

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

Justin Cooper

Submitted in fulfillment of the requirements for the degree

Magister Scientiæ

in the Faculty of Natural and Agricultural Sciences

Department of Physics

University of the Free State

South Africa

Date of submission: July 2024

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

# Abstract

TODO:

- Done last
- Flow from use of SALT and pipeline and basics of its science implementations into why a more streamlined wavelength calibration is an improvement.
- Give summary of results.
- Aim for a paragraph ( $\sim 600$ ) 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

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Spectropolarimetry and the SALT RSS</b>	<b>3</b>
2.1	Spectroscopy . . . . .	3
2.1.1	Telescope Optics . . . . .	3
2.1.2	Slit . . . . .	4
2.1.3	Collimator . . . . .	4
2.1.4	Dispersion Element . . . . .	4
2.1.5	Camera Optics . . . . .	5
2.1.6	Detector . . . . .	5
2.1.7	Dispersion of Light . . . . .	5
2.1.8	Detector and Spectroscopic Calibrations . . . . .	9
2.2	Polarimetry . . . . .	15
2.2.1	Polarization . . . . .	16
2.2.2	Polarization Measurement . . . . .	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
<b>3</b>	<b>Existing and Developed Software</b>	<b>31</b>
3.1	POLSALT . . . . .	31
3.1.1	Basic CCD Reductions . . . . .	32
3.1.2	Wavelength Calibrations . . . . .	32
3.1.3	Spectral Extraction . . . . .	33
3.1.4	Raw Stokes Calculations . . . . .	33
3.1.5	Final Stokes Calculations . . . . .	34
3.1.6	Visualization . . . . .	34
3.1.7	Post-Processing Analysis . . . . .	34
3.1.8	POLSALT Limitations and the Need for Supplementary Tools . . . . .	35
3.2	IRAF . . . . .	36
3.2.1	Identify . . . . .	37

3.2.2	Reidentify . . . . .	37
3.2.3	Fitcoords . . . . .	39
3.2.4	Transform . . . . .	40
3.3	STOPS . . . . .	41
3.3.1	Splitting . . . . .	41
3.3.2	Joining . . . . .	43
3.3.3	Sky Line Checks . . . . .	47
3.3.4	Cross Correlation . . . . .	47
3.4	General Reduction Procedure . . . . .	50
3.4.1	Initial Setup . . . . .	50
3.4.2	POLSLAT Pre-Reductions . . . . .	51
3.4.3	Wavelength Calibration . . . . .	52
3.4.4	POLSLAT Reduction Completion . . . . .	54
<b>4</b>	<b>Testing and Application</b>	<b>57</b>
4.1	Testing STOPS . . . . .	57
4.1.1	Testing the split Method . . . . .	57
4.1.2	Testing the join Method . . . . .	58
4.2	Wavelength Solution Checks . . . . .	58
4.2.1	Cross Correlation Checks . . . . .	58
4.2.2	Sky Line Checks . . . . .	58
4.3	Application of STOPS . . . . .	59
4.3.1	Polarization Parameters . . . . .	59
4.3.2	Spectropolarimetric Standards . . . . .	59
4.4	Application in Publications . . . . .	59
<b>5</b>	<b>Conclusions</b>	<b>61</b>
5.1	Future Work . . . . .	61
<b>A</b>	<b>The Modified Reduction Process</b>	<b>63</b>
<b>B</b>	<b>STOPS Source Code</b>	<b>71</b>
	<b>Bibliography</b>	<b>121</b>

# 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 and Data Unit
HET	Hobby-Eberly Telescope
HgAr	Mercury-Argon
HRS	High Resolution Spectrograph
IRAF	<i>Image Reduction and Analysis Facility</i>
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
POLSAIT	<i>Polarimetric reductions for SALT</i>
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	<i>Supplementary Tools for POLSALT Spectropolarimetry</i>
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  $\lambda$  is the wavelength of an incident monochromatic beam and Full Width at Half Maximum (FWHM) refers to the width of the feature on the detector at half of its maximum intensity.

#### Prism

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

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

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



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

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

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

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

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

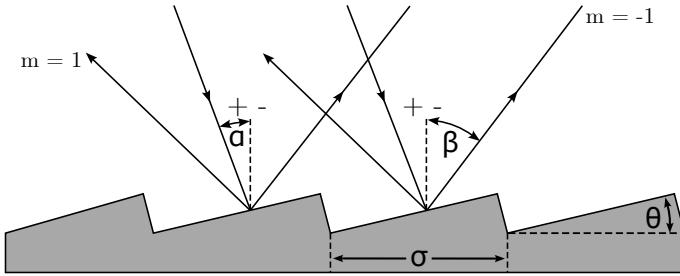
$$\frac{\partial \theta}{\partial \lambda} = -\frac{B}{a} \frac{2B_C}{\lambda^3} \propto -\lambda^{-3}, \quad (2.4)$$

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

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

---

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



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

## Diffraction Grating

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

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

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

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

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

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

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

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



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

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

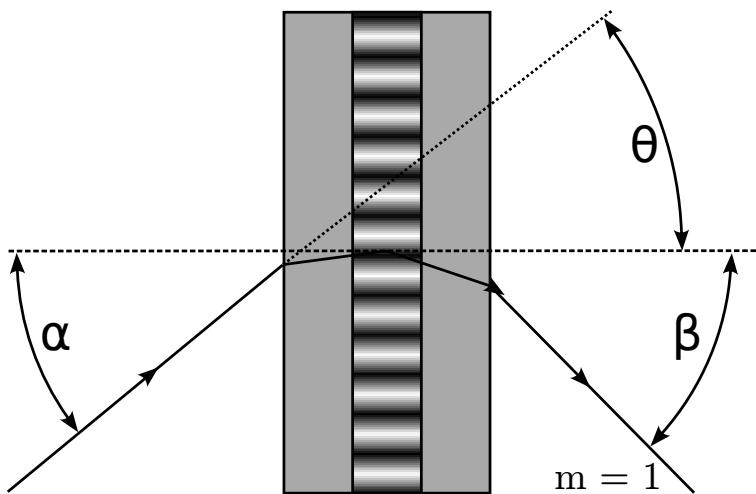
$$\frac{\partial \beta}{\partial \lambda} = \frac{\sin(\alpha) + \sin(\beta)}{\lambda \cos(\beta)} \propto \lambda^{-1}. \quad (2.8)$$

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

### Alternate Diffraction Elements

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

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



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

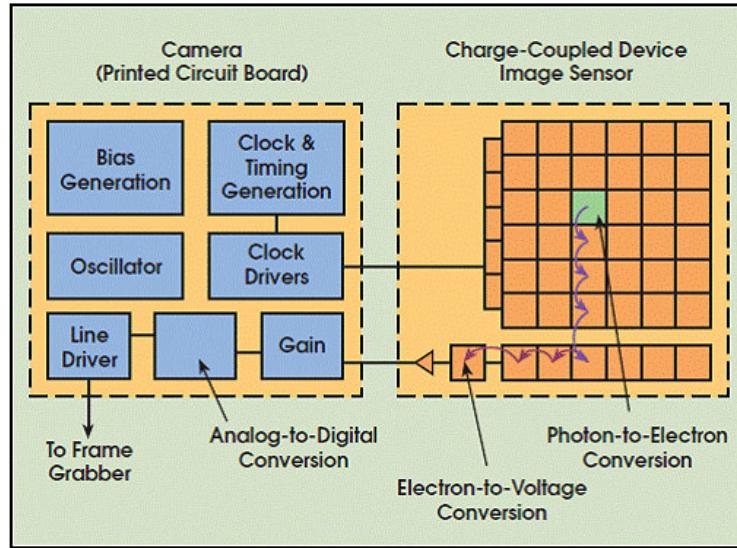
efficiency is decreased. An echelle grating refers to a diffraction grating with higher groove spacing which is optimized for use at high orders. The high order of the diffracted beam allows for greater angular dispersion which is most useful when combined with another dispersion element to cross-disperse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

### 2.1.8 Detector and Spectroscopic Calibrations

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

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

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



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

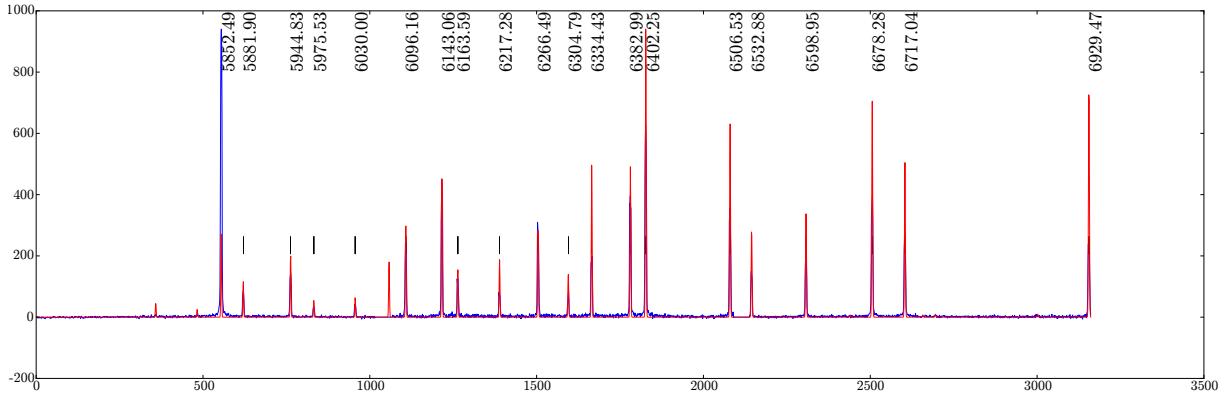
Additive noise, such as bias and dark currents, is inherent to CCD images, and as such needs to be subtracted out first when performing calibrations. Bias currents can be found by taking a bias image or by adding an overscan region to each image. A bias image is an image where the charges on the CCD 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  $34.625^\circ$ , an articulation angle of  $69.258^\circ$ , and covering a wavelength range of  $\sim 5600 - 6900 \text{ \AA}$ . Plot adapted from SALT’s published Longslit Line Atlases, (2023).<sup>2</sup>

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,  $t_D$ .  $\text{med}_{lp}$  is a low-pass median filter which smoothes out any rapid changes in the pixel-to-pixel response, removing the illumination contribution.

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

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

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

---

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



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

## 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.

---

<sup>3</sup>Excellent resources on Chebyshev and Legendre polynomials are available digitally at [www.numerical.recipes/book](http://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  $n$ .<sup>4</sup> An important property of Chebyshev polynomials is that they are orthogonal polynomials. This means that the inner product of any two differing Chebyshev polynomials,  $T_i(x)$  and  $T_j(x)$ , over the range  $[-1, 1]$  is zero, as shown by:

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

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

---

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



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

**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 ( $\lambda$ ,  $y$ -pixel) coordinate system.

## 2.2 Polarimetry

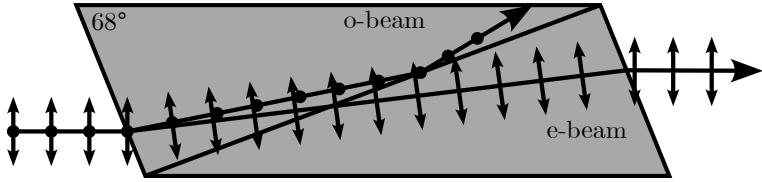
Both Huygens and Newton came to the conclusion that light demonstrates transversal properties (Huygens, 1690; Newton and Innys, 1730), which was later further investigated and coined as ‘polarization’ by Malus (Malus, 1809). Malus also investigated the polarization effects of multiple materials including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

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

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

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  $\theta_i$ ,  $\theta_t$ , and  $\theta_r$  are the angles of incidence, transmission, and reflection, respectively (Fresnel, 1870).

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



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

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

### 2.2.1 Polarization

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

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

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

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

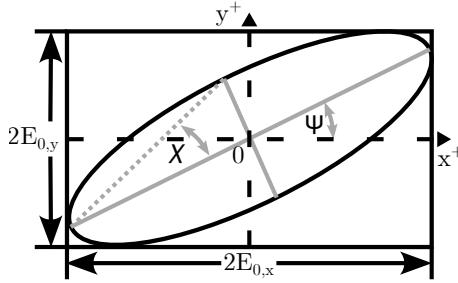
where  $E_x$ ,  $E_y$ ,  $\Phi_x$ , and  $\Phi_y$  are all parameters describing the amplitude and phase of the electric field vector in the  $(x, y)$  plane, and with the magnetic field vector proportional and perpendicular to the electric field vector (Griffiths, 2005).

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

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

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

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



**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  $\mathbf{E}_{0,x}$ ,  $\mathbf{E}_{0,y}$ ,  $\Phi_x$ , and  $\Phi_y$  describe the wave, the polarization ellipse that results from these parameters is fixed as the wave continues to propagate.

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

Several or more frames taken under differing configurations may be used to reduce a system of equations to extract all four Stokes polarization parameters, but it is possible to extract the  $I$ ,  $Q$  and  $U$  polarization parameters using only four frames, or two dual-beam frames, for well-chosen configurations and assuming ideal components. This ideal configuration varies the retarder angle such that  $\Delta\alpha = \pi/8$  while keeping the polarizer stationary. More frames for additional retarder angles are advisable and often necessary, however, as they correct for any differences in sensitivity, such as may arise in a polarized flat field and which is further discussed in § 2.2.3 (Patat and Romaniello, 2006). From Equation 2.39 we see that the linear retarder element is the driving element of a polarizer as the first three Stokes parameters ( $S_{0-2}$ , or  $I$ ,  $Q$ , and  $U$ ) may be found by changing only the angle of retardance,  $\alpha$ .

**Wave Plates** Wave plates, also commonly referred to as retarders, are generally made from optically transparent birefringent crystals. A wave plate has a fast and slow axis, which are perpendicular to one another and both perpendicular to an incident beam. Due to the birefringence of the wave plate medium, the phase velocity of the beam polarized



**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,  $\gamma$ , which is given by:

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

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

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  $\theta_i$  at angle  $i$ . These intensities thus relate to the wavelength dependent Stokes parameters as:

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

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

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

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

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

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

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

which allows Equation 2.41 to be written, as

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

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

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

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

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

### 2.3.2 Spectropolarimetric Calibrations

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

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



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

## 2.4 The Southern African Large Telescope

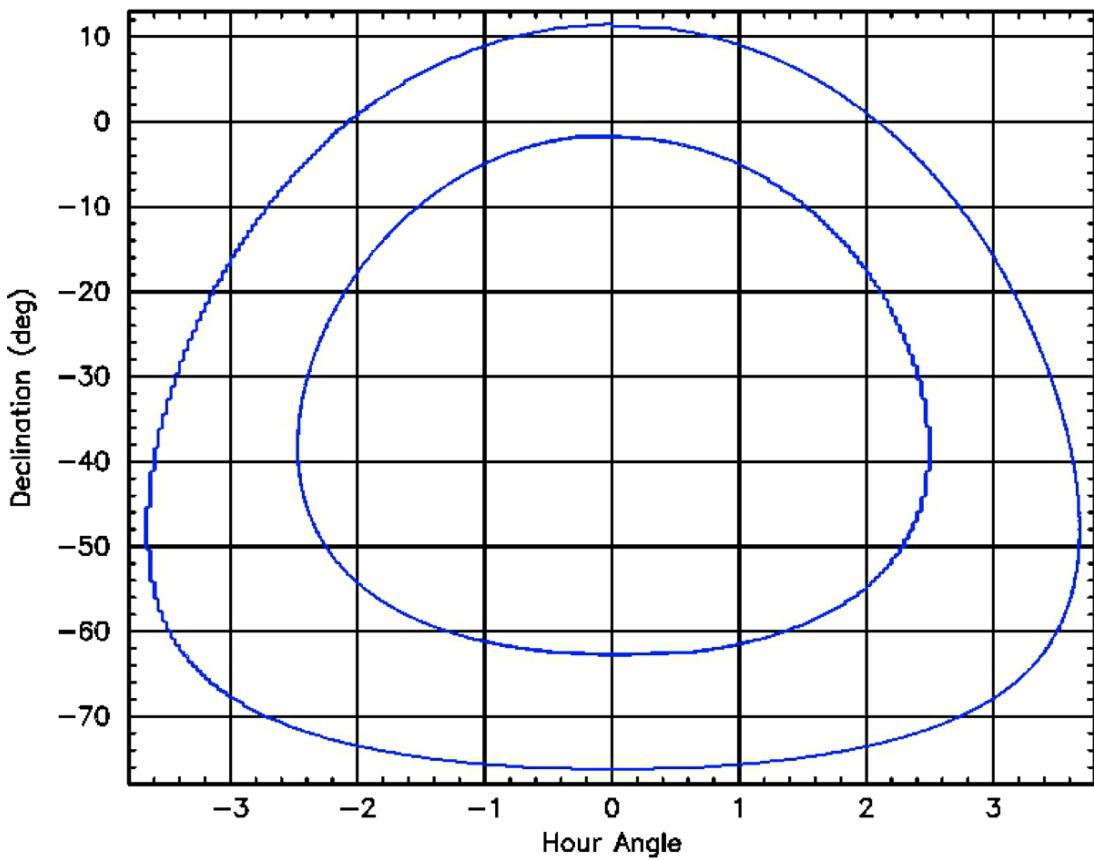
Southern African Large Telescope (SALT) is a 10 m class optical/near-infrared telescope situated at the South African Astronomical Observatory (SAAO) field station near Sutherland, South Africa (Burgh et al., 2003). The operational design was based on the Hobby-Eberly Telescope (HET) situated at McDonald Observatory, Texas, which limits the pointing of the telescope’s primary mirror to a fixed elevation ( $37^\circ$  from zenith in the case of SALT) while still allowing for full azimuthal rotation (Ramsey et al., 1998). Both SALT and HET utilize a spherical primary mirror which is stationary during observations and a tracker housing most of the instrumentation that tracks the primary mirrors spherically shaped focal path. Figure 2.17 depicts SALT’s tracker (top left), supporting structure, and primary mirror (bottom right).

### 2.4.1 The Primary Mirror

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

---

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



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

### 2.4.2 Tracker and Tracking

During observations the primary mirror is stationary and the tracker tracks celestial objects across the sky by moving along the primary focus. The tracker is capable of 6 degrees of freedom with an accuracy of  $5 \mu\text{m}$  and is capable of tracking  $\pm 6^\circ$  from the optimal central track position. Targets at declinations from  $10.5^\circ$  to  $-75.3^\circ$ , as shown in Figure 2.18 are accessible during windows of opportunity. As the tracker moves along the track the effective collecting area varies and thus SALT has a varying effective diameter of  $\sim 7 \text{ m}$  to  $9 \text{ m}$  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  $3200 \text{ \AA}$ . 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).

---

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



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

### 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.

---

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

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

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

## 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.<sup>9</sup>

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

<sup>9</sup>Commission status 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*

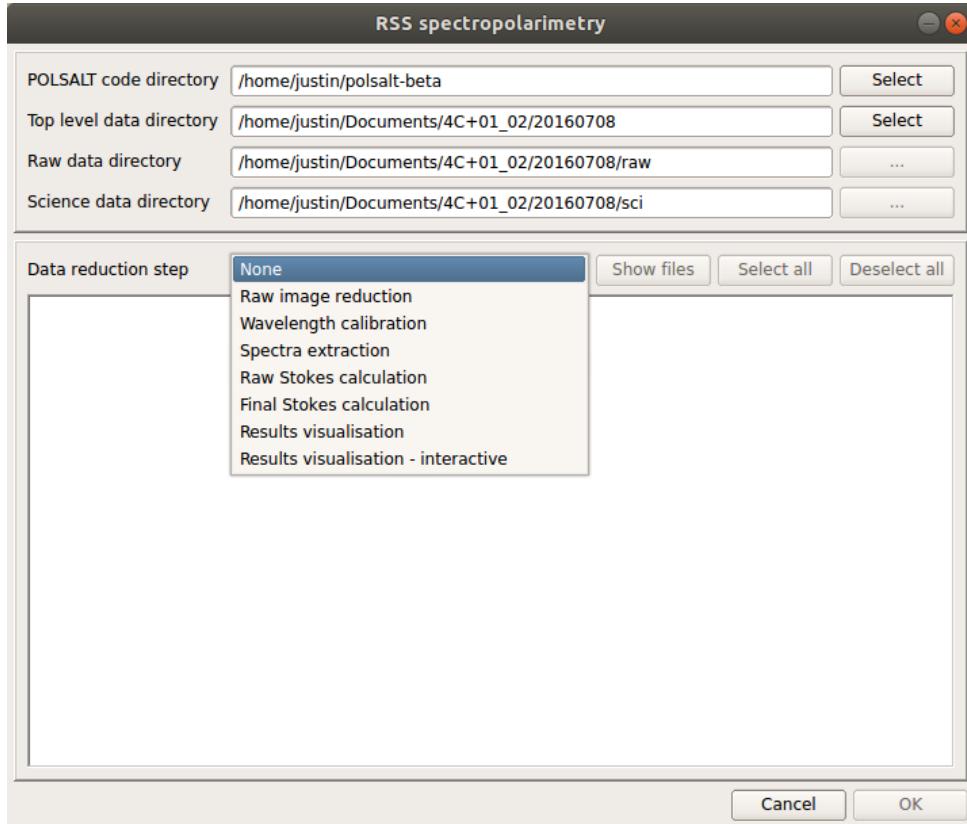
The POLSALT (*Polarimetric reductions for SALT*) pipeline is the official reduction pipeline for spectropolarimetric data taken using the SALT RSS.<sup>1</sup> The newest version of the software, aptly named the ‘beta version’ (‘version’ 23 January 2020), was the version adapted in this study. It includes a GUI, depicted in Figure 3.1, which allows for limited interactivity during key steps in the reduction process.<sup>2</sup>

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.<sup>3</sup>

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

<sup>2</sup>Installation 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.

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



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

### 3.1.1 Basic CCD Reductions

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

### 3.1.2 Wavelength Calibrations

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



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

### 3.1.3 Spectral Extraction

Background subtraction and spectral extraction is run via `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 showing the interactive window used for spectral extraction. The user, using the brightest trace in the image as a reference, defines regions which span the wavelength axis which define the background and trace regions for the sky subtraction and spectral extraction. Files with background and geometric corrections applied are saved with “c” prepended to their names and files which contain the extracted one dimensional spectrum have “e” further prepended to their names.

### 3.1.4 Raw Stokes Calculations

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

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

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

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; most notably the indexes of the related waveplate pairs, from Table 2.2, are included in the file names.



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

### 3.1.5 Final Stokes Calculations

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

### 3.1.6 Visualization

Plotting the results of the spectropolarimetric reduction process is done using `specpolview.py`, which generates a plot of the Intensity, Linear Polarization (%), and Equatorial Polarization Angle ( $^\circ$ ) against a shared wavelength axis, as seen in Figure 3.4. This step is interactive allowing the user control over various options, such as the wavelength range, binning, etc., with the GUI shown in Figure 3.3.

### 3.1.7 Post-Processing Analysis

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



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

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.

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

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

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

Since PG0300 provided the widest wavelength range and highest throughput, it was almost exclusively used for observations of flaring blazars, resulting in a large backlog of unanalyzed data. The only arc available for the PG0300 grating with a close enough articulation and grating angle ( $\sim 10.68^\circ$  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,



**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.<sup>4</sup>

since minor deviations of identified spectral features resulted in large deviations in regions with no spectral features. To only further compound the difficulty of the wavelength calibrations, the spectrum of the Ar arc lamp contains a partial overlap of a higher order at longer wavelengths (§ 2.1.7, Equation 2.5).

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

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

## 3.2 IRAF - *Image Reduction and Analysis Facility*

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

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

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

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

as a file which can later be called as a script,<sup>6</sup> the `implot` task in the `plot` package which allows the rows or columns of an image to be interactively displayed,<sup>7</sup> and the `eparam` task in the `language` package which allows the parameters of a task to be edited within the IRAF CLI.<sup>8</sup>

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

### 3.2.1 Identify

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

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

### 3.2.2 Reidentify

The `reidentify` task is used to run the `identify` task autonomously and repeatedly across the entirety of the arc frame at defined (row) intervals.<sup>11</sup> 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`.

---

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

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

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

<sup>9</sup>Help documentation for the `identify` task may be found at [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/noao.onedspec.identify.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.onedspec.identify.html).

<sup>10</sup>See also <https://iraf.net/irafdocs/formats/identify.php> for an explanation of the database contents.

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

Listing 3.1: An example of the identify database contents.<sup>10</sup>

```
# Thu 15:19:16 13-May-2021
begin    identify arc00057[*,237]
  id arc00057
  task      identify
  image     arc00057[*,237]
  units    Angstroms
  features   35
      53.61 5944.74989  5944.834 13.0 1 1 15257
      140.19 6029.9793  6029.997 13.0 1 1 4652
      185.30 6074.34644  6074.338 13.0 1 1 13396
      207.49 6096.15873  6096.161 13.0 1 1 21700
      255.23 6143.0493  6143.063 13.0 1 1 33330
      276.13 6163.56995  6163.594 13.0 1 1 11344
      330.89 6217.29293  6217.281 13.0 1 1 13705
      381.10 6266.51524  6266.495 13.0 1 1 21747
      420.21 6304.8113  6304.789 13.0 1 1 10226
      450.49 6334.45415  6334.428 13.0 1 1 36235
      500.18 6383.04826  6382.991 13.0 1 1 35824
      519.85 6402.26802  6402.248 13.0 1 1 70163
      626.70 6506.56147  6506.528 13.0 1 1 46165
      653.73 6532.91083  6532.882 13.0 1 1 21413
      721.60 6598.98642  6598.953 13.0 1 1 26396
      803.21 6678.31069  6678.277 13.0 1 1 51338
      843.15 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
```

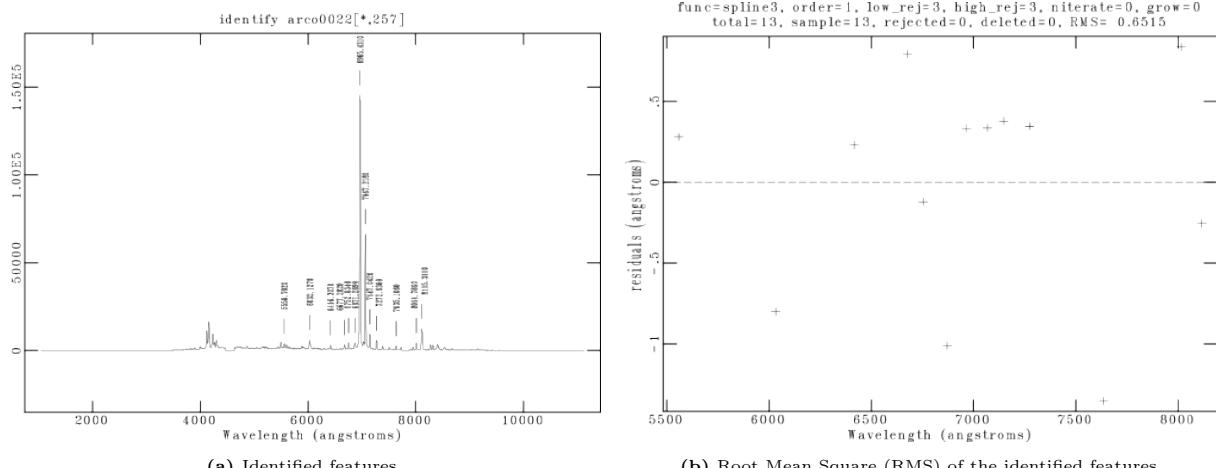


Figure 3.6: A plot and the RMS of the identified features found using the IRAF identify task. Figures created using the IRAF identify task.

**Listing 3.2:** An example of the `fitcoords` database contents.<sup>13</sup>

```
# Thu 15:26:55 13-May-2021
begin    arc00057
task      fitcoords
axis      1
units     angstroms
surface   33
      1.
      5.
      5.
      1.
      1.
      3199.
      1.
      474.
      7419.096745914063
      1510.03933621895
      -21.10886852752348
      -2.079553916887794
      0.06772631420528228
      0.7720164913117386
      -1.506773900054024
      0.1341878190232142
      -0.01659697703758917
      0.0251087019569153
      -3.318493303995171
      -0.3612632489821799
      0.003270665801371641
      -0.0157962041414068
      -0.003073690871589242
      0.007533453962924031
      0.02839687304474069
      -0.003233465769521899
      0.00174111456659807
      0.00645177595090841
      0.0105080093855621
      -0.01157827440314294
      -0.007789479002470706
      -0.006562085282926231
      -0.002321476801926803
```

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.<sup>12</sup> The usage of `fitcoords` is similar to that of `identify` and consists of examining the distribution of identified points and eliminating any points that `reidentify` may have misidentified (see Figure 3.7).

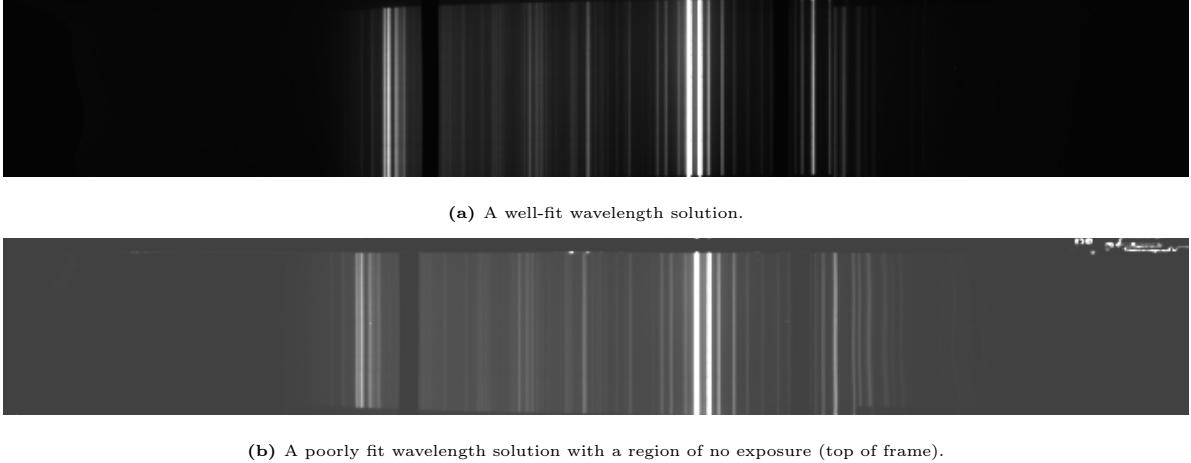
By eliminating outliers with bad residuals and modifying the two-dimensional surface function’s type and degree, the overall error of the fit is decreased, aligning more closely to what the ‘true’ wavelength solution is. This surface function is the final two-dimensional wavelength solution for each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

<sup>12</sup>Help documentation for the `fitcoords` task may be found at [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/noao.twodspec.longslit.fitcoords.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.twodspec.longslit.fitcoords.html).

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



**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.



**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.

### 3.2.4 Transform

The `transform` task is the optional final step in the IRAF wavelength calibration process.<sup>14</sup> Simply put, `transform` converts the  $(x_p, y_p)$  pixel units of an exposure to  $(\lambda, y_p)$  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.

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.

<sup>14</sup>Help documentation for the `transform` task may be found at [https://astro.uni-bonn.de/~sysstw/lfa\\_html/iraf/noao.twodspec.longslit.transform.html](https://astro.uni-bonn.de/~sysstw/lfa_html/iraf/noao.twodspec.longslit.transform.html).

### 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**).

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

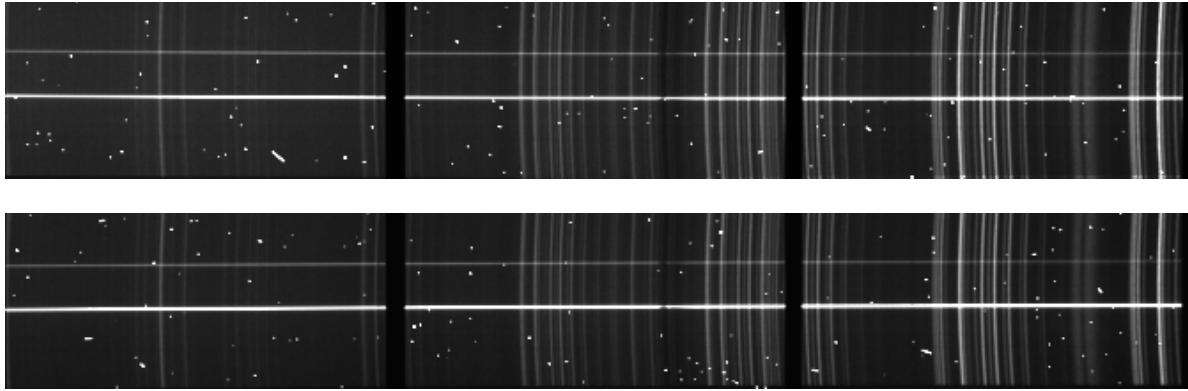

---

```

25 """
26 The `Split` class allows for the splitting of `polsalt` FITS files
27 based on the polarization beam. The FITS files must have basic
28 `polsalt` pre-reductions already applied (`mzgbp...` FITS files).
29
30 Parameters
31 -----
32 data_dir : str / Path
33     The path to the data to be split
34 fits_list : list[str], optional
35     A list of pre-reduced `polsalt` FITS files to be split within `data_dir`.
36     (The default is None, `Split` will search for `mzgbp*.fits` files)
37 split_row : int, optional
38     The row along which to split the data of each extension in the FITS file.
39     (The default is SPLIT_ROW (See Notes), the SALT RSS CCD's middle row)
40 no_arc : bool, optional
41     Decides whether the arc frames should be recombined.
42     (The default is False, `polsalt` has no use for the arc after wavelength calibrations)
43 save_prefix : dict[str, list[str]], optional
44     The prefix with which to save the O & E beams.
45     Setting `save_prefix` = ``None`` does not save the split O & E beams.
46     (The default is SAVE_PREFIX (See Notes))
47
48 Attributes
49 -----
50 arc : str
51     Name of arc FITS file within `data_dir`.
52     `arc` = `""` if `no_arc` or not detected in `data_dir`.
53 o_files, e_files : list[str]
54     A list of the `O`- and `E`-beam FITS file names.
55     The first entry is the arc file if `arc` defined.
56 data_dir
57 fits_list
58 split_row
59 save_prefix
60
61 Methods
62 -----
63 split_file(file: os.PathLike)
64     -> tuple[astropy.io.fits.HDUList]
65     Handles creation and saving the separated FITS files
66 split_ext(hdulist: astropy.io.fits.HDUList, ext: str = 'SCI')
67     -> astropy.io.fits.HDUList
68     Splits the data in the `ext` extension along the `split_row`
69 crop_file(hdulist: astropy.io.fits.HDUList, crop: int = CROP_DEFAULT (See Notes))
70     -> tuple[numpy.ndarray]
71     Crops the data along the edge of the frame, that is,
72     `O`-beam cropped as [crop:], and
73     `E`-beam cropped as [:crop].
74 update_beam_lists(o_name: str, e_name: str)
75     -> None
76     Updates `o_files` and `e_files`.
77 save_beam_lists(file_suffix: str = 'frames')
78     -> None
79     Creates (Overwrites if exists) and writes the `o_files` and `e_files` to files named
80     `o_{file_suffix}` and `e_{file_suffix}`, respectively.
81 process()
82     -> None
83     Calls `split_file` and `save_beam_lists` on each file in `fits_list` for automation.
84
85 Other Parameters
86 -----
87 **kwargs : dict
88     keyword arguments. Allows for passing unpacked dictionary to the class constructor.
89
90 Notes
91 -----
92 Constants Imported (See utils.Constants):
93     SAVE_PREFIX
94     CROP_DEFAULT
95     SPLIT_ROW
96
97
98 """

```

---



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

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 and 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. To aid the scripting of the IRAF wavelength calibration process, the `split` method also performs row cropping to exclude CCD regions which are not exposed to light, and creates files listing the split *O*- and *E*-beam FITS files which may be passed to the IRAF task inputs. Row cropping was decided on as IRAF does not handle rows with no exposure well, specifically when it comes to the autonomous `reidentify` task. The full STOPS `split` class docstring is given in Listing 3.3.

### 3.3.2 Joining

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

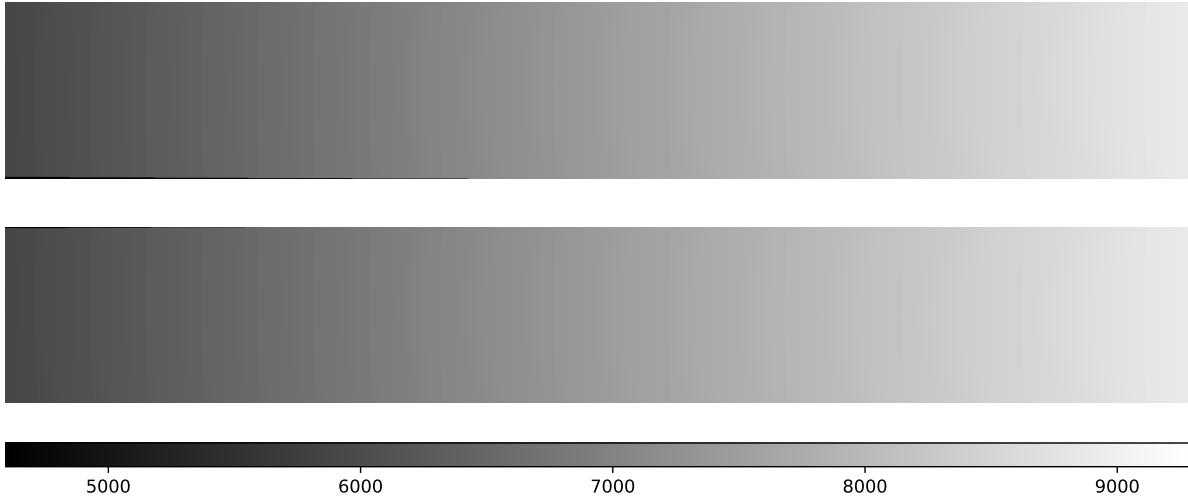
First, `join` parses the wavelength database file, described in § 3.2.3, for either a ‘Chebyshev’ or ‘Legendre’ solution, and calculates the wavelength at each  $(x_p, y_p)$  pixel position. This new image containing the corresponding wavelength values, seen in Figure 3.10, is appended to the wavelength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

Listing 3.4: The ‘docstring’ for join.py

```

31 """
32
33 The `Join` class allows for the joining of previously split files and the
34 appending of an external wavelength solution in the `polsalt` FITS file format.
35
36 Parameters
37 -----
38 data_dir : str / Path
39     The path to the data to be joined
40 database : str, optional
41     The name of the `IRAF` database folder.
42     (The default is "database")
43 fits_list : list[str], optional
44     A list of pre-reduced `polsalt` FITS files to be joined within `data_dir`.
45     (The default is ``None``, `Join` will search for `mgbp*.fits` files)
46 solutions_list: list[str], optional
47     A list of solution filenames from which the wavelength solution is created.
48     (The default is ``None``, `Join` will search for `fc*` files within the `database` directory)
49 split_row : int, optional
50     The row along which the data of each extension in the FITS file was split.
51     Necessary when Joining cropped files.
52     (The default is 517, the SALT RSS CCD's middle row)
53 save_prefix : dict[str, list[str]], optional
54     The prefix with which the previously split `O`- & `E`-beams were saved.
55     Used for detecting if cropping was applied during the splitting procedure.
56     (The default is SAVE_PREFIX (See Notes))
57 verbose : int, optional
58     The level of verbosity to use for the Cosmic ray rejection
59     (The default is 30, I.E. logging.INFO)
60
61 Attributes
62 -----
63 fc_files : list[str]
64     Valid solutions found from `solutions_list`.
65 custom : bool
66     Internal flag for whether `solutions_list` uses the `IRAF` or a custom format.
67     See Notes for custom solution formatting.
68     (Default (inherited from `solutions_list`) is False)
69 arc : str
70     Deprecated. Name of arc FITS file within `data_dir`.
71 data_dir
72 database
73 fits_list
74 split_row
75 save_prefix
76
77 Methods
78 -----
79 get_solutions(wavlist: list / None, prefix: str = "fc") -> (fc_files, custom): tuple[list[str], bool]
80     Parse `solutions_list` and return valid solution files and if they are non-`IRAF` solutions.
81 parse_solution(fc_file: str, x_shape: int, y_shape: int) -> tuple[dict[str, int], np.ndarray]
82     Loads the wavelength solution file and parses keywords necessary for creating the wavelength extension.
83 join_file(file: os.PathLike) -> None
84     Joins the files,
85     attaches the wavelength solutions,
86     performs cosmic ray cleaning,
87     masks the extension,
88     and checks cropping performed in `Split`.
89     Writes the FITS file in a `polsalt` valid format.
90 check_crop(hdu: pyfits.HDUList, o_file: str, e_file: str) -> int
91     Opens the split `O`- and `E`-beam FITS files and returns the amount of cropping that was performed.
92 process() -> None
93     Calls `join_file` on each file in `fits_list` for automation.
94
95
96 Other Parameters
97 -----
98 no_arc : bool, optional
99     Deprecated. Decides whether the arc frames should be processed.
100    (The default is False, `polsalt` has no use for the arc after wavelength calibrations)
101 **kwargs : dict
102     keyword arguments. Allows for passing unpacked dictionary to the class constructor.
103
104 Notes
105 -----
106 Constants Imported (See utils.Constants):
107     DATADIR, SAVE_PREFIX, SPLIT_ROW, CR_PARAMS
108
109 Custom wavelength solutions must be formatted containing:
110     'x', 'y', *coefficients...
111 where a solution are of order ('x' by 'y') and must contain x*y coefficients,
112 all separated by newlines. The name of the custom wavelength solution file
113 must contain either "cheb" or "leg" for Chebyshev or Legendre
114 wavelength solutions, respectively.
115
116 Cosmic ray rejection is performed using lacosmic [1]_ implemented in ccdproc via astroscrappy [2]_.
117
118 References
119 -----
120 .. [1] van Dokkum 2001, PASP, 113, 789, 1420 (article : https://adsabs.harvard.edu/abs/2001PASP..113.1420V)
121 .. [2] https://zenodo.org/records/1482019
122
123 """
124

```



**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.

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

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

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

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

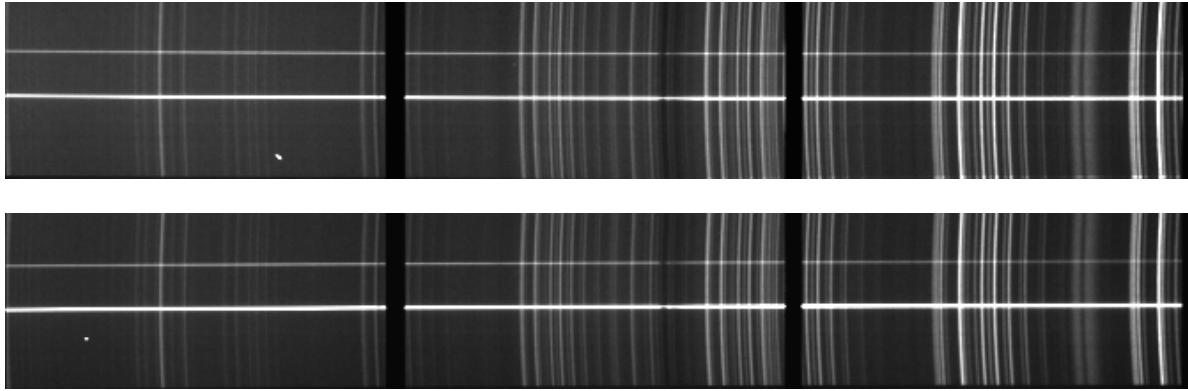

---

```

37 """
38     """
39     Class representing the Skylines object.
40
41     Parameters
42     -----
43     data_dir : Path
44         The directory containing the data files.
45     filenames : list[str]
46         The list of filenames to be processed.
47     beam : str, optional
48         The beam mode, by default "OE".
49     plot : bool, optional
50         Flag indicating whether to plot the continuum, by default False.
51     save_prefix : Path / None, optional
52         The prefix for saving the data, by default None.
53     **kwargs
54         Additional keyword arguments.
55
56     Attributes
57     -----
58     data_dir : Path
59         The directory containing the data files.
60     fits_list : list[str]
61         The list of fits file paths.
62     beams : 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
71     Methods
72     -----
73     checkLoad(self, path1: str) -> np.ndarray:
74         Checks and loads the data from the given path.
75     transform(self, wav_sol: np.ndarray, spec: np.ndarray) -> np.ndarray:
76         Transforms the input wavelength and spectral data based on
77         the given wavelength solution.
78     rmvCont(self) -> np.ndarray:
79         Removes the continuum from the spectrum.
80     process(self) -> None:
81         Placeholder method for processing the data.
82 """

```

---



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

### 3.3.3 Sky Line Checks

The `skyline` method has been implemented to compare the position of the sky lines on the science extension, or arc lines in the arc exposure, to the known positions of the sky lines, or arc lines, as measured by SALT, respectively.<sup>16</sup> 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.

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`,<sup>17</sup> and compares the peak wavelength position of the sky lines to the reference sky lines as measured by SALT. Finally, a figure is created containing a plot of the background spectra (offset by the respective legend entries superscript) with the known sky lines marked with a vertical line, and a plot of the closest identified peaks of said spectra.

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

### 3.3.4 Cross Correlation

The `skyline` method allows for confirmation of a single wavelength solution, but has no means for comparing how the wavelength solutions of two polarization beams differ from each other. As the Stokes results, and thus final polarization results, are determined by the difference between the *O*- and *E*-beams, a direct comparison is not appropriate.

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

<sup>16</sup>Both sky and arc lines are available at <https://astronomers.salt.ac.za/data/salt-longslit-line-atlas/>.

<sup>17</sup>The transformation applied by the `skyline` method uses linear interpolation and is thus less accurate at flux conservation than the transformation applied by the `transform` method.

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


---

```

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

```

---

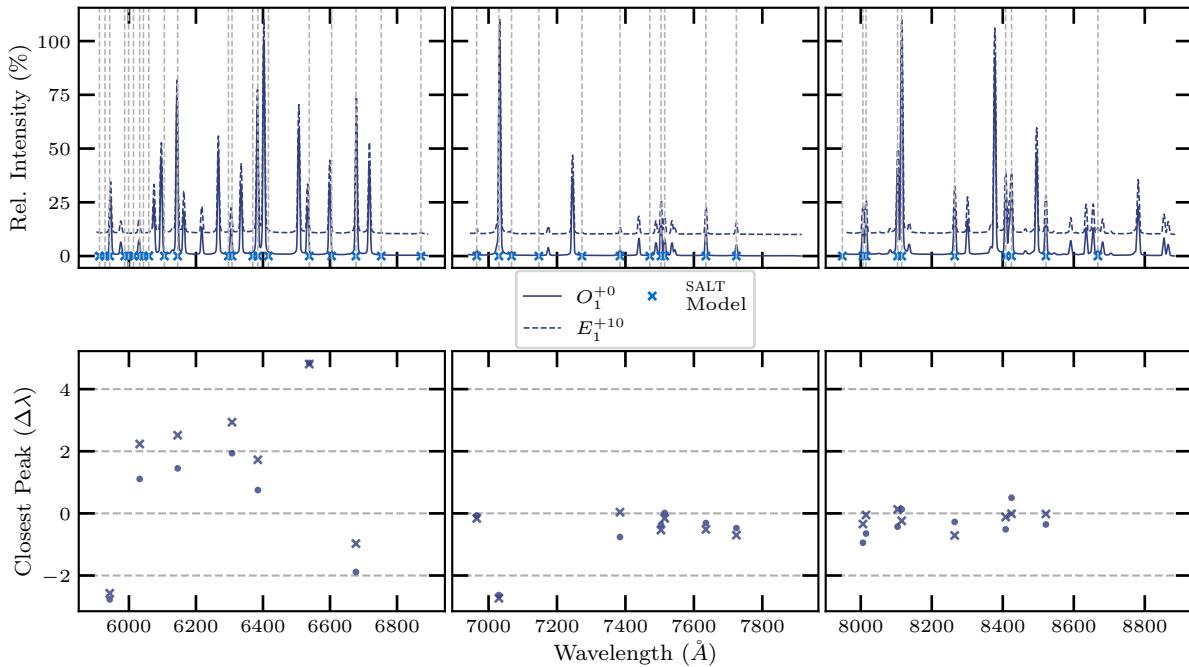


**Figure 3.12:** An example of a well calibrated wavelength solution, specifically shown for science images. Figure created via the `STOPS skyline` method.

Any observed unpolarized light, however, will reflect equally in both polarization beams and so the general trend of the two spectra may reasonably be expected to follow one another. The `correlate` method was created to allow for comparisons between the wavelength solutions of the *O*- and *E*-beams of a single exposure or the *O*- or *E*-beams of differing exposures by cross correlating the spectra.

The `correlate` method loads the `POLSALT spectra extraction` FITS files, removes the continuum and separates the CCD regions. The relevant CCD regions are cross correlated and the correlation peak is plotted and specified in the plot legend, as seen in Figure 3.14.

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

## 3.4 General Reduction Procedure

This section aims to provide a comprehensive discussion of the modified reduction procedure, an example of which is provided in Appendix A. As users all employ a variety of operating systems, language environments, and software setups, not much emphasis will be placed on how to get the software running or the managing of files; instead, the general order, seen in Figure 3.15, and commands necessary to complete each step of the reduction process are discussed, assuming that the software is running as intended.

### 3.4.1 Initial Setup

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

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



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

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.

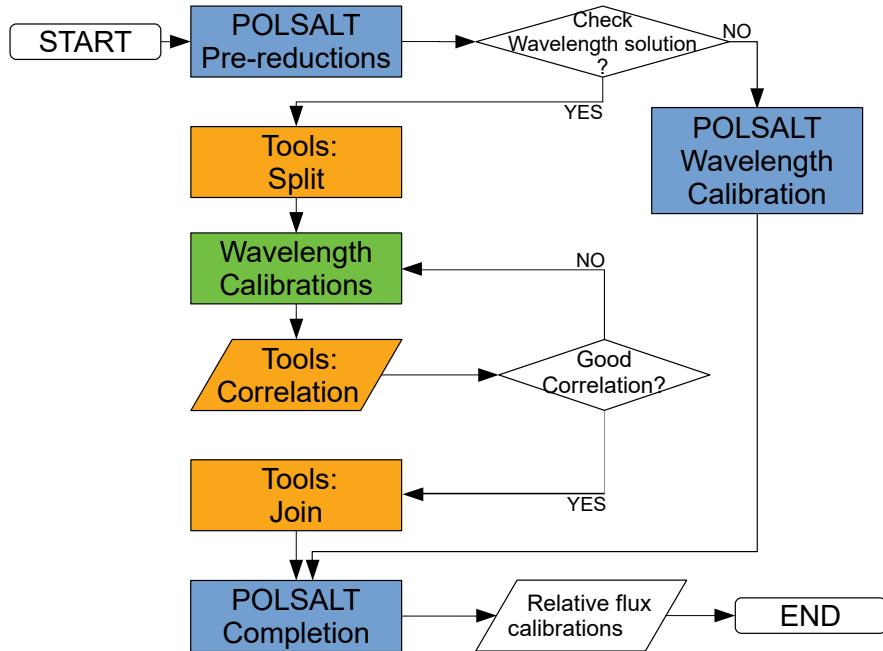
Alternatively, if the data already includes ‘mxgbp’ FITS files in the `YYYYMMDD/sci/` working directory, a CLI may be used to complete the initial pre-reductions using

---

```
$ 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.



**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).

### 3.4.3 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.

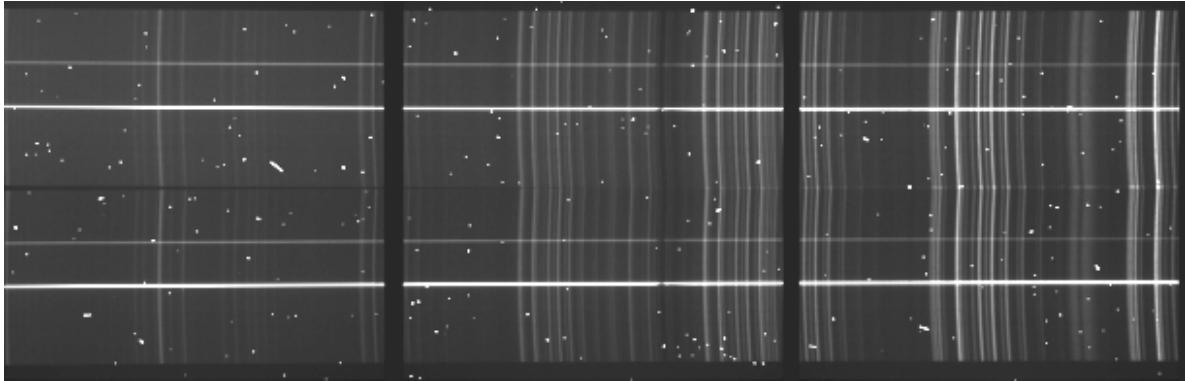
#### Preparing the Data for IRAF

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:<sup>18</sup>

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



**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.

---

```
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 `mkscrip` task which interactively asks for a task to script and parameters to use. Multiple tasks may be appended to an IRAF script, allowing for the parameters of both beams to be tracked.

Running an IRAF script may be done by running:

---

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

---

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

### Preparing the Data for POLSALT

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

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

### Sky Line Checks

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

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

### Cross Correlation Checks

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

### Cleaning Up the IRAF and STOPS Output

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

---

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

---

### 3.4.4 POLSALT Reduction Completion

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

#### POLSALT Beta in the GUI

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

---

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

**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)

```

### POLSALT Beta through a CLI

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

---

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

---

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

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

The POLSALT analysis scripts may be run using:

```
$ python script.py
```

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

---

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

# Chapter 4

## Testing and Application

This chapter contains an overview of the testing performed for the development of STOPS (§ 4.1) and the checking of the replaced wavelength solutions (§ 4.2), as well as the application of STOPS on observations (§ 4.3) and its application in publications (§ 4.4).

### 4.1 Testing STOPS

The main challenge faced when developing STOPS was ensuring that the software was compatible with both the POLSALT and IRAF file structures. As development is an iterative process, STOPS was continually checked to ensure compatibility such that the varying STOPS method inputs were correctly parsed, and that their outputs were parsable by the relevant IRAF tasks or POLSALT methods.

To this end, observations which were verified to have been accurately reduced were duplicated for testing purposes, allowing for continual checks of the STOPS pipeline to be made during the development process. As the STOPS `split` and `join` methods are designed to convert between the POLSALT and IRAF file structures, greater emphasis was made to ensure that the output of both methods provided accurate and consistent results.

#### 4.1.1 Testing the split Method

The STOPS `split` method accepts any POLSALT pre-reduced ('mxgbp-' prefixed) FITS files as input and outputs IRAF compatible file structures. As no 'split' FITS files are created during pure POLSALT reductions, the STOPS `split` method was tested by comparing the pre-reduced POLSALT files to the `split` methods output files, ensuring the correct structure and data integrity of the files handed off to IRAF.

Table 4.1 shows the FITS file information for the files before and after splitting. The split 'beam\*.fits' files contain the split 'SCI' extension data as well as the same header as the pre-reduced file. The only changes to the data, other than the exclusion of the other perpendicular polarization beam, is a cropping which defaults to 40 pixels, introduced to the top- and bottom-most rows of the data (see § 3.3.1). The header is left mostly untouched, and only updated to represent the new data type and shape: the 'BITPIX'

Filename	No.	Name	Type	Cards	Dimensions	Format
mxgbpP-20170328-0054.fits	0	PRIMARY	PrimaryHDU	161	()	
	1	SCI	ImageHDU	19	(3199, 1028)	float32
	2	VAR	ImageHDU	8	(3199, 1028)	float32
	3	BPM	ImageHDU	8	(3199, 1028)	uint8
beam0054.fits	0	PRIMARY	PrimaryHDU	162	(3199, 474)	float32
beame0054.fits	0	PRIMARY	PrimaryHDU	162	(3199, 474)	float32

**Table 4.1:** A comparison of the POLSALT pre-reduced `mxgbpP201703280054.fits` file, and the STOPS `split beam0054.fits` and `beame0054.fits` file contents.

value is updated, from 8 to  $-32$ , and the ‘NAXIS’ value is updated, from 0 to 2; the ‘NAXIS1’ and ‘NAXIS2’ keywords are added, and their values are set to the data shape; and the ‘EXTEND’ keyword is removed.<sup>1</sup>

### 4.1.2 Testing the join Method

The `join` method accepts IRAF wavelength solution database files as well as the POLSALT pre-reduced files and outputs a POLSALT `spectra extraction` compatible file structure. Ensuring that the output format was correct was paramount as the POLSALT `spectra extraction` method would not be able to process the files otherwise, thus halting the reduction process.

## 4.2 Wavelength Solution Checks

The secondary challenge encountered when developing STOPS was ensuring that the wavelength solutions parsed by STOPS were unaffected by the pipeline and that they were consistent with those created by POLSALT. This was achieved through the `correlate` and `skylines` methods, which were designed to validate the wavelength solutions produced by IRAF.

Before the POLSALT wavelength calibrations were replaced with the IRAF wavelength calibrations, the accuracy of the new wavelength solutions needed to be validated. This was done both through the IRAF tasks, ensuring an accurate wavelength solution, and through the STOPS `correlate` and `skylines` methods, allowing the integration of the wavelength solutions to be validated.

### 4.2.1 Cross Correlation Checks

The `correlate` method returns plots validating the wavelength solutions and so only has to accept the POLSALT `spectra extraction` method output files as input.

### 4.2.2 Sky Line Checks

The `skylines` method returns plots validating the wavelength solutions and so only has to accept either the IRAF `transform` task or STOPS `join` method output files as input.

---

<sup>1</sup>The ‘EXTEND’ keyword indicates that the FITS file contains multiple extensions while the ‘NAXIS1’ and ‘NAXIS2’ keywords indicate the shape and size of the data stored in the relevant extension.

## 4.3 Application of STOPS

### 4.3.1 Polarization Parameters

### 4.3.2 Spectropolarimetric Standards

## 4.4 Application in Publications



# 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

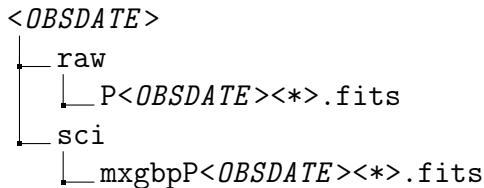
## The Modified Reduction Process

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` task<sup>1</sup> along with cursor commands, but may also be found using FITS viewing software, such as `ds9`.<sup>2</sup> The `identify` task may be run using the commands in Listing 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.<sup>3</sup> 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="")
```

---

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

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

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

The `reidentify` task will run autonomously so long as the `interactive` parameter is set to “no”.<sup>4</sup> 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).<sup>5</sup> 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.<sup>6</sup> 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

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

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

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

defaults to *GAINSET*=‘FAINT’, and *ROSPEED*=‘SLOW’. If the gain and read noise values differ from the defaults, the parameters should be updated when running `join`.<sup>7</sup>

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’.

---

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

The STOPS `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.

```

<OBSDATE>
├── raw
│   └── P<OBSDATE><*>.fits
└── sci
    ├── database
    │   ├── 01_identify.cl
    │   ├── 02_reidentify.cl
    │   ├── 03_fitcoords.cl
    │   ├── 04_transform.cl
    │   ├── deletions.db
    │   ├── fcarrE00<##>
    │   ├── fcarr000<##>
    │   ├── idarcE00<##>
    │   ├── idarc000<##>
    │   └── <*>.eps
    ├── split_files
    │   ├── arcE00<##>.fits
    │   ├── arc000<##>.fits
    │   ├── beamE<*>.fits
    │   ├── beam0<*>.fits
    │   ├── tarcE00<##>.fits
    │   ├── tarc000<##>.fits
    │   ├── tbeamE<*>.fits
    │   ├── tbeam0<*>.fits
    │   ├── <OBSDATE>_geom.txt
    │   ├── <OBSDATE>_filtered.txt
    │   ├── cwmxgbpP<OBSDATE><*>.fits
    │   ├── ecwmxgbpP<OBSDATE><*>.fits
    │   ├── mxgbpP<OBSDATE><*>.fits
    │   ├── wmxgbpP<OBSDATE><*>.fits
    │   └── <*>.log
    ├── <OBJ>_c0_h<*>_01.fits
    ├── <OBJ>_c0_1_stokes.fits
    ├── <OBJ>_c0_1_stokes_<BIN>_Ipt.txt
    └── <OBJ>_c0_1_stokes_<BIN>_Ipt.pdf

```

**Figure 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 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 """Module for splitting ``polsalt`` FITS files."""
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     #-----split0-----
25
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 (`mxgbp...` FITS files).
30
31     Parameters
32     -----
33     data_dir : str / Path
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 `→
37         data_dir`.
38         (The default is None, `Split` will search for `mxgbp*.fits` `→
39         files)
40     split_row : int, optional
41         The row along which to split the data of each extension in the
42         `→` FITS file.
43         (The default is SPLIT_ROW (See Notes), the SALT RSS CCD's
44         `→` middle row)
45     no_arc : bool, optional
46         Decides whether the arc frames should be recombined.
47         (The default is False, `polsalt` has no use for the arc after
48         `→` wavelength calibrations)
49     save_prefix : dict[str, list[str]], optional
50         The prefix with which to save the O & E beams.
51         Setting `save_prefix` = ``None`` does not save the split O & E
52         `→` beams.
53         (The default is SAVE_PREFIX (See Notes))
54
55     Attributes
56     -----

```



```

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

```

```

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

```

```

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

```

```

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

```

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

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 """Module for joining the split FITS files with an external wavelength
5    solution."""
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     """join0"""
31
32     """
33         The `Join` class allows for the joining of previously split files
34         and the
35         appending of an external wavelength solution in the `polsalt` FITS
36         file format.
37
38     Parameters
39     -----
40     data_dir : str / Path
41         The path to the data to be joined
42     database : str, optional
43         The name of the `IRAF` database folder.
44         (The default is "database")
45     fits_list : list[str], optional
46         A list of pre-reduced `polsalt` FITS files to be joined within `data_dir`.
47         (The default is ``None``, `Join` will search for `mxbp*.fits` files)
48     solutions_list: list[str], optional
49         A list of solution filenames from which the wavelength solution
        is created.
        (The default is ``None``, `Join` will search for `fc*` files
        within the `database` directory)
50     split_row : int, optional

```

```

50      The row along which the data of each extension in the FITS file
51      ↪ 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
56          ↪ were saved.
57          Used for detecting if cropping was applied during the splitting
58          ↪ procedure.
59          (The default is SAVE_PREFIX (See Notes))
60      verbose : int, optional
61          The level of verbosity to use for the Cosmic ray rejection
62          (The default is 30, I.E. logging.INFO)
63
64      Attributes
65      -----
66      fc_files : list[str]
67          Valid solutions found from `solutions_list`.
68      custom : bool
69          Internal flag for whether `solutions_list` uses the `IRAF` or a
70          ↪ custom format.
71          See Notes for custom solution formatting.
72          (Default (inherited from `solutions_list`) is False)
73      arc : str
74          Deprecated. Name of arc FITS file within `data_dir`.
75      data_dir
76      database
77      fits_list
78      split_row
79      save_prefix
80
81      Methods
82      -----
83      get_solutions(wavlist: list | None, prefix: str = "fc") ->
84          ↪ (fc_files, custom): tuple[list[str], bool]
85          Parse `solutions_list` and return valid solution files and if
86          ↪ they are non-`IRAF` solutions.
87      parse_solution(fc_file: str, x_shape: int, y_shape: int) ->
88          ↪ tuple[dict[str, int], np.ndarray]
89          Loads the wavelength solution file and parses keywords
90          ↪ necessary for creating the wavelength extension.
91      join_file(file: os.PathLike) -> None
92          Joins the files,
93          attaches the wavelength solutions,
94          performs cosmic ray cleaning,
95          masks the extension,
96          and checks cropping performed in `Split`.
97          Writes the FITS file in a `polsalt` valid format.
98      check_crop(hdu: pyfits.HDUList, o_file: str, e_file: str) -> int
99          Opens the split `O`- and `E`-beam FITS files and returns the
100         ↪ amount of cropping that was performed.
101     process() -> None
102         Calls `join_file` on each file in `fits_list` for automation.
103
104     Other Parameters
105     -----

```

```

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
102        ↳ wavelength calibrations)
103    **kwargs : dict
104        keyword arguments. Allows for passing unpacked dictionary to
105        ↳ the class constructor.
106
107    Notes
108    -----
109    Constants Imported (See utils.Constants):
110        DATADIR, SAVE_PREFIX, SPLIT_ROW, CR_PARAMS
111
112    Custom wavelength solutions must be formatted containing:
113        `x`, `y`, *coefficients...
114    where a solution are of order (`x` by `y`) and must contain  $x*y$ 
115    ↳ coefficients,
116    all separated by newlines. The name of the custom wavelength
117    ↳ solution file
118    must contain either "cheb" or "leg" for Chebyshev or Legendre
119    wavelength solutions, respectively.
120
121    Cosmic ray rejection is performed using lacosmic [1]_
122    ↳ in ccdproc via astroscrappy [2]_.
123
124    References
125    -----
126    .. [1] van Dokkum 2001, PASP, 113, 789, 1420 (article :
127    ↳ https://adsabs.harvard.edu/abs/2001PASP..113.1420V)
128    .. [2] https://zenodo.org/records/1482019
129
130    """
131
132    # MARK: Join init
133    def __init__(
134        self,
135        data_dir: Path,
136        database: str = "database",
137        fits_list: list[str] = None,
138        solutions_list: list[Path] = None,
139        split_row: int = SPLIT_ROW,
140        no_arc: bool = True,
141        save_prefix: Path | None = None,
142        verbose: int = 30,
143        **kwargs,
144    ) -> None:
145        self.data_dir = data_dir
146        self.database = Path(data_dir) / database
147        self.fits_list = find_files(
148            data_dir=self.data_dir,
149            filenames=fits_list,
150            prefix="mxbp",
151            ext="fits",
152        )
153        self.fc_files, self.custom = self.get_solutions(solutions_list)
154        self.split_row = split_row

```

```

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

```

```

207             f"dir.: {self.database}"
208         )
209         logging.error(msg)
210         raise FileNotFoundError(msg)
211
212     sols = {
213         i: j for i, j in zip(['O', 'E'], sorted(ws,
214                               ↪ reverse=reverse))
215     }
216     logging.debug(f"get_solutions - Found {sols} in
217                   ↪ {self.database}")
218
219     return sorted(ws, reverse=reverse), False
220
221 # Custom solution
222 if len(wavlist) >= 2:
223     if len(wavlist) > 2:
224         logging.warning(f" Too many solutions, only
225                         ↪ {wavlist[:2]} are considered")
226         wavlist = wavlist[:2]
227
228     for fl in wavlist:
229         if not os.path.isfile(os.path.join(self.data_dir, fl)):
230             msg = (
231                 f"{fl} not found in the "
232                 f"data directory {self.data_dir}"
233             )
234             logging.error(msg)
235             raise FileNotFoundError(msg)
236
237     sols = {
238         i: j for i, j in zip(['O', 'E'], sorted(wavlist,
239                               ↪ reverse=reverse))
240     }
241     logging.debug(f"get_solutions - Found {sols} in
242                   ↪ {self.database}")
243
244     return sorted(wavlist, reverse=reverse), True
245
246 # MARK: Parse 2D WAV Function
247 def parse_solution(
248     self,
249     fc_file: str,
250     x_shape: int,
251     y_shape: int
252 ) -> tuple[dict[str, int], np.ndarray]:
253     """
254     Parse the 2D wavelength solution function from `fc_file`.
255
256     Parameters
257     -----
258     fc_file : str
259         The filename of the wavelength solutions file.
260     x_shape : int
261         The x-order of the 2D solution.
262     y_shape : int
263         The y-order of the 2D solution.
264
265     Returns
266     -------
267     dict[str, int]
268         A dictionary mapping wavelengths to their corresponding
269         solution values.
270     np.ndarray
271         A 2D NumPy array representing the wavelength solution.
272
273     Raises
274     ----
275     FileNotFoundError
276         If the specified file does not exist.
277     ValueError
278         If the file contains invalid data or has an unexpected
279         structure.
280     """
281
282     # Read the file and parse the data
283     # ...
284
285     return sols, np_array
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
759

```

```

260     Returns
261     -----
262     tuple[dict[str, int], np.ndarray]
263         A tuple containing a dictionary of
264         the parameters of the solution function
265         and the function coefficients.
266
267     """
268     fit_params = {}
269
270     if self.custom:
271         # Load coefficients
272         coeff = np.loadtxt(fc_file)
273
274         fit_params["xorder"] = int(coeff[0].astype(int))
275         fit_params["yorder"] = int(coeff[1].astype(int))
276         coeff = coeff[2:]
277
278         f_type = 3
279         if "cheb" in str(fc_file):
280             f_type = 1
281         elif "leg" in str(fc_file):
282             f_type = 2
283         fit_params["function"] = f_type
284
285         fit_params["xmin"], fit_params["xmax"] = 1, x_shape
286         fit_params["ymin"], fit_params["ymax"] = 1, y_shape
287
288     else:
289         # Parse IRAF fc database files
290         file_contents = []
291         with open(self.database / fc_file) as fcfile:
292             for i in fcfile:
293                 file_contents.append(re.sub(r"\n\t\s*", "", i))
294
295         if file_contents[9] != "1.": # xterms - Cross-term type
296             msg = (
297                 "Cross-term not recognised (always 1 for "
298                 "'FITCOORDS'), redo FITCOORDS or change manually."
299             )
300             raise Exception(msg)
301
302         fit_params["function"] = int(file_contents[6][-1])
303
304         fit_params["xorder"] = int(file_contents[7][-1])
305         fit_params["yorder"] = int(file_contents[8][-1])
306
307         fit_params["xmin"] = int(file_contents[10][-1])
308         fit_params["xmax"] = x_shape
309         # int(file_contents[11][-1])# stretch fit over x
310         fit_params["ymin"] = int(file_contents[12][-1])
311         fit_params["ymax"] = y_shape
312         # int(file_contents[13][-1])# stretch fit over y
313
314         coeff = np.array(file_contents[14:], dtype=float)
315
316         coeff = np.reshape(
317             coeff,

```

```

318         (fit_params["xorder"], fit_params["yorder"])
319     )
320
321     return fit_params, coeff
322
323 # MARK: Join Files
324 def join_file(self, file: os.PathLike) -> None:
325     """
326     Join the `O`- and `E`-beams, attach the wavelength solutions,
327     perform cosmic ray cleaning, mask the extensions,
328     and checks cropping performed by `Split`.
329     Write the FITS file in a `polsalt` valid format.
330
331     Parameters
332     -----
333     file : os.PathLike
334         The path of the FITS file to be joined.
335
336     See Also
337     -----
338     IRAF - `fitcoords` task
339         https://iraf.net/irafdocs/formats/fitcoords.php,
340     numpy 2D grid functions
341         https://numpy.org/doc/stable/reference/generated/numpy
342         Chebyshev: + '.polynomial.chebyshev.chebgrid2d.html'">'
343         Legendre: + '.polynomial.legendre.leggrid2d.html'">'
344
345     """
346     # Create empty wavelength appended hdu list
347     whdu = pyfits.HDUList()
348
349     # Handle prefix and names
350     pref = "arc" if file == self.arc else "beam"
351     o_file = self.save_prefix[pref][0] + file.name[-9:]
352     e_file = self.save_prefix[pref][1] + file.name[-9:]
353
354     # Open file
355     with pyfits.open(file) as hdu:
356         # Check if file has been cropped
357         cropsize = self.check_crop(hdu, o_file, e_file)
358
359         y_shape = int(hdu["SCI"].data.shape[0] / 2) - cropsize
360         x_shape = hdu["SCI"].data.shape[1]
361
362         # No differences in "PRIMARY" extention header
363         primary_ext = hdu["PRIMARY"]
364         whdu.append(primary_ext)
365
366         for ext in ["SCI", "VAR", "BPM"]:
367             whdu.append(pyfits.ImageHDU(name=ext))
368             whdu[ext].header = hdu[ext].header.copy()
369             whdu[ext].header["CTYPE3"] = "O,E"
370
371         # Create empty extention with correct order and format
372         if ext == "BPM":
373             whdu[ext].data = np.zeros(
374                 (2, y_shape, x_shape),
375                 dtype="uint8"
376             )

```

```

376
377         )
378     whdu[ext].header["BITPIX"] = "-uint8"
379 else:
380     whdu[ext].data = np.zeros(
381         (2, y_shape, x_shape),
382         dtype=">f4"
383     )
384     whdu[ext].header["BITPIX"] = "-32"
385
386 # Fill in empty extention
387 if cropsize:
388     temp_split = split_sci(
389         hdu,
390         self.split_row,
391         ext=ext
392     )[ext].data
393     whdu[ext].data[0] = temp_split[0, cropsize:]
394     whdu[ext].data[1] = temp_split[1, 0:-cropsize]
395
396 else:
397     whdu[ext].data = split_sci(
398         hdu,
399         self.split_row,
400         ext=ext
401     )[ext].data
402 # End of hdu calls, close hdu
403
404 # MARK: Join (Wav. Ext.)
405 whdu.append(pyfits.ImageHDU(name="WAV"))
406 wav_header = whdu["SCI"].header.copy()
407 wav_header["EXTNAME"] = "WAV"
408 wav_header["CTYPE3"] = "O,E"
409 whdu["WAV"].header = wav_header
410
411 whdu["WAV"].data = np.zeros(
412     whdu["SCI"].data.shape,
413     dtype=">f4"
414 )
415
416 for num, fname in enumerate(self.fc_files):
417     params, coeffs = self.parse_solution(
418         fname,
419         x_shape,
420         y_shape
421     )
422
423     if params["function"] == 1: # Function type (1 = chebyshev)
424         # Set wavelength extention values to function
425         whdu["WAV"].data[num] = chebgrid2d(
426             x=np.linspace(-1, 1, params["ymax"]),
427             y=np.linspace(-1, 1, params["xmax"]),
428             c=coeffs,
429         )
430
431     elif params["function"] == 2: # Function type (2 =
432         # legendre)
433         # Set wavelength extention values to function
434         whdu["WAV"].data[num] = leggrid2d(

```

```

433             x=np.linspace(-1, 1, params["ymax"]),
434             y=np.linspace(-1, 1, params["xmax"]),
435             c=coeffs,
436         )
437
438     else:
439         msg = (
440             "Function type not recognised, please wavelength "
441             "calibrate using either Chebyshev or Legendre."
442         )
443         raise Exception(msg)
444
445     # MARK: Cosmic Ray Cleaning
446     # See utils.Constants for `CR_PARAMS` discussion
447     whdu["SCI"].data[num] = lacosmic(
448         whdu["SCI"].data[num],
449         # contrast=CR_PARAMS['CR_CONTRAST'],
450         # threshold=CR_PARAMS['CR_THRESHOLD'],
451         # neighbor_threshold=CR_PARAMS['CR_NEIGH_THRESH'],
452         # effective_gain=CR_PARAMS['GAIN'],
453         # background=CR_PARAMS['BACKGROUND'],
454         readnoise=CR_PARAMS['READNOISE'],
455         gain=CR_PARAMS['GAIN'],
456         verbose=self.verbose,
457     )[0]
458
459     # MARK: WAV masking
460     # Left & Right Crop
461     whdu["WAV"].data[whdu["WAV"].data[:] < 3_000] = 0.0
462     whdu["WAV"].data[whdu["WAV"].data[:] >= 10_000] = 0.0
463
464     # Top & Bottom Crop (shift\tilt)
465     rpix_oc, cols, rbin, lam_c = read_wollaston(
466         whdu,
467         DATADIR + "wollaston.txt"
468     )
469
470     drow_oc = (rpix_oc - rpix_oc[:, int(cols / 2)][:, None]) / rbin
471
472     # Cropping as suggested
473     for c, col in enumerate(drow_oc[0]):
474         if np.isnan(col):
475             continue
476
477         if int(col) < 0:
478             whdu["WAV"].data[0, int(col):, c] = 0.0
479         elif int(col) > cropsize:
480             whdu["WAV"].data[0, 0: int(col) - cropsize, c] = 0.0
481
482     for c, col in enumerate(drow_oc[1]):
483         if np.isnan(col):
484             continue
485
486         if int(col) > 0:
487             whdu["WAV"].data[1, 0: int(col), c] = 0.0
488         elif (int(col) < 0) & (abs(int(col)) > cropsize):
489             whdu["WAV"].data[1, int(col) + cropsize:, c] = 0.0
490

```

```

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

549         ↪ attribute"""

```

```
548     for target in self.fits_list:
549         logging.debug(f"Processing {target}")
550         self.join_file(target)
551
552     return
553
554
555 def main(argv) -> None:
556     """Main function."""
557
558     return
559
560
561 if __name__ == "__main__":
562     main(sys.argv[1:])
```

---

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

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 """Module for cross correlating polarization beams."""
5
6 from __init__ import __author__, __email__, __version__
7
8 # MARK: Imports
9 import sys
10 import logging
11 import itertools as iters
12 from pathlib import Path
13 from typing import Callable
14
15 import numpy as np
16 from numpy.polynomial import chebyshev
17 import matplotlib.pyplot as plt
18 import matplotlib.axes
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
31 # MARK: Correlate class
32 class CrossCorrelate:
33
34     #-----corr0-----
35
36     """
37     Cross correlate allows for comparing the extensions of multiple
38     FITS files, or comparing the O and E beams of a single FITS file.
39
40     Parameters
41     -----
42     data_dir : str / Path
43         The path to the data to be cross correlated
44     filenames : list[str]
45         The ecwmmagbp*.fits files to be cross correlated.
46         If only one filename is defined, correlation is done against
47         → the two polarization beams.
48     split_ccd : bool, optional
49         Decides whether the CCD regions should each be individually
50         → cross correlated.
51         (The default is True, which splits the spectrum up into its
52         → separate CCD regions)
53     cont_ord : int, optional
54         The degree of a chebyshev to fit to the continuum.
55         (The default is 11)
56     plot : bool, optional

```

```

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

62  Attributes
63  -----
64  data_dir
65  fits_list
66  beams : str
67      The mode of correlation.
68      'OE' for same file, and 'O' or 'E' for different files but same
69      ↪ extension.
70  ccds : int
71      The number of CCD's in the data. Used to split the CCD's if
72      ↪ split_ccd is True.
73  cont_ord : int
74      The degree of the chebyshev to fit to the continuum.
75  can_plot : bool
76      Decides whether the continuum fitting should be plotted
77  offset : int
78      The amount the spectrum is shifted, mainly to test the effect
79      ↪ of the cross correlation
80      (The default is 0, I.E. no offset introduced)
81  save_prefix
82  wav_unit : str
83      The units of the wavelength axis.
84      (The default is Angstroms)
85  wav_cdel : int
86      The wavelength increment.
87      (The default is 1)
88  alt : Callable
89      An alternate method of cross correlating the data.
90      (The default is None)

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

```

```

105     Other Parameters
106     -----
107     offset : int, optional
108         The amount the spectrum is shifted, mainly to test the effect
109         ↪ of the cross correlation
110         (The default is 0, I.E. no offset introduced)
111     **kwargs : dict
112         keyword arguments. Allows for passing unpacked dictionary to
113         ↪ the class constructor.
114     ftcs : bool, optional
115         Decides whether the Fourier Transform should be used for
116         ↪ cross correlation.

117     See Also
118     -----
119     scipy
120         https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.correlate.html
121
122     Notes
123     -----
124     Constants Imported (See utils.Constants):
125         SAVE_CORR
126
127     """
128
129     #-----corr1-----
130
131     # MARK: Correlate init
132     def __init__(
133         self,
134         data_dir: Path,
135         filenames: list[str],
136         beams: str = "OE",
137         split_ccd: bool = True,
138         cont_ord: int = 11,
139         plot: bool = False,
140         offset: int = 0,
141         save_prefix: Path | None = None,
142         **kwargs
143     ) -> None:
144         self.data_dir = data_dir
145         self.fits_list = find_files(
146             data_dir=self.data_dir,
147             filenames=filenames,
148             prefix="ecwmxgbp",
149             ext="fits",
150         )
151         self._beams = None
152         self.beams = beams
153         self.ccds = 1
154         if split_ccd:
155             # BPM == 2 near center of CCD if CCD count varies
156             with pyfits.open(self.fits_list[0]) as hdu:
157                 self.ccds = sum(hdu["BPM"].data.sum(axis=1)[0] == 2)
158
159         self.cont_ord = cont_ord

```

```

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

```



```

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: np.ndarray,
284         wav: np.ndarray,
285         bpm: np.ndarray,
286         plot_cont: bool
287     ) -> np.ndarray:
288         """
289             Remove the continuum from the data.
290
291         Parameters
292         -----
293         spec : np.ndarray
294             The spectrum to remove the continuum from.
295         wav : np.ndarray
296             The wavelength of the spectrum.
297         bpm : np.ndarray
298             The bad pixel mask.
299         plot_cont : bool
300             Decides whether the continuum fitting should be plotted
301
302         Returns
303         -----
304         spec : np.ndarray
305
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     ) -> tuple[np.ndarray, np.ndarray, np.ndarray, list[list],
332                 list[list]]:
333         """
334

```



```

389         # Invert BPM (and account for 2); zero bad pixels
390         sig.append((
391             spec[ext, lb:ub]
392             * abs(bpm[ext, lb:ub] * -1 + 1)
393         ))
394
395     # Finally(!!!) cross correlate signals and scale max -> 1
396     corrdb[ccd] = signal.correlate(*sig) if not alt else
397     ↪ alt(*sig)
398     corrdb[ccd] /= np.max(corrdb[ccd])
399     # noinspection PyTypeChecker
400     lagsdb[ccd] = signal.correlation_lags(
401         sig[0].shape[-1],
402         sig[1].shape[-1]
403     ) * self.wav_cdel
404
405     return spec, wav, bpm, corrdb, lagsdb
406
407     # MARK: ftcs alternate
408     def ftcs(
409         self,
410         signal1: np.ndarray,
411         signal2: np.ndarray
412     ) -> np.ndarray:
413         """
414             Cross correlates the data using the Fourier Transform.
415
416             Parameters
417             -----
418             signal1 : np.ndarray
419                 The first signal to cross correlate.
420             signal2 : np.ndarray
421                 The second signal to cross correlate.
422
423             Returns
424             -----
425             np.ndarray
426                 The correlation data using the Fourier Transform.
427
428             """
429             logging.debug(
430                 f"ftcs - data shape:\n{spec/wav/bpm: {signal1.shape}}"
431             )
432
433         # Invert BPM (and account for 2); zero bad pixels
434         ft_spec1 = np.fft.fft(signal1)
435         ft_spec2 = np.fft.fft(signal2)
436
437         if self.can_plot:
438             plt.plot(ft_spec1)
439             plt.plot(ft_spec2)
440             plt.show()
441
442         # Cross correlate signals
443         # ft_spectrum1 * np.conj(ft_spectrum2)
444         corr_entry = signal.correlate(ft_spec1, ft_spec2)
445
446         return np.fft.ifft(corr_entry)

```

```

446
447     # MARK: Plot
448     def plot(self, spec, wav, bpm, corrdb, lagsdb) -> None:
449         """
450             Plot the data.
451
452             Parameters
453             -----
454             spec : np.ndarray
455                 The spectrum.
456             wav : np.ndarray
457                 The wavelength.
458             bpm : np.ndarray
459                 The bad pixel mask.
460             corrdb : np.ndarray
461                 The cross correlation data.
462             lagsdb : np.ndarray
463                 The `lags` data.
464
465             Returns
466             -----
467             None
468
469         """
470         plt.style.use([
471             Path(__file__).parent.resolve() / 'utils/STOPS.mplstyle',
472             Path(__file__).parent.resolve() /
473             ↪ 'utils/STOPS_correlate.mplstyle',
474         ])
475         bounds = self.get_bounds(bpm)
476
477         fig, axs = plt.subplots(2, self.ccds, sharey="row")
478
479         if self.ccds == 1:
480             # Convert axs to a 2D array
481             # noinspection PyTypeChecker
482             axs: np.ndarray[matplotlib.axes.Axes] =
483             ↪ np.swapaxes(np.atleast_2d(axs), 0, 1)
484
485         # for ext, ccd in iters.product(range(2), range(self.ccds)):
486
487             for ccd in range(self.ccds):
488                 axs[0, ccd].plot(
489                     lagsdb[ccd],
490                     corrdb[ccd] * 100,
491                     color='C4',
492                     label=f"max lag @ {lagsdb[ccd][corrdb[ccd].argmax()]} - "
493                     ↪ (bounds[1, ccd, 0] - bounds[0, ccd, 0])",
494                 )
495
496                 for ext in range(2):
497                     lb, ub = bounds[ext, ccd]
498                     logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
499                     ↪ 1]}")
500
501                     axs[1, ccd].plot(
502                         wav[ext, lb:ub],
503                         spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
504                         1
505                     )
506
507             for ext in range(2):
508                 lb, ub = bounds[ext, 0]
509                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
510                     ↪ 1]}")
511
512                 axs[1, 0].plot(
513                     wav[ext, lb:ub],
514                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
515                     1
516                 )
517
518             for ext in range(2):
519                 lb, ub = bounds[0, ext]
520                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
521                     ↪ 1]}")
522
523                 axs[1, ext].plot(
524                     wav[ext, lb:ub],
525                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
526                     1
527                 )
528
529             for ext in range(2):
530                 lb, ub = bounds[0, 0]
531                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
532                     ↪ 1]}")
533
534                 axs[1, 0].plot(
535                     wav[ext, lb:ub],
536                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
537                     1
538                 )
539
540             for ext in range(2):
541                 lb, ub = bounds[0, 0]
542                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
543                     ↪ 1]}")
544
545                 axs[1, 0].plot(
546                     wav[ext, lb:ub],
547                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
548                     1
549                 )
550
551             for ext in range(2):
552                 lb, ub = bounds[0, 0]
553                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
554                     ↪ 1]}")
555
556                 axs[1, 0].plot(
557                     wav[ext, lb:ub],
558                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
559                     1
560                 )
561
562             for ext in range(2):
563                 lb, ub = bounds[0, 0]
564                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
565                     ↪ 1]}")
566
567                 axs[1, 0].plot(
568                     wav[ext, lb:ub],
569                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
570                     1
571                 )
572
573             for ext in range(2):
574                 lb, ub = bounds[0, 0]
575                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
576                     ↪ 1]}")
577
578                 axs[1, 0].plot(
579                     wav[ext, lb:ub],
580                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
581                     1
582                 )
583
584             for ext in range(2):
585                 lb, ub = bounds[0, 0]
586                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
587                     ↪ 1]}")
588
589                 axs[1, 0].plot(
590                     wav[ext, lb:ub],
591                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
592                     1
593                 )
594
595             for ext in range(2):
596                 lb, ub = bounds[0, 0]
597                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
598                     ↪ 1]}")
599
600                 axs[1, 0].plot(
601                     wav[ext, lb:ub],
602                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
603                     1
604                 )
605
606             for ext in range(2):
607                 lb, ub = bounds[0, 0]
608                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
609                     ↪ 1]}")
610
611                 axs[1, 0].plot(
612                     wav[ext, lb:ub],
613                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
614                     1
615                 )
616
617             for ext in range(2):
618                 lb, ub = bounds[0, 0]
619                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
620                     ↪ 1]}")
621
622                 axs[1, 0].plot(
623                     wav[ext, lb:ub],
624                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
625                     1
626                 )
627
628             for ext in range(2):
629                 lb, ub = bounds[0, 0]
630                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
631                     ↪ 1]}")
632
633                 axs[1, 0].plot(
634                     wav[ext, lb:ub],
635                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
636                     1
637                 )
638
639             for ext in range(2):
640                 lb, ub = bounds[0, 0]
641                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
642                     ↪ 1]}")
643
644                 axs[1, 0].plot(
645                     wav[ext, lb:ub],
646                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
647                     1
648                 )
649
650             for ext in range(2):
651                 lb, ub = bounds[0, 0]
652                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
653                     ↪ 1]}")
654
655                 axs[1, 0].plot(
656                     wav[ext, lb:ub],
657                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
658                     1
659                 )
660
661             for ext in range(2):
662                 lb, ub = bounds[0, 0]
663                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
664                     ↪ 1]}")
665
666                 axs[1, 0].plot(
667                     wav[ext, lb:ub],
668                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
669                     1
670                 )
671
672             for ext in range(2):
673                 lb, ub = bounds[0, 0]
674                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
675                     ↪ 1]}")
676
677                 axs[1, 0].plot(
678                     wav[ext, lb:ub],
679                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
680                     1
681                 )
682
683             for ext in range(2):
684                 lb, ub = bounds[0, 0]
685                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
686                     ↪ 1]}")
687
688                 axs[1, 0].plot(
689                     wav[ext, lb:ub],
690                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
691                     1
692                 )
693
694             for ext in range(2):
695                 lb, ub = bounds[0, 0]
696                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
697                     ↪ 1]}")
698
699                 axs[1, 0].plot(
700                     wav[ext, lb:ub],
701                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
702                     1
703                 )
704
705             for ext in range(2):
706                 lb, ub = bounds[0, 0]
707                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
708                     ↪ 1]}")
709
710                 axs[1, 0].plot(
711                     wav[ext, lb:ub],
712                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
713                     1
714                 )
715
716             for ext in range(2):
717                 lb, ub = bounds[0, 0]
718                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
719                     ↪ 1]}")
720
721                 axs[1, 0].plot(
722                     wav[ext, lb:ub],
723                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
724                     1
725                 )
726
727             for ext in range(2):
728                 lb, ub = bounds[0, 0]
729                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
730                     ↪ 1]}")
731
732                 axs[1, 0].plot(
733                     wav[ext, lb:ub],
734                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
735                     1
736                 )
737
738             for ext in range(2):
739                 lb, ub = bounds[0, 0]
740                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
741                     ↪ 1]}")
742
743                 axs[1, 0].plot(
744                     wav[ext, lb:ub],
745                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
746                     1
747                 )
748
749             for ext in range(2):
750                 lb, ub = bounds[0, 0]
751                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
752                     ↪ 1]}")
753
754                 axs[1, 0].plot(
755                     wav[ext, lb:ub],
756                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
757                     1
758                 )
759
760             for ext in range(2):
761                 lb, ub = bounds[0, 0]
762                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
763                     ↪ 1]}")
764
765                 axs[1, 0].plot(
766                     wav[ext, lb:ub],
767                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
768                     1
769                 )
770
771             for ext in range(2):
772                 lb, ub = bounds[0, 0]
773                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
774                     ↪ 1]}")
775
776                 axs[1, 0].plot(
777                     wav[ext, lb:ub],
778                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
779                     1
780                 )
781
782             for ext in range(2):
783                 lb, ub = bounds[0, 0]
784                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
785                     ↪ 1]}")
786
787                 axs[1, 0].plot(
788                     wav[ext, lb:ub],
789                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
790                     1
791                 )
792
793             for ext in range(2):
794                 lb, ub = bounds[0, 0]
795                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
796                     ↪ 1]}")
797
798                 axs[1, 0].plot(
799                     wav[ext, lb:ub],
800                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
801                     1
802                 )
803
804             for ext in range(2):
805                 lb, ub = bounds[0, 0]
806                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
807                     ↪ 1]}")
808
809                 axs[1, 0].plot(
810                     wav[ext, lb:ub],
811                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
812                     1
813                 )
814
815             for ext in range(2):
816                 lb, ub = bounds[0, 0]
817                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
818                     ↪ 1]}")
819
820                 axs[1, 0].plot(
821                     wav[ext, lb:ub],
822                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
823                     1
824                 )
825
826             for ext in range(2):
827                 lb, ub = bounds[0, 0]
828                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
829                     ↪ 1]}")
830
831                 axs[1, 0].plot(
832                     wav[ext, lb:ub],
833                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
834                     1
835                 )
836
837             for ext in range(2):
838                 lb, ub = bounds[0, 0]
839                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
840                     ↪ 1]}")
841
842                 axs[1, 0].plot(
843                     wav[ext, lb:ub],
844                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
845                     1
846                 )
847
848             for ext in range(2):
849                 lb, ub = bounds[0, 0]
850                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
851                     ↪ 1]}")
852
853                 axs[1, 0].plot(
854                     wav[ext, lb:ub],
855                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
856                     1
857                 )
858
859             for ext in range(2):
860                 lb, ub = bounds[0, 0]
861                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
862                     ↪ 1]}")
863
864                 axs[1, 0].plot(
865                     wav[ext, lb:ub],
866                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
867                     1
868                 )
869
870             for ext in range(2):
871                 lb, ub = bounds[0, 0]
872                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
873                     ↪ 1]}")
874
875                 axs[1, 0].plot(
876                     wav[ext, lb:ub],
877                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
878                     1
879                 )
880
881             for ext in range(2):
882                 lb, ub = bounds[0, 0]
883                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
884                     ↪ 1]}")
885
886                 axs[1, 0].plot(
887                     wav[ext, lb:ub],
888                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
889                     1
890                 )
891
892             for ext in range(2):
893                 lb, ub = bounds[0, 0]
894                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
895                     ↪ 1]}")
896
897                 axs[1, 0].plot(
898                     wav[ext, lb:ub],
899                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
900                     1
901                 )
902
903             for ext in range(2):
904                 lb, ub = bounds[0, 0]
905                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
906                     ↪ 1]}")
907
908                 axs[1, 0].plot(
909                     wav[ext, lb:ub],
910                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
911                     1
912                 )
913
914             for ext in range(2):
915                 lb, ub = bounds[0, 0]
916                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
917                     ↪ 1]}")
918
919                 axs[1, 0].plot(
920                     wav[ext, lb:ub],
921                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
922                     1
923                 )
924
925             for ext in range(2):
926                 lb, ub = bounds[0, 0]
927                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
928                     ↪ 1]}")
929
930                 axs[1, 0].plot(
931                     wav[ext, lb:ub],
932                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
933                     1
934                 )
935
936             for ext in range(2):
937                 lb, ub = bounds[0, 0]
938                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
939                     ↪ 1]}")
940
941                 axs[1, 0].plot(
942                     wav[ext, lb:ub],
943                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
944                     1
945                 )
946
947             for ext in range(2):
948                 lb, ub = bounds[0, 0]
949                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
950                     ↪ 1]}")
951
952                 axs[1, 0].plot(
953                     wav[ext, lb:ub],
954                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
955                     1
956                 )
957
958             for ext in range(2):
959                 lb, ub = bounds[0, 0]
960                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
961                     ↪ 1]}")
962
963                 axs[1, 0].plot(
964                     wav[ext, lb:ub],
965                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
966                     1
967                 )
968
969             for ext in range(2):
970                 lb, ub = bounds[0, 0]
971                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
972                     ↪ 1]}")
973
974                 axs[1, 0].plot(
975                     wav[ext, lb:ub],
976                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
977                     1
978                 )
979
980             for ext in range(2):
981                 lb, ub = bounds[0, 0]
982                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
983                     ↪ 1]}")
984
985                 axs[1, 0].plot(
986                     wav[ext, lb:ub],
987                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
988                     1
989                 )
990
991             for ext in range(2):
992                 lb, ub = bounds[0, 0]
993                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
994                     ↪ 1]}")
995
996                 axs[1, 0].plot(
997                     wav[ext, lb:ub],
998                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
999                     1
1000                 )
1001
1002             for ext in range(2):
1003                 lb, ub = bounds[0, 0]
1004                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1005                     ↪ 1]}")
1006
1007                 axs[1, 0].plot(
1008                     wav[ext, lb:ub],
1009                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1010                     1
1011                 )
1012
1013             for ext in range(2):
1014                 lb, ub = bounds[0, 0]
1015                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1016                     ↪ 1]}")
1017
1018                 axs[1, 0].plot(
1019                     wav[ext, lb:ub],
1020                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1021                     1
1022                 )
1023
1024             for ext in range(2):
1025                 lb, ub = bounds[0, 0]
1026                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1027                     ↪ 1]}")
1028
1029                 axs[1, 0].plot(
1030                     wav[ext, lb:ub],
1031                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1032                     1
1033                 )
1034
1035             for ext in range(2):
1036                 lb, ub = bounds[0, 0]
1037                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1038                     ↪ 1]}")
1039
1040                 axs[1, 0].plot(
1041                     wav[ext, lb:ub],
1042                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1043                     1
1044                 )
1045
1046             for ext in range(2):
1047                 lb, ub = bounds[0, 0]
1048                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1049                     ↪ 1]}")
1050
1051                 axs[1, 0].plot(
1052                     wav[ext, lb:ub],
1053                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1054                     1
1055                 )
1056
1057             for ext in range(2):
1058                 lb, ub = bounds[0, 0]
1059                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1060                     ↪ 1]}")
1061
1062                 axs[1, 0].plot(
1063                     wav[ext, lb:ub],
1064                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1065                     1
1066                 )
1067
1068             for ext in range(2):
1069                 lb, ub = bounds[0, 0]
1070                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1071                     ↪ 1]}")
1072
1073                 axs[1, 0].plot(
1074                     wav[ext, lb:ub],
1075                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1076                     1
1077                 )
1078
1079             for ext in range(2):
1080                 lb, ub = bounds[0, 0]
1081                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1082                     ↪ 1]}")
1083
1084                 axs[1, 0].plot(
1085                     wav[ext, lb:ub],
1086                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1087                     1
1088                 )
1089
1090             for ext in range(2):
1091                 lb, ub = bounds[0, 0]
1092                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1093                     ↪ 1]}")
1094
1095                 axs[1, 0].plot(
1096                     wav[ext, lb:ub],
1097                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1098                     1
1099                 )
1100
1101             for ext in range(2):
1102                 lb, ub = bounds[0, 0]
1103                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1104                     ↪ 1]}")
1105
1106                 axs[1, 0].plot(
1107                     wav[ext, lb:ub],
1108                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1109                     1
1110                 )
1111
1112             for ext in range(2):
1113                 lb, ub = bounds[0, 0]
1114                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1115                     ↪ 1]}")
1116
1117                 axs[1, 0].plot(
1118                     wav[ext, lb:ub],
1119                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1120                     1
1121                 )
1122
1123             for ext in range(2):
1124                 lb, ub = bounds[0, 0]
1125                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1126                     ↪ 1]}")
1127
1128                 axs[1, 0].plot(
1129                     wav[ext, lb:ub],
1130                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1131                     1
1132                 )
1133
1134             for ext in range(2):
1135                 lb, ub = bounds[0, 0]
1136                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1137                     ↪ 1]}")
1138
1139                 axs[1, 0].plot(
1140                     wav[ext, lb:ub],
1141                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1142                     1
1143                 )
1144
1145             for ext in range(2):
1146                 lb, ub = bounds[0, 0]
1147                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1148                     ↪ 1]}")
1149
1150                 axs[1, 0].plot(
1151                     wav[ext, lb:ub],
1152                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1153                     1
1154                 )
1155
1156             for ext in range(2):
1157                 lb, ub = bounds[0, 0]
1158                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1159                     ↪ 1]}")
1160
1161                 axs[1, 0].plot(
1162                     wav[ext, lb:ub],
1163                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1164                     1
1165                 )
1166
1167             for ext in range(2):
1168                 lb, ub = bounds[0, 0]
1169                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1170                     ↪ 1]}")
1171
1172                 axs[1, 0].plot(
1173                     wav[ext, lb:ub],
1174                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1175                     1
1176                 )
1177
1178             for ext in range(2):
1179                 lb, ub = bounds[0, 0]
1180                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1181                     ↪ 1]}")
1182
1183                 axs[1, 0].plot(
1184                     wav[ext, lb:ub],
1185                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1186                     1
1187                 )
1188
1189             for ext in range(2):
1190                 lb, ub = bounds[0, 0]
1191                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1192                     ↪ 1]}")
1193
1194                 axs[1, 0].plot(
1195                     wav[ext, lb:ub],
1196                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1197                     1
1198                 )
1199
1200             for ext in range(2):
1201                 lb, ub = bounds[0, 0]
1202                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1203                     ↪ 1]}")
1204
1205                 axs[1, 0].plot(
1206                     wav[ext, lb:ub],
1207                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1208                     1
1209                 )
1210
1211             for ext in range(2):
1212                 lb, ub = bounds[0, 0]
1213                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1214                     ↪ 1]}")
1215
1216                 axs[1, 0].plot(
1217                     wav[ext, lb:ub],
1218                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1219                     1
1220                 )
1221
1222             for ext in range(2):
1223                 lb, ub = bounds[0, 0]
1224                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1225                     ↪ 1]}")
1226
1227                 axs[1, 0].plot(
1228                     wav[ext, lb:ub],
1229                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1230                     1
1231                 )
1232
1233             for ext in range(2):
1234                 lb, ub = bounds[0, 0]
1235                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1236                     ↪ 1]}")
1237
1238                 axs[1, 0].plot(
1239                     wav[ext, lb:ub],
1240                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1241                     1
1242                 )
1243
1244             for ext in range(2):
1245                 lb, ub = bounds[0, 0]
1246                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1247                     ↪ 1]}")
1248
1249                 axs[1, 0].plot(
1250                     wav[ext, lb:ub],
1251                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1252                     1
1253                 )
1254
1255             for ext in range(2):
1256                 lb, ub = bounds[0, 0]
1257                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1258                     ↪ 1]}")
1259
1260                 axs[1, 0].plot(
1261                     wav[ext, lb:ub],
1262                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1263                     1
1264                 )
1265
1266             for ext in range(2):
1267                 lb, ub = bounds[0, 0]
1268                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1269                     ↪ 1]}")
1270
1271                 axs[1, 0].plot(
1272                     wav[ext, lb:ub],
1273                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1274                     1
1275                 )
1276
1277             for ext in range(2):
1278                 lb, ub = bounds[0, 0]
1279                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1280                     ↪ 1]}")
1281
1282                 axs[1, 0].plot(
1283                     wav[ext, lb:ub],
1284                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1285                     1
1286                 )
1287
1288             for ext in range(2):
1289                 lb, ub = bounds[0, 0]
1290                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1291                     ↪ 1]}")
1292
1293                 axs[1, 0].plot(
1294                     wav[ext, lb:ub],
1295                     spec[ext, lb:ub] * abs(bpm[ext, lb:ub] * -1 + 1) +
1296                     1
1297                 )
1298
1299             for ext in range(2):
1300                 lb, ub = bounds[0, 0]
1301                 logging.debug(f"fl-{ext}: {wav[ext, lb]}:{wav[ext, ub - "
1302                     ↪ 1]}")
1303
1304                 axs[1, 0].plot(
1305                     wav[ext, lb:ub],
1306                     spec[ext, lb:ub]
```

```

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

```

```
      ↪ alt=self.alt)
      self.plot(spec, wav, bpm, corr, lags)

556
557     return

558     # O/E `mode` (diff. files, same ext.)
559     for fl1, fl2 in iter.combinations(self.fits_list, 2):
560         logging.info(f"{self.beams} correlation of {fl1} vs {fl2}.")
561         spec, wav, bpm, corr, lags = self.correlate(fl1, fl2,
562             ↪ alt=self.alt)
563         self.plot(spec, wav, bpm, corr, lags)

564     return

565
566
567
568 # MARK: Main function
569 def main(argv) -> None:
570     return

571
572
573 if __name__ == "__main__":
574     main(sys.argv[1:])
```

---

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

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Module for analyzing the sky lines of a wavelength calibrated image.
5 """
6
7
8 from __init__ import __author__, __email__, __version__
9
10 # MARK: Imports
11 import sys
12 import logging
13 from pathlib import Path
14
15 import numpy as np
16 import matplotlib.pyplot as plt
17 import matplotlib.axes
18 from astropy.io import fits as pyfits
19 from scipy import signal
20
21 from utils.SharedUtils import find_files
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
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     beams : 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
71     Methods
72     -----
73     checkLoad(self, path1: str) -> np.ndarray:
74         Checks and loads the data from the given path.
75     transform(self, wav_sol: np.ndarray, spec: np.ndarray) ->
76         np.ndarray:
77         Transforms the input wavelength and spectral data based on
78         the given wavelength solution.
79     rmvCont(self) -> np.ndarray:
80         Removes the continuum from the spectrum.
81     process(self) -> None:
82         Placeholder method for processing the data.
83
84     #-----sky1-----
85
86     # MARK: Skylines init
87     def __init__(
88         self,
89         data_dir: Path,
90         filenames: list[str],
91         beams: str = "OE",
92         split_ccd: bool = False,
93         cont_ord: int = 11,
94         plot: bool = False,
95         transform: bool = True,
96         save_prefix: Path | None = None,
97         **kwargs,
98     ) -> None:
99         self.data_dir = data_dir
100        self.fits_list, self.arc_list = find_files(
101            data_dir=self.data_dir,
102            filenames=filenames,
103            prefix="wmxgbp", # t[ole]beam
104            ext="fits",
105            sep_arc=True,
106        )
107        self._beams = None
108        self.beams = beams
109        self.ccds = 1
110        if split_ccd:
111            # See cross_correlate for initial implementation
112            self.ccds = 3
113

```

```

114     self.cont_ord = cont_ord
115     self.can_plot = plot
116     self.must_transform = transform
117
118     self.save_prefix = save_prefix
119     # Handle directory save name
120     if self.save_prefix and self.save_prefix.is_dir():
121         self.save_prefix /= SAVE_SKY
122         logging.warning((
123             f"Skylines save name resolves to a directory. "
124             f"Saving under {self.save_prefix}"
125         ))
126
127     self.max_difference = 5
128
129     self.wav_unit = "$\AA$"
130
131     logging.debug("__init__ - \n", self.__dict__)
132
133     return
134
135     # MARK: Beams property
136     @property
137     def beams(self) -> str:
138         return self._beams
139
140     @beams.setter
141     def beams(self, mode: str) -> None:
142         if mode not in ['O', 'E', 'OE']:
143             err_msg = f"Correlation mode '{mode}' not recognized."
144             logging.error(err_msg)
145             raise ValueError(err_msg)
146
147         self._beams = mode
148
149     return
150
151     # MARK: Find Peaks
152     def find_peaks(
153         self,
154         spec: np.ndarray,
155         axis: int | None = None,
156         min_height: float = 0.5,
157         **kwargs,
158     ) -> tuple[list[np.ndarray], list[dict]]:
159         """
160             Finds the peaks in the given spectral data.
161
162             Parameters
163             -----
164             spec : np.ndarray
165                 The spectral data.
166             axis: int / None, optional
167                 The axis along which the peaks are found.
168             min_height : float, optional
169                 The minimum height of the peaks, by default 0.5.
170
171             Returns

```

```

172     -----
173     peaks, properties : tuple[list[np.ndarray], list[dict]]
174         The peaks and their properties.
175
176     """
177     peaks = []
178     props = []
179     row_means = []
180
181     for ext in range(len(self.beams)):
182         row_mean = spec[ext] if axis is None else
183             np.mean(spec[ext], axis=axis)
184
185         peak, prop = signal.find_peaks(
186             row_mean,
187             prominence=min_height * np.max(row_mean),
188             width=0,
189             **kwargs,
190         )
191         peaks.append(peak)
192         props.append(prop)
193         row_means.append(row_mean)
194
195     if self.can_plot:
196         fig, axs = plt.subplots(2, 1)
197         for ext in range(len(self.beams)):
198             axs[ext].plot(
199                 row_means[ext],
200                 label=f"'E' if ext else 'O'"
201             )
202             axs[ext].plot(
203                 peaks[ext],
204                 row_means[peaks[ext]],
205                 "x",
206                 label=f"'E' if ext else 'O'" peaks"
207             )
208             axs[ext].legend()
209             plt.show()
210
211         logging.debug(f"find_peaks - peaks: {[len(i) for i in peaks]}")
212         logging.debug(f"find_peaks - props: {[key for key in
213             props[0].keys()]}")
214
215     return peaks, props
216
217     # MARK: Min. of Diff. Matrix
218     @staticmethod
219     def min_diff_matrix(
220         a: np.ndarray,
221         b: np.ndarray,
222         max_diff: int = 100
223     ) -> tuple[np.ndarray, np.ndarray, np.ndarray]:
224         """
225             Find the minimum difference between the elements of two arrays.
226
227             Parameters
228             -----
229             a : np.ndarray

```

```

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

```

```

283         exts = [0, 1] if len(self.beams) == 2 else 0 +
284         ↪ int(self.beams == 'E')
285         spec2d = np.atleast_3d(hdul["SCI"].data[exts])
286         wav2d = np.atleast_3d(hdul["WAV"].data[exts])
287         bpm2d = np.atleast_3d(hdul["BPM"].data[exts]).astype(bool))
288
289         logging.info(
290             f"load_file_data - {filename.name} - shape:
291             ↪ {spec2d.shape}"
292         )
293
294     return spec2d, wav2d, bpm2d
295
296 # MARK: Load Sky or Arc Lines
297 @staticmethod
298 def load_lines(
299     filename: str | Path | None = None,
300     dtype: list[tuple] = [('wav', float), ('flux', float)],
301     skip_header: int = 3,
302     skip_footer: int = 1
303 ) -> np.ndarray:
304     """
305         Loads the sky or arc lines from the given file.
306
307     Parameters
308     -----
309     filename : str / Path / None, optional
310         The path to the file to be loaded.
311         Defaults to loading the skylines from `utils/sky.salt`.
312     dtype : list[tuple], optional
313         The data type of the sky lines.
314         (Default is [(‘wav’, float), (‘flux’, float)])
315     skip_header : int, optional
316         The amount of lines to skip of the `filenames` header.
317         (Default is 3)
318     skip_footer : int, optional
319         The amount of lines to skip of the `filename`’s footer.
320         (Default is 1)
321
322     Returns
323     -----
324     sky_lines : np.ndarray[‘wav’, ‘flux’]
325         The sky lines from the file.
326
327     """
328     usecols = None
329     if filename:
330         filename = Path(__file__).parent.resolve() / filename
331         usecols = (0, 1)
332     else:
333         filename = Path(__file__).parent.resolve() /
334         ↪ 'utils/sky.salt'
335
336     lines = np.genfromtxt(
337         filename,
338         dtype=dtype,
339         skip_header=skip_header,
340         skip_footer=skip_footer,

```

```

338         usecols=usecols
339     )
340
341     logging.debug(
342         f"load_lines - {filename.name} - shape: {lines.shape}"
343     )
344
345     return lines
346
347 # MARK: Mask Traces
348 def mask_traces(
349     self,
350     spec: np.ndarray,
351     bpm: np.ndarray,
352     max_traces: int = 1,
353     tr_pad: int = 5,
354     bg_margin: int = 10,
355     lr_margins: list[int] = [10, 10],
356     h_min: float = 0.5,
357     h_rel: float = 1 - 0.05,
358 ) -> np.ndarray:
359     """
360         Masks the traces in the bad pixel mask.
361
362     Parameters
363     -----
364     spec : np.ndarray
365         The spectral data.
366     bpm : np.ndarray
367         The bad pixel mask.
368     max_traces : int, optional
369         The maximum number of traces to be masked.
370         (Default is 1)
371     tr_pad : int, optional
372         The amount to pad traces by.
373         (Default is 5)
374     bg_margin : int, optional
375         The margin size for the background.
376         (Default is 10)
377     lr_margins : list[int, int], optional
378         The left and right background margins at the spectrum edge.
379         (Default is [10, 10])
380     h_min: float, optional
381         The minimum height of a detected peak.
382         (Default is 0.5)
383     h_rel: float, optional
384         The relative height for the properties of a detected peak.
385         (Default is 1)
386
387     Returns
388     -----
389     bpm : np.ndarray
390         The updated bad pixel mask.
391
392     """
393     # Base mask
394     bpm[:, :bg_margin] = True
395     bpm[:, -bg_margin:] = True

```

```

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

```



```

509
510         )
511         axx.plot(
512             np.median(cs[ext], axis=0),
513             "r",
514             label=f"median {'E' if ext else 'O'}"
515         )
516         axx.legend()
517     plt.show()
518
519     logging.info(f"transform - {cs.shape} transformed.")
520
521     return cs, cw
522
523     # MARK: Plot
524     def plot(
525         self,
526         spectra,
527         wavelengths,
528         peaks,
529         properties,
530         arc: bool = False,
531     ) -> None:
532         plt.style.use([
533             Path(__file__).parent.resolve() / 'utils/STOPS.mplstyle',
534             Path(__file__).parent.resolve() /
535             'utils/STOPS_skylines.mplstyle'
536         ])
537         plt.rcParams['figure.subplot.hspace'] *= len(self.beams)
538
539     def norm(vals):
540         return (vals - np.min(vals)) / (np.max(vals) - np.min(vals))
541
542     # Load known lines
543     if arc:
544         lines =
545             self.load_lines(filename=f'utils/RSS_arc_files/{ARC_FILE}')
546     else:
547         lines = self.load_lines()
548
549     # noinspection PyTypeChecker
550     lines = lines[
551         (lines['wav'] > wavelengths[1][0][0].min()) &
552         (lines['wav'] < wavelengths[1][0][0].max())
553     ]
554
555     # Create plot for results
556     fig, axs = plt.subplots(2, self.ccds, sharex='col',
557                           sharey='row')
558
559     # Convert axs to a 2D array if ccd count is 1
560     if self.ccds == 1:
561         # noinspection PyTypeChecker
562         axs: np.ndarray[matplotlib.axes.Axes] =
563             np.swapaxes(np.atleast_2d(axs), 0, 1)
564
565     for fl in range(len(self.arc_list if arc else self.fits_list)):
566
567         # set color cycle

```

```

color = next(axs[0, 0]._get_lines.prop_cycler)]['color']

for ext in range(len(self.beams)):

    ccdrange = spectra[1][fl][ext].shape[-1] // self.ccds
    for ccd in range(self.ccds):
        # MARK: plot spectrum
        # (transformed)
        axs[0, ccd].plot(
            wavelengths[1][fl][ext][
                ccdrange * ccd:ccdrange * (ccd + 1)
            ],
            norm(spectra[1][fl][ext][
                ccdrange * ccd:ccdrange * (ccd + 1)
            ]) * 100 + 10 * ext + 30 * fl,
            color=color,
            linestyle='dashed' if ext else 'solid',
            label=f"${{{self.beams[ext]}}}_{{{fl + 1}}}^{{{{10 * ext + 30 * fl}}}}$" if ccd == 0 else
            None,
        )

# MARK: plot dev
# noinspection PyTypeChecker
sky_wavs, dev, peak_idx = self.min_diff_matrix(
    lines['wav'],
    wavelengths[1][fl][ext][peaks[1][fl][ext]],
    max_diff=self.max_difference,
)

# # MARK: width/width_init
# width = properties[1][fl][ext]['widths'][peak_idx]
# width_i = np.zeros_like(width)

# sky_i, i_dev, i_idx = self.min_diff_matrix(
#     lines['wav'],
#     wavelengths[0][fl][ext][peaks[0][fl][ext]],
#     max_diff=self.max_difference,
# )

# width_i = np.array([
#     properties[0][fl][ext]['widths'][
#         np.where(wav == sky_i)[0][0]
#     ]
#     if wav in sky_i else 1000
#     for wav in sky_wavs
# ])
# width_ratio = (width / width_i) - 1
# width_ratio[width_ratio < 0] = 0

# ylolims = width_ratio > self.max_difference
# width_ratio[
#     width_ratio > self.max_difference
# ] = self.max_difference // 2

ok = np.where(
    (sky_wavs >
     ↪ wavelengths[1][fl][ext].data[ccdrange *

```

```

618             ↪ ccd]) &
619             (sky_wavs <=
620             ↪ wavelengths[1][fl][ext].data[ccdrange * (ccd
621             ↪ + 1)])
622         )
623         axs[1, ccd].plot(
624             sky_wavs[ok],
625             dev[ok],
626             # yerr=(width_ratio[ok] * 0, width_ratio[ok]),
627             # lolims=ylolims[ok],
628             ".." if ext else "x",
629             # fmt="." if ext else "x",
630             alpha=0.8,
631             color=color,
632             # markeredgewidth=0.5,
633             # label=f"${self.beams[ext]}_{{{fl + 1}}}$",
634         )
635
636         logging.debug(f"plot - RMS:
637             ↪ {np.sqrt(np.mean(dev[ok] ** 2)):.2f}")
638
639     for ccd in range(self.ccds):
640         ccdrange = spectra[1][0][0].shape[-1] // self.ccds
641
642         # spectrum
643         # noinspection PyTypeChecker
644         ok = np.where(
645             (lines['wav'] >= wavelengths[1][0][0].data[ccdrange *
646             ↪ ccd]) &
647             (lines['wav'] <= wavelengths[1][0][0].data[ccdrange *
648             ↪ (ccd + 1)])
649         )
650         # noinspection PyTypeChecker
651         axs[0, ccd].plot(
652             lines['wav'][ok],
653             lines['flux'][ok] * 0,
654             'x',
655             color='C4',
656             label="\text{sc}{{salt}}\nModel" if ccd == 0 else None,
657         )
658         # noinspection PyTypeChecker
659         for x in lines['wav'][ok]:
660             axs[0, ccd].axvline(x, ls='dashed', c='0.7')
661
662         axs[0, 0].set_ylabel("Rel. Intensity ($\%$)")
663         axs[1, 0].set_ylabel(
664             "Closest Peak ($\Delta\lambda$)")
665
666     # for ax in axs[:, 0]:
667     #     ax.legend(loc='upper left', ncols=(fl + 1) * (ext + 1) +
668     ↪ 1)
669     leg = fig.legend(
670         loc='center',
671         ncol=min(8, len(spectra[0])) + 1,
672         columnspacing=0.5,
673         bbox_to_anchor=(
674             np.mean((

```

```

669         plt.rcParams['figure.subplot.left'],
670         plt.rcParams['figure.subplot.right']
671     )),
672     np.mean((
673         plt.rcParams['figure.subplot.bottom'],
674         plt.rcParams['figure.subplot.top']
675     ))
676 ),
677 )
678 leg.set_draggable(True)
679 for ax in axs[1, :]:
680     ax.grid(axis='y')
681
682 # fig.add_subplot(111, frameon=False)
683 # # hide tick and tick label of the big axis
684 # plt.tick_params(
685 #     labelcolor='none',
686 #     which='both',
687 #     top=False,
688 #     bottom=False,
689 #     left=False,
690 #     right=False
691 # )
692 axs[-1, 0 if self.ccds == 1 else 1].set_xlabel(
693     f"Wavelength ({self.wav_unit})"
694 )
695
696 # plt.tight_layout()
697
698 plt.show()
699
700 # Save results
701 if self.save_prefix:
702     fig.savefig(fname=self.save_prefix)
703
704 return
705
706 # MARK: Process all listed images
707
708 def process(self, arc: bool = False) -> None:
709     files = self.fits_list
710     if arc:
711         files = self.arc_list
712
713     logging.info(f"Processing '{self.beams}', lines.")
714
715     spectra = [[], []]
716     wavs = [[], []]
717     peaks = [[], []]
718     peak_props = [[], []]
719
720     for fl in files:
721         # Load data
722         spec2d, wav2d, bpm2d = self.load_file_data(fl)
723
724         # Mask traces in BPM
725         bpm2d = self.mask_traces(
726             spec2d,

```

```

727         bpm2d ,
728         max_traces=0 ,
729         bg_margin=15 ,
730         h_min=0.05
731     )
732     m_spec2d = np.ma.masked_array(spec2d, mask=bpm2d) # spec2d
733     m_wav2d = np.ma.masked_array(wav2d, mask=bpm2d) # wav2d
734
735     # Initial spectra
736     spec_i = np.mean(m_spec2d, axis=-2)
737     wav_i = np.mean(m_wav2d, axis=-2)
738
739     # Transform data
740     t_spec2d, t_wav = self.transform(
741         m_spec2d,
742         m_wav2d,
743         res_plot=self.can_plot
744     )
745
746     # Final spectra
747     spec_f = np.mean(t_spec2d, axis=-2)
748     wav_f = t_wav
749
750     # Find peaks
751     peaks_i, props_i = self.find_peaks(
752         spec_i,
753         **FIND_PEAK_PARAMS
754     )
755     peaks_f, props_f = self.find_peaks(
756         spec_f,
757         **FIND_PEAK_PARAMS
758     )
759
760     spectra[0].append([*spec_i])
761     spectra[1].append([*spec_f])
762     wavs[0].append([*wav_i])
763     wavs[1].append([*wav_f])
764     peaks[0].append([*peaks_i])
765     peaks[1].append([*peaks_f])
766     peak_props[0].append([*props_i])
767     peak_props[1].append([*props_f])
768
769     # Plot results
770     self.plot(spectra, wavs, peaks, peak_props, arc=arc)
771
772     if arc:
773         return
774     elif self.arc_list:
775         self.process(arc=True)
776
777     return
778
779
780 # MARK: Main function
781 def main(argv) -> None:
782     return
783
784

```

```
785 if __name__ == "__main__":
786     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.