Spectroscopic Data Reduction Pipeline for the Goodman High Throughput Spectrograph

Simón Torres-Robledo,[1] César Briceño,[1,2] and Bruno Quint[1]

[1]*SOAR Telescope, La Serena, Región de Coquimbo, Chile;*
storres@ctio.noao.edu

[2]*Cerro Tololo Interamerican Observatory, Casilla 603, La Serena 1700000, Chile*

Abstract. The Goodman High Throughput Spectrograph (Goodman Spectrograph) is a highly versatile instrument in operation at the SOAR Telescope on Cerro Pachon, Chile. It is capable of doing low to mid-resolution spectroscopy in a range from 3200 Å to 9000 Å. The data reduction pipeline is conceived as an easy-to-run software, that can process an entire night worth of data by execution of a simple command, with some arguments, from a terminal window. It is written almost entirely in Python, following several Python standard conventions. We aim towards using exclusively standard Python packages, such as Astropy and Astropy-affiliated packages, while at the same time allowing for fast and efficient computing. In its present form the pipeline produces fully reduced, wavelength-calibrated spectra. Flux calibration will be an add-on option for a later release.

1. Introduction

The Goodman Spectrograph (Clemens et al. 2004) is a highly customizable spectrograph in operation at the 4-meter SOAR Telescope on Cerro Pachon, northern Chile. It is capable of delivering spectra in a wavelength range from 3200 Å to 9000 Å with resolving power ($R = \lambda/\Delta\lambda$) ranging from 800 to 14,000 depending on the configuration. The Goodman Spectrograph is the workhorse instrument of the SOAR Telescope, with about 65% of the available time on the sky. Hence the need of a dedicated pipeline. Although IRAF is considered by many the standard tool in data reduction, its scope is to provide data reduction and processing capabilities to a rather wide variety of cases, thus is a bit hard to implement as an automatic pipeline. The SOAR Telescope in general is in a path to increased automation, looking forward to be an efficient follow up facility in the LSST era. We chose Python because of its versatility, completeness and because it allows for fast development and robust products. The existence and maturity of Astropy and its affiliated packages was also a big motivation to choose Python. Our initial plan considered a Python-only project but we had to change that because LaCosmic (van Dokkum 2001) did not provide satisfactory results and we had to shift to using a program called DCR (Pych 2004) which is written in C, works very well but does not integrate into Python easily, ideally this should be converted into a python C++ extension. LaCosmic implementation is still part of the pipeline and can be selected with a

command line argument. Also we wanted the pipeline to be open source and available for everyone; in fact the full project is available on GitHub.

Compared to other tools for data reduction the Goodman Spectroscopic Pipeline will provide clear advantages to users of the Goodman Spectrograph, by allowing them to run full processing of their spectra with a one-line command, that produces wavelength-calibrated, science quality results. Ours is not the first or only effort in providing a pipeline for reducing Goodman spectra, however, it is the first one developed not as a solution for one specific project, but rather as a community tool. It has been conceived to be simple to use and very well documented, so users can clearly understand each processing step, and even further develop and modify the software.

2. Pipeline Concept & Structure

The pipeline project started as two parallel projects, one for image reduction and one for spectra reduction. Very soon we realized they could not be isolated but should be part of an integrated system. They evolved into a single project with two main packages that work together. In the current stable version (1.0b1) the pipeline is structured as two packages with two scripts that call them.

The static structure still needs some refactoring and for sure there will be modifications to the dynamic structure.

For the next release we are developing a single-package structure that will contain sub-packages with new features, such as a telemetry package that can be used to monitor the instrument performance by comparing wavelength calibration parameters. Another very important feature that we are developing is integrated code testing.

3. Pipeline Use

The Goodman pipeline has been developed to be easy to use. There are two scripts or commands that you have to run in order to get fully processed data: `redccd` and `redspec`. `redccd` is for the initial image processing, such as bias subtraction and flat correction and `redspec` is for the spectroscopic part; identification, trace, extraction and wavelength calibration. Both scripts can be called with no arguments and should work well for most general use cases and well behaved data, but there is also the possibility of customizing the behavior up to a certain level. As we mentioned in the introduction, we had to deviate from our goal of having a purely Python-written pipeline and go with DCR which is a C program that works very well. Although the execution sequence was gracefully integrated into the pipeline's workflow there is a rigidity regarding parameter variation, because it uses a parameter file with a specific name and at a specific location.

For parsing command line arguments we use the standard python library `argparse` which also allows easy integration with other ways of calling the script, from a GUI for instance or even from a Jupyter Notebook. Also, it provides nicely formatted help text upon argument failure or by using `-h` or `--help`

3.1. redspec

The spectroscopic reduction code was probably the most complex to develop, given that Astropy's libraries were not ready at the moment. It starts by classifying the spec-

troscopic data, the output from `redccd`, using a set of keywords from the header, an important point to highlight here is that this pipeline does not rely on file names to classify the spectra, it only uses prefixes in order to filter data and for the user to easily keep track of the processing done, otherwise, it trusts that the header information is correct. As a result of the classification process it creates an object whose attributes contain information regarding the location of the data, time reference points, such as, the sunset and sunrise time. It also contain lists of `pandas.DataFrame` instances that group all the images that "belong together" according to the classification scheme. It will consider three possible scenarios: A comparison lamp that is not associated with any science target. A science target that does not have associated any comparison lamp. Groups of science targets and comparison lamps that are related to each other.

The target identification works by detecting the n-most intense peaks in the spatial profile obtained by collapsing the spectrum image along the dispersion direction using a median average. If there are any background features they are removed assuming the background should be flat. An equivalent implementation should be done for extended sources.

The tracing of the target is done by keeping track of the path of the spectrum's highest point. This is one of the parts that need more development in order to work for fainter sources or sources near very bright neighbors.

The current stable version has a simple extraction method only, meaning, it simply sums the counts along the spatial direction within a rectangular region that fully contains the spectrum. A background subtraction is done by selecting a clear zone outside this rectangular region and with the same number of pixels, if it is possible to get the background level at both sides of the spectrum, an average of both background regions is calculated and if only one region is possible it is averaged and multiplied by the width in pixels of the rectangular region that contains the spectrum.

Perhaps one of the most interesting parts of the code is the interactive and automatic wavelength calibration. It relies on a library of previously calibrated lamps, this makes things extremely easier compared to other methods you might think. The interactive mode was developed to avoid the *big wall* that the automatic wavelength calibration meant at that early stage of development. It allowed us to understand how the data behaved in the wavelength regime. Also and very importantly gave us the most useful insight that allowed us to later implement the automatic mode. In order to obtain a wavelength solution in interactive mode, the user has to click-identify the lines in two plots one with a reference lamp plus reference line values and another with the new lamp and marks of detected lines. The clicks are stored in two lists that later are used to calculate the fit of a low order Chebyshev polynomial. In Python this results in a callable object that you can give a pixel axis as argument and will return an axis in Angstrom. The use of the reference lamp is only for *visual reference*.

The automated wavelength calibration instead detects emission lines in the comparison lamps and then splits the spectrum in an even number of parts according to the number of lines detected. The reference lamp is also split. Equivalent portions are then cross-correlated thus obtaining an offset value for the corresponding center value, still in pixels, of the lines detected in the new comparison lamp. Using the mathematical model of the reference lamp's wavelength-solution the equivalent line center in angstrom is obtained using the following equation:

$$angstrom = model(pixel + offset) \tag{1}$$

As a first order filter for mismatched lines the cross-correlation values, which are stored separated, will be sigma-clipped using 2-sigmas and one-iteration only. This eliminates the most obvious mismatched lines. The ejected values also eject their pixel and angstrom equivalents allowing to calculate a cleaner new wavelength solution.

Using the Angstrom values previously found and the detected lines plus the newly calculated solution, the differences in angstrom are calculated to which values a new sigma-clipping is applied, again one iteration two-sigmas, since the distributions are not necessarily normal distributions.

Once these values are cleaned of rejected values the final solution is calculated using a low order Chebyshev polynomial.

4. Results

The pipeline has met our expectation of being an efficient and fast spectroscopic data processing tool. Although its scientific usefulness has not been validated yet, we believe we are close because the results are repeatable and consistent. Since there is no other production-ready pipeline available for the Goodman Spectrograph data, at the moment we can only compare it to IRAF. Figure 1 contains the result obtained for the same spectrum using the pipeline in automatic mode versus IRAF, by two different users. The good match is evident even more when zooming in. We find that, using the 400 l/mm grating with the 1″ slit, which yields a resolution R~ 800 (FWHM~ 6Å), the pipeline produces an automated wavelength solution with RMS ~ 0.3Å. This is roughly the same RMS obtained with IRAF.

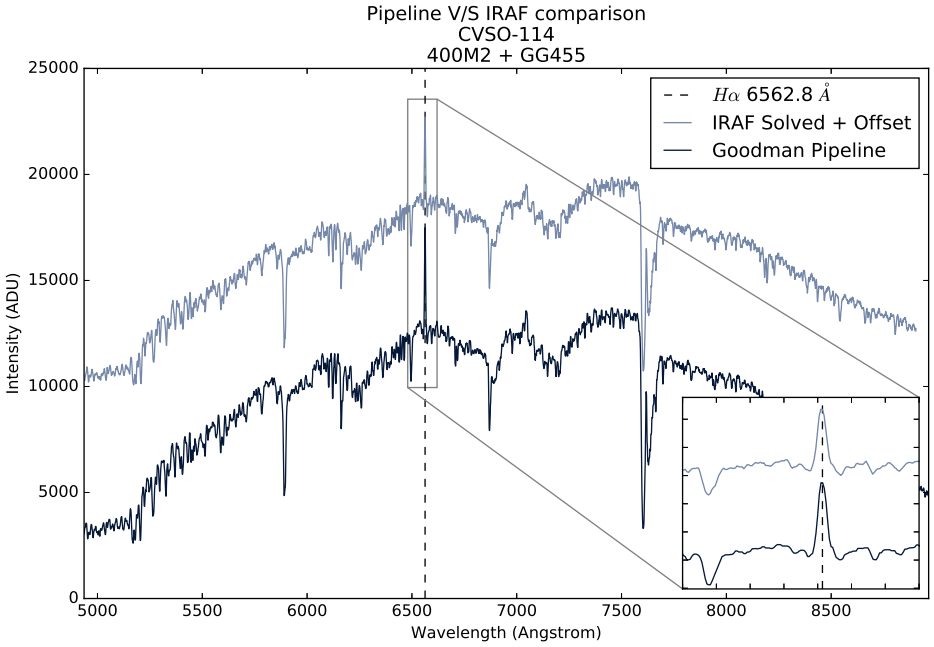

Figure 1. Results obtained using the Goodman Pipeline compared to IRAF. There is an arbitrary vertical offset to clearly set apart both results. The zoomed portion focuses on $H\alpha$, in emission, also allowing to view nearby features.

5. Future Work

At the moment of writing the pipeline is in its first beta version and as expected there are some features still missing. They should appear in future releases. Examples are flux calibration, and a complete implementation of fractional pixel extraction and optimal extraction (Marsh 1989) and (Horne 1986). We also want to change the Matplotlib UI by Qt UIs.

Other planned changes are: to modify the overall architecture of the pipeline, since it started as two separated projects and evolved into a single one, there is a development version that integrates everything as a single package. The ultimate goal in this regard is to make it comply with the Astropy Affiliated packages standard although we need to evaluate how quickly we want that.

Talking about Astropy, there should be a replacement of tasks that are becoming available in Astropy and Astropy Affiliated Packages as they mature, wavelength solution storage for instance.

On the coding side itself we must implement test code and in fact we might switch to a *test driven development* philosophy.

Acknowledgments. This research made use of Astropy, a community-developed core Python package for Astronomy (Astropy Collaboration, 2013). IRAF is distributed by the National Optical Astronomy Observatories, which are operated by the Association of Universities for Research in Astronomy, Inc., under cooperative agreement with the National Science Foundation.

References

Clemens, J. C., Crain, J. A., & Anderson, R. 2004, in Ground-based Instrumentation for Astronomy, vol. 5492 of Proceedings of SPIE, 331

Horne, K. 1986, PASP, 98, 609

Marsh, T. R. 1989, PASP, 101, 1032

Pych, W. 2004, PASP, 116, 148. astro-ph/0311290

van Dokkum, P. G. 2001, PASP, 113, 1420. astro-ph/0108003