

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: June 28, 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 → 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

1	Introduction	1
2	Spectropolarimetry and the SALT RSS	3
2.1	Spectroscopy	3
2.1.1	Telescope Optics	3
2.1.2	Slit	4
2.1.3	Collimator	4
2.1.4	Dispersion Element	4
2.1.5	Camera Optics	5
2.1.6	Detector	5
2.1.7	Dispersion of Light	5
2.1.8	Detector and Spectroscopic Calibrations	9
2.2	Polarimetry	15
2.2.1	Polarization	16
2.2.2	Polarization Measurement	19
2.2.3	Polarimetric Calibrations	22
2.3	Spectropolarimetry	23
2.3.1	Spectropolarimetric Measurement	24
2.3.2	Spectropolarimetric Calibrations	25
2.4	The Southern African Large Telescope	26
2.4.1	The Primary Mirror	26
2.4.2	Tracker and Tracking	27
2.4.3	SALT Instrumentation	28
3	Existing and Developed Software	31
3.1	POLSALT	31
3.1.1	Basic CCD Reductions	32
3.1.2	Wavelength Calibrations	32
3.1.3	Spectral Extraction	33
3.1.4	Raw Stokes Calculations	33
3.1.5	Final Stokes Calculations	34
3.1.6	Visualization	34
3.1.7	Post-Processing Analysis	34
3.1.8	POLSALT Limitations and the Need for Supplementary Tools	35
3.2	IRAF	36
3.2.1	Identify	37

3.2.2	Reidentify	39
3.2.3	Fitcoords	39
3.2.4	Transform	39
3.3	STOPS	41
3.3.1	Splitting	42
3.3.2	Joining	45
3.3.3	Sky Line Checks	46
3.3.4	Cross Correlation	48
3.4	General Reduction Procedure	51
3.4.1	Initial Setup	51
3.4.2	POLSLAT Pre-Reductions	52
3.4.3	Wavelength Calibration	53
3.4.4	POLSLAT Reduction Completion	55
4	Testing and Application	57
5	Conclusions	59
5.1	Future Work	59
A	The Modified Reduction Process	61
B	STOPS Source Code	69
	Bibliography	117

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
POLSLAT	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 POLSLAT Spectro-polarimetry

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.

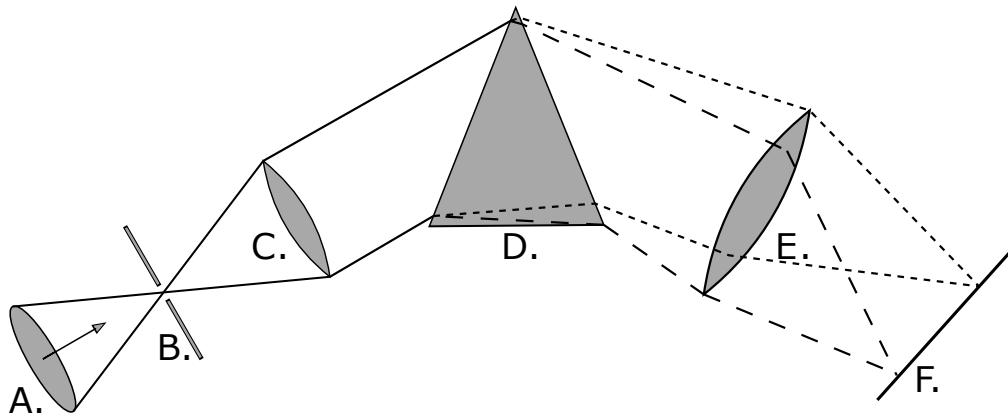


Figure 2.1: Layout depicting the light path through a spectrometer. Diagram adapted from Birney et al. (2006).

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 minuscule losses.

2.1.3 Collimator

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

2.1.4 Dispersion Element

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

2.1.5 Camera Optics

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

2.1.6 Detector

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

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

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

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:

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

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:

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

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

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

which shows that the angular dispersion of a prism is wavelength dependent and furthermore that longer wavelengths are dispersed less than shorter wavelengths (Birney et al., 2006; Hecht, 2017). The dependence of the angular dispersion, $d\theta/d\lambda$, on the wavelength, 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

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



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

Diffraction Grating

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

An example of a reflective blazed grating is illustrated in Figure 2.3. Here a monochromatic beam is incident on the grating at an angle of 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:

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

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:

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

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,

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



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

Substituting m/σ with the grating equation results in

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

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

Alternate Diffraction Elements

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

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

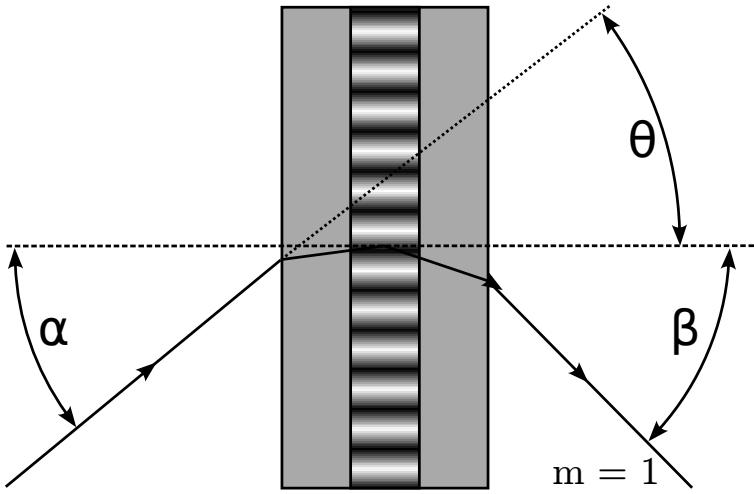


Figure 2.5: Diagram of a VPH grating for an incident monochromatic beam of light. Diagram adapted from Birney et al. (2006).

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

2.1.8 Detector and Spectroscopic Calibrations

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

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

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



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 telescope's 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.

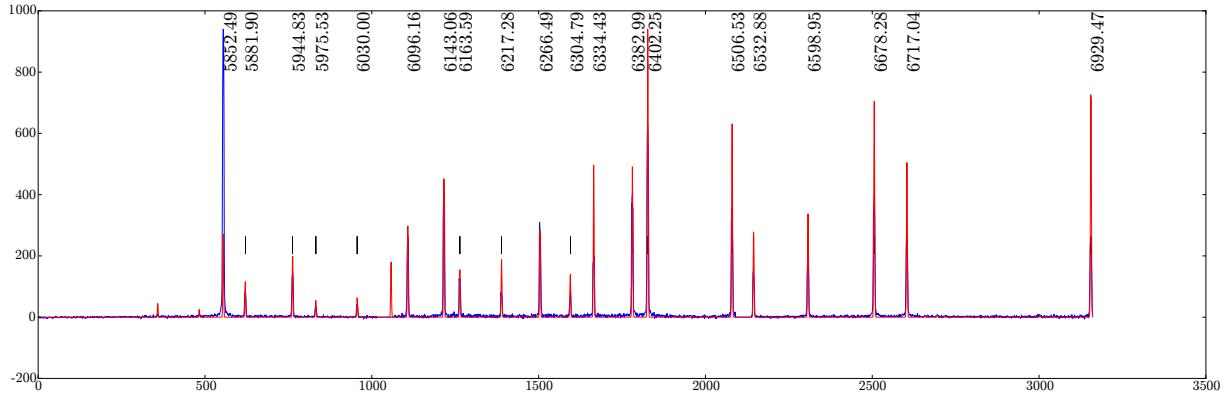


Figure 2.7: Example of an arc spectrum for NeAr taken with SALT’s RSS using the PG1800 grating at a grating angle of 34.625° , an articulation angle of 69.258° , and covering a wavelength range of $\sim 5600 - 6900 \text{ \AA}$. Plot adapted from SALT’s published Longslit Line Atlases, (2023).²

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 $\sim 20^\circ$ 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, $F_\lambda^n(x, y)$, can be calculated as:

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

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

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

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

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

²NeAr 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.

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

Chebyshev Polynomials The Chebyshev polynomials are defined explicitly as:

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

or recursively as:

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

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:

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

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

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

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

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

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

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

⁴Chebyshev 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:

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

or recursively as:

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

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:

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

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:

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

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

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

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

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

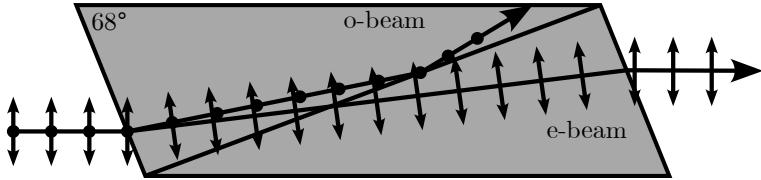


Figure 2.10: Diagram of a Nicol prism for incident non-polarized light. Diagram adapted from Fred the Oyster, CC BY-SA 4.0, via Wikimedia Commons (2023).

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:

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

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

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

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:

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

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

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

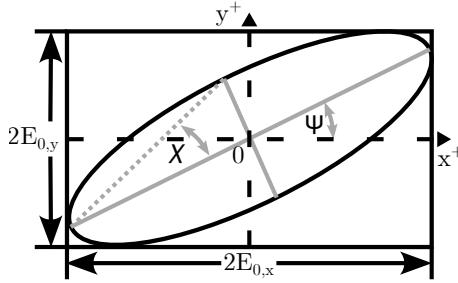


Figure 2.11: The polarization ellipse for an electric field vector propagating through free space. Diagram adapted from Inductiveload, PDM 1.0, via Wikimedia Commons (2023).

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:

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

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:

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

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

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

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

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

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

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

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

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

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

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

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

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

where I denotes the total intensity, P denotes the degree of polarization, or the ratio of polarized to non-polarized light in the wave-packet, 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 ($\mathbf{M}_0, \dots, \mathbf{M}_N$) results in an outgoing beam \mathbf{S}' such that:

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

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

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

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

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 retardance 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 \leftrightarrow and \odot 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:

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

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

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

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

Wollaston Prisms The Wollaston prism consists of two right-angle prisms consisting of a birefringent monoaxial material, cemented together with an optically transparent adhesive along their hypotenuses with their optical axes orthogonal, as seen in Figure 2.15. The Wollaston prism is a common optical polarizing element in astrophysical polarimetry which separates an incident beam into two linearly polarized O - and E -beams, orthogonal to one another, and deviated from their common axis equally. The deviation angle of the



Figure 2.15: Diagram of a Wollaston prism. Included in the diagram are the optical axes of the differing sections of the birefringent material as well as polarizing directions of the incident beam, denoted using the \leftrightarrow and $\downarrow\uparrow$ 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 Romanielo, 2006; Peinado et al., 2010).

Dual-Beam Extraction and Alignment

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

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

Sky Subtraction

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

2.3 Spectropolarimetry

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

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

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



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:

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

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

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

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

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

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

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

which allows Equation 2.41 to be written, as

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

in terms of the normalized Stokes parameters, or, alternatively, the degree of polarization, P , and polarization angle, 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:

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

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

2.3.2 Spectropolarimetric Calibrations

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

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



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).

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

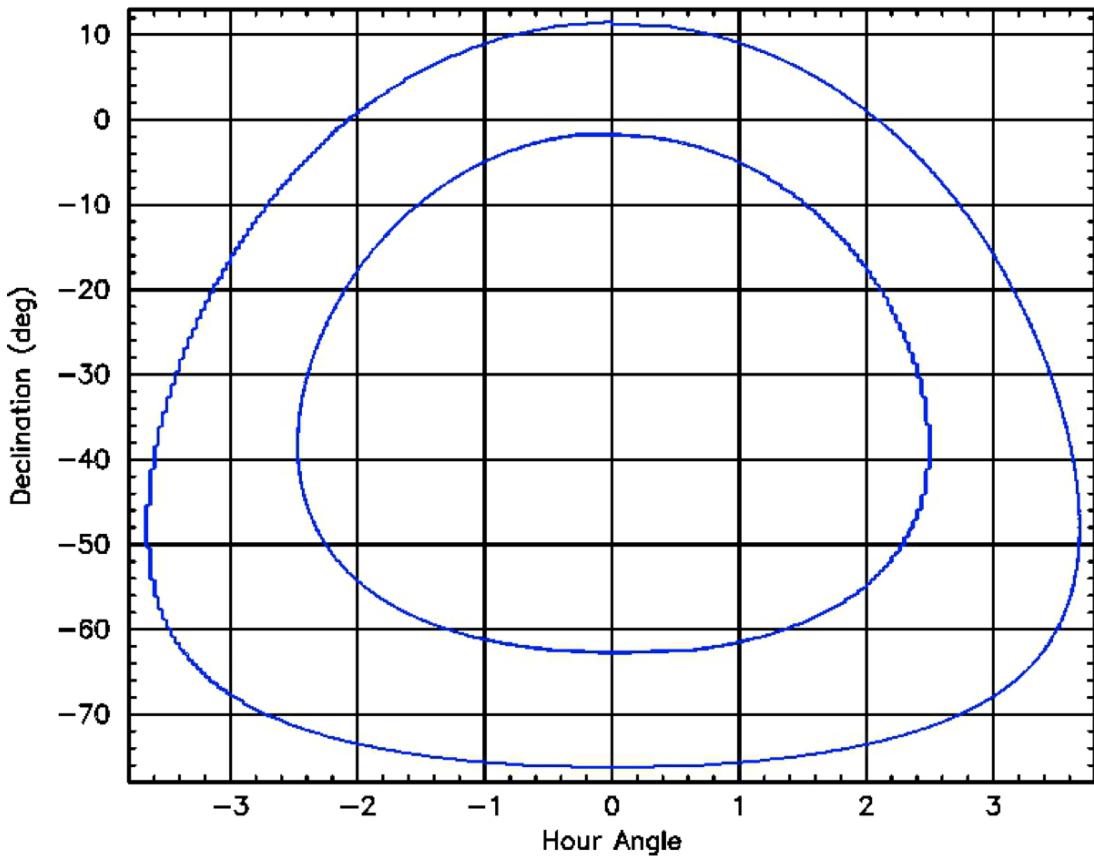


Figure 2.18: The visibility annulus of objects observable by SALT. Figure adapted from the SALT call for proposals (2013).⁶

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 $5 \mu\text{m}$ and is capable of tracking $\pm 6^\circ$ 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 $\sim 7 \text{ m}$ 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 \AA . 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).

⁶https://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.

⁷https://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

⁸The 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.

⁹Commission status sourced from the latest ‘Polarimetry Observers Guide’ (2024).

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

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

Chapter 3

Existing and Developed Software: An Overview of 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

¹POLSALT 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.

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

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

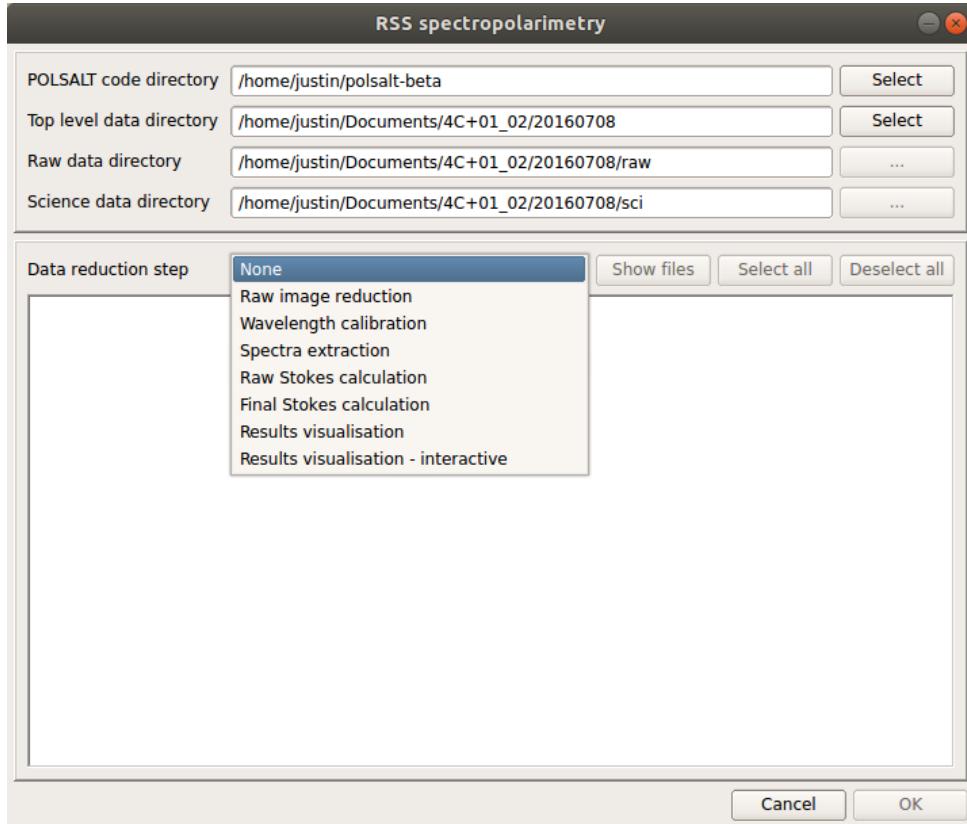


Figure 3.1: The layout of the `POLSALT` Graphical User Interface (GUI), including the contents of the reduction steps accessible via the dropdown menu. Note that there is no trailing forward slash after the ‘Top level data directory’. Figure created from a local instance of the `POLSALT` GUI.

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.

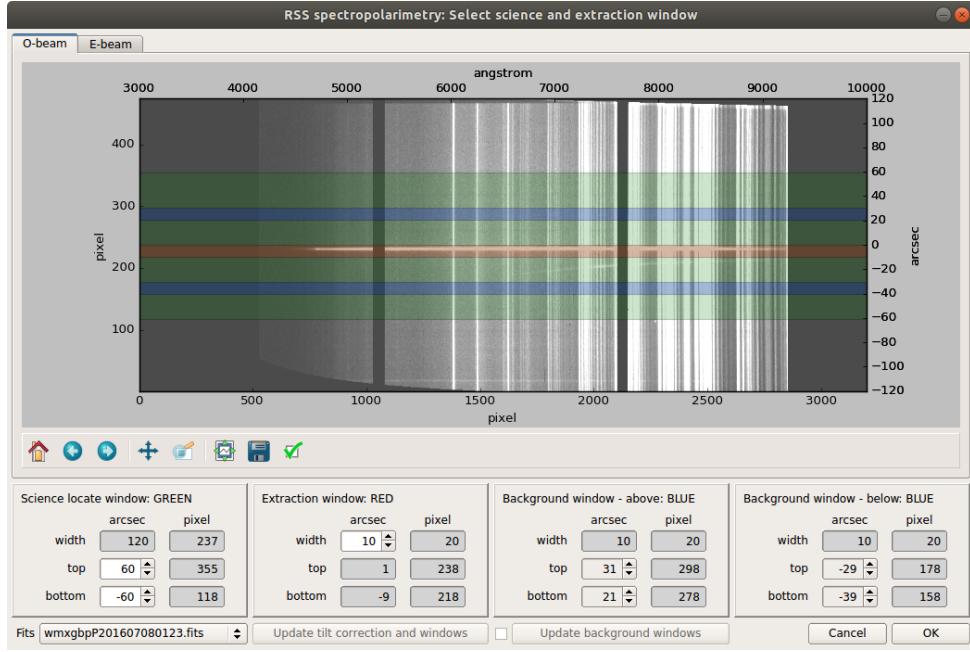


Figure 3.2: The layout of the interactive POLSALT spectra extraction GUI after selecting the ‘update tilt correction and windows’ button along the bottom border of the window. Figure created from a local instance of the POLSALT GUI.

3.1.3 Spectral Extraction

Background subtraction and spectral extraction is run via `specpoleextract_dev.py` which corrects for the beam-splitter distortion and tilt, performs sky subtraction, and extracts a one dimensional wavelength dependent spectrum for each beam sub-extension. This step is interactive with Figure 3.2 depicting 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:

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

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

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



Figure 3.3: The layout of the interactive POLSALT visualization GUI after selecting the ‘Plot results’ button along the bottom border of the window. Figure created from 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 uses `specpolview.py` and generates a plot of the Intensity, Linear Polarization (%), and Equatorial Polarization Angle ($^\circ$) against a shared wavelength axis, as seen in Figure 3.4. This step is interactive and various options, such as the wavelength range, binning, etc., are available.

3.1.7 Post-Processing Analysis

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

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



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

Synthetic filtering is calculated via `specpolfilter.py` and computes the synthetically filtered polarization results. 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 SALTICAM *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 any amount of reductions larger than a handful of observations. This is particularly true for observations performed with the SALT PG0300 grating as the sparse spectral features of the Ar arc lamp are not handled well by the POLSALT pipeline.

Since PG0300 provided the widest wavelength range and highest throughput, it was almost exclusively used for observations of flaring blazars, resulting in a large backlog of unanalyzed data. The only arc available for the PG0300 grating with a close enough articulation and grating angle ($\sim 10.68^\circ$ and $\sim 5.38^\circ$, respectively), was the 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 result in large deviations in regions with no spectral features. To only further compound the difficulty of the wavelength calibrations, the spectrum of the Ar arc lamp contains a partial overlap of a higher order at large wavelengths (§ 2.1.7, Equation 2.5).

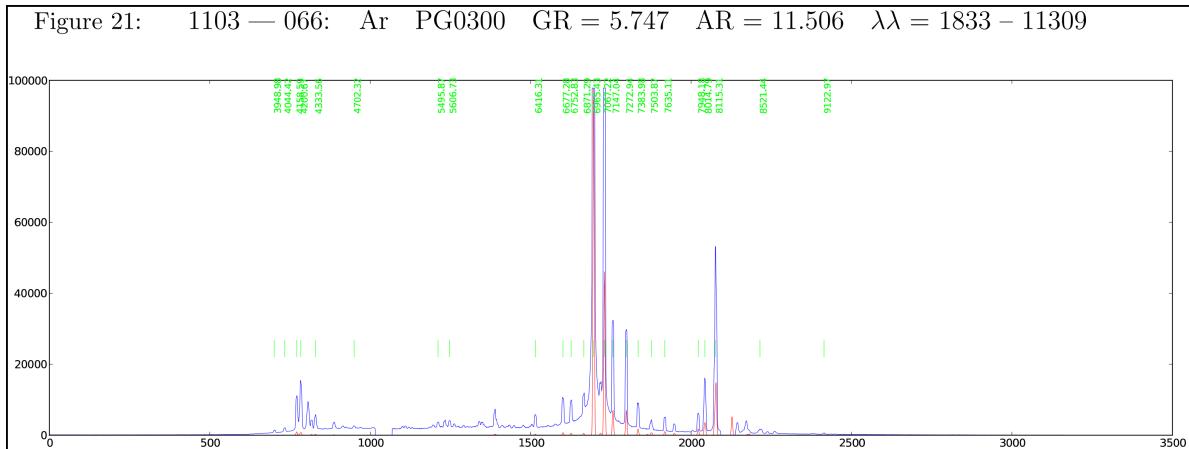


Figure 3.5: One of many Ar arc lamp spectra as provided by SALT for line identification. Plot adapted from SALT’s published Longslit Line Atlases (as of 2024), resized to fit within the document margins but otherwise unchanged.⁴

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 allows the rows or columns of an image to be interactively displayed,⁷ and the `eparam`

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

⁵Online help for IRAF is available at <https://iraf.net/irafdocs/>

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

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

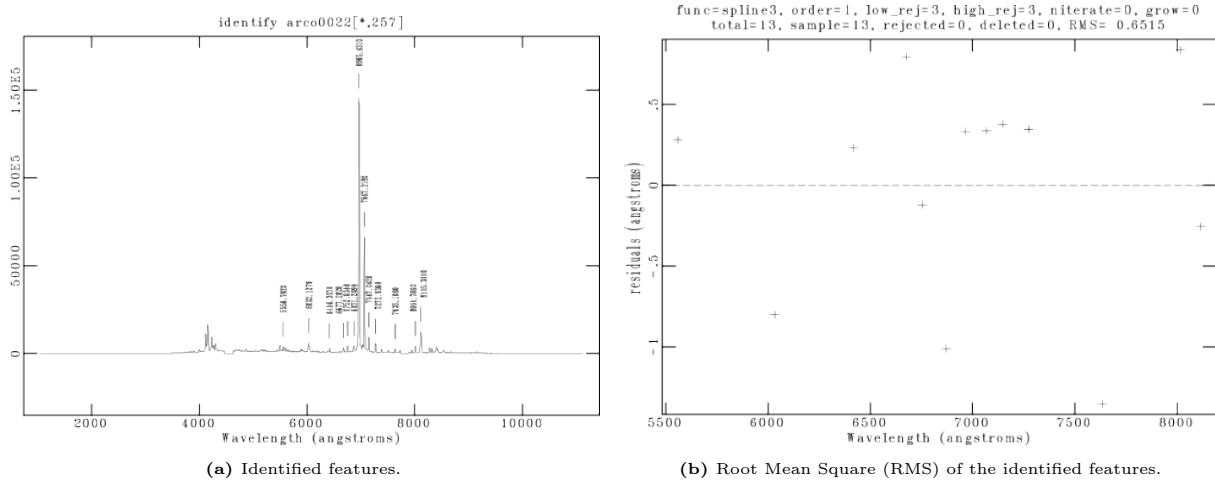


Figure 3.6: A plot and the RMS of the identified features found using the IRAF `identify` task. Figures created using the IRAF `identify` task.

task in the `language` package which allows the parameters of a task to be edited within the IRAF CLI.⁸

For wavelength calibrations 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 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 identified features which negatively impact the wavelength solution. A balance must be found between the number of identified features and parameters of the fit against the deviation of the fit from the known features.

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

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

¹⁰See 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 exposure at a defined (row) interval.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.

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.

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

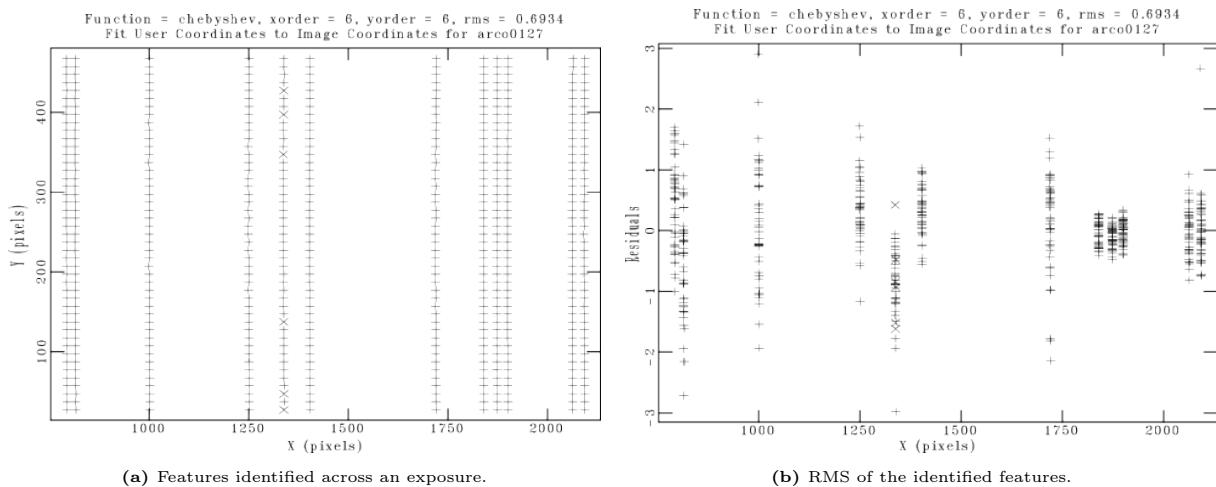
¹²Help documentation for the `fitcoords` task may be found at https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.twodspec.longslit.fitcoords.html.

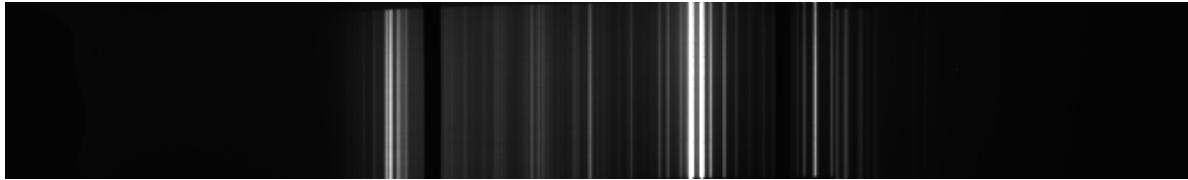
¹³See also <https://iraf.net/irafdocs/formats/fitcoords.php> for an explanation of the database contents.

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

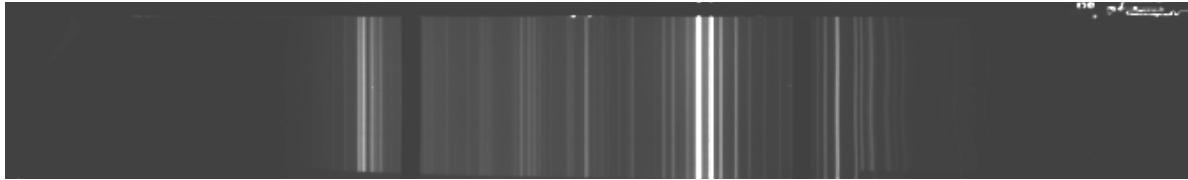
Listing 3.2: An example of the `fitcoords` database contents.¹³

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

**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) displaying a poorly fit wavelength solution.

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.

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.

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. This is especially necessary when considering the amount of exposures taken for long term studies.

To aid the scripting of the IRAF wavelength calibration process, the `split` method also performs row cropping to exclude CCD regions 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.

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

```

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

```

Listing 3.4: The ‘docstring’ for join.py

```

32 """
33
34 The `Join` class allows for the joining of previously split files and the
35 appending of an external wavelength solution in the `polsalt` FITS file format.
36
37 Parameters
38 -----
39 data_dir : str
40     The path to the data to be joined
41 database : str, optional
42     The name of the `IRAF` database folder.
43     (The default is "database")
44 fits_list : list[str], optional
45     A list of pre-reduced `polsalt` FITS files to be joined within `data_dir`.
46     (The default is ``None``, `Join` will search for `m*gbp*.fits` files)
47 solutions_list: list[str], optional
48     A list of solution filenames from which the wavelength solution is created.
49     (The default is ``None``, `Join` will search for `fc*` files within the `database` directory)
50 split_row : int, optional
51     The row along which the data of each extension in the FITS file was split.
52     Necessary when Joining cropped files.
53     (The default is 517, the SALT RSS CCD's middle row)
54 save_prefix : dict[str, list[str]], optional
55     The prefix with which the previously split `O`- & `E`-beams were saved.
56     Used for detecting if cropping was applied during the splitting procedure.
57     (The default is SAVE_PREFIX (See Notes))
58 verbose : int, optional
59     The level of verbosity to use for the Cosmic ray rejection
60     (The default is 30, I.E. logging.INFO)
61
62 Attributes
63 -----
64 fc_files : list[str]
65     Valid solutions found from `solutions_list`.
66 custom : bool
67     Internal flag for whether `solutions_list` uses the `IRAF` or a custom format.
68     See Notes for custom solution formatting.
69     (Default (inherited from `solutions_list`) is False)
70 arc : str
71     Deprecated. Name of arc FITS file within `data_dir`.
72 data_dir
73 database
74 fits_list
75 split_row
76 save_prefix
77
78 Methods
79 -----
80 get_solutions(wavlist: list / None, prefix: str = "fc") -> (fc_files, custom): tuple[list[str], bool]
81     Parse `solutions_list` and return valid solution files and if they are non-`IRAF` solutions.
82 parse_solution(fc_file: str, xshape: int, yshape: int) -> tuple[dict[str, int], np.ndarray]
83     Loads the wavelength solution file and parses keywords necessary for creating the wavelength extension.
84 join_file(file: os.PathLike) -> None
85     Joins the files,
86     attaches the wavelength solutions,
87     performs cosmic ray cleaning,
88     masks the extension,
89     and checks cropping performed in `Split`.
90     Writes the FITS file in a `polsalt` valid format.
91 check_crop(hdu: pyfits.HDUList, o_file: str, e_file: str) -> int
92     Opens the split `O`- and `E`-beam FITS files and returns the amount of cropping that was performed.
93 process() -> None
94     Calls `join_file` on each file in `fits_list` for automation.
95
96
97 Other Parameters
98 -----
99 no_arc : bool, optional
100     Deprecated. Decides whether the arc frames should be processed.
101     (The default is False, `polsalt` has no use for the arc after wavelength calibrations)
102 **kwargs : dict
103     keyword arguments. Allows for passing unpacked dictionary to the class constructor.
104
105 Notes
106 -----
107 Constants Imported (See utils.Constants):
108     DATADIR, SAVE_PREFIX, SPLIT_ROW, CR_PARAMS
109
110 Custom wavelength solutions must be formatted containing:
111     `x`, `y`, *coefficients...
112 where a solution are of order (x by y) and must contain x*y coefficients,
113 all separated by newlines. The name of the custom wavelength solution file
114 must contain either "cheb" or "leg" for Chebychev or Legendre
115 wavelength solutions, respectively.
116
117 Cosmic ray rejection is performed using lacosmic [1]_ implemented in ccdproc via astroscreappy [2]_.
118
119 References
120 -----
121 .. [1] van Dokkum 2001, PASP, 113, 789, 1420 (article : https://adsabs.harvard.edu/abs/2001PASP..113.1420V)
122 .. [2] https://zenodo.org/records/1482019
123
124 """

```

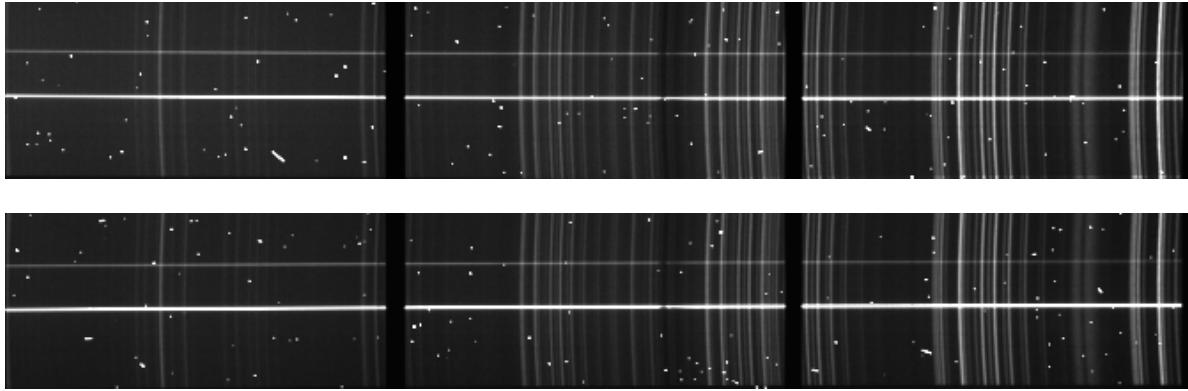


Figure 3.9: The split *O*- and *E*-beams as handed to IRAF. Figure created from the `STOPS split` method output.

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 suggestions¹⁵ 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

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

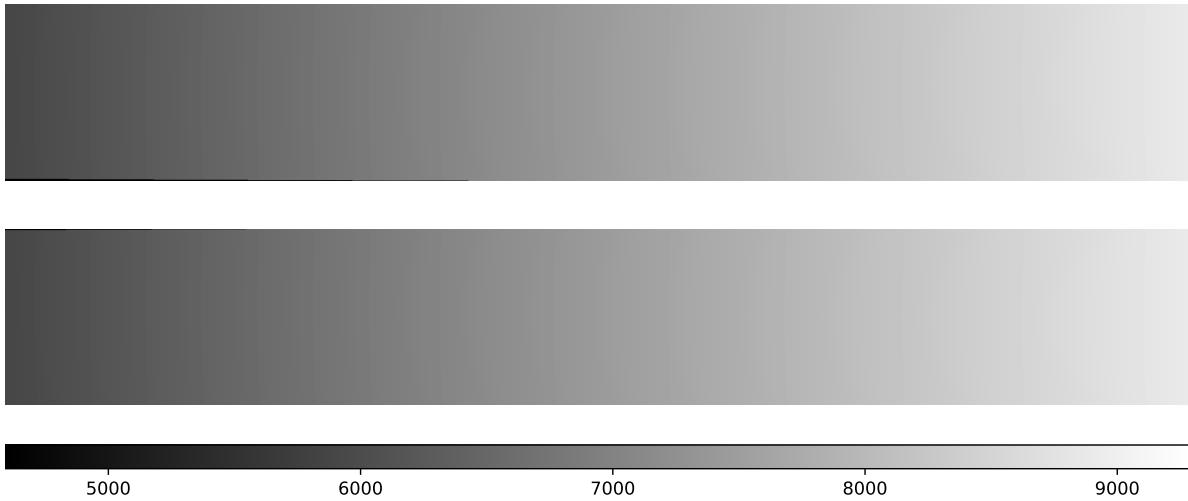


Figure 3.10: A representative wavelength extension of a FITS file, for the *O*- (top) and *E*-beam (bottom), ready to be processed by the POLSALT pipeline. The color bar shows the wavelength in Å. Note that regions which fall outside the exposed region are masked by setting the corresponding pixel values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

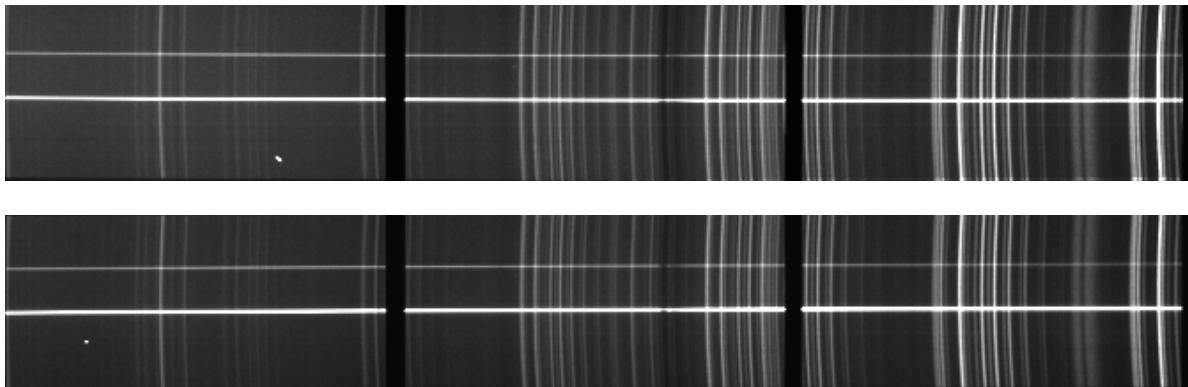


Figure 3.11: A representative science extension of a FITS file, for the *O*- (top) and *E*-beam (bottom), ready to be processed by the POLSALT pipeline. The observed intensity is displayed via the grayscale value at each pixel. Figure created from the `STOPSjoin` method output.

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.

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.

¹⁶Both 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     Class representing the Skylines object.
39
40     Parameters
41     -----
42     data_dir : Path
43         The directory containing the data files.
44     filenames : list[str]
45         The list of filenames to be processed.
46     beam : str, optional
47         The beam mode, by default "OE".
48     plot : bool, optional
49         Flag indicating whether to plot the continuum, by default False.
50     save_prefix : Path / None, optional
51         The prefix for saving the data, by default None.
52     **kwargs
53         Additional keyword arguments.
54
55     Attributes
56     -----
57     data_dir : Path
58         The directory containing the data files.
59     fits_list : list[str]
60         The list of fits file paths.
61     beam : str
62         The beam mode.
63     can_plot : bool
64         Flag indicating whether to plot the continuum.
65     save_prefix : Path / None
66         The prefix for saving the data.
67     wav_unit : str
68         The unit of wavelength.
69     rawWav : np.ndarray
70         The raw wavelength data.
71     rawSpec : np.ndarray
72         The raw spectral data.
73     rawBpm : np.ndarray
74         The raw bad pixel mask data.
75     corrWav : np.ndarray
76         The corrected wavelength data.
77     corrSpec : np.ndarray
78         The corrected spectral data.
79     spec : np.ndarray
80         The median spectrum.
81     normSpec : np.ndarray
82         The normalized spectrum.
83
84     Methods
85     -----
86     checkLoad(self, path1: str) -> np.ndarray:
87         Checks and loads the data from the given path.
88     transform(self, wav_sol: np.ndarray, spec: np.ndarray) -> np.ndarray:
89         Transforms the input wavelength and spectral data based on
90         the given wavelength solution.
91     rmvCont(self) -> np.ndarray:
92         Removes the continuum from the spectrum.
93     process(self) -> None:
94         Placeholder method for processing the data.
95
96 """

```

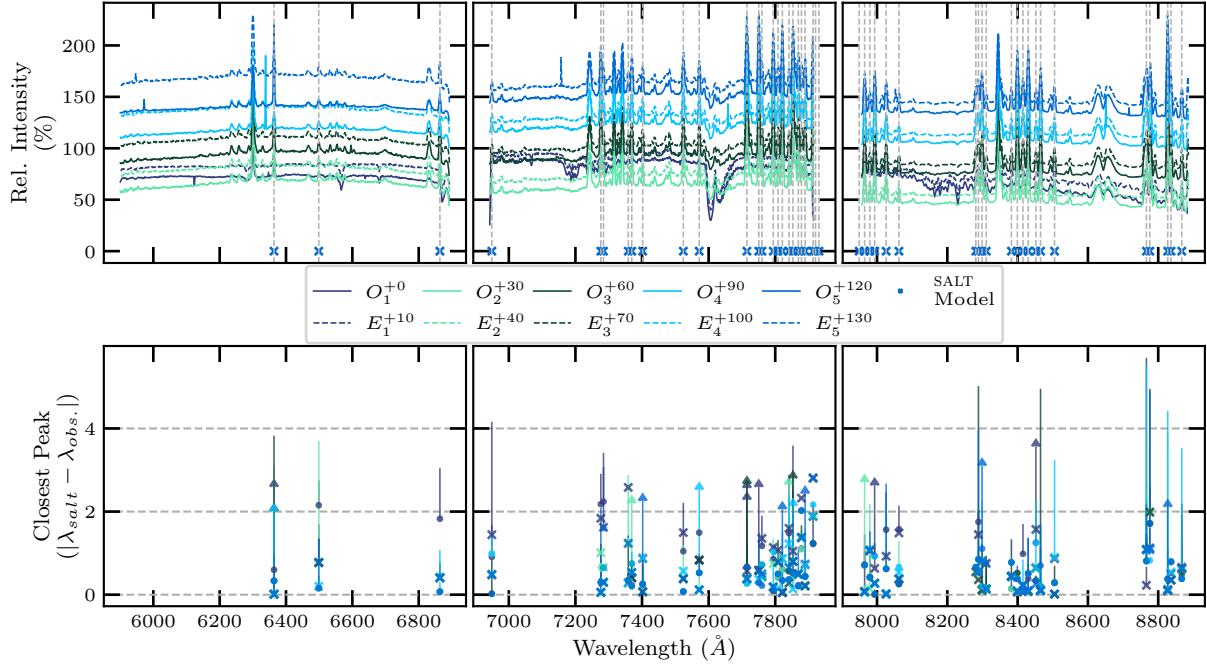


Figure 3.12: An example of a well calibrated wavelength solution, specifically shown for science images. Figure created via the `STOPS skyline` method.

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 $(\text{\AA}, y_p)$ 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 and are heavily influenced by the differences in the spectra of the different O - and E -beams, a direct comparison is not appropriate. Any observed unpolarized light, however, will reflect equally in both polarization beams and so the general trend of the two spectra may reasonably be expected to follow one another. The `correlate` method was created to allow for comparisons between the wavelength solutions of the O - and E -beams of a single exposure or the O - or E -beams of differing exposures by cross correlating the spectra.

The `correlate` method loads the `POLSALT spectra extraction` FITS files, removes the continuum and separates the CCD regions. The relevant CCD regions are cross

¹⁷The 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 Cross correlate allows for comparing the extensions of multiple
36 FITS files, or comparing the O and E beams of a single FITS file.
37
38 Parameters
39 -----
40 data_dir : str
41     The path to the data to be cross correlated
42 filenames : list[str]
43     The ecwmagbp*.fits files to be cross correlated.
44     If only one filename is defined, correlation is done against the two polarization beams.
45 split_ccd : bool, optional
46     Decides whether the CCD regions should each be individually cross correlated.
47     (The default is True, which splits the spectrum up into its separate CCD regions)
48 cont_ord : int, optional
49     The degree of a chebyshev to fit to the continuum.
50     (The default is 11)
51 plot : bool, optional
52     Decides whether or not the continuum fitting should be plotted
53     (The default is False, so no continua plots are displayed)
54 save_prefix : str, optional
55     The name or directory to save the figure produced to.
56     "" saves a default name to the current working. A default name is also used when save_prefix is a directory.
57     (The default is None, I.E. The figure is not saved, only displayed)
58
59 Attributes
60 -----
61 data_dir
62 fits_list
63 beams : str
64     The mode of correlation.
65     'OE' for same file, and 'O' or 'E' for different files but same ext's.
66 ccds : int
67     The number of CCD's in the data. Used to split the CCD's if split_ccd is True.
68 cont_ord : int
69     The degree of the chebyshev to fit to the continuum.
70 can_plot : bool
71     Decides whether or not the continuum fitting should be plotted
72 offset : int
73     The amount the spectrum is shifted, mainly to test the effect of the cross correlation
74     (The default is 0, I.E. no offset introduced)
75 save_prefix
76 wav_unit : str
77     The units of the wavelength axis.
78     (The default is Angstroms)
79 wav_cdel : int
80     The wavelength increment.
81     (The default is 1)
82 alt : Callable
83     An alternate method of cross correlating the data.
84     (The default is None)
85
86 Methods
87 -----
88 load_file(filename: Path) -> tuple[np.ndarray, np.ndarray, np.ndarray]
89     Loads the data from a FITS file.
90 get_bounds(bpm: np.ndarray) -> np.ndarray
91     Finds the bounds for the CCD regions.
92 remove_cont(spec: list, wav: list, bpm: list, plot_cont: bool) -> None
93     Removes the continuum from the data.
94 correlate(filename1: Path, filename2: Path | None = None) -> None
95     Cross correlates the data.
96 FTCS(filename1: Path, filename2: Path | None = None) -> None
97     Cross correlates the data using the Fourier Transform.
98 plot(spec, wav, bpm, corrdb, lagsdb) -> None
99     Plots the data.
100 process() -> None
101     Processes the data.
102
103 Other Parameters
104 -----
105 offset : int, optional
106     The amount the spectrum is shifted, mainly to test the effect of the cross correlation
107     (The default is 0, I.E. no offset introduced)
108 **kwargs : dict
109     keyword arguments. Allows for passing unpacked dictionary to the class constructor.
110 FTCS : bool, optional
111     Decides whether the Fourier Transform should be used for cross correlation.
112
113 See Also
114 -----
115 scipy
116     https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.correlate.html
117
118 Notes
119 -----
120 Constants Imported (See utils.Constants):
121     SAVE_CORR
122
123 """

```

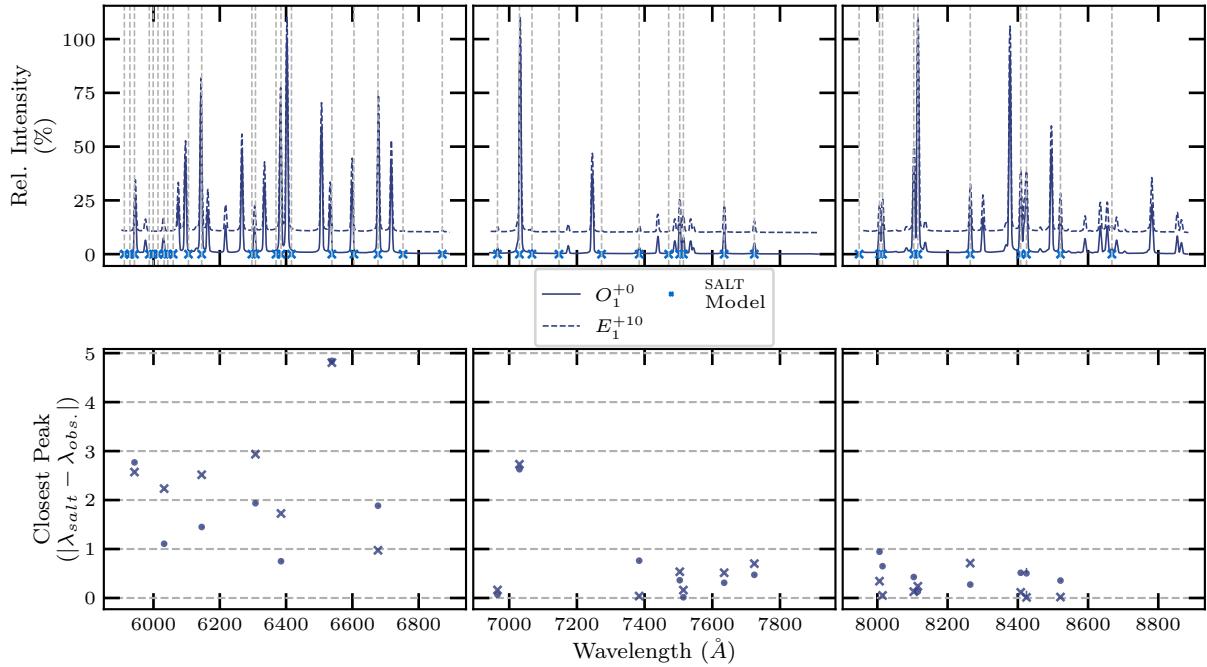


Figure 3.13: An example of a wavelength solution with a poor fit at lower wavelengths (bottom left), specifically shown for an arc image. Figure created via the `STOPS skyline` method.

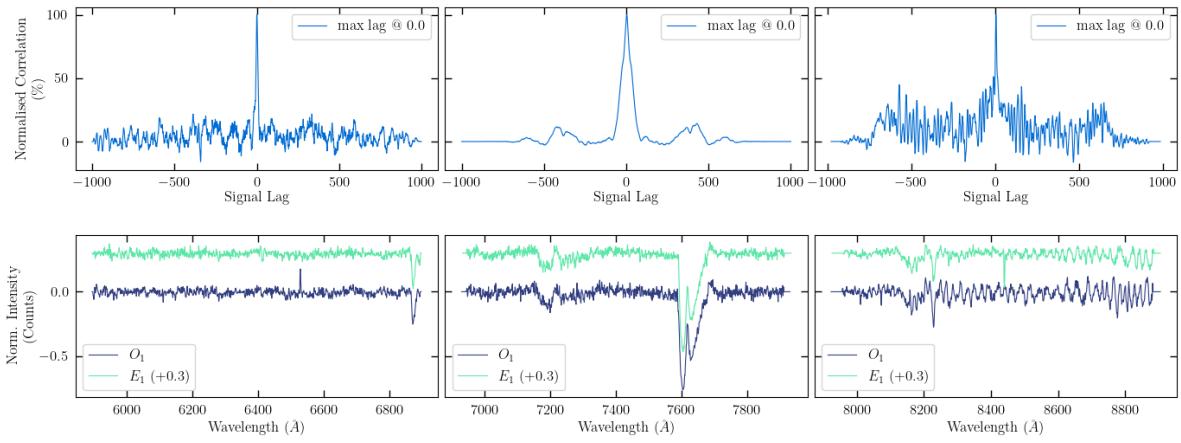


Figure 3.14: The resultant output plot of the `STOPS correlate` method. Figure created via the `STOPS correlate` method.

correlated and the correlation peak is plotted and specified in the plot legend, as seen in 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.

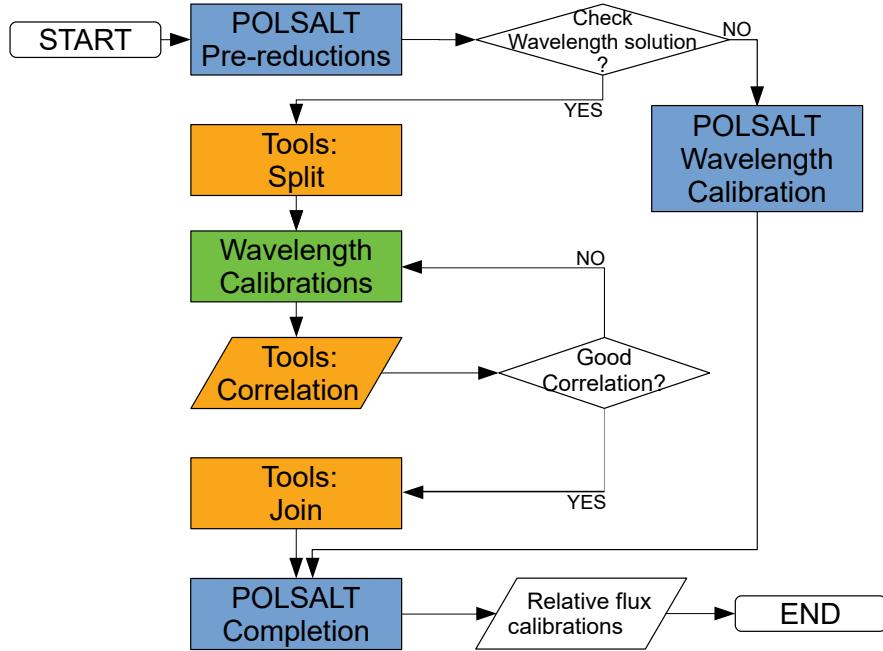


Figure 3.15: A general workflow for data reductions using a combination of POLSALT, IRAF, and the developed supplementary tools. Diagram adapted from Cooper et al. (2022).

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 Listing in § 3.4.2 below), the `anaconda` environment is activated at the start of the Listing, otherwise it is assumed the previously specified environment is still active.

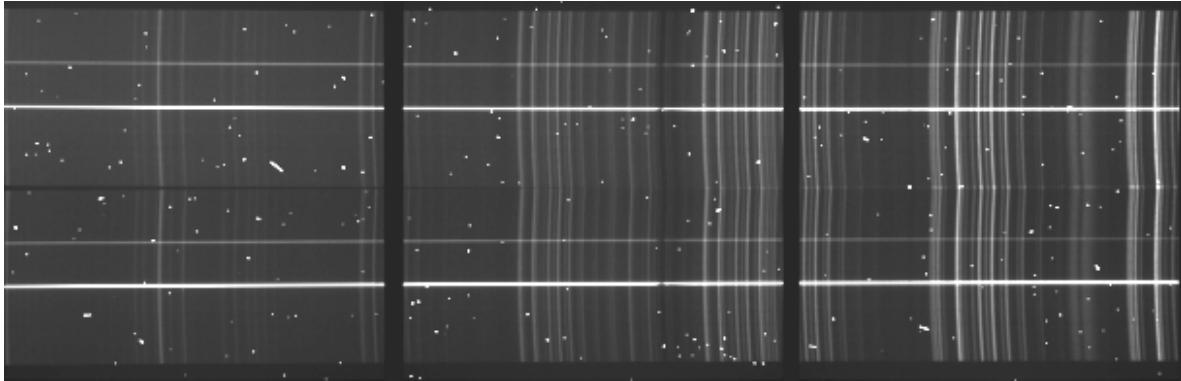


Figure 3.16: The science extension of a typical spectropolarimetric FITS file taken with the SALT RSS, after basic POLSALT CCD reductions have been completed. Figure created from the `STOPs split` output.

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

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 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.

¹⁸Please 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. 66 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 `reducetpoldata_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 reducetpoldata_sc.py YYYYMMDD
```

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

The POLSALT beta `reducetpoldata_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

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

²⁰Please 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

Short intro to chapter contents.

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

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

Chapter 5

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 ‘c1>’ and ‘>>>’ tokens are used for IRAF’s ‘xgterm’ single- and multi-line commands, respectively.

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

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

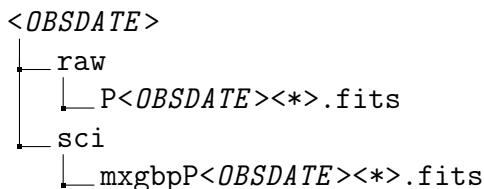


Figure A.1: The typical minimal file structure of data provided by SALT.

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` task¹ 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$idheneare.dat", units="", nsum="10", match=-3.,
>>> maxfeatures=50, zwidth=100., ftype="emission", fwidth=8.,
>>> cradius=5., threshold=0., minsep=2., function="spline3", order=2,
>>> sample="*", niterate=0, low_reject=3., high_reject=3., grow=0.,
>>> autowrite=no, graphics="stdgraph", cursor="", aidpars="")
```

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

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$idheneare.dat",
>>> match=-3., maxfeatures=50, minsep=2., database="database",
>>> logfiles="logfile", plotfile="", verbose=yes, graphics="stdgraph",
>>> cursor="", aidpars="")
```

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

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

³See 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 $\langle O\text{-beam } ARC \rangle$ with $\langle E\text{-beam } ARC \rangle$ 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 $\langle O\text{-beam } ARC \rangle$ with $\langle E\text{-beam } ARC \rangle$.

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 $\langle O\text{-beam } FILES \rangle$ and $\langle O\text{-beam } ARC \rangle$ with $\langle E\text{-beam } FILES \rangle$ and $\langle E\text{-beam } ARC \rangle$ 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

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

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

⁶See 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’.

⁷The 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.

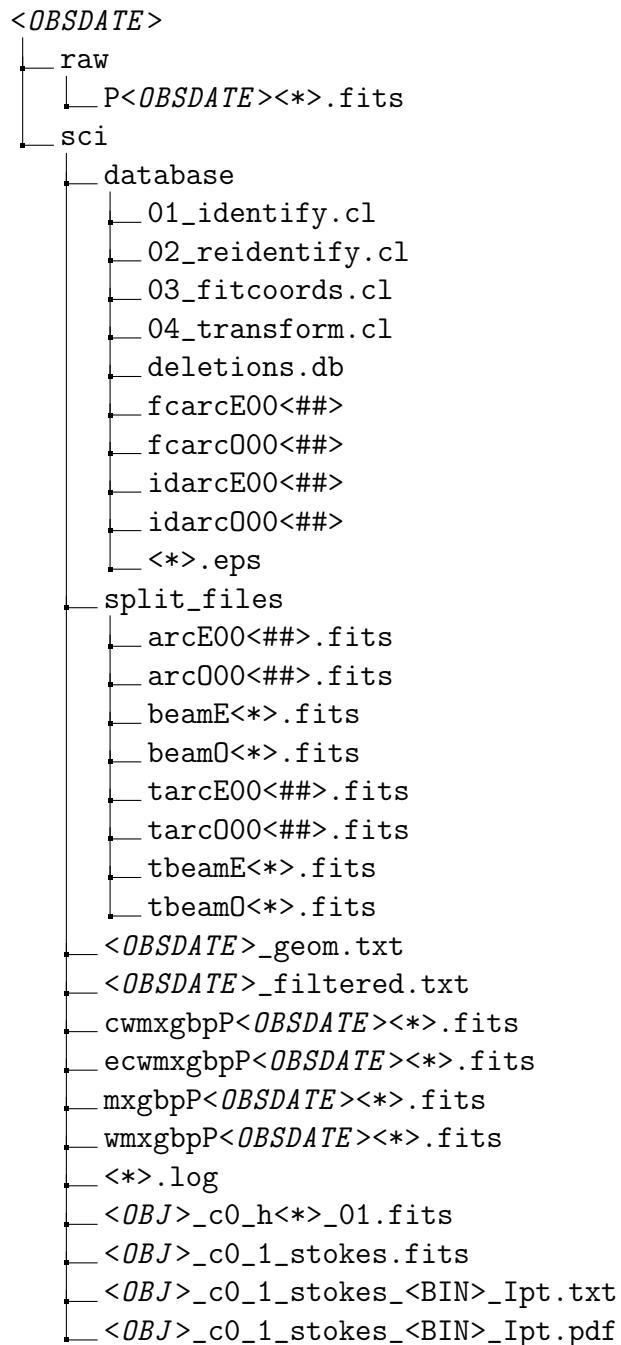


Figure A.2: The typical file structure after completing the reduction process.

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`

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

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

Listing B.2: The source code for `split.py`

```

1  """Module for splitting ``polsalt`` FITS files."""
2
3 #!/usr/bin/env python3
4 # -*- coding: utf-8 -*-
5
6 from __init__ import __author__, __email__, __version__
7
8 # MARK: Imports
9 import os
10 import sys
11 import logging
12 from copy import deepcopy
13 from pathlib import Path
14
15 import numpy as np
16 from astropy.io import fits as pyfits
17
18 from utils.SharedUtils import find_files, find_arc
19 from utils.Constants import SAVE_PREFIX, CROP_DEFAULT, SPLIT_ROW
20
21
22 # MARK: Split Class
23 class Split:
24
25     #-----split0-----
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
41         FITS file.
42         (The default is SPLIT_ROW (See Notes), the SALT RSS CCD's
43         middle row)
44     no_arc : bool, optional
45         Decides whether the arc frames should be recombined.
46         (The default is False, `polsalt` has no use for the arc after
47         wavelength calibrations)
48     save_prefix : dict[str, list[str]], optional
49         The prefix with which to save the O & E beams.
50         Setting `save_prefix` = ``None`` does not save the split O & E
51         beams.
52         (The default is SAVE_PREFIX (See Notes))
53
54     Attributes
55 
```

```

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


```

```

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

```

```

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

```

```

219     padbins = (ind_rc < offset) | (ind_rc > rows + offset)
220
221     # Roll split_row to be centre row
222     image_rc = np.roll(hdu[ext].data[:rows, :], -offset, axis=0)
223     image_rc[padbins] = 0.0
224
225     # Split columns equally
226     hdu[ext].data = image_rc.reshape((2, int(rows / 2), cols))
227
228     return hdu
229
230
231     # MARK: Crop files
232     def crop_file(
233         self,
234         hdulist: pyfits.HDUList,
235         crop: int = CROP_DEFAULT
236     ) -> tuple[np.ndarray]:
237         """
238             Crop the data with respect to the `O`/`E` beam.
239
240             Parameters
241             -----
242             hdulist : astropy.io.fits.HDUList
243                 The HDUList containing the data to be cropped.
244             crop : int, optional
245                 The number of rows to be cropped from the bottom and top
246                 of the `O` and `E` beam, respectively.
247                 (Defaults to 40)
248
249             Returns
250             -----
251             tuple[numpy.ndarray]
252                 Tuple containing the cropped O and E beam data arrays.
253
254             """
255             o_data = hdulist["SCI"].data[1, 0:-crop]
256             e_data = hdulist["SCI"].data[0, crop:]
257
258             return o_data, e_data
259
260     # MARK: Update beam lists
261     def update_beam_lists(
262         self,
263         o_name,
264         e_name,
265         arc: bool = True
266     ) -> None:
267         """
268             Update the `o_files` and `e_files` attributes.
269
270             Parameters
271             -----
272             o_name : str
273                 The filename of the O beam.
274             e_name : str
275                 The filename of the E beam.
276             arc : bool, optional
277                 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`

```

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

```



```

99      -----
100     no_arc : bool, optional
101         Deprecated. Decides whether the arc frames should be processed.
102         (The default is False, `polsalt` has no use for the arc after
103         ↪ wavelength calibrations)
104     **kwargs : dict
105         keyword arguments. Allows for passing unpacked dictionary to
106         ↪ the class constructor.
107
108     Notes
109     -----
110     Constants Imported (See utils.Constants):
111         DATADIR, SAVE_PREFIX, SPLIT_ROW, CR_PARAMS
112
113     Custom wavelength solutions must be formatted containing:
114         `x`, `y`, *coefficients...
115     where a solution are of order (`x` by `y`) and must contain x*y
116     ↪ coefficients,
117     all separated by newlines. The name of the custom wavelength
118     ↪ solution file
119     must contain either "cheb" or "leg" for Chebychev or Legendre
120     wavelength solutions, respectively.
121
122     Cosmic ray rejection is performed using lacosmic [1]_
123     ↪ in ccdproc via astroscrappy [2]_.
124
125     References
126     -----
127     .. [1] van Dokkum 2001, PASP, 113, 789, 1420 (article :
128         ↪ https://adsabs.harvard.edu/abs/2001PASP..113.1420V)
129     .. [2] https://zenodo.org/records/1482019
130
131     """
132
133     # -----join1-----
134
135     # MARK: Join init
136     def __init__(
137         self,
138         data_dir: Path,
139         database: str = "database",
140         fits_list: list[str] = None,
141         solutions_list: list[Path] = None,
142         split_row: int = SPLIT_ROW,
143         no_arc: bool = True,
144         save_prefix: Path | None = None,
145         verbose: int = 30,
146         **kwargs,
147     ) -> None:
148         self.data_dir = data_dir
149         self.database = Path(data_dir) / database
150         self.fits_list = find_files(
151             data_dir=self.data_dir,
152             filenames=fits_list,
153             prefix="mxgbp",
154             ext="fits",
155         )
156         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
179         the `database` directory.
180     prefix : str, optional
181         The prefix of the wavelength solution files.
182         (Defaults to "fc")
183
184     Returns
185     -----
186     tuple[list[str], bool]
187         A tuple containing the list of wavelength solutions files
188         and
189         a boolean indicating whether custom solutions were provided.
190
191     """
192     # No custom solutions
193     if not wavlist:
194         # Handle finding solutions
195         ws = []
196         for fl in os.listdir(self.database):
197             if os.path.isfile(self.database / fl) and (prefix ==
198                 fl[0:len(prefix)]):
199                 ws.append(fl)
200
201         if len(ws) != 2:
202             # Handle incorrect number of solutions found
203             msg = (
204                 f"\"Incorrect amount of wavelength solutions \""
205                 f"\"{len(ws)} fc... files) found in the solution \""
206                 f"dir.: {self.database}\""
207             )
208             logging.error(msg)

```

```

206             raise FileNotFoundError(msg)
207
208         sols = {i: j for i, j in zip(['0', 'E'], sorted(ws,
209                                     reverse=reverse))}
210         logging.debug(f"get_solutions - Found {sols} in
211                                     {self.database}")
212
213     # Custom solution
214     if len(wavlist) >= 2:
215         if len(wavlist) > 2:
216             logging.warning(f" Too many solutions, only
217                             {wavlist[:2]} are considered")
218             wavlist = wavlist[:2]
219
220         for fl in wavlist:
221             if not os.path.isfile(os.path.join(self.data_dir, fl)):
222                 msg = (
223                     f"{fl} not found in the "
224                     f"data directory {self.data_dir}"
225                 )
226                 logging.error(msg)
227                 raise FileNotFoundError(msg)
228
229         sols = {i: j for i, j in zip(['0', 'E'], sorted(wavlist,
230                                     reverse=reverse))}
231         logging.debug(f"get_solutions - Found {sols} in
232                                     {self.database}")
233
234     return (sorted(wavlist, reverse=reverse), True)
235
236 # MARK: Parse 2D WAV Function
237 def parse_solution(
238     self,
239     fc_file: str,
240     xshape: int,
241     yshape: int
242 ) -> tuple[dict[str, int], np.ndarray]:
243     """
244     Parse the 2D wavelength solution function from `fc_file`.
245
246     Parameters
247     -----
248     fc_file : str
249         The filename of the wavelength solutions file.
250     xshape : int
251         The x-order of the 2D solution.
252     yshape : int
253         The y-order of the 2D solution.
254
255     Returns
256     -----
257     tuple[dict[str, int], np.ndarray]
258         A tuple containing a dictionary of the parameters of the
259         solution function
260         and the function coefficients.

```

```

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     return (fit_params, coeff)
311
312 # MARK: Join Files
313 def join_file(self, file: os.PathLike) -> None:
314     """
315

```

```

316 Join the `O`- and `E`-beams, attach the wavelength solutions,
317 perform cosmic ray cleaning, mask the extensions,
318 and checks cropping performed by `Split`.
319 Write the FITS file in a `polsalt` valid format.
320
321 Parameters
322 -----
323 file : os.PathLike
324 The path of the FITS file to be joined.
325
326 See Also
327 -----
328 IRAF - `fitcoords` task
329 https://iraf.net/irafdocs/formats/fitcoords.php,
330 numpy 2D grid functions
331 https://numpy.org/doc/stable/reference/generated/numpy
332 Chebyshev: + '.polynomial.chebyshev.chebgrid2d.html'
333 Legendre: + '.polynomial.legendre.leggrid2d.html'
334
335 """
336 # Create empty wavelength appended hdu list
337 whdu = pyfits.HDUList()
338 primary_ext = ""
339
340 # Handle prefix and names
341 pref = "arc" if file == self.arc else "beam"
342 o_file = self.save_prefix[pref][0] + file.name[-9:]
343 e_file = self.save_prefix[pref][1] + file.name[-9:]
344
345 # Open file
346 with pyfits.open(file) as hdu:
347     # Check if file has been cropped
348     cropsize = self.check_crop(hdu, o_file, e_file)
349
350     y_shape = int(hdu["SCI"].data.shape[0] / 2) - cropsize
351     x_shape = hdu["SCI"].data.shape[1]
352
353     # No differences in "PRIMARY" extention header
354     primary_ext = hdu["PRIMARY"]
355     whdu.append(primary_ext)
356
357     for ext in ["SCI", "VAR", "BPM"]:
358         whdu.append(pyfits.ImageHDU(name=ext))
359         whdu[ext].header = hdu[ext].header.copy()
360         whdu[ext].header["CTYPE3"] = "O,E"
361
362     # Create empty extentions with correct order and format
363     if ext == "BPM":
364         whdu[ext].data = np.zeros(
365             (2, y_shape, x_shape),
366             dtype="uint8"
367         )
368         whdu[ext].header["BITPIX"] = "-uint8"
369     else:
370         whdu[ext].data = np.zeros(
371             (2, y_shape, x_shape),
372             dtype=">f4"
373         )

```

```

374         whdu[ext].header["BITPIX"] = "-32"
375
376     # Fill in empty extention
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 =
422             # legendre)
423             # Set wavelength extention values to function
424             whdu["WAV"].data[num] = leggrid2d(
425                 x=np.linspace(-1, 1, params["ymax"]),
426                 y=np.linspace(-1, 1, params["xmax"]),
427                 c=coeffs,
428             )
429
430         else:
431             msg = (

```

```

431             "Function type not recognised, please wavelength "
432             "calibrate using either chebychev or legendre."
433         )
434         raise Exception(msg)
435
436     # MARK: Cosmic Ray Cleaning
437     # See utils.Constants for `CR_PARAMS` discussion
438     whdu["SCI"].data[num] = lacosmic(
439         whdu["SCI"].data[num],
440         # contrast=CR_PARAMS['CR_CONTRAST'],
441         # threshold=CR_PARAMS['CR_THRESHOLD'],
442         # neighbor_threshold=CR_PARAMS['CR_NEIGH_THRESH'],
443         # effective_gain=CR_PARAMS['GAIN'],
444         # background=CR_PARAMS['BACKGROUND'],
445         readnoise=CR_PARAMS['READNOISE'],
446         gain=CR_PARAMS['GAIN'],
447         verbose=self.verbose,
448     )[0]
449
450     # MARK: WAV masking
451     # Left & Right Crop
452     whdu["WAV"].data[whdu["WAV"].data[:] < 3_000] = 0.0
453     whdu["WAV"].data[whdu["WAV"].data[:] >= 10_000] = 0.0
454
455     # Top & Bottom Crop (shift\tilt)
456     rpix_oc, cols, rbin, lam_c = read_wollaston(
457         whdu,
458         DATADIR + "wollaston.txt"
459     )
460
461     drow_oc = (rpix_oc - rpix_oc[:, int(cols / 2)][:, None]) / rbin
462
463     ## Cropping as suggested
464     for c, col in enumerate(drow_oc[0]):
465         if np.isnan(col):
466             continue
467
468         if int(col) < 0:
469             whdu["WAV"].data[0, int(col) :, c] = 0.0
470         elif int(col) > cropsize:
471             whdu["WAV"].data[0, 0 : int(col) - cropsize, c] = 0.0
472
473     for c, col in enumerate(drow_oc[1]):
474         if np.isnan(col):
475             continue
476
477         if int(col) > 0:
478             whdu["WAV"].data[1, 0 : int(col), c] = 0.0
479         elif (int(col) < 0) & (abs(int(col)) > cropsize):
480             whdu["WAV"].data[1, int(col) + cropsize :, c] = 0.0
481
482     # MARK: BPM masking
483     whdu["BPM"].data[0] = np.where(
484         whdu["WAV"].data[0] == 0,
485         1,
486         whdu["BPM"].data[0]
487     )
488     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`  

541     ↳ attribute"""
542     for target in self.fits_list:
543         logging.debug(f"Processing {target}")
544         self.join_file(target)
545

```

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

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

```

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

```

```

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

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

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

108    Other Parameters

```

```

105     -----
106     offset : int, optional
107         The amount the spectrum is shifted, mainly to test the effect
108         ↵ of the cross correlation
109         (The default is 0, I.E. no offset introduced)
110     **kwargs : dict
111         keyword arguments. Allows for passing unpacked dictionary to
112         ↵ the class constructor.
113     FTCS : bool, optional
114         Decides whether the Fourier Transform should be used for
115         ↵ cross correlation.
116
117     See Also
118     -----
119     scipy
120         https://docs.scipy.org/doc/scipy/reference/generated/
121         correlation: scipy.signal.correlate.html
122
123     Notes
124     -----
125     Constants Imported (See utils.Constants):
126         SAVE_CORR
127
128     #-----corr1-----
129
130     # MARK: Correlate init
131     def __init__(
132         self,
133         data_dir: Path,
134         filenames: list[str],
135         beams: str = "OE",
136         split_ccd: bool = True,
137         cont_ord: int = 11,
138         plot: bool = False,
139         offset: int = 0,
140         save_prefix: Path | None = None,
141         **kwargs
142     ) -> None:
143         self.data_dir = data_dir
144         self.fits_list = find_files(
145             data_dir=self.data_dir,
146             filenames=filenames,
147             prefix="ecwmxgbp",
148             ext="fits",
149         )
150         self._beams = None
151         self.beams = beams
152         self.ccds = 1
153         if split_ccd:
154             # BPM == 2 near center of CCD if CCD count varies
155             with pyfits.open(self.fits_list[0]) as hdu:
156                 self.ccds = sum(hdu["BPM"].data.sum(axis=1)[0] == 2)
157
158         self.cont_ord = cont_ord
159         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],
170         #     ↪ axis=-1
171         # )[ :, : self.spec1.shape[-1]]
172
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.

```



```

275             min(mid + bpm.shape[-1] // CCDs, bpm.shape[-1])
276         )
277
278     return bounds.astype(int)
279
280     # MARK: Remove Continua
281 def remove_cont(
282     self,
283     spec: list,
284     wav: list,
285     bpm: list,
286     plot_cont: bool
287 ) -> np.ndarray:
288     """
289     Remove the continuum from the data.
290
291     Parameters
292     -----
293     spec : list
294         The spectrum to remove the continuum from.
295     wav : list
296         The wavelength of the spectrum.
297     bpm : list
298         The bad pixel mask.
299     plot_cont : bool
300         Decides whether or not the continuum fitting should be
301         ↪ plotted
302
303     Returns
304     -----
305     None
306
307     """
308     # Mask out the bad pixels for fitting continua
309     okwav = np.where(bpm != 1)
310
311     # Define continua
312     ctm = continuum(
313         wav[okwav],
314         spec[okwav],
315         deg=self.cont_ord,
316         plot=plot_cont,
317     )
318
319     # Normalise spectra
320     spec /= chebyshev.chebval(wav, ctm)
321     spec -= 1
322
323     return spec
324
325     # MARK: Correlate
326 def correlate(
327     self,
328     filename1: Path,
329     filename2: Path | None = None,
330     alt: Callable = None
331 ) -> None:
332     """

```

```

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: O E -> '01' & 'E1', O -> '01' & '02', 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{spec/wav/bpm: {spec.shape}}"
371     )
372
373     corrdb = [[] for _ in range(self.ccdb)]
374     lagsdb = [[] for _ in range(self.ccdb)]
375     for ccd in range(self.ccdb):
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((

```

```

390             spec[ext, lb:ub]
391             * abs(bpm[ext, lb:ub] * -1 + 1)
392         ))
393
394     # Finally!!!! cross correlate signals and scale max -> 1
395     corrdb[ccd] = signal.correlate(*sig) if not alt else
396     ↪ alt(*sig)
397     corrdb[ccd] /= np.max(corrdb[ccd])
398     lagsdb[ccd] = signal.correlation_lags(
399         sig[0].shape[-1],
400         sig[1].shape[-1]
401     ) * self.wav_cdel
402
403     return (spec, wav, bpm), (corrdb, lagsdb)
404
405 # MARK: FTCS alternate
406 def FTCS(
407     self,
408     signal1: np.ndarray,
409     signal2: np.ndarray
410 ) -> None:
411     """
412         Cross correlates the data using the Fourier Transform.
413
414     Parameters
415     -----
416     signal1 : np.ndarray
417         The first signal to cross correlate.
418     signal2 : np.ndarray
419         The second signal to cross correlate.
420
421     Returns
422     -----
423     np.ndarray
424         The correlation data using the Fourier Transform.
425
426     """
427     logging.debug(
428         f"FTCS - data shape:\n{tspec/wav/bpm: {signal1.shape}}"
429     )
430
431     # Invert BPM (and account for 2); zero bad pixels
432     ft_spec1 = np.fft.fft(signal1)
433     ft_spec2 = np.fft.fft(signal2)
434
435     if self.can_plot:
436         plt.plot(ft_spec1)
437         plt.plot(ft_spec2)
438         plt.show()
439
440     # Cross correlate signals
441     # ft_spectrum1 * np.conj(ft_spectrum2)
442     corr_entry = signal.correlate(ft_spec1, ft_spec2)
443
444     return np.fft.ifft(corr_entry)
445
446 # MARK: Plot
447 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(Path(__file__).parent.resolve() /
469                   'utils/STOPS.mplstyle')
470     bounds = self.get_bounds(bpm)
471
472     fig, axs = plt.subplots(2, self.ccds, sharey="row")
473
474     if self.ccds == 1:
475         # Convert axs to a 2D array
476         axs = np.swapaxes(np.atleast_2d(axs), 0, 1)
477
478     # for ext, ccd in iters.product(range(2), range(self.ccds)):
479
480         for ccd in range(self.ccds):
481             axs[0, ccd].plot(
482                 lagsdb[ccd],
483                 corrdb[ccd] * 100,
484                 color='C4',
485                 label=f"max lag @ {lagsdb[ccd][corrdb[ccd].argmax()]} - "
486                       f"({bounds[1, ccd, 0] - bounds[0, ccd, 0]})",
487             )
488
489         for ext in range(2):
490             lb, ub = bounds[ext, ccd]
491             logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
492                           f"1]}")
493
494             axs[1, ccd].plot(
495                 wav[ext, lb:ub],
496                 spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
497                 OFFSET * ext,
498                 label=(
499                     f"${self.beams if self.beams != 'OE' else "
500                     f"self.beams[{ext}]}"
501                     f"_{{ext + 1 if self.beams != 'OE' else 1}}$"
502                     f"{{{('+' + str(OFFSET * ext)) + ')')}} if ext > 0 "
503                     f"else ''}"
504             ),

```

```

499
500
501     )
502
503     axs[0, 0].set_ylabel("Normalised Correlation\n(\%)")
504     for ax in axs[0, :]:
505         ax.set_xlabel("Signal Lag")
506     for ax in axs[1:, 0]:
507         ax.set_ylabel("Norm. Intensity\n(Counts)")
508     for ax in axs[-1, :]:
509         ax.set_xlabel(f"Wavelength ({self.wav_unit})")
510     for ax in axs.flatten():
511         ax.legend()
512
513     # plt.tight_layout()
514     # fig1 = plt.gcf()
515     # DPI = fig1.get_dpi()
516     # fig1.set_size_inches(700.0/float(DPI), 250.0/float(DPI))
517     plt.show()
518
519     # Handle do not save
520     if not self.save_prefix:
521         return
522
523     # Handle save
524     fig.savefig(fname=self.save_prefix)
525
526     return
527
528     # MARK: Process all listed images
529     def process(self) -> None:
530         """
531             Process the data.
532
533             Returns
534             -----
535             None
536
537             """
538
539         if self.beams != 'OE' and len(self.fits_list) == 1:
540             # change mode to OE with warning
541             logging.warning((
542                 f"`{self.beams}` correlation not possible for "
543                 "a single file. correlation `mode` changed to 'OE'."
544             ))
545             self.beams = 'OE'
546
547         # OE `mode` (same file, diff. ext.)
548         if self.beams == 'OE':
549             for fl in self.fits_list:
550                 logging.info(f"`OE` correlation of {fl}.")
551                 (spec, wav, bpm), (corr, lags) = self.correlate(fl,
552                     ↪ alt=self.alt)
553                 self.plot(spec, wav, bpm, corr, lags)
554
555         return
556
557         # O/E `mode` (diff. files, same ext.)
558         for fl1, fl2 in iters.combinations(self.fits_list, 2):
559             logging.info(f"`{self.beams}` correlation of {fl1} vs {fl2}.")

```

```
556     (spec, wav, bpm), (corr, lags) = self.correlate(f11, f12,
557     ↪ alt=self.alt)
558     self.plot(spec, wav, bpm, corr, lags)
559
560
561
562 # MARK: Main function
563 def main(argv) -> None:
564     return
565
566 if __name__ == "__main__":
567     main(sys.argv[1:])
```

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

```

1 """
2 Module for analyzing the sky lines of a wavelength calibrated image.
3 """
4
5 #!/usr/bin/env python3
6 # -*- coding: utf-8 -*-
7
8 from __init__ import __author__, __email__, __version__
9
10 # MARK: Imports
11 import os
12 import sys
13 import logging
14 from pathlib import Path
15
16 import numpy as np
17 import matplotlib.pyplot as plt
18 from astropy.io import fits as pyfits
19 from scipy import signal, stats, interpolate
20
21 from utils.SharedUtils import find_files, continuum
22 from utils.Constants import SAVE_SKY, FIND_PEAK_PARAMS, ARC_FILE
23
24 # print(
25 #     [logging.getLogger(name) for name in logging.root.manager.loggerDict]
26 # )
27 mpl_logger = logging.getLogger('matplotlib')
28 mpl_logger.setLevel(logging.INFO)
29 pil_logger = logging.getLogger('PIL')
30 pil_logger.setLevel(logging.INFO)
31 # plt.rcParams['figure.figsize'] = (20, 4)
32
33 # MARK: Skylines Class
34 class Skylines:
35
36     #-----sky0-----
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) ->
90         np.ndarray:
91         Transforms the input wavelength and spectral data based on
92         the given wavelength solution.
93     rmvCont(self) -> np.ndarray:
94         Removes the continuum from the spectrum.
95     process(self) -> None:
96         Placeholder method for processing the data.
97
98     #-----sky1-----
99
100    # MARK: Skylines init
101    def __init__(
102        self,
103        data_dir: Path,
104        filenames : list[str],
105        beams: str = "OE",
106        split_ccd: bool = False,
107        cont_ord: int = 11,
108        plot: bool = False,
109        transform: bool = True,
110        save_prefix: Path | None = None,
111        **kwargs,
112    ) -> None:
113        self.data_dir = data_dir

```

```

114     self.fits_list, self.arc_list = find_files(
115         data_dir=self.data_dir,
116         filenames=filenames,
117         prefix="wmxgbp", # t[o/e]beam
118         ext="fits",
119         sep_arc=True,
120     )
121     self._beams = None
122     self.beams = beams
123     self.ccds = 1
124     if split_ccd:
125         # See cross_correlate for initial implementation
126         self.ccds = 3
127
128     self.cont_ord = cont_ord
129     self.can_plot = plot
130     self.must_transform = transform
131
132     self.save_prefix = save_prefix
133     # Handle directory save name
134     if self.save_prefix and self.save_prefix.is_dir():
135         self.save_prefix /= SAVE_SKY
136         logging.warning((
137             f"Skylines save name resolves to a directory. "
138             f"Saving under {self.save_prefix}"
139         ))
140
141     self.max_difference = 5
142
143     self.wav_unit = "$\AA$"
144
145     logging.debug("__init__ - \n", self.__dict__)
146
147     return
148
149     # MARK: Beams property
150     @property
151     def beams(self) -> str:
152         return self._beams
153
154     @beams.setter
155     def beams(self, mode: str) -> None:
156         if mode not in ['O', 'E', 'OE']:
157             errMsg = f"Correlation mode '{mode}' not recognized."
158             logging.error(errMsg)
159             raise ValueError(errMsg)
160
161         self._beams = mode
162
163     return
164
165     # MARK: Find Peaks
166     def find_peaks(
167         self,
168         spec: np.ndarray,
169         axis: int | None = None,
170         min_height: float = 0.5,
171         **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
229                         → props[0].keys()]}")

```

```

229         return peaks, props
230
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 -1.
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
257                 arrays.
258             min_idxs : np.ndarray (len(A))
259                 The indices of the minimum difference between
260                 the elements of the two arrays.
261
262             """
263             # Compute the difference matrix using transpose
264             diff = np.abs(A - B[:, np.newaxis])
265
266             # Find the minimum value in each row (A) of `diff`
267             min_vals = np.min(diff, axis=0)
268             min_idxs = np.argmin(diff, axis=0)
269             # TODO: Recalculate min_val after
270             # selecting best min_val and removing the corresponding
271             # row/column
272
273             logging.debug(f"min_diff_matrix - min_vals: {np.round(min_vals,
274             2)}")
275             logging.debug(f"min_diff_matrix - min_idxs: {min_idxs}")
276
277             max_mask = min_vals <= max_diff
278
279             return A[max_mask], min_vals[max_mask], min_idxs[max_mask]
280
281
282     # MARK: Load File Data
283     def load_file_data(
284         self,
285         filename: Path
286     ) -> tuple[np.ndarray, np.ndarray, np.ndarray]:
287         """

```

```

284     Loads the data from the given file.
285
286     Parameters
287     -----
288     filename : Path
289         The path to the file to be loaded.
290
291     Returns
292     -----
293     spec, wav, bpm : tuple[np.ndarray, np.ndarray, np.ndarray]
294         The wavelength, spectral, and bad pixel mask data.
295
296     """
297     # Load data from self.beams extension
298     with pyfits.open(filename) as hdul:
299         exts = [0, 1] if len(self.beams) == 2 else 0 + self.beams
300         ↪ == 'E'
301         spec2D = np.atleast_3d(hdul["SCI"].data[exts])
302         wav2D = np.atleast_3d(hdul["WAV"].data[exts])
303         bpm2D = np.atleast_3d(hdul["BPM"].data[exts].astype(bool))
304
305         logging.info(
306             f"load_file_data - {filename.name} - shape:
307             ↪ {spec2D.shape}"
308         )
309
310     return spec2D, wav2D, bpm2D
311
312     # MARK: Load Sky or Arc Lines
313     def load_lines(
314         self,
315         filename: Path | None = None,
316         dtype: list[tuple] = [('wav', float), ('flux', float)],
317         skip_header: int = 3,
318         skip_footer: int = 1
319     ) -> np.ndarray:
320     """
321         Loads the sky or arc lines from the given file.
322
323         Parameters
324         -----
325         filename : Path / None, optional
326             The path to the file to be loaded.
327             Defaults to loading the skylines from `utils/sky.salt`
328
329         Returns
330         -----
331         sky_lines : np.ndarray['wav', 'flux']
332             The sky lines from the file.
333
334         usecols = None
335         if filename:
336             filename = Path(__file__).parent.resolve() / filename
337             usecols = (0, 1)
338         else:
339             filename = Path(__file__).parent.resolve() /
340             ↪ 'utils/sky.salt'
```

```

339
340     lines = np.genfromtxt(
341         filename,
342         dtype=dtype,
343         skip_header=skip_header,
344         skip_footer=skip_footer,
345         usecols=usecols
346     )
347
348     logging.debug(
349         f"load_lines - {filename.name} - shape: {lines.shape}"
350     )
351
352     return lines
353
354 # MARK: Mask Traces
355 def mask_traces(
356     self,
357     spec: np.ndarray,
358     bpm: np.ndarray,
359     max_traces: int = 1,
360     tr_pad: int = 5,
361     bg_margin: int = 10,
362     lr_margins: list[int] = [10, 10],
363     h_min: float = 0.5,
364     h_rel: float = 1 - 0.05,
365 ) -> np.ndarray:
366     """
367     Masks the traces in the bad pixel mask.
368
369     Parameters
370     -----
371     spec : np.ndarray
372         The spectral data.
373     bpm : np.ndarray
374         The bad pixel mask.
375
376     Returns
377     -----
378     bpm : np.ndarray
379         The updated bad pixel mask.
380
381     """
382     # Base mask
383     bpm[:, :bg_margin] = True
384     bpm[:, -bg_margin:] = True
385     bpm[:, :, :lr_margins[0]] = True
386     bpm[:, :, -lr_margins[1]:] = True
387
388     # Get the traces
389     traces, tr_props = self.find_peaks(
390         spec,
391         axis=1,
392         min_height=h_min,
393         rel_height=h_rel
394     )
395
396     for ext in range(len(self.beams)):

```

```

397         # Mask the traces
398         for i in range(len(traces[ext][:max_traces])):
399             lb = max(
400                 0,
401                 int(tr_props[ext]['left_ips'][i]) - tr_pad
402             )
403             ub = min(
404                 spec.shape[-1],
405                 int(tr_props[ext]['right_ips'][i]) + tr_pad
406             )
407             bpm[ext, lb : ub] = True
408             # TODO: Relocate targets after initial masking
409
410     logging.info(f"mask_traces - {min(max_traces, len(traces))} of
411     ↪ {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
452                     ↪ to)
453                     avg_max = np.mean(spec[ext], axis=1).argmax()

```

```

453     # Correct extensions based on wavelength
454     # Get wavelength values at row with most trace
455     cw[ext] = wav_sol[ext, avg_max]
456
457     # Spec ext
458     for row in range(cs.shape[1]):
459         cs[ext, row] = np.interp(
460             cw[ext],
461             wav_sol[ext, row],
462             spec[ext, row]
463         )
464         # f_2d = interpolate.interp2d(
465         #     wav_sol[ext, row],
466         #     np.arange(rows),
467         #     spec[ext],
468         # )
469         # cs[ext] = f_2d(cw[ext], np.arange(rows))
470
471     # Plot results
472     if resPlot:
473         fig, axs = plt.subplots(2, 1, figsize=[20, 4])
474         for ext in range(len(self.beams)):
475             axs[ext].imshow(
476                 cs[ext],
477                 vmax=cs[ext].mean() + 2*cs[ext].std(),
478                 vmin=cs[ext].mean() - 2*cs[ext].std()
479             )
480
481             logging.debug(f"{'E' if ext else 'O'} Average continuum
482             ↪ = {np.median(np.median(cs[ext], axis=0)):.4f}")
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,

```

```

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

```

```

562     )
563
564     # deviation
565     sky_wavs, dev, peak_idx = self.min_diff_matrix(
566         lines['wav'],
567         wavelengths[1][fl][ext][peaks[1][fl][ext]],
568         max_diff=self.max_difference,
569     )
570
571     # width/initial width
572     width = properties[1][fl][ext]['widths'][peak_idx]
573     width_i = np.zeros_like(width)
574
575     sky_i, i_dev, i_idx = self.min_diff_matrix(
576         lines['wav'],
577         wavelengths[0][fl][ext][peaks[0][fl][ext]],
578         max_diff=self.max_difference,
579     )
580
581     width_i = np.array([
582         properties[0][fl][ext]['widths'][
583             np.where(wav == sky_i)[0][0]
584         ]
585         if wav in sky_i else 1000
586         for wav in sky_wavs
587     ])
588     width_ratio = (width / width_i) - 1
589     width_ratio[width_ratio < 0] = 0
590
591     ylolims = width_ratio > self.max_difference
592     width_ratio[
593         width_ratio > self.max_difference
594     ] = self.max_difference // 2
595
596     ok = np.where(
597         (sky_wavs >=
598          ↳ wavelengths[1][fl][ext].data[ccdrange*ccd]) &
599         (sky_wavs <=
600          ↳ wavelengths[1][fl][ext].data[ccdrange*(ccd+1)])
601     )
602     axs[1, ccd].errorbar(
603         sky_wavs[ok],
604         dev.data[ok],
605         yerr=(width_ratio[ok] * 0, width_ratio[ok]),
606         lolims=ylolims[ok],
607         fmt="." if ext else "x",
608         alpha=0.8,
609         color=color,
610         # markeredgecolor='white',
611         # markeredgewidth=0.5,
612         # label=f"${self.beams[ext]}_{\{{\{fl + 1}\}}}$",
613     )
614
615     logging.debug(f"plot - RMS:
616           ↳ {np.sqrt(np.mean(dev**2)):.2f}")
617
618     for ccd in range(self.ccds):
619         # spectrum

```

```

617         ok = np.where(
618             (lines['wav'] >=
619              ↳ wavelengths[1][fl][0].data[ccdrange*ccd]) &
620             (lines['wav'] <=
621              ↳ wavelengths[1][fl][0].data[ccdrange*(ccd+1)])
622         )
623         axs[0, ccd].plot(
624             lines['wav'][ok],
625             lines['flux'][ok] * 0,
626             'x',
627             color='C4',
628             label="\textsc{salt}\nModel" if ccd == 0 else None,
629         )
630         for x in lines['wav'][ok]: axs[0, ccd].axvline(x,
631             ↳ ls='dashed', c='0.7')
632
633         axs[0, 0].set_ylabel("Rel. Intensity\n($\%$)")
634         axs[1, 0].set_ylabel(
635             "Closest Peak\n($|\lambda_{salt} - \lambda_{obs.}|$)")
636
637     # for ax in axs[:, 0]:
638     #     ax.legend(loc='upper left', ncols=(fl + 1) * (ext + 1) +
639     #             1)
640     leg = fig.legend(
641         loc='center',
642         ncol=min(8, len(spectra[0])) + 1,
643         columnspacing=0.5,
644         bbox_to_anchor=(
645             np.mean((
646                 plt.rcParams['figure.subplot.left'],
647                 plt.rcParams['figure.subplot.right']
648             )),
649             np.mean((
650                 plt.rcParams['figure.subplot.bottom'],
651                 plt.rcParams['figure.subplot.top']
652             ))
653         ),
654     )
655     leg.set_draggable(True)
656     for ax in axs[1, :]:
657         ax.grid(axis='y')
658
659     # fig.add_subplot(111, frameon=False)
660     # # hide tick and tick label of the big axis
661     # plt.tick_params(
662     #     labelcolor='none',
663     #     which='both',
664     #     top=False,
665     #     bottom=False,
666     #     left=False,
667     #     right=False
668     # )
669     axs[-1, 0 if self.ccds == 1 else 1].set_xlabel(
670         f"Wavelength ({self.wav_unit})"
671     )
672
673     # plt.tight_layout()

```

```

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

```

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

Bibliography

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

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

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

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

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

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

- Observational Astronomy - 2nd Edition. Cambridge University Press, 2006. doi: 10.2277/0521853702.
- Janus D. Brink, Moses K. Mogotsi, Melanie Saayman, Nicolaas M. Van der Merwe, Jonathan Love, and Alrin Christians. Preparing the SALT for near-infrared observations. In Heather K. Marshall, Jason Spyromilio, and Tomonori Usuda, editors, Ground-based and Airborne Telescopes IX, volume 12182 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, page 121822E, August 2022. doi: 10.1117/12.2627328.
- 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, Daragh O'Donoghue, Michael P. Smith, and Jeffrey W. Percival. Prime Focus Imaging Spectrograph for the Southern African Large Telescope: optical design. In Masanori Iye and Alan F. M. Moorwood, editors, Instrument Design and Performance for Optical/Infrared Ground-based Telescopes, volume 4841 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, pages 1463–1471, March 2003. doi: 10.1117/12.460312.
- Subrahmanyan Chandrasekhar. Radiative transfer, 1950.
- Marshall H. Cohen. Genesis of the 1000-foot Arecibo dish. Journal of Astronomical History and Heritage, 12(2):141–152, July 2009.
- E. Collett. Field Guide to Polarization. Field Guides. SPIE Press, 2005. ISBN 9780819458681. URL <https://books.google.co.za/books?id=51JwcCsLbLsC>.
- 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 chebyshев polynomials. *Advances in Engineering Software*, 28(2):133–141, 1997. ISSN 0965-9978. doi: [https://doi.org/10.1016/S0965-9978\(96\)00043-9](https://doi.org/10.1016/S0965-9978(96)00043-9). URL <https://www.sciencedirect.com/science/article/pii/S0965997896000439>.
- Dave Litwiller. Ccd vs. cmos. *Photonics spectra*, 35(1):154–158, 2001.
- Dongyue Liu and Bryan M. Hennelly. Improved wavelength calibration by modeling the spectrometer. *Applied Spectroscopy*, 76(11):1283–1299, 2022. doi: 10.1177/0003702822111796. URL <https://doi.org/10.1177/0003702822111796>. 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=1aA0dzK3FegC>.
- J. R. Priebe. Operational form of the mueller matrices. J. Opt. Soc. Am., 59(2):176–180, Feb 1969. doi: 10.1364/JOSA.59.000176. URL <https://opg.optica.org/abstract.cfm?URI=josa-59-2-176>.
- Lawrence W. Ramsey, M. T. Adams, Thomas G. Barnes, John A. Booth, Mark E. Cornell, James R. Fowler, Niall I. Gaffney, John W. Glaspey, John M. Good, Gary J. Hill, Philip W. Kelton, Victor L. Krabbendam, L. Long, Phillip J. MacQueen, Frank B. Ray, Randall L. Ricklefs, J. Sage, Thomas A. Sebring, W. J. Spiesman, and M. Steiner. Early performance and present status of the Hobby-Eberly Telescope. In Larry M. Stepp, editor, Advanced Technology Optical/IR Telescopes VI, volume 3352 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, pages 34–42, August 1998. doi: 10.1117/12.319287.
- Maria C. Simon. Wollaston prism with large split angle. Appl. Opt., 25(3):369–376, Feb 1986. doi: 10.1364/AO.25.000369. URL <https://opg.optica.org/ao/abstract.cfm?URI=ao-25-3-369>.
- G. G. Stokes. On the Composition and Resolution of Streams of Polarized Light from different Sources. Transactions of the Cambridge Philosophical Society, 9:399, January 1852.
- Doug Tody. The IRAF Data Reduction and Analysis System. In David L. Crawford, editor, Instrumentation in astronomy VI, volume 627 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, page 733, January 1986. doi: 10.1117/12.968154.
- Doug Tody. IRAF in the Nineties. In R. J. Hanisch, R. J. V. Brissenden, and J. Barnes, 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.