also expected to vary. The accuracy of correlation may decrease as features with differences in the polarized component of the polarization beams change, and it is up to the user to determine the validity of the correlation result taking into consideration the two spectra being correlated. The differences in the features of the different spectra are often negligible when compared to the overall continuum of the spectra and are generally only reflected in a change in the intensity of said features when the continuum is removed. The full STOPS `split` class docstring is given in Listing 3.6.

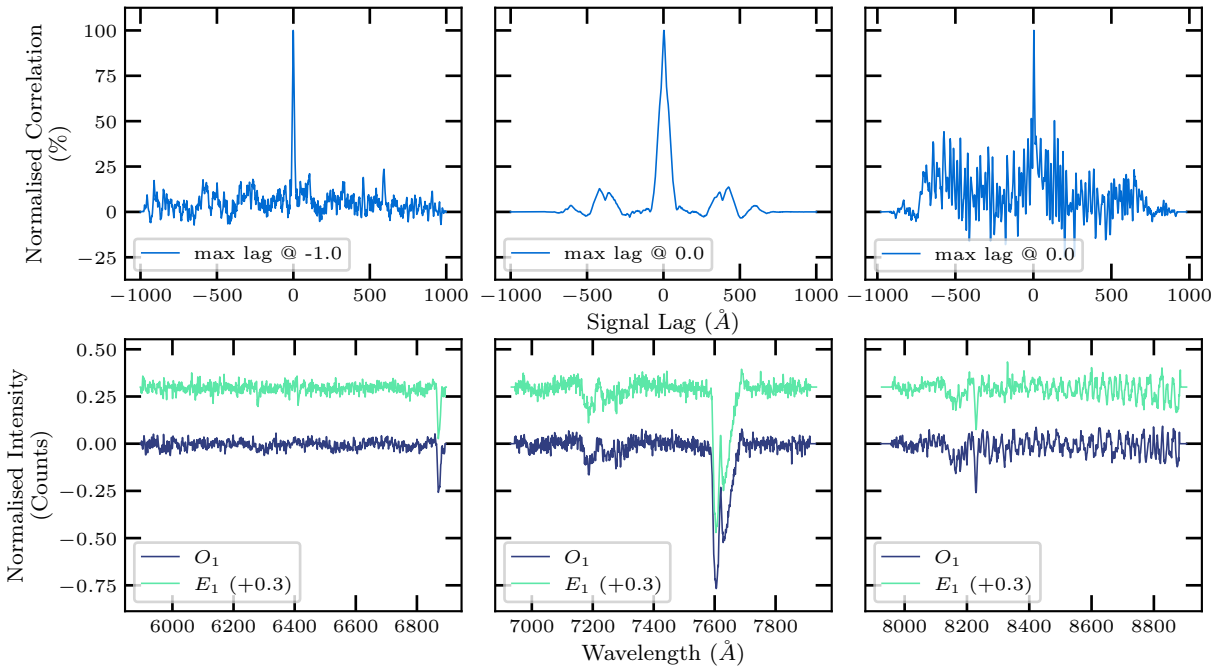


Figure 3.14: The resultant output plot of the STOPS `correlate` method. Figure created via the STOPS `correlate` method.

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 A. 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 general order, seen in Figure 3.15, and commands necessary to complete each step of the reduction process are discussed, assuming that the software is running as intended.

3.4.1 Initial Setup

It is important to note that while POLSALT was developed in Python 2 (2.7), the STOPS supplementary tools were developed in, and require, Python 3 (3.11+), as well as the other requirements mentioned in § 3.3. While managing multiple versions of Python introduces some extra complication, it would not have been reasonable to develop STOPS in Python 2, as it has been deprecated, nor would it have been reasonable to update POLSALT to Python 3.

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

It is recommended to use POLSALT through the GUI as it provides a user-friendly environment while also sequentially listing each step of the reduction process in a dropdown

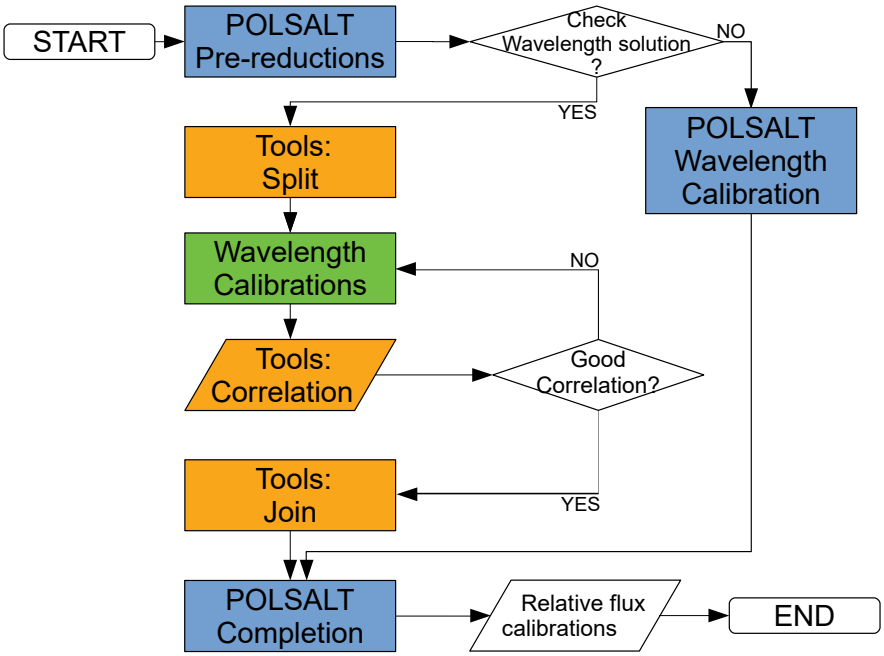


Figure 3.15: A general workflow for data reductions using a combination of POLSALT, IRAF, and the developed supplementary tools. Diagram adapted from Cooper et al. (2022).

menu, as seen in Figure 3.1. Reductions are possible, however, purely through the CLI using the POLSALT 'beta' scripts.

3.4.2 POLSALT Pre-Reductions

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

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

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

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

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

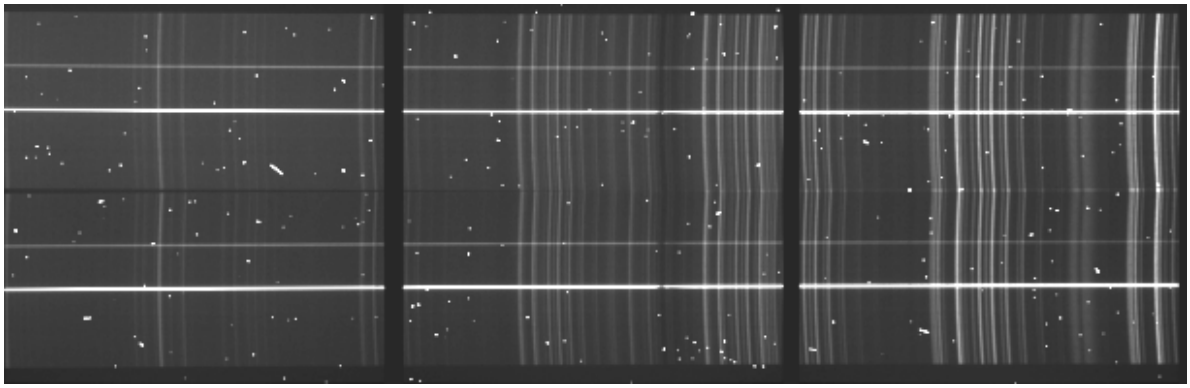


Figure 3.16: The science extension of a typical spectropolarimetric FITS file taken with the SALT RSS, after basic POLSALT CCD reductions have been completed. Figure created from the STOPS `split` output.

is followed.

3.4.3 Wavelength Calibration

The wavelength calibrations may now be completed in IRAF. This section concerns the procedure for parsing the FITS files to and from both IRAF and POLSALT, as well as the relevant task names and methods to be run to complete the calibrations. A base working case of each of the tasks and methods are presented in Listing A.2 to A.8, but it should be noted that the art of wavelength calibration consists of modifying the parameters to achieve a well-fit calibration function.

Preparing the Data for IRAF

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

IRAF Wavelength Calibrations

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

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

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

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

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

[18] 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.

Running an IRAF script may be done by running:

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

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

Preparing the Data for POLSALT

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

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

Sky Line Checks

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

The `skyline` method is run in the CLI following Listing A.9. As with the rest of STOPS, default parameters describe the overplotting behavior for the *O*- and *E*-beams, the skylines provided by SALT, and the calculated variation of the wavelength axis of a frame.

Cross Correlation Checks

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

Cleaning Up the IRAF and STOPS Output

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

```
$ rm beam*.fits arc*.fits frame* <any user created files>
```

3.4.4 POLSALT Reduction Completion

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

POLSALT Beta in the GUI

The reduction process using the POLSALT GUI is completed by selecting and, when applicable, interactively modifying the reduction step through the interactive windows, one-by-one, from the GUI's dropdown menu, as explained in Appendix A (p. 68 onwards).[19]

POLSALT Beta through a CLI

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

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

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

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`, for the different analysis modes, respectively.[20]

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

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

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

```
import os, sys, glob, shutil
poldir = '/home/justin/polsalt-beta/'                                  # Will differ according to user

reddir=poldir+'polsalt/'
scrdir=poldir+'scripts/'
sys.path.extend((reddir,scrdir,poldir))

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

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

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

#basic polarimetric reductions
logfile='specpol'+obsdate+'.log'

                                                                       # The following lines may be removed or commented out as below
# #wavelength map
# infilelist = sorted(glob.glob('m*fits'))
# linelistlib=""
# specpolwavmap(infilelist, linelistlib=linelistlib, logfile=logfile)

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

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

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

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

Chapter 4

Testing and Application

This chapter contains an overview of the testing performed for and during the development of STOPS as well as its application in publications. Software tests cover ... The tests cover ...

- No POLSALT or data tests
  - POLSALT is trusted to be accurate. See ... (reference tests of POLSALT, i.e. published papers, etc.)
- Testing `split`
  - POLSALT to IRAF file structure conversion.
  - Show changes to data files are intended (I.E. only splitting the data, no changes to the data itself)
  - Tested over multiple grating/articulation angles to ensure robustness.
  - Mention any header updates skipped specific to POLSALT (if any)
  - Figure showing split fits file contents difference (cropped rows shown, etc.)
- Testing IRAF wavelength solution
  - IRAF is trusted to be accurate. See ... (reference tests of IRAF, i.e. published papers, etc.)
  - The `skylines` and `correlate` outputs are tests of the wavelength calibration.
    - Testing correlate functionality using 'offset', comparisons of arcs, FSRQ's and BLLac's.
    - Any figures showing correlation tests?
    - Testing skylines using known spectral sky lines.
    - Any figures showing skyline tests?
- Testing `join`
  - IRAF to POLSALT file structure conversion
  - Show changes to data files are intended (I.E. only joining the data, and 'WAV' appended. No changes to the data itself)
  - Wollaston correction of wavelength and bpm extensions
  - update to python 3 of 'polsalt' functions
  - Mention any header differences (if any)
  - Figure showing joined fits file contents difference (cropped rows shown once again, bpm differences due to CRR and NO wollaston bpm differences, etc.)

- Testing reduction results not negatively impacted by STOPS.
  - General discussion of testing (I.E. not this test was done specifically this source, more along the lines of these tests were done to check this issue, seen here using this source for example.)
  - Wavelength solution validation from correlate and skylines results.
    * Figures showing Correlate and Skyline results.
    * RMS comparisons between POLSALT and IRAF to quantify differences.
    * Any Figures for RMS comparisons or wavelength validation?
  - Polarization parameters validation from known sources.
    * Polarization tested using 3C 279, 4C+01.02, and preliminary testing data provided by David.
    * Polarization tested using spectropolarimetric standards (4 highly polarized, 2 non-polarized).
    * Tabulate sources used, with their properties.
    * Figures showing comparison plots of polarization parameters from POLSALT and STOPS.

- Background information on each object.
- Detailed reductions steps performed on each object.
- Comparison of POLSALT results to those obtained using the STOPS pipeline.

- Application to Spectropolarimetric Standards
  - Science results, what the results can tell us and why it is useful, also comparison of results to FORS1/2 published data, focus on the polarization results.
- Application in Publications
  - Summarize the results of the publications appended to appendix.

Chapter 5

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.

5.1 Future Work

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

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

Appendix A

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>`' and '`>>>`' tokens are used for IRAF's 'xgterm' single- and multi-line commands, respectively.

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

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

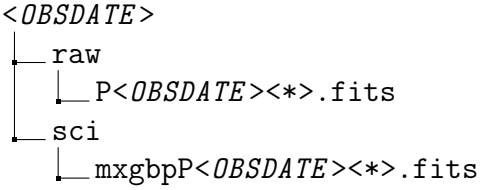


Figure A.1: The typical minimal file structure of data provided by SALT.

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

Listing A.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.

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

Listing A.2: Splitting data using STOPS

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

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

Listing A.3: Launching IRAF in xgterm

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

The IRAF `identify` task requires an average feature width, 'fwidth', as a parameter. The width of a feature may be found in IRAF using the `implot` task[1] along with cursor commands, but may also be found using FITS viewing software, such as `ds9`.[2] The `identify` task may be run using the commands in Listing A.4.

Listing A.4: Running the IRAF `identify` task

```
cl> mkscript 01_identify.cl
cl> # Add identify to 01_identify.cl twice, for both beams
cl> # Edit the parameters of 01_identify.cl in a text editor
cl> # Paste an identify script into the CLI, resulting in:
cl>
cl> identify ("<O-beam ARC>",
>>> "", "", section="middle line", database="database",
>>> coordlist="linelists$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="")
```

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

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

Listing A.5: Running the IRAF `reidentify` task

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

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

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

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

The `reidentify` task will run autonomously so long as the `interactive` parameter is set to "no".[4] Repeat the `reidentify` procedure, replacing <*O-beam ARC*> with <*E-beam ARC*> at both the 'reference' and 'image' parameter locations.

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

Listing A.6: Running the IRAF `fitcoords` task

```
cl> mkscript 03_fitcoords.cl
cl> # Add fitcoords to 03_fitcoords.cl twice, for both beams
cl> # Edit the parameters of 03_fitcoords.cl in a text editor
cl> # Paste a fitcoords script into the CLI, resulting in:
cl>
cl> fitcoords ("<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="")
```

The `fitcoords` task will launch an interactive window. The x- and y-axis being plotted may be changed using the 'x' or 'y' cursor commands followed by the desired data axis ('x' for the x-axis, 'y' for the y-axis, or 'r' for the residuals).[5] Repeat the `fitcoords` procedure, replacing <*O-beam ARC*> with <*E-beam ARC*>.

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

Listing A.7: Running the IRAF `transform` task

```
cl> mkscript 04_transform.cl
cl> # Add transform to 04_transform.cl twice, for both beams
cl> # Edit the parameters of 04_transform.cl in a text editor
cl> # Paste a transform script into the CLI, resulting in:
cl>
cl> transform ("@<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")
```

The `transform` task will run autonomously.[6] Repeat the `transform` procedure, replacing the <*O-beam FILES*> and <*O-beam ARC*> with <*E-beam FILES*> and <*E-beam ARC*> at both parameter locations. Inspect the transformed images, most notably the arc images, using any FITS viewer as a cursory check that the wavelength calibrations were completed without error.

The 'gain' and 'read noise' are now needed as the cosmic-ray rejection of the STOPS `join` method accepts them as parameters. These parameters may be found using the '*GAINSET*' and '*ROSPEED*' keywords in the FITS headers. The cosmic ray rejection

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

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

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

defaults to *GAINSET*='FAINT', and *ROSPEED*='SLOW'. If the gain and read noise values differ from the defaults, the parameters should be updated when running `join`.[7]

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

Listing A.8: Joining the data using STOPS

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

The STOPS `skylines` method may be run on any 'joined' or transformed FITS files, *<FILE(S)>*, using the commands in Listing A.9.

Listing A.9: Running the STOPS `skylines` method

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

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

Listing A.10: Directory cleanup for POLSALT

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

The POLSALT `spectra extraction` is now run. If the POLSALT GUI was closed it should now be reopened using Listing A.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'.

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

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

Listing A.11: Running the STOPS `correlate` method

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

The POLSALT `raw Stokes calculation`, `final Stokes calculation`, and `results visualisation` may now be completed. For the last time, if the POLSALT GUI was closed it should now be reopened using Listing A.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 A.2.

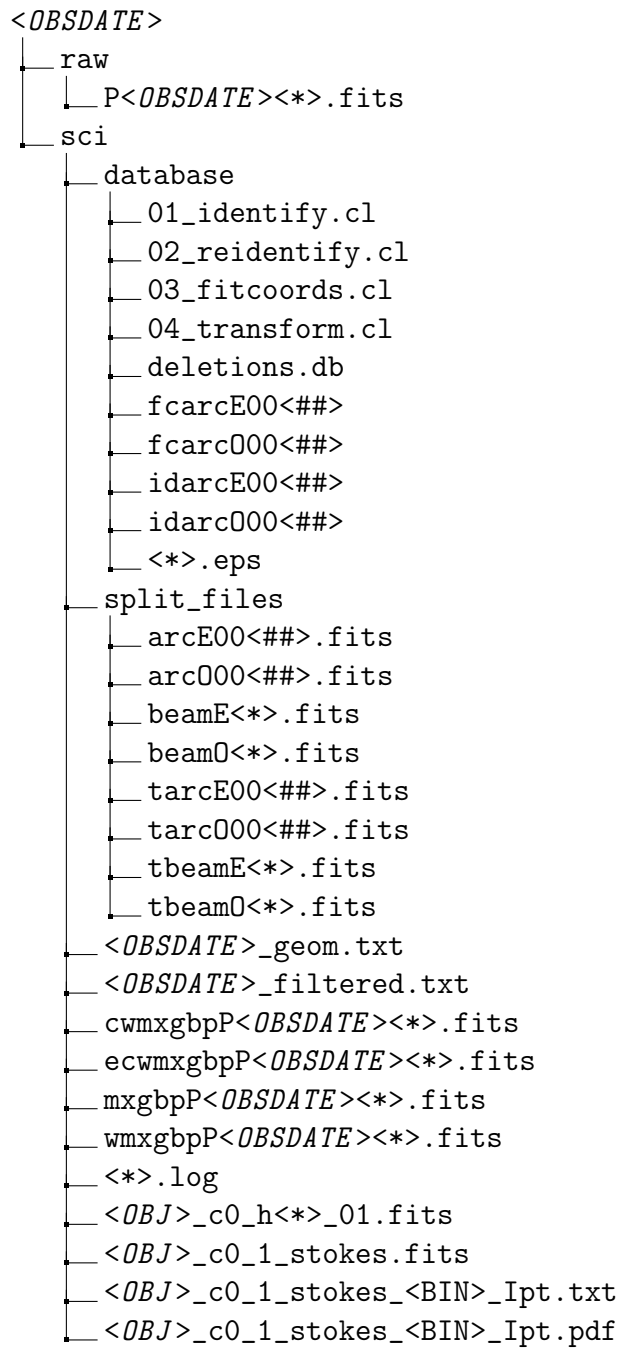


Figure A.2: The typical file structure after completing the reduction process.

Appendix B

STOPS Source Code

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

Listing B.1: The source code for **__main__.py**

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

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

from __init__ import __version__, __author__, __email__

# MARK: Imports
import os
import sys
import argparse
import logging
from pathlib import Path

from split import Split
from join import Join
from cross_correlate import CrossCorrelate
from skylines import Skylines

from utils import ParserUtils as pu
from utils.Constants import SPLIT_ROW, PREFIX, PARSE, SAVE_CORR,
↪ SAVE_SKY

# 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
```

```
"""


# 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 of the logging file. "
        "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
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: Correlate\Skylines Parent
corr_sky_args = argparse.ArgumentParser(add_help=False)
corr_sky_args.add_argument(
    "filenames",
    action="store",
    nargs="+",
    type=pu.parse_file,
    help=(
        "File name(s) of FITS file(s) to be processed."
        "A minimum of one filename is required."
    ),
)
corr_sky_args.add_argument(
    "-b",
    "--beams",
    choices=["O", "E", "OE"],
    type=str.upper,
    default=PARSE['BEAMS'],
    help=(
        "Beams to process. "
        f"Defaults to {PARSE['BEAMS']}, but "
        "may be given 'O', 'E', or 'OE' to "
        "determine which beams are processed."
    ),
)
corr_sky_args.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_sky_args.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_sky_args.add_argument(
    "-p",
    "--plot",
    action="store_true",
    help="Flag for additional plot outputs.",
)
corr_sky_args.add_argument(
    "-s",
    "--save_prefix",
    action="store",
    nargs="?",
    type=lambda path: Path(path).expanduser().resolve(),
    const=SAVE_CORR,
    help=(
        "Prefix used when saving plot. "
        "Excluding flag does not save output plot, "
        f"flag usage of option uses default prefix, "
        "and a provided prefix overwrites default 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,
)


# 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_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,
)


# MARK: Correlate Subparser
corr_parser = subparsers.add_parser(
    "correlate",
    aliases=["x"],
    help="Cross correlation mode",
    parents=[corr_sky_args],
)
# 'children' join args here
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.)"
    ),
)
# Change defaults here
corr_parser.set_defaults(
    mode="correlate",
    func=CrossCorrelate,
)


# MARK: Skyline Subparser
sky_parser = subparsers.add_parser(
    "skylines",
    aliases=["sky"],
    help="Sky line check mode",
    parents=[corr_sky_args],
)
# 'children' skyline args here
```

```
sky_parser.add_argument(
    "-t",
    "--transform",
    action="store_false",
    help=(
        "Flag to force transform images. "
        "Recommended to use only when input image(s) "
        "are prefixed 't' but are not yet transformed."
    ),
)
# Change defaults here
sky_parser.set_defaults(
    mode="skyline",
    func=Skylines,
)


# MARK: Keyword Clean Up
args = parser.parse_args()

if len(sys.argv) == 1:
    parser.print_help(sys.stderr)
    sys.exit(2)

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 and type(args.solutions_list) == list:
    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 B.2: The source code for **split.py**

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

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

from __init__ import __author__, __email__, __version__

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

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:

    #----------split0----------

    """
    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
```

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

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

        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.
```

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

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

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

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

from __init__ import __author__, __email__, __version__

# MARK: Imports
import os
import sys
import logging
import re
from pathlib import Path

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 # Replaced: 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 Class
class Join:

    #----------join0----------

    """
    The `Join` class allows for the joining of previously split files
    ↪ and the
    appending of an external wavelength solution in 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 containing:
        `x`, `y`, *coefficients...
    where a solution are of order (`x` by `y`) and must contain x*y
    ↪ coefficients,
    all separated by newlines. 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 :
    ↪ https://adsabs.harvard.edu/abs/2001PASP..113.1420V)
    .. [2] https://zenodo.org/records/1482019

    """

    #----------join1----------

    # MARK: Join init
    def __init__(
        self,
        data_dir: Path,
        database: str = "database",
        fits_list: list[str] = None,
        solutions_list: list[Path] = None,
        split_row: int = SPLIT_ROW,
        no_arc: bool = True,
        save_prefix: Path | None = None,
        verbose: int = 30,
        **kwargs,
    ) -> None:
        self.data_dir = data_dir
        self.database = Path(data_dir) / 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)
```

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

```
206                 raise FileNotFoundError(msg)
207
208             sols = {i: j for i, j in zip(['O', 'E'], sorted(ws,
                ↪ reverse=reverse))}
209             logging.debug(f"get_solutions - Found {sols} in
                ↪ {self.database}")
210
211             return (sorted(ws, reverse=reverse), False)
212
213         # Custom solution
214         if len(wavlist) >= 2:
215             if len(wavlist) > 2:
216                 logging.warning(f" Too many solutions, only
                    ↪ {wavlist[:2]} are considered")
217                 wavlist = wavlist[:2]
218
219             for fl in wavlist:
220                 if not os.path.isfile(os.path.join(self.data_dir, fl)):
221                     msg = (
222                         f"{fl} not found in the "
223                         f"data directory {self.data_dir}"
224                     )
225                     logging.error(msg)
226                     raise FileNotFoundError(msg)
227
228             sols = {i: j for i, j in zip(['O', 'E'], sorted(wavlist,
                ↪ reverse=reverse))}
229             logging.debug(f"get_solutions - Found {sols} in
                ↪ {self.database}")
230
231             return (sorted(wavlist, reverse=reverse), True)
232
233     # MARK: Parse 2D WAV Function
234     def parse_solution(
235         self,
236         fc_file: str,
237         xshape: int,
238         yshape: int
239     ) -> tuple[dict[str, int], np.ndarray]:
240         """
241         Parse the 2D wavelength solution function from `fc_file`.
242
243         Parameters
244         ----------
245         fc_file : str
246             The filename of the wavelength solutions file.
247         xshape : int
248             The x-order of the 2D solution.
249         yshape : int
250             The y-order of the 2D solution.
251
252         Returns
253         -------
254         tuple[dict[str, int], np.ndarray]
255             A tuple containing a dictionary of the parameters of the
                ↪ solution function
256             and the function coefficients.
257
```

```
258         """
259         fit_params = {}
260         coeff = []
261 
262         if self.custom:
263             # Load coefficients
264             coeff = np.loadtxt(fc_file)
265 
266             fit_params["xorder"] = coeff[0].astype(int)
267             fit_params["yorder"] = coeff[1].astype(int)
268             coeff = coeff[2:]
269 
270             f_type = 3
271             if "cheb" in str(fc_file): f_type = 1
272             elif "leg" in str(fc_file): f_type = 2
273             fit_params["function"] = f_type
274 
275             fit_params["xmin"], fit_params["xmax"] = 1, xshape
276             fit_params["ymin"], fit_params["ymax"] = 1, yshape
277 
278         else:
279             # Parse IRAF fc database files
280             file_contents = []
281             with open(self.database / fc_file) as fcfile:
282                 for i in fcfile:
283                     file_contents.append(re.sub(r"[\n\t\s]*", "", i))
284 
285             if file_contents[9] != "1.":  # xterms - Cross-term type
286                 msg=(
287                     "Cross-term not recognised (always 1 for "
288                     "FITCOORDS), redo FITCOORDS or change manually."
289                 )
290                 raise Exception(msg)
291 
292             fit_params["function"] = int(file_contents[6][:-1])
293 
294             fit_params["xorder"] = int(file_contents[7][:-1])
295             fit_params["yorder"] = int(file_contents[8][:-1])
296 
297             fit_params["xmin"] = int(file_contents[10][:-1])
298             fit_params["xmax"] = xshape
299             # int(file_contents[11][:-1])# stretch fit over x
300             fit_params["ymin"] = int(file_contents[12][:-1])
301             fit_params["ymax"] = yshape
302             # int(file_contents[13][:-1])# stretch fit over y
303 
304             coeff = np.array(file_contents[14:], dtype=float)
305 
306         coeff = np.reshape(
307             coeff,
308             (fit_params["xorder"], fit_params["yorder"])
309         )
310 
311         return (fit_params, coeff)
312 
313     # MARK: Join Files
314     def join_file(self, file: os.PathLike) -> None:
315         """
```

```
        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 2D grid functions
            https://numpy.org/doc/stable/reference/generated/numpy
            Chebyshev: + '.polynomial.chebyshev.chebgrid2d.html'
            Legendre: + '.polynomial.legendre.leggrid2d.html'

        """
        # 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"
                    )
```

```
374                     whdu[ext].header["BITPIX"] = "-32"
375
376                 # Fill in empty extentions
377                 if cropsize:
378                     temp_split = split_sci(
379                         hdu,
380                         self.split_row,
381                         ext=ext
382                     )[ext].data
383                     whdu[ext].data[0] = temp_split[0, cropsize:]
384                     whdu[ext].data[1] = temp_split[1, 0:-cropsize]
385
386                 else:
387                     whdu[ext].data = split_sci(
388                         hdu,
389                         self.split_row,
390                         ext=ext
391                     )[ext].data
392         # End of hdu calls, close hdu
393
394         # MARK: Join (Wav. Ext.)
395         whdu.append(pyfits.ImageHDU(name="WAV"))
396         wav_header = whdu["SCI"].header.copy()
397         wav_header["EXTNAME"] = "WAV"
398         wav_header["CTYPE3"] = "O,E"
399         whdu["WAV"].header = wav_header
400
401         whdu["WAV"].data = np.zeros(
402             whdu["SCI"].data.shape,
403             dtype=">f4"
404         )
405
406         for num, fname in enumerate(self.fc_files):
407             params, coeffs = self.parse_solution(
408                 fname,
409                 x_shape,
410                 y_shape
411             )
412
413             if params["function"] == 1:  # Function type (1 = chebyshev)
414                 # Set wavelength extention values to function
415                 whdu["WAV"].data[num] = chebgrid2d(
416                     x=np.linspace(-1, 1, params["ymax"]),
417                     y=np.linspace(-1, 1, params["xmax"]),
418                     c=coeffs,
419                 )
420
421             elif params["function"] == 2:  # Function type (2 =
                ↪ legendre)
422                 # Set wavelength extention values to function
423                 whdu["WAV"].data[num] = leggrid2d(
424                     x=np.linspace(-1, 1, params["ymax"]),
425                     y=np.linspace(-1, 1, params["xmax"]),
426                     c=coeffs,
427                 )
428
429             else:
430                 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_NEIGH_THRESH'],
                # 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(
```

```
489             whdu["WAV"].data[1] == 0,
490             1,
491             whdu["BPM"].data[1]
492         )
493 
494         whdu.writeto(f"w{os.path.basename(file)}", overwrite="True")
495 
496         return
497 
498     # MARK: Check Crop
499     def check_crop(
500         self,
501         hdu: pyfits.HDUList,
502         o_file: str,
503         e_file: str
504     ) -> int:
505         """
506         Check if cropping is necessary when joining `O`- and `E`-beams.
507 
508         Parameters
509         ----------
510         hdu : astropy.io.fits.HDUList
511             The HDUList to check for cropping.
512         o_file : str
513             The name of the previously split `O`-beam FITS file.
514         e_file : str
515             The name of the previously split `E`-beam FITS file.
516 
517         Returns
518         -------
519         int
520             The number of rows which were cropped by `Split`.
521 
522         """
523         cropsize = 0
524         o_y = 0
525         e_y = 0
526 
527         with pyfits.open(o_file) as o, \
528              pyfits.open(e_file) as e:
529             o_y = o[0].data.shape[0]
530             e_y = e[0].data.shape[0]
531 
532         if hdu["SCI"].data.shape[0] != (o_y + e_y):
533             # Get crop size, assuming crop same on both sides
534             cropsize = int((hdu["SCI"].data.shape[0] - o_y - e_y) / 2)
535 
536         return cropsize
537 
538     # MARK: Process all Listed Images
539     def process(self) -> None:
540         """Process all FITS images stored in the `fits_list`
        ↪ attribute"""
541         for target in self.fits_list:
542             logging.debug(f"Processing {target}")
543             self.join_file(target)
544 
545         return
```

```
546
547
548 def main(argv) -> None:
549     """Main function."""
550
551     return
552
553
554 if __name__ == "__main__":
555     main(sys.argv[1:])
```

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

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

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

from __init__ import __author__, __email__, __version__

# MARK: Imports
import os
import sys
import logging
import itertools as iters
from pathlib import Path
from typing import Callable

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 find_files, continuum
from utils.Constants import SAVE_CORR

OFFSET = 0.3

mpl_logger = logging.getLogger('matplotlib')
mpl_logger.setLevel(logging.INFO)

# MARK: Correlate class
class CrossCorrelate:

    #----------corr0----------

    """
    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
    filenames : 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)
    plot : bool, optional
        Decides whether or not the continuum fitting should be plotted
```

```
54          (The default is False, so no continua plots are displayed)
55      save_prefix : str, optional
56          The name or directory to save the figure produced to.
57          "." saves a default name to the current working. A default name
            ↪ is also used when save_prefix is a directory.
58          (The default is None, I.E. The figure is not saved, only
            ↪ displayed)
59
60      Attributes
61      ----------
62      data_dir
63      fits_list
64      beams : str
65          The mode of correlation.
66          'OE' for same file, and 'O' or 'E' for different files but same
            ↪ ext's.
67      ccds : int
68          The number of CCD's in the data. Used to split the CCD's if
            ↪ split_ccd is True.
69      cont_ord : int
70          The degree of the chebyshev to fit to the continuum.
71      can_plot : bool
72          Decides whether or not the continuum fitting should be plotted
73      offset : int
74          The amount the spectrum is shifted, mainly to test the effect
            ↪ of the cross correlation
75          (The default is 0, I.E. no offset introduced)
76      save_prefix
77      wav_unit : str
78          The units of the wavelength axis.
79          (The default is Angstroms)
80      wav_cdelt : int
81          The wavelength increment.
82          (The default is 1)
83      alt : Callable
84          An alternate method of cross correlating the data.
85          (The default is None)
86
87      Methods
88      -------
89      load_file(filename: Path) -> tuple[np.ndarray, np.ndarray,
        ↪ np.ndarray]
90          Loads the data from a FITS file.
91      get_bounds(bpm: np.ndarray) -> np.ndarray
92          Finds the bounds for the CCD regions.
93      remove_cont(spec: list, wav: list, bpm: list, plot_cont: bool) ->
        ↪ None
94          Removes the continuum from the data.
95      correlate(filename1: Path, filename2: Path | None = None) -> None
96          Cross correlates the data.
97      FTCS(filename1: Path, filename2: Path | None = None) -> None
98          Cross correlates the data using the Fourier Transform.
99      plot(spec, wav, bpm, corrdb, lagsdb) -> None
100         Plots the data.
101     process() -> None
102         Processes the data.
103
104     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.
        FTCS : bool, optional
            Decides whether the Fourier Transform should be used for
            ↪ cross correlation.

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


    Notes
    -----
    Constants Imported (See utils.Constants):
        SAVE_CORR

    """

    #----------corr1----------

    # MARK: Correlate init
    def __init__(
        self,
        data_dir: Path,
        filenames: list[str],
        beams: str = "OE",
        split_ccd: bool = True,
        cont_ord: int = 11,
        plot: bool = False,
        offset: int = 0,
        save_prefix: Path | None = None,
        **kwargs
    ) -> None:
        self.data_dir = data_dir
        self.fits_list = find_files(
            data_dir=self.data_dir,
            filenames=filenames,
            prefix="ecwmxgbp",
            ext="fits",
        )
        self._beams = None
        self.beams = beams
        self.ccds = 1
        if split_ccd:
            # BPM == 2 near center of CCD if CCD count varies
            with pyfits.open(self.fits_list[0]) as hdu:
                self.ccds = sum(hdu["BPM"].data.sum(axis=1)[0] == 2)

        self.cont_ord = cont_ord
        self.can_plot = plot
```

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

```
217
218         Returns
219         -------
220         tuple[np.ndarray, np.ndarray, np.ndarray]
221             The spectrum, wavelength, and bad pixel mask.
222
223         """
224         spec, wav, bpm = None, None, None
225
226         # Open HDU
227         with pyfits.open(filename) as hdul:
228             spec = hdul["SCI"].data.sum(axis=1)
229             wav = (
230                 np.arange(spec.shape[-1])
231                 * hdul["SCI"].header["CDELT1"]
232                 + hdul["SCI"].header["CRVAL1"]
233             )
234             wav = np.array((wav, wav))
235             bpm = hdul["BPM"].data.sum(axis=1)
236
237             self.wav_cdelt = float(hdul["SCI"].header["CDELT1"])
238
239             if hdul["SCI"].header["CTYPE1"] != 'Angstroms':
240                 self.wav_unit = hdul["SCI"].header["CTYPE1"]
241
242         return spec, wav, bpm
243
244     # MARK: Get bounds
245     def get_bounds(self, bpm: np.ndarray) -> np.ndarray:
246         """
247         Find the bounds for a file based on the CCD count.
248
249         Parameters
250         ----------
251         bpm : np.ndarray
252             The bad pixel mask.
253
254         Returns
255         -------
256         np.ndarray
257             The bounds for the CCD regions.
258
259         """
260         # bounds.shape -> (O|E, CCD's, low./up. bound)
261         if self.ccds == 1:
262             return np.array(
263                 [[(0, bpm[0].shape[-1])], [(0, bpm[1].shape[-1])]]
264             ).astype(int)
265
266         bounds = np.zeros((2, self.ccds, 2))
267
268         # Get lower and upper bound for each ccd, save to bounds
269         # Lower -> min is zero, Upper -> max is bpm length
270         for ext, ccd in iters.product(range(2), range(self.ccds)):
271             mid = np.where(bpm[ext] == 2)[0][ccd]
272             CCDs = self.ccds * 2
273             bounds[ext, ccd] = (
274                 max(mid - bpm.shape[-1] // CCDs, 0),
```

```
                min(mid + bpm.shape[-1] // CCDs, bpm.shape[-1])
            )

        return bounds.astype(int)

    # MARK: Remove Continua
    def remove_cont(
        self,
        spec: list,
        wav: list,
        bpm: list,
        plot_cont: bool
    ) -> np.ndarray:
        """
        Remove the continuum from the data.

        Parameters
        ----------
        spec : list
            The spectrum to remove the continuum from.
        wav : list
            The wavelength of the spectrum.
        bpm : list
            The bad pixel mask.
        plot_cont : bool
            Decides whether or not the continuum fitting should be
            ↪ plotted

        Returns
        -------
        None

        """
        # Mask out the bad pixels for fitting continua
        okwav = np.where(bpm != 1)

        # Define continua
        ctm = continuum(
            wav[okwav],
            spec[okwav],
            deg=self.cont_ord,
            plot=plot_cont,
        )

        # Normalise spectra
        spec /= chebyshev.chebval(wav, ctm)
        spec -= 1

        return spec

    # MARK: Correlate
    def correlate(
        self,
        filename1: Path,
        filename2: Path | None = None,
        alt: Callable = None
    ) -> None:
        """
```

```
332         Cross correlates the data.
333
334         Parameters
335         ----------
336         filename1 : Path
337             The name of the first FITS file to cross correlate.
338         filename2 : Path, optional
339             The name of the second FITS file to cross correlate.
340             (Defaults to None)
341         alt : Callable, optional
342             An alternate method of cross correlating the data.
343             (Defaults to None)
344
345         Returns
346         -------
347         None
348
349         """
350         #  mode: OE -> 'O1' & 'E1', O -> 'O1' & 'O2', E -> 'E1' & 'E2'
351         # Load data
352         spec, wav, bpm = self.load_file(filename1)
353         if filename2 and self.beams != 'OE':
354             unpack = lambda ext, *args: [arr[ext] for arr in args]
355
356             if self.beams == 'O':
357                 spec[-1], wav[-1], bpm[-1] = unpack(
358                     0, *self.load_file(filename2)
359                 )
360
361             else:
362                 spec[0], wav[0], bpm[0] = spec[-1], wav[-1], bpm[-1]
363                 spec[-1], wav[-1], bpm[-1] = unpack(
364                     -1, *self.load_file(filename2)
365                 )
366
367         bounds = self.get_bounds(bpm)
368
369         logging.debug(
370             f"correlate - data shape:\n\tspec/wav/bpm: {spec.shape}"
371         )
372
373         corrdb = [[] for _ in range(self.ccds)]
374         lagsdb = [[] for _ in range(self.ccds)]
375         for ccd in range(self.ccds):
376             sig = []
377             for ext in range(2):
378                 lb, ub = bounds[ext, ccd]
379
380                 if self.cont_ord > 0:
381                     spec[ext, lb:ub] = self.remove_cont(
382                         spec[ext, lb:ub],
383                         wav[ext, lb:ub],
384                         bpm[ext, lb:ub],
385                         self.can_plot
386                     )
387
388                 # Invert BPM (and account for 2); zero bad pixels
389                 sig.append((
```

```
                    spec[ext, lb:ub]
                    * abs(bpm[ext, lb:ub] * -1 + 1)
                ))

            # Finally(!!!) cross correlate signals and scale max -> 1
            corrdb[ccd] = signal.correlate(*sig) if not alt else
            ↪ alt(*sig)
            corrdb[ccd] /= np.max(corrdb[ccd])
            lagsdb[ccd] = signal.correlation_lags(
                sig[0].shape[-1],
                sig[1].shape[-1]
            ) * self.wav_cdelt

        return (spec, wav, bpm), (corrdb, lagsdb)

    # MARK: FTCS alternate
    def FTCS(
        self,
        signal1: np.ndarray,
        signal2: np.ndarray
    ) -> None:
        """
        Cross correlates the data using the Fourier Transform.

        Parameters
        ----------
        signal1 : np.ndarray
            The first signal to cross correlate.
        signal2 : np.ndarray
            The second signal to cross correlate.

        Returns
        -------
        np.ndarray
            The correlation data using the Fourier Transform.

        """
        logging.debug(
            f"FTCS - data shape:\n\tspec/wav/bpm: {signal1.shape}"
        )

        # Invert BPM (and account for 2); zero bad pixels
        ft_spec1 = np.fft.fft(signal1)
        ft_spec2 = np.fft.fft(signal2)

        if self.can_plot:
            plt.plot(ft_spec1)
            plt.plot(ft_spec2)
            plt.show()

        # Cross correlate signals
        # ft_spectrum1 * np.conj(ft_spectrum2)
        corr_entry = signal.correlate(ft_spec1, ft_spec2)

        return np.fft.ifft(corr_entry)

    # MARK: Plot
    def plot(self, spec, wav, bpm, corrdb, lagsdb) -> None:
```

```
447         """
448         Plot the data.
449 
450         Parameters
451         ----------
452         spec : np.ndarray
453             The spectrum.
454         wav : np.ndarray
455             The wavelength.
456         bpm : np.ndarray
457             The bad pixel mask.
458         corrdb : np.ndarray
459             The cross correlation data.
460         lagsdb : np.ndarray
461             The lags data.
462 
463         Returns
464         -------
465         None
466 
467         """
468         plt.style.use([
469             Path(__file__).parent.resolve() / 'utils/STOPS.mplstyle',
470             Path(__file__).parent.resolve() /
                ↪ 'utils/STOPS_correlate.mplstyle'
471         ])
472         bounds = self.get_bounds(bpm)
473 
474         fig, axs = plt.subplots(2, self.ccds, sharey="row")
475 
476         if self.ccds == 1:
477             # Convert axs to a 2D array
478             axs = np.swapaxes(np.atleast_2d(axs), 0, 1)
479 
480         # for ext, ccd in iters.product(range(2), range(self.ccds)):
481 
482         for ccd in range(self.ccds):
483             axs[0, ccd].plot(
484                 lagsdb[ccd],
485                 corrdb[ccd] * 100,
486                 color='C4',
487                 label=f"max lag @ {lagsdb[ccd][corrdb[ccd].argmax()] -
                    ↪ (bounds[1, ccd, 0] - bounds[0, ccd, 0])}",
488             )
489 
490             for ext in range(2):
491                 lb, ub = bounds[ext, ccd]
492                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub -
                    ↪ 1]}")
493 
494                 axs[1, ccd].plot(
495                     wav[ext, lb:ub],
496                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
                        ↪ OFFSET * ext,
497                     label=(
498                         f"${self.beams if self.beams != 'OE' else
                            ↪ self.beams[ext]}"
499                         f"_{ext + 1 if self.beams != 'OE' else 1}$"
```

```
                        f"{(' (+' + str(OFFSET * ext) + ')') if ext > 0
                        ↪ else ''}"
                    ),
                )

        axs[0, 0].set_ylabel("Normalised Correlation\n(\%)")
        for ax in axs[1:, 0]:
            ax.set_ylabel("Normalised Intensity\n(Counts)")
        xcol = int(self.ccds != 1)
        axs[0, xcol].set_xlabel(f"Signal Lag ({self.wav_unit})")
        axs[-1, xcol].set_xlabel(f"Wavelength ({self.wav_unit})")
        for ax in axs.flatten():
            leg = ax.legend()
            leg.set_draggable(True)

        # plt.tight_layout()
        # fig1 = plt.gcf()
        # DPI = fig1.get_dpi()
        # fig1.set_size_inches(700.0/float(DPI), 250.0/float(DPI))
        plt.show()

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

        # Handle save
        fig.savefig(fname=self.save_prefix)

        return

    # MARK: Process all listed images
    def process(self) -> None:
        """
        Process the data.

        Returns
        -------
        None

        """
        if self.beams != 'OE' and len(self.fits_list) == 1:
            # change mode to OE with warning
            logging.warning((
                f"`{self.beams}` correlation not possible for "
                "a single file. correlation `mode` changed to 'OE'."
            ))
            self.beams = 'OE'

        # OE `mode` (same file, diff. ext.)
        if self.beams == 'OE':
            for fl in self.fits_list:
                logging.info(f"'OE' correlation of {fl}.")
                (spec, wav, bpm), (corr, lags) = self.correlate(fl,
                ↪ alt=self.alt)
                self.plot(spec, wav, bpm, corr, lags)

            return

```

```
556         # O|E `mode` (diff. files, same ext.)
557         for fl1, fl2 in iters.combinations(self.fits_list, 2):
558             logging.info(f"{self.beams} correlation of {fl1} vs {fl2}.")
559             (spec, wav, bpm), (corr, lags) = self.correlate(fl1, fl2,
                ↪ alt=self.alt)
560             self.plot(spec, wav, bpm, corr, lags)
561 
562         return
563 
564 
565 # MARK: Main function
566 def main(argv) -> None:
567     return
568 
569 if __name__ == "__main__":
570     main(sys.argv[1:])
```

Listing B.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 __init__ import __author__, __email__, __version__

# MARK: Imports
import os
import sys
import logging
from pathlib import Path

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

from utils.SharedUtils import find_files, continuum
from utils.Constants import SAVE_SKY, FIND_PEAK_PARAMS, ARC_FILE

# print(
#  [logging.getLogger(name) for name in logging.root.manager.loggerDict]
# )
mpl_logger = logging.getLogger('matplotlib')
mpl_logger.setLevel(logging.INFO)
pil_logger = logging.getLogger('PIL')
pil_logger.setLevel(logging.INFO)
# plt.rcParams['figure.figsize'] = (20, 4)

# MARK: Skylines Class
class Skylines:

    #----------sky0----------

    """
    Class representing the Skylines object.

    Parameters
    ----------
    data_dir : Path
        The directory containing the data files.
    filenames : list[str]
        The list of filenames to be processed.
    beam : str, optional
        The beam mode, by default "OE".
    plot : bool, optional
        Flag indicating whether to plot the continuum, by default False.
    save_prefix : Path | None, optional
        The prefix for saving the data, by default None.
    **kwargs
        Additional keyword arguments.

    Attributes
```

```
    ----------
    data_dir : Path
        The directory containing the data files.
    fits_list : list[str]
        The list of fits file paths.
    beam : str
        The beam mode.
    can_plot : bool
        Flag indicating whether to plot the continuum.
    save_prefix : Path | None
        The prefix for saving the data.
    wav_unit : str
        The unit of wavelength.
    rawWav : np.ndarray
        The raw wavelength data.
    rawSpec : np.ndarray
        The raw spectral data.
    rawBpm : np.ndarray
        The raw bad pixel mask data.
    corrWav : np.ndarray
        The corrected wavelength data.
    corrSpec : np.ndarray
        The corrected spectral data.
    spec : np.ndarray
        The median spectrum.
    normSpec : np.ndarray
        The normalized spectrum.

    Methods
    -------
    checkLoad(self, path1: str) -> np.ndarray:
        Checks and loads the data from the given path.
    transform(self, wav_sol: np.ndarray, spec: np.ndarray) ->
    ↪ np.ndarray:
        Transforms the input wavelength and spectral data based on
        the given wavelength solution.
    rmvCont(self) -> np.ndarray:
        Removes the continuum from the spectrum.
    process(self) -> None:
        Placeholder method for processing the data.
    """

    #----------sky1----------

    # MARK: Skylines init
    def __init__(
        self,
        data_dir: Path,
        filenames : list[str],
        beams: str = "OE",
        split_ccd: bool = False,
        cont_ord: int = 11,
        plot: bool = False,
        transform: bool = True,
        save_prefix: Path | None = None,
        **kwargs,
    ) -> None:
        self.data_dir = data_dir
```

```
        self.fits_list, self.arc_list = find_files(
            data_dir=self.data_dir,
            filenames=filenames,
            prefix="wmxgbp", # t[o|e]beam
            ext="fits",
            sep_arc=True,
        )
        self._beams = None
        self.beams = beams
        self.ccds = 1
        if split_ccd:
            # See cross_correlate for initial implementation
            self.ccds = 3

        self.cont_ord = cont_ord
        self.can_plot = plot
        self.must_transform = transform

        self.save_prefix = save_prefix
        # Handle directory save name
        if self.save_prefix and self.save_prefix.is_dir():
            self.save_prefix /= SAVE_SKY
            logging.warning((
                f"Skylines save name resolves to a directory. "
                f"Saving under {self.save_prefix}"
            ))

        self.max_difference = 5

        self.wav_unit = "$\AA$"

        logging.debug("__init__ - \n", self.__dict__)

        return

    # MARK: Beams property
    @property
    def beams(self) -> str:
        return self._beams

    @beams.setter
    def beams(self, mode: str) -> None:
        if mode not in ['O', 'E', 'OE']:
            errMsg = f"Correlation mode '{mode}' not recognized."
            logging.error(errMsg)
            raise ValueError(errMsg)

        self._beams = mode

        return

    # MARK: Find Peaks
    def find_peaks(
        self,
        spec: np.ndarray,
        axis: int | None = None,
        min_height: float = 0.5,
        **kwargs,
```

```
172     ) -> tuple[np.ndarray, np.ndarray]:
173         """
174         Finds the peaks in the given spectral data.
175 
176         Parameters
177         ----------
178         spec : np.ndarray
179             The spectral data.
180         bpm : np.ndarray
181             The bad pixel mask.
182         min_height : float, optional
183             The minimum height of the peaks, by default 0.5.
184         rel_height : float, optional
185             The relative height of the peaks, by default 0.05.
186 
187         Returns
188         -------
189         peaks, properties : tuple[np.ndarray, dict]
190             The peaks and their properties.
191 
192         """
193         peaks = []
194         props = []
195 
196         for ext in range(len(self.beams)):
197             if axis is None:
198                 row_mean = spec[ext]
199             else:
200                 row_mean = np.mean(spec[ext], axis=axis)
201 
202             peak, property = signal.find_peaks(
203                 row_mean,
204                 prominence=min_height * np.max(row_mean),
205                 width=0,
206                 **kwargs,
207             )
208             peaks.append(peak)
209             props.append(property)
210 
211         if self.can_plot:
212             fig, axs = plt.subplots(2, 1)
213             for ext in range(len(self.beams)):
214                 axs[ext].plot(
215                     row_mean,
216                     label=f"{'E' if ext else 'O'}"
217                 )
218                 axs[ext].plot(
219                     peak,
220                     row_mean[peak],
221                     "x",
222                     label=f"{'E' if ext else 'O'} peaks"
223                 )
224                 axs[ext].legend()
225                 plt.show()
226 
227         logging.debug(f"find_peaks - peaks: {[len(i) for i in peaks]}")
228         logging.debug(f"find_peaks - props: {[key for key in
            ↪ props[0].keys()]}")
```

```
229
230         return peaks, props
231
232     # MARK: Min. of Diff. Matrix
233     @staticmethod
234     def min_diff_matrix(
235         A: np.ndarray,
236         B: np.ndarray,
237         max_diff: int = 100
238     ) -> np.ndarray:
239         """
240         Find the minimum difference between the elements of two arrays.
241
242         Parameters
243         ----------
244         A : np.ndarray
245             The first 1d array.
246         B : np.ndarray
247             The second 1d array.
248         max_diff : int, optional
249             The maximum difference allowed, by default 100.
250
251         Returns
252         -------
253         A : np.ndarray (len(A))
254             The elements of the first array.
255         min_vals : np.ndarray (len(A))
256             The minimum difference between the elements of the two
                ↪ arrays.
257         min_idxs : np.ndarray (len(A))
258             The indices of the minimum difference between
259             the elements of the two arrays.
260
261         """
262         # Compute the difference matrix using transpose
263         diff = A - B[:, np.newaxis]
264
265         # Find the minimum value in each row (A) of `diff`
266         min_idxs = np.abs(diff).argmin(axis=0)
267         print(min_idxs.shape, diff.shape)
268         min_vals = np.array([diff[j, i] for i, j in
            ↪ enumerate(min_idxs)])
269         # TODO: Recalculate min_val after selecting best min_val and
270         # removing the corresponding row/column
271
272         logging.debug(f"min_diff_matrix - min_vals: {np.round(min_vals,
            ↪ 2)}")
273         logging.debug(f"min_diff_matrix - min_idxs: {min_idxs}")
274
275         max_mask = (min_vals <= max_diff) & (min_vals >= -1 * max_diff)
276
277         return A[max_mask], min_vals[max_mask], min_idxs[max_mask]
278
279     # MARK: Load File Data
280     def load_file_data(
281         self,
282         filename: Path
283     ) -> tuple[np.ndarray, np.ndarray, np.ndarray]:
```

```
        """
        Loads the data from the given file.

        Parameters
        ----------
        filename : Path
            The path to the file to be loaded.

        Returns
        -------
        spec, wav, bpm : tuple[np.ndarray, np.ndarray, np.ndarray]
            The wavelength, spectral, and bad pixel mask data.

        """
        # Load data from self.beams extension
        with pyfits.open(filename) as hdul:
            exts = [0, 1] if len(self.beams) == 2 else 0 + self.beams
            ↪ == 'E'
            spec2D = np.atleast_3d(hdul["SCI"].data[exts])
            wav2D = np.atleast_3d(hdul["WAV"].data[exts])
            bpm2D = np.atleast_3d(hdul["BPM"].data[exts].astype(bool))

            logging.info(
                f"load_file_data - {filename.name} - shape:
                ↪ {spec2D.shape}"
            )

            return spec2D, wav2D, bpm2D

    # MARK: Load Sky or Arc Lines
    def load_lines(
            self,
            filename: Path | None = None,
            dtype: list[tuple] = [('wav', float), ('flux', float)],
            skip_header: int = 3,
            skip_footer: int = 1
        ) -> np.ndarray:
        """
        Loads the sky or arc lines from the given file.

        Parameters
        ----------
        filename : Path | None, optional
            The path to the file to be loaded.
            Defaults to loading the skylines from `utils/sky.salt`

        Returns
        -------
        sky_lines : np.ndarray['wav', 'flux']
            The sky lines from the file.

        """
        usecols = None
        if filename:
            filename = Path(__file__).parent.resolve() / filename
            usecols = (0, 1)
        else:
            filename = Path(__file__).parent.resolve() /
```

```
            ↪ 'utils/sky.salt'

        lines = np.genfromtxt(
            filename,
            dtype=dtype,
            skip_header=skip_header,
            skip_footer=skip_footer,
            usecols=usecols
        )

        logging.debug(
            f"load_lines - {filename.name} - shape: {lines.shape}"
        )

        return lines

    # MARK: Mask Traces
    def mask_traces(
            self,
            spec: np.ndarray,
            bpm: np.ndarray,
            max_traces: int = 1,
            tr_pad: int = 5,
            bg_margin: int = 10,
            lr_margins: list[int] = [10, 10],
            h_min: float = 0.5,
            h_rel: float = 1 - 0.05,
        ) -> np.ndarray:
        """
        Masks the traces in the bad pixel mask.

        Parameters
        ----------
        spec : np.ndarray
            The spectral data.
        bpm : np.ndarray
            The bad pixel mask.

        Returns
        -------
        bpm : np.ndarray
            The updated bad pixel mask.

        """
        # Base mask
        bpm[:, :bg_margin] = True
        bpm[:, -bg_margin:] = True
        bpm[:, :, :lr_margins[0]] = True
        bpm[:, :, -lr_margins[1]:] = True

        # Get the traces
        traces, tr_props = self.find_peaks(
            spec,
            axis=1,
            min_height=h_min,
            rel_height=h_rel
        )

```

```
397         for ext in range(len(self.beams)):
398             # Mask the traces
399             for i in range(len(traces[ext][:max_traces])):
400                 lb = max(
401                     0,
402                     int(tr_props[ext]['left_ips'][i]) - tr_pad
403                 )
404                 ub = min(
405                     spec.shape[-1],
406                     int(tr_props[ext]['right_ips'][i]) + tr_pad
407                 )
408                 bpm[ext, lb : ub] = True
409                 # TODO: Relocate targets after initial masking
410 
411         logging.info(f"mask_traces - {min(max_traces, len(traces))} of
            ↪ {len(traces)} traces masked.")
412 
413         return bpm
414 
415     # MARK: Transform Spectra
416     def transform(
417         self,
418         spec: np.ndarray,
419         wav_sol: np.ndarray,
420         row_max: int | None = None,
421         resPlot: bool = False,
422     ) -> np.ndarray:
423         """
424         Transforms the input wavelength and spectral data
425         based on the given wavelength solution.
426 
427         Parameters
428         ----------
429         spec : np.ndarray
430             The spectral data.
431         wav_sol : np.ndarray
432             The wavelength solution.
433         resPlot : bool, optional
434             Flag indicating whether to plot the results, default False.
435 
436         Returns
437         -------
438         spec, wav : np.ndarray
439             The transformed wavelength and spectral data.
440 
441         """
442         # Create arrays to return
443         cs = np.zeros_like(spec)
444         cw = np.zeros_like(wav_sol.mean(axis=1))
445 
446         for ext in range(len(self.beams)):
447 
448             if row_max:
449                 avg_max = row_max
450             else:
451                 # Get middle row (to interpolate the rest of the rows
                    ↪ to)
452                 avg_max = np.mean(spec[ext], axis=1).argmax()
```

```
453
454             # Correct extensions based on wavelength
455             # Get wavelength values at row with most trace
456             cw[ext] = wav_sol[ext, avg_max]
457
458             # Spec ext
459             for row in range(cs.shape[1]):
460                 cs[ext, row] = np.interp(
461                     cw[ext],
462                     wav_sol[ext, row],
463                     spec[ext, row]
464                 )
465                 # f_2d = interpolate.interp2d(
466                 #     wav_sol[ext, row],
467                 #     np.arange(rows),
468                 #     spec[ext],
469                 # )
470                 # cs[ext] = f_2d(cw[ext, row], np.arange(rows))
471
472         # Plot results
473         if resPlot:
474             fig, axs = plt.subplots(2, 1, figsize=[20, 4])
475             for ext in range(len(self.beams)):
476                 axs[ext].imshow(
477                     cs[ext],
478                     vmax=cs[ext].mean() + 2*cs[ext].std(),
479                     vmin=cs[ext].mean() - 2*cs[ext].std()
480                 )
481
482                 logging.debug(f"{'E' if ext else 'O'} Average continuum
                    ↪ = {np.median(np.median(cs[ext], axis=0)):4.3f}")
483
484                 axx = axs[ext].twinx()
485                 axx.hlines(
486                     np.median(np.median(cs[ext], axis=0)),
487                     0,
488                     cs[ext].shape[-1],
489                     colors='black'
490                 )
491                 axx.plot(
492                     cs[ext].mean(axis=0),
493                     "k",
494                     label=f"mean {'E' if ext else 'O'}"
495                 )
496                 axx.plot(
497                     np.median(cs[ext], axis=0),
498                     "r",
499                     label=f"median {'E' if ext else 'O'}"
500                 )
501                 axx.legend()
502             plt.show()
503
504         logging.info(f"transform - {cs.shape} transformed.")
505
506         return cs, cw
507
508     # MARK: Plot
509     def plot(
```

```
510             self,
511             spectra,
512             wavelengths,
513             peaks,
514             properties,
515             arc: bool = False,
516         ) -> None:
517         plt.style.use([
518             Path(__file__).parent.resolve() / 'utils/STOPS.mplstyle',
519             Path(__file__).parent.resolve() /
                ↪ 'utils/STOPS_skylines.mplstyle'
520         ])
521         plt.rcParams['figure.subplot.hspace'] *= len(self.beams)
522 
523         def norm(x):
524             return (x - np.min(x)) / (np.max(x) - np.min(x))
525 
526         # Load known lines
527         if arc:
528             lines =
                ↪ self.load_lines(filename=f'utils/RSS_arc_files/{ARC_FILE}')
529         else:
530             lines = self.load_lines()
531 
532         lines = lines[
533             (lines['wav'] > wavelengths[1][0][0].min()) &
534             (lines['wav'] < wavelengths[1][0][0].max())
535         ]
536 
537 
538         # Create plot for results
539         fig, axs = plt.subplots(2, self.ccds, sharex='col',
                ↪ sharey='row')
540 
541         # Convert axs to a 2D array if ccd count is 1
542         if self.ccds == 1:
543             axs = np.swapaxes(np.atleast_2d(axs), 0, 1)
544 
545         for fl in range(len(self.arc_list if arc else self.fits_list)):
546 
547             # set color cycle
548             color=next(axs[0, 0]._get_lines.prop_cycler)['color']
549 
550             for ext in range(len(self.beams)):
551 
552                 for ccd in range(self.ccds):
553 
554                     # MARK: plot spectrum
555                     # (transformed)
556                     ccdrange = spectra[1][fl][ext].shape[-1] //
                        ↪ self.ccds
557                     axs[0, ccd].plot(
558                         wavelengths[1][fl][ext][
559                             ccdrange*ccd:ccdrange*(ccd+1)
560                         ],
561                         norm(spectra[1][fl][ext][
562                             ccdrange*ccd:ccdrange*(ccd+1)
563                         ]) * 100 + 10 * ext + 30 * fl,
```

```
564                         color=color,
565                         linestyle='dashed' if ext else 'solid',
566                         label = f"${{{self.beams[ext]}}}_{{{fl +
                            ↪ 1}}}^{{+ {10*ext + 30*fl}}}$" if ccd == 0
                            ↪ else None,
567                     )
568
569                     # MARK: plot dev
570                     sky_wavs, dev, peak_idx = self.min_diff_matrix(
571                         lines['wav'],
572                         wavelengths[1][fl][ext][peaks[1][fl][ext]],
573                         max_diff=self.max_difference,
574                     )
575
576                     # # MARK: width/initial width
577                     # width = properties[1][fl][ext]['widths'][peak_idx]
578                     # width_i = np.zeros_like(width)
579
580                     # sky_i, i_dev, i_idx = self.min_diff_matrix(
581                     #     lines['wav'],
582                     #     wavelengths[0][fl][ext][peaks[0][fl][ext]],
583                     #     max_diff=self.max_difference,
584                     # )
585
586                     # width_i = np.array([
587                     #     properties[0][fl][ext]['widths'][
588                     #         np.where(wav == sky_i)[0][0]
589                     #     ]
590                     #     if wav in sky_i else 1000
591                     #     for wav in sky_wavs
592                     # ])
593                     # width_ratio = (width / width_i) - 1
594                     # width_ratio[width_ratio < 0] = 0
595
596                     # ylolims = width_ratio > self.max_difference
597                     # width_ratio[
598                     #     width_ratio > self.max_difference
599                     # ] = self.max_difference // 2
600
601                     ok = np.where(
602                         (sky_wavs >=
                            ↪ wavelengths[1][fl][ext].data[ccdrange*ccd]) &
603                         (sky_wavs <=
                            ↪ wavelengths[1][fl][ext].data[ccdrange*(ccd+1)])
604                     )
605                     axs[1, ccd].plot(
606                         sky_wavs[ok],
607                         dev[ok],
608                         # yerr=(width_ratio[ok] * 0, width_ratio[ok]),
609                         # lolims=ylolims[ok],
610                         "." if ext else "x",
611                         # fmt="." if ext else "x",
612                         alpha=0.8,
613                         color=color,
614                         # markeredgecolor='white',
615                         # markeredgewidth=0.5,
616                         # label=f"${self.beams[ext]}_{{{fl + 1}}}$",
617                     )
```

```
618
619                     logging.debug(f"plot - RMS:
                        ↪ {np.sqrt(np.mean(dev[ok]**2)):2.3f}")
620
621             for ccd in range(self.ccds):
622                 # spectrum
623                 ok = np.where(
624                     (lines['wav'] >=
                        ↪ wavelengths[1][fl][0].data[ccdrange*ccd]) &
625                     (lines['wav'] <=
                        ↪ wavelengths[1][fl][0].data[ccdrange*(ccd+1)])
626                 )
627                 axs[0, ccd].plot(
628                     lines['wav'][ok],
629                     lines['flux'][ok] * 0,
630                     'x',
631                     color='C4',
632                     label="\\textsc{salt}\nModel" if ccd == 0 else None,
633                 )
634                 for x in lines['wav'][ok]: axs[0, ccd].axvline(x,
                    ↪ ls='dashed', c='0.7')
635
636             axs[0, 0].set_ylabel("Rel. Intensity ($\%$)")
637             axs[1, 0].set_ylabel(
638                 "Closest Peak ($\Delta\lambda$)"
639             )
640             # for ax in axs[:, 0]:
641             #     ax.legend(loc='upper left', ncols=(fl + 1) * (ext + 1) +
                ↪ 1)
642             leg = fig.legend(
643                 loc='center',
644                 ncol=min(8, len(spectra[0]) + 1),
645                 columnspacing=0.5,
646                 bbox_to_anchor=(
647                     np.mean((
648                         plt.rcParams['figure.subplot.left'],
649                         plt.rcParams['figure.subplot.right']
650                     )),
651                     np.mean((
652                         plt.rcParams['figure.subplot.bottom'],
653                         plt.rcParams['figure.subplot.top']
654                     ))
655                 ),
656             )
657             leg.set_draggable(True)
658             for ax in axs[1, :]:
659                 ax.grid(axis='y')
660
661             # fig.add_subplot(111, frameon=False)
662             # # hide tick and tick label of the big axis
663             # plt.tick_params(
664             # labelcolor='none',
665             #    which='both',
666             #    top=False,
667             #    bottom=False,
668             #    left=False,
669             #    right=False
670             # )
```

```
671         axs[-1, 0 if self.ccds == 1 else 1].set_xlabel(
672             f"Wavelength ({self.wav_unit})"
673         )
674 
675         # plt.tight_layout()
676 
677         plt.show()
678 
679         # Save results
680         if self.save_prefix:
681             fig.savefig(fname=self.save_prefix)
682 
683         return
684 
685     # MARK: Process all listed images
686     def process(self, arc: bool=False) -> None:
687         files = self.fits_list
688         if arc:
689             files = self.arc_list
690 
691         logging.info(f"Processing '{self.beams}' lines.")
692 
693         spectra = [[], []]
694         wavs = [[], []]
695         peaks = [[], []]
696         peak_props = [[], []]
697 
698         for fl in files:
699             # Load data
700             spec2d, wav2d, bpm2d = self.load_file_data(fl)
701 
702             # Mask traces in BPM
703             bpm2d = self.mask_traces(
704                 spec2d,
705                 bpm2d,
706                 max_traces=0,
707                 bg_margin=15,
708                 h_min=0.05
709             )
710             m_spec2d = np.ma.masked_array(spec2d, mask=bpm2d) # spec2d
711             m_wav2d = np.ma.masked_array(wav2d, mask=bpm2d) # wav2d
712 
713             # Initial spectra
714             spec_i = np.mean(m_spec2d, axis=-2)
715             wav_i = np.mean(m_wav2d, axis=-2)
716 
717             # Transform data
718             t_spec2d, t_wav = self.transform(
719                 m_spec2d,
720                 m_wav2d,
721                 resPlot=self.can_plot
722             )
723 
724             # Final spectra
725             spec_f = np.mean(t_spec2d, axis=-2)
726             wav_f = t_wav
727 
728             # Find peaks
```

```
729             peaks_i, props_i = self.find_peaks(
730                 spec_i,
731                 **FIND_PEAK_PARAMS
732             )
733             peaks_f, props_f = self.find_peaks(
734                 spec_f,
735                 **FIND_PEAK_PARAMS
736             )
737 
738             spectra[0].append([*spec_i])
739             spectra[1].append([*spec_f])
740             wavs[0].append([*wav_i])
741             wavs[1].append([*wav_f])
742             peaks[0].append([*peaks_i])
743             peaks[1].append([*peaks_f])
744             peak_props[0].append([*props_i])
745             peak_props[1].append([*props_f])
746 
747         # Plot results
748         self.plot(spectra, wavs, peaks, peak_props, arc=arc)
749 
750         if arc:
751             return
752         elif self.arc_list:
753             self.process(arc=True)
754 
755         return
756 
757 
758 # MARK: Main function
759 def main(argv) -> None:
760     return
761 
762 if __name__ == "__main__":
763     main(sys.argv[1:])
```

Bibliography

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

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

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

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

Whitmore, J. Woillez, V. Zabalza, and Astropy Contributors. The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package. AJ, 156(3):123, September 2018. doi: 10.3847/1538-3881/aabc4f.

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

S. Bagnulo, M. Landolfi, J. D. Landstreet, E. Landi Degl'Innocenti, L. Fossati, and M. Sterzik. Stellar spectropolarimetry with retarder waveplate and beam splitter devices. Publications of the Astronomical Society of the Pacific, 121(883):993, aug 2009. doi: 10.1086/605654. URL `https://dx.doi.org/10.1086/605654`.

Erasmus Bartholinus. Experimenta crystalli islandici dis-diaclastici, quibus mira et insolita refractio detegitur (copenhagen, 1670). Edinburgh Philosophical Journal, 1:271, 1670.

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

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

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

D. A. H. Buckley, J. Brink, N. S. Loaring, A. Swat, and H. L. Worters. The Southern African Large Telescope (SALT) calibration system. In Ian S. McLean and Mark M. Casali, editors, Ground-based and Airborne Instrumentation for Astronomy II, volume 7014, page 70146H. International Society for Optics and Photonics, SPIE, 2008. doi: 10.1117/12.790385. URL `https://doi.org/10.1117/12.790385`.

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.

M. Craig, S. Crawford, M. Seifert, T. Robitaille, B. Sipőcz, J. Walawender, Z. Vinícius,

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

G. Dahlquist and Å. Björck. Numerical Methods. Dover Books on Mathematics. Dover Publications, 2003. ISBN 9780486428079. URL https://books.google.co.ls/books?id=armfeHpJIwAC.

E. Landi Degl'Innocenti, S. Bagnulo, and L. Fossati. Polarimetric standardization, 2006.

Egidio Landi Degl'Innocenti. The physics of polarization. Proceedings of the International Astronomical Union, 10(S305):1–1, 2014.

Egidio Landi Degl'Innocenti and M. Landolfi. Polarization in Spectral Lines, volume 307. Springer Dordrecht, 2004. doi: 10.1007/978-1-4020-2415-3.

Königlich Bayerische Akademie der Wissenschaften. Denkschriften der Königlichen Akademie der Wissenschaften zu München für das Jahre 1820 und 1821, volume 8. Die Akademie, 1824. URL https://books.google.co.za/books?id=k-EAAAAAYAAJ.

J. F. Donati, M. Semel, B. D. Carter, D. E. Rees, and A. Collier Cameron. Spectropolarimetric observations of active stars. MNRAS, 291(4):658–682, November 1997. doi: 10.1093/mnras/291.4.658.

I. V. Florinsky and A. N. Pankratov. Digital terrain modeling with the chebyshev polynomials. Machine Learning and Data Analysis, 1(12):1647 – 1659, 2015. doi: 10.48550/ARXIV.1507.03960. URL https://arxiv.org/abs/1507.03960.

Augustin Fresnel. Oeuvres completes d'Augustin Fresnel: 3. Imprimerie impériale, 1870.

L. M. Freyhammer, M. I. Andersen, T. Arentoft, C. Sterken, and P. Nørregaard. On Cross-talk Correction of Images from Multiple-port CCDs. Experimental Astronomy, 12(3):147–162, January 2001. doi: 10.1023/A:1021820418263.

David J Griffiths. Introduction to electrodynamics, 2005.

George E. Hale. The Zeeman Effect in the Sun. PASP, 20(123):287, December 1908. doi: 10.1086/121847.

George E. Hale. 16. On the Probable Existence of a Magnetic Field in Sun-Spots, pages 96–105. Harvard University Press, Cambridge, MA and London, England, 1979. ISBN 9780674366688. doi: doi:10.4159/harvard.9780674366688.c19. URL https://doi.org/10.4159/harvard.9780674366688.c19.

P. D. Hale and G. W. Day. Stability of birefringent linear retarders(waveplates). Appl.

Opt., 27(24):5146–5153, Dec 1988. doi: 10.1364/AO.27.005146. URL https://opg.optica.org/ao/abstract.cfm?URI=ao-27-24-5146.

C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. Nature, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

E. Hecht. Optics. Pearson Education, Incorporated, 2017. ISBN 9780133977226. URL https://books.google.co.za/books?id=ZarLoQEACAAJ.

Steve B. Howell. Handbook of CCD Astronomy, volume 5. Cambridge University Press, 2006.

J. D. Hunter. Matplotlib: A 2d graphics environment. Computing in Science & Engineering, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

Christian Huygens. Treatise on light, 1690. translated by Thompson, s. p., 1690. URL https://www.gutenberg.org/files/14725/14725-h/14725-h.htm.

Mourad E. H. Ismail. Classical and Quantum Orthogonal Polynomials in One Variable. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2005. doi: 10.1017/CBO9781107325982.

James Janesick, James T. Andrews, and Tom Elliott. Fundamental performance differences between CMOS and CCD imagers: Part 1. In David A. Dorn and Andrew D. Holland, editors, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, volume 6276 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, page 62760M, June 2006. doi: 10.1117/12.678867.

F.A. Jenkins and H.E. White. Fundamentals of Optics. International student edition. McGraw-Hill, 1976. ISBN 9780070323308. URL https://books.google.co.za/books?id=dCdRAAAAMAAJ.

Christoph U. Keller. Instrumentation for astrophysical spectropolarimetry. Astrophysical Spectropolarimetry, 1:303–354, 2002.

G. Kirchhoff and R. Bunsen. Chemische Analyse durch Spectralbeobachtungen. Annalen der Physik, 189(7):337–381, January 1861. doi: 10.1002/andp.18611890702.

Henry A. Kobulnicky, Kenneth H. Nordsieck, Eric B. Burgh, Michael P. Smith, Jeffrey W. Percival, Ted B. Williams, and Darragh O'Donoghue. Prime focus imaging spectrograph for the Southern African large telescope: operational modes. In Masanori Iye and Alan F. M. Moorwood, editors, Instrument Design and Performance for Optical/Infrared Ground-based Telescopes, volume 4841 of Society of Photo-Optical Instrumentation

Engineers (SPIE) Conference Series, pages 1634–1644, March 2003. doi: 10.1117/12.460315.

Gerard Leng. Compression of aircraft aerodynamic database using multivariable chebyshev polynomials. Advances in Engineering Software, 28(2):133–141, 1997. ISSN 0965-9978. doi: https://doi.org/10.1016/S0965-9978(96)00043-9. URL https://www.sciencedirect.com/science/article/pii/S0965997896000439.

Dave Litwiller. Ccd vs. cmos. Photonics spectra, 35(1):154–158, 2001.

Dongyue Liu and Bryan M. Hennelly. Improved wavelength calibration by modeling the spectrometer. Applied Spectroscopy, 76(11):1283–1299, 2022. doi: 10.1177/00037028221111796. URL https://doi.org/10.1177/00037028221111796. PMID: 35726593.

Etienne L. Malus. Sur une propriété de la lumière réfléchie. Mém. Phys. Chim. Soc. d'Arcueil, 2:143–158, 1809.

Curtis McCully, Steve Crawford, Gabor Kovacs, Erik Tollerud, Edward Betts, Larry Bradley, Matt Craig, James Turner, Ole Streicher, Brigitta Sipocz, Thomas Robitaille, and Christoph Deil. astropy/astroscrappy: v1.0.5 zenodo release, November 2018. URL https://doi.org/10.5281/zenodo.1482019.

I. Newton and W. Innys. Opticks:: Or, A Treatise of the Reflections, Refractions, Inflections and Colours of Light. Opticks:: Or, A Treatise of the Reflections, Refractions, Inflections and Colours of Light. William Innys at the West-End of St. Paul's., 1730. URL https://books.google.co.za/books?id=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.

Doug Tody. The IRAF Data Reduction and Analysis System. In David L. Crawford, editor, Instrumentation in astronomy VI, volume 627 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, page 733, January 1986. doi: 10.1117/12.968154.

Doug Tody. IRAF in the Nineties. In R. J. Hanisch, R. J. V. Brissenden, and J. Barnes, ed-

itors, Astronomical Data Analysis Software and Systems II, volume 52 of Astronomical Society of the Pacific Conference Series, page 173, January 1993.

Stephen F. Tonkin. Practical Amateur Spectroscopy. The Patrick Moore Practical Astronomy Series. Springer London, 2013. ISBN 9781447101277. URL https://books.google.fr/books?id=b2fgBwAAQBAJ.

Pieter G. van Dokkum. Cosmic-Ray Rejection by Laplacian Edge Detection. PASP, 113 (789):1420–1427, November 2001. doi: 10.1086/323894.

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

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

Marsha J. Wolf, Matthew A. 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.

Glossary

Johnson-Cousins photometric system

U The ultraviolet filter, typically centered around 3640 Å.

B The blue filter, typically centered around 4420 Å.

V The visual filter, designed to approximate human visual sensitivity and typically centered around 5400 Å.

R The red filter, typically centered around 6580 Å.

I The infrared filter, typically centered around 8060 Å.