8-2015

The local Cygnus cold cloud and further constraints on a local hot bubble.

Geoffrey Robert Lentner
University of Louisville

THE LOCAL CYGNUS COLD CLOUD AND FUTHER CONSTRAINTS ON A LOCAL HOT BUBBLE

By

Geoffrey Robert Lentner
B.S., Purdue University, 2013

A Thesis
Submitted to the Faculty of the
College of Arts and Sciences of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science
in Physics

Department of Physics and Astronomy
University of Louisville
Louisville, Kentucky

August 2015

THE LOCAL CYGNUS COLD CLOUD AND FUTHER CONSTRAINTS ON A LOCAL HOT BUBBLE

By

Geoffrey Robert Lentner
B.S., Purdue University, 2013

A Thesis Approved On

July 20, 2015

by the following Thesis Committee:

Thesis Director
James Lauroesch

Lutz Haberzettl

John Kielkopf

Ryan Gill

DEDICATION

This work is dedicated to my two beautiful boys, Henry and Charlie.
Their lives give mine meaning and motivation.

ACKNOWLEDGEMENTS

One does not simply earn an advanced degree without an innumerable quantity of assists, both in life and in academics. There have been and continue to be individuals who selflessly promote and support me in the pursuit of my interests. Though I cannot possibly acknowledge every person who influenced me in a way that inexorably lead to my current situation, I want to highlight the contributions of many people that have been instrumental.

There will have been countless minute situations and occurrences that have shaped my growing mind to prepare me for what ultimately became a passion. Despite being what were completely unguided forces, I am thankful to have been born into a world and society where I am enabled to spend my life pursuing knowledge.

Two individuals, though, who indeed had my best interests at heart and mind were my mother and father. I would like to formally express my debt and gratitude to my parents Robert William Lentner III and Jennifer Anne Wedge. I suspect my sudden and spontaneous leap into natural science was not expected when it occurred, or at least my eventual decision to apply for graduate studies. I truly believe that throughout my life my parents have supported me in my decisions and interests, whatever they may be. I have always felt and continue to feel loved and encouraged by them. They will tell me how proud they are of what I've accomplished and what I will accomplish. I want to express a heartfelt thank you, to say that through it all I always felt supported.

There is another person from my childhood that I must recognize as being import to my growth, both intellectually and in other areas. We did not necessarily see eye-to-eye at times but as we grew older I saw him as an inspiration. My brother, Aaron David Lentner, is arguably one of the most import people to have helped me get where I am at the present moment. It is literally the case that I could not have pursued science as a career without

his help and I am forever thankful for his vote of confidence in me when I needed it most.

I cannot neglect to acknowledge the love and support of my incredible wife, Whitney Leigh Lentner. She is my first and forever romance. She has been my encouragement even before I started on the path that lead to this thesis. Well into my undergraduate career I had the audacious idea to drop everything I had been working on and study physics. With all things in my life, she was the first person I came to. It was a complete upheaval and added much uncertainty to our future. She supported me then, and now. With our two beautiful boys Henry Francis Lentner and Charlie Alexander Lentner, graduate studies have been for me an especially rigorous challenge, particularly these past few months. I want to thank her for being such a caring and supportive wife and mother.

Academically, there are many individuals who contributed to my current success. There were several teaching assistants and professors who taught me both the substance of physics and what it really meant to be a scientist. In particular I want to mention two faculty from Purdue University that are directly responsible for getting me into a graduate program in physics. As a professor, Dr. Jaehyon Rhee peaked my motivation to pursue science. He provided my first research experience with nothing to go on but my passion. Without this initial spring board, I could not have ultimately joined the academic world. Also, Dr. Rebecca Lindell saw in me what at times I did not see myself. Without her support and constant confidence I would not have been accepted by a graduate school.

Here at the University of Louisville, many have given me both their support and collaboration. Chief among them is my advisor and mentor, Dr. James Lauroesch. We have made and will continue to make a great team. Both his intellect and kindness have gone unmatched. He has always taken an interest in my thoughts and ideas; in my two years in the department I have worked on many projects, even ones not related to our research into the ISM. It is often the case that graduate students are at the mercy of the advisor they find themselves working with. I can say with confidence that I could not have had a better advisor, intellectually or otherwise. Others have as well been in integral component to my experience. In particular, I want to acknowledge both the collaboration

and constructive criticism of the faculty and graduate students in the astrophysics group. Wherever my career takes me, I will maintain a resolute bridge between me and the department that gave me my start.

This research is based on spectral data retrieved from the ELODIE archive at Observatoire de Haute-Provence (OHP) (Moultaka et al., 2004) and has made use of Matplotlib, a 2D graphics package used for Python for application development, interactive scripting, and publication-quality image generation across user interfaces and operating systems (Hunter, 2007); SciPy, open source scientific tools for Python (Jones et al., 2001–); NumPy, a structure for efficient numerical computation (van der Walt et al., 2011); Astropy, a community Python package for astronomy (Astropy Collaboration et al., 2013); APLpy, an open-source plotting package for Python hosted at http://aplpy.github.com; and the SIMBAD database, operated at CDS, Strasbourg, France (Wenger et al., 2000).

ABSTRACT

THE LOCAL CYGNUS COLD CLOUD AND FUTHER CONSTRAINTS ON A LOCAL HOT BUBBLE

Geoffrey Robert Lentner

July 20, 2015

Recent studies of the local interstellar medium have identified regions of nearby cold neutral gas with unexpected low temperatures and high pressures well within the boundaries of the local cavity (Meyer et al., 2006). Now, a multi-wavelength study of the local Leo cold cloud (LLCC) has strengthened our understanding of this apparent conflict with the conventional view of this environment (Peek et al., 2011, Meyer et al., 2012). The soft X-ray background observed here cannot be completely accounted for by a local hot bubble model (Snowden et al., 2015).

Interstellar absorption of Na I (D2 λ5889.591 and D1 λ5895.924) was found towards δ Cygni (Welty et al., 1994). The present study will cover an extensive preliminary search of a 40 deg^2 field centered here. Using archival data from ELODIE, 41 of 284 targets have measurable interstellar absorption. Three of these occur along site lines to stars within the local cavity and represent the identification of a new nearby cloud.

Further, the author has developed a suite of precision software tools packaged as an open source library for Python 3. This library builds on other popular modern packages for Python in astronomy and may help to accelerate the pace of investigation and discovery in spectroscopy.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 The Interstellar Medium

1.1.1 Background

Before embarking on a study of a particular constituent of the local interstellar medium and our search for nearby cold clouds, it is instructional to begin with a brief description of what exactly the interstellar medium (ISM) refers to. Informative texts include Draine (2011), [1] Spitzer (1978), [2] and the earlier Aller et al. (1968). [3]

Generally, we can regard essentially all matter and interactions in the galaxy *not of the stars* as being a constituent of the ISM, or at least contributing to it, though a more limited grouping is more useful. The Milky Way galaxy is approximately 100,000 light years (30.7 kiloparsecs) in diameter and home to on the order of a few hundred billion stars. However much mass is contained within all the stars in the galaxy, there is comparable mass in the diffuse gas between them. On average, there is approximately one atom/molecule per cubic centimeter in the ISM. This is in fact a better vacuum than most modern laboratories achieve. And yet, across the vast distances of interstellar space, this material together forms intricate structure on both small and large scales.

Both the composition and dynamics in the ISM are the subject of research, old and new. We know that the ISM is mostly hydrogen and helium (in proportions we expect

[1] *Physics of the Interstellar and Intergalactic Medium*
[2] *Physical Processes in the Interstellar Medium*
[3] *Nebula and Interstellar Matter*, Stars and Stellar Systems, Vol VII

from knowledge of the stars and Big Bang nucleosynthesis), more or less 3/4 and 1/4 respectively. The hydrogen in our galaxy can be broken into two major components. The first of these is referred to as HII regions (though this notation is somewhat unique to astronomy) and is composed of hot, photoionized (H^+) hydrogen that surrounds young O and B type stars and is also distributed in the disk and halo. This gas is even more diffuse than described previously and lives at temperatures on the order of millions of kelvin. In contrast, large portions of the galaxy contain regions of hydrogen gas referred to as HI. This is cold ($\sim$ 10 K) neutral hydrogen. These regions trace much of the structure of our galaxy and constitute a considerable fraction of the mass of the ISM. This gas can emit a signature 21-cm line from hyperfine atomic transitions. There is so much that could be said about the rich physics played out across the galaxy – emission, absorption, extinction, gravitation, magnetic fields, electrostatics, star formation; a complete survey would include a substantial fraction of modern astrophysics.

One important component within the ISM though is the abundance and distribution of *metals*, the elements heavier than helium ($z > 2$). These element paint a picture of the history and evolution of our galaxy. The first stars contained precious little heavier elements, relatively speaking, and in their death polluted the ISM with their enriched material. Every generation of stars that has followed continued this cycle of enrichment. These heavier elements offer tracers of particular physics. We can use absorption studies to not only understand the chemical evolution of our galaxy but to learn the local structure and dynamics in the immediate vicinity of the Sun.

1.1.2 The Local Bubble

The Local Bubble refers to a relatively evacuated region (or void) immediately surrounding the Solar System. Within the past 25-60 million years an expanding wave from supernovae explosions blew out an already low density region created by massive

stellar winds and displaced the local ISM to create large magnetized bubbles hundreds of parsecs across (Frisch, 2007). The boundaries of this cavity have been traced using several techniques including color excess, HI radio emission, etc. Generally, this void is thought to be filled with hot, diffuse, X-ray emitting gas at around 10^6 K (Snowden et al., 1998). Cool, more dense molecular clouds dot the boundaries and have been compressed to create star forming regions.

Our understanding of this local environment has taken considerable shape over the past few decades (and even just in the past few years). Recently, we have found that there are several warm (T $\sim 10^4$ K) clouds inside this cavity; and even that the Solar System itself lives in one of these warm clouds (Lallement et al., 2003). It was generally believed that there was no colder interstellar material in this Local Hot Bubble (LHB); however, one cloud originally identified by Verschuur (1969), has presented a challenge to this view for many years. Meyer et al. (2006) used observations of Na I (D2 λ5889.591 and D1 λ5895.924) and found cold interstellar absorption towards 23 stars behind the cloud. The distance to the closest of these stars ($\sim$ 42 parsecs) places this local Leo cold cloud (LLCC) well within the boundaries of the local cavity. Peek et al. (2011), using optical observations of the LLCC, constrains the distance to this material between 11 and 24.5 parsecs. Meyer et al. (2012) used observations from the *Hubble Space Telescope* of interstellar C I to directly measure the pressure in the cloud. The 40,000-80,000 cm^3 K thermal pressure found is much greater than was expected and indicates that it is not in thermal pressure equilibrium with its surrounding environment. Further, analysis of *Rosat* All Sky-Survey 1/4 keV (C-band) data (Peek et al., 2011, Snowden et al., 2015) shows only weak shadowing of X-ray flux towards the LLCC, indicating that a LHB of diffuse X-ray emitting gas cannot be the only source of the soft X-ray background radiation (SXBR).

1.1.3 Searching for Interstellar Clouds

It is difficult to make generalizations about the consequences and constraints placed on a LHB using only a single object. It is important to search for other, possibly similar, cold clouds in the vicinity of the Sun to further strengthen any conclusions drawn here. To this end, we are interested in conduct such a search. Recently, another field of astronomy has enjoyed much success and popularity – the search for extrasolar planets. Several new archives of moderate resolution spectra have become available and public access. These archives offer an opportunity for science outside the domain of radial velocity studies. This work consists of one such preliminary search for a nearby cold cloud.

Welty et al. (1994) has shown absorption of interstellar sodium towards δ Cygni. The hypothesis is now that there exists another of these cold neutral clouds in the direction of Cygnus. To investigate the possibility of placing further constraints on a Local Hot Bubble, an extensive search of a 40 degree square field centered here consisting of 284 spectra from the ELODIE archive has yielded numerous targets for which absorption is present; the closest of these is HD 184006 at $\sim$ 37 parsecs.

1.2 Thesis Overview

1.2.1 Content

Each major component of this study is discussed in the following chapters, largely in procedural order. All the data used here was retrieved from the ELODIE archive. In Chapter 2, I discuss the exhaustive procedures necessary to acquire *all* relevant spectra to the present work. Every spectrum needed to be calibrated, including the removal of telluric absorption lines and performing heliocentric velocity corrections. These steps in and of themselves (considering the quantity of data) in addition to some measure of quality assurance were essentially the motivation for creating a new library of precision

software tools in a modern language and style (to be discussed). I cover these steps in Chapter 3. Further, the actual practice of fitting absorption profiles and extracting accurate measurements with appropriate uncertainties necessitated the development of an interacting, flexible fitting routine. The details of this routine as well as numerical and visual representations of the results of this work are provided in Chapters 4 and 5, respectively. In Chapter 6, I discuss the relevance of these results to the Local Hot Bubble model and our understanding of the soft X-ray background (SXRB). There is still much work to do in this area, specifically as it relates to the local Cygnus cold cloud (hereafter LCCC); A recent *Hubble Space Telescope* proposal is attached in Appendix A (the author is a co-investigator on the proposal). Further, I am publishing all the relevant metadata for all of the observations used in this study. This includes the file indices for accessing the data from the archive as well as the date and time of observation, the exposure time, and the signal-to-noise ratio. This metadata is provided in Appendix B in Tables B.1 - B.11. The tables in Chapter 5 and Appendix B that correspond to targets with measured interstellar Na I absorption are annotated with index numbers that label markers in Figures 5.2 and 5.3. Last, in Appendix C the full source code for the entire library is included for completion (with a few exceptions). This code is up-to-date as of August 10, 2015. Though, most of the code is provided, some auxiliary components have been neglected for reasons of practicality.

1.2.2 SLiPy

We are interested in continuing to identify other cold clouds; therefore, the ELODIE archive has been parsed and retrieved for every unique stellar target - one file with the highest signal to noise. This constitutes 5, 296 targets (if we restrict ourselves to only HD, HR, GC, GJ, and BD catalogues). There are 284 targets within 20° of δ Cygni (the field of interest). With this many data elements, it is import to not only be consistent,

but to be efficient. Modern astronomy is headed in the direction of so called *Big Data* and it is becoming more critical for astronomers to develop new technical and software skills and strategies to meet this demand. Here, this quantity of spectra is at the limit for what is possible to analyze iteratively and with consistent quality. **SLiPy** (the Spectroscopy and astrophysics Library for Python 3) was created to accelerate my work in managing, analyzing, visualizing, and extracting measurements from spectra. It is open source (under the GNU General Public License, version 3) and is free to download from its website.

The focus of this research has been on the science; however, the reality is that the bulk of the work done here has been related to data management. We not only need to do good science but we need to do it efficiently and consistently. One of the core components of science is reproducibility. The idea behind creating much of this software has been that if it is at all possible to automate a particular procedure, the quality and efficiency of the analysis increases. Further, later work can much more easily confirm or reject these and other results with confidence because the process for making or failing to make a detection is reproducible.

In an effort to simultaneously lay bear a census of my research and document the syntax of using this library (SLiPy) I will provide code snippets along the way. The alternative would be to provide another appendix of documentation and usage details that would be referenced throughout. In-text examples however are more easily comprehensible. I will necessarily assume that the reader is familiar with the Python language to the extent that basic concepts will not be explained here. In keeping with the syntax style of packages/modules/functions/methods in Python, all of these will be referenced using bold face type. For example, **Algorithms.Functions.Lorentzian()** is referencing the Lorentzian *function* inside the Functions *module* contained within the Algorithms *sub-package* of the SLiPy package. Leading punctuation will be used to indicate a function that is a *method* for a *class* (e.g., **.resample()** is a method of the

Spectrum class). Also, the name of explicit arguments to functions will be given via italics. For example, *filepath* and *keyword* are arguments to the **Fits.Header()** function. Last, the snippets will at times compound sequentially. That is, previous variables and imports will not be reproduced in every snippet. Further, the *representation* of the result on the final line of the snippet may be printed immediately below it. For example:

```
from slipy import Simbad

# retrieve the right ascension and declination from SIMBAD
ra, dec = Simbad.Position("Alf Lyr")
```

```
[<Quantity 279.23473479 deg>, <Quantity 38.78368896 deg>]
```

In the above code snippet, the numerical output below the horizontal line is the result of issuing `[ra, dec]` at the interpreter.

CHAPTER 2

DATA

2.1 Archival Data

The recent growth in the study and search for extrasolar planets has produced several spectral archives, much of which is public access (i.e., no longer proprietary). These archives offer a number of opportunities to do good science outside the domain of extrasolar planets. This study is one such opportunity. We have an interest in searching both the Keck HIRES (Vogt et al., 1994) and ELODIE (Baranne et al., 1996) archives for absorption features indicative of local interstellar clouds. The present study principally relies on the ELODIE archive, as the data products offered are already reduced and ready for investigation. The following sections outline the use of this archive. Future work will include the automated reduction and investigation of the Keck HIRES data.

2.2 The Elodie Spectrograph

This study is based on data retrieved from the ELODIE archive. ELODIE is a fibre-fed, cross-dispersed echelle spectrograph that was mounted [1] at the 1.93 meter telescope at the Observatoire de Haute-Provence, France (Baranne et al., 1996, Moultaka et al., 2004, Prugniel et al., 2007). Its design purpose was to provide high accuracy radial velocity measurements (needed to search for brown-dwarfs and giant planets around

[1] The spectrograph was replaced by the SOPHIE (Spectrographe pour l'Observation des Phénomènes des Intérieurs stellaires et des Exoplanètes) echelle spectrograph at the 1.93 meter telescope in 2006 (Bouchy and Sophie Team, 2006).

nearby stars) and has resolution 42000.[2]

The ELODIE archive is an online repository of all the data taken by the spectrograph. The database contains 35, 517 spectra. [3] Users can access content via an intuitive web interface that allows searching for specific targets, by position, as well as some advanced search options. The search results provide a list of names of targets resolved by the archive with a preview option and a customization tool that allows for some pipeline processes before downloading. Among these is the ability to resample the spectrum (e.g., 5850-5950 Å at 0.01 Å pixel^{-1}) and to perform continuum normalization. The high accuracy radial velocity spectra in addition to the already reduced data products with further pipeline options made ELODIE an attractive option for this research and was the reason the decision was made to rely on it.

2.3 Acquisition Methods

In addition to the web interface, the ELODIE archive can be accessed directly using url scripting. For example:

```
http://atlas.obs-hp.fr/elodie/E.cgi?&c=i&o=elodie:20031001/0016
    &z=wrs|fca[1,nor]|wrs[1,5850,5950,0.01]&a=mime:application/x-fits
```

The above url has several components to it. First, `20031001/0016` is the index of a particular file in the archive (like those listed in Tables B.1 - B.11 in Appendix B). Second, the `z=wrs|fca[1,nor]|wrs[1,5850,5950,0.01]` segment is a pipeline command. `fca[1,nor]` says to perform the continuum normalization. `wrs[1,5858,5950,0.01]` is a wavelength resampling command; in this case we have requested to resample as in the example used previously from 5850 - 5950 Å at 0.01 Å pixel^{-1}.

[2] In astronomy, resolution is defined as $R = \lambda/\Delta\lambda$.

[3] Previously, the archive contained around 30, 600 spectra and only about half were public access. In the past few months however all spectra have been made public.

Previous work has identified an interstellar Na I D1 component [4] towards δ Cygni at a distance of 51 pc within the edge of the Local Bubble (Lallement et al., 2003, Welty et al., 1994). As such, we were interested in searching a field centered here to identify whether there was a large interstellar cloud in this vicinity. A 20 degree radius (or more precisely, everything with a right ascension and declination within 20 degrees of δ Cygni) provided ample sampling of the area. The difficulty arises when trying to browse such extensive search results manually to decide which spectra are best and to only select one for a unique stellar target. To this end, the *entire* archive was accessed to the extent that by giving the positional search form the whole sky, every single file in the archive was returned in the results. This text was exported and used to construct an ascii catalogue of the entire archive containing a listing of all files. These files are organized according to the identifier they belong to. Information also listed is the signal-to-noise ratio for the spectrum reported by the archive. With this information we can use scripting techniques to comprehend the archive using any criteria of interested. How exactly this is done is covered in the next section.

After deciding what spectra (the indices of the spectra we want) to download, the actual process of downloading them has been incorporated into SLiPy. It is quite useful to have a piece of software be able to access data on your behalf in an intuitive way via a standardized API [5]. Within the SLiPy package, there is now a dedicated **Data** sub-package with a few functioning data management products. One of these is the **Elodie** module (see Appendix C.5 for the source). In the future, the goal is to incorporate many other modules for accessing data from astronomical archives.

The **Elodie.Archive** is an object that once instantiated reads in the raw ascii catalog mentioned previously (from *../Data/Archives/Elodie.csv* by default) and parses the

[4] W_λ = 20.2 mÅ, $v_\odot$ = -18.54 km s^{-1}, N = 29.6 10^{10} cm^{-2}, b = 0.42 km s^{-1}. (Welty et al., 1994)

[5] Wolfram's Mathematica © for example is a very sophisticated language that has built in data accessor methods.

data into a set of two Python *dictionaries*, **.files** and **.data**. These are indexed by unique target names. The **.data** member has a list of pairs consisting of the name of the file and the signal to noise for that spectrum. **.files** contains the reduced archive and by default consists only of HD, BD, HR, GC, and GJ objects, [6] choosing the file pertaining to the spectra with the highest signal-to-noise ratio available.

```
from slipy.Data import Elodie

archive = Elodie.Archive()

"HD187642" in archive.files # returns True (Altair)
"HD045348" in archive.files # returns False (Canopus)

altair = archive.data["HD187642"]
```

```
[['elodie:19980702/0024', 206],
 ['elodie:19960902/0012', 239],
 ['elodie:20031001/0017', 228],
 ['elodie:20031001/0016', 280],
 ['elodie:20031001/0018', 245],
 ['elodie:20031001/0014', 180],
 ['elodie:20031001/0015', 201]]
```

Also included within the module are the **Elodie.Script()** and **Elodie.Download()** functions. **Elodie.Script()** was not really intended as an end-user product, but is a convenient helper-function to **Elodie.Download()**. It merely takes the properly formatted file index of a spectrum from the archive (and optionally a pipeline command-string) and constructs the necessary url script to download that file to your machine. **Elodie.Download()** is a high-level function that actually retrieves the file for you. The idea here is that in the future the **SLiPy.Data** sub-package will be expanded to provide access to numerous archives, surveys, and data sets - all with similar accessor methods. So while the content of the different archive members will invariably be distinct, the *methods* for retrieving that data from within the software will be uniform. For this study, the **Elodie** module allowed me to concisely crawl and sort the archive using **Simbad** (a module

[6] Henry Draper, Bonner Durchmusterung, Bright Star (though abbreviated using the name of its predecessor the Harvard Revised Photometry Catalogue), General Catalogue, and Gliese-Jahreiss (respectively) are all the names of stellar catalogues.

discussed in the following section). Ultimately, I was able to automatically identify the 5, 296 unique stellar targets [7] and retrieve the corresponding spectrum with the highest signal-to-noise ratio available. [8]

Elodie.Download() takes an arbitrary number of file names (in the format represented above) and actually downloads them to your machine for you. There are a few options as well. First, both the continuum normalization and wavelength resampling pipeline options are accessible. By default, *normalize* is set to `True` and *resample* is left unspecified (leaving the spectrum at the original wavelength calibration). To specify a desired resampling, pass a tuple of length three to *resample* (starting wavelength, ending wavelength, and resolution in angstroms and angstroms per pixel respectively). Further, by default the FITS files will be downloaded to your current working directory. To specify an alternative location, give the desired relative path as *outpath*. Also, all files downloaded will have names matching the pattern, "elodie-yyyymmdd-nnnn.fits". If you provide a list of file *names* equal in length to the number of files requested, the downloaded FITS files will be given these alternative names instead. Last, this can be a non-trival length of time needed to download a data set. By default *verbose* is set to `True` and a progress bar is displayed while the files are being downloaded and an estimated time of completion is given.

```
# the "data" has the actual file name as the first "entry" in each pair
regulus = [ entry[0] for entry in archive.files["HD087901A"] ]

Elodie.Download( *regulus, resample=(5850, 5950, 0.01),
    outpath="./Data/")
```

```
Downloading 6 files from Elodie ...
[=================>                ]   50.00 %  ETC: 2015-06-18 @ 14:24
```

[7] Sometimes in the ELODIE archive there are stars which are the same but are labeled with an appended "A" or "B" (not necessarily corresponding to binary systems). As such, about a dozen of the targets from the 284 identified in the Cygnus field are actually duplicates. This has been handled appropriately.

[8] While not necessarily representing the absolute best piece of data, using the S/N as a benchmark for automatically sorting the data quality is useful. In a few instances, despite having the highest S/N, spectra from the archive were of too poor quality to provide any meaningful results.

In the above code snippet the `*regulus` syntax means to expand the list as if we had typed them in manually. Here, we would have downloaded the six files from the Elodie archive for Regulus [9] to a "Data" directory within the current working directory.

2.4 Sorting and Data Management

As alluded to previously, there are a number of modules developed for SLiPy that provide tools for handling and managing astronomical data. Among these (to be discussed here) are the **Simbad**, **Fits**, and **Spectrum** modules.

2.4.1 The Simbad Database

The **Simbad** (see Appendix C.17) module allows the user to query the SIMBAD astronomical database from *inside* Python or shell commands and scripts. It's four current major functions **Position**, **Distance**, **Sptype**, and **IDList** return real variables with appropriate types ready for use. While the module's implementation was remarkably strait forward, its impact for this research was nearly unmatched in terms of what it afforded to me in my ability to quickly and confidently put data elements into context and organize them. A full 85% of the content provided in Tables 5.1 - 5.11 and B.1 - B.11 were auto generated using these functions.

The approach taken is similar to the accessor method from the **Elodie** module. Here, a **Simbad.Query** class uses the **Simbad.Script()** and **Simbad.URLEncode()** helper functions to put together a url script recognized by the SIMBAD astronomical database. When querying the database, a file is returned with some standard retrieval information and statistics along with the results of your query. The format for this file is quite standardized and allowed for a common query object that is instantiated by each of

[9] Unfortunately, as described previously, some of the targets have an "A" appended to them. If you think a star is in the archive and it comes up false, try adding an "A".

the specialized functions. That is, the **Simbad.Distance()**, **Simbad.Position()**, **Simbad.Sptype()**, and **Simbad.IDList()** functions are wrappers to creating the query object. They each take a string value specifying the identifier of the target of interest. Each function then creates the query object with a specialized parameter string [10] specific to the attribute of interest and retrieves the file from SIMBAD. The final step is to extract the relavent information from the file and return the data with an appropriate type (either a string or a **Quantity** [11]).

The following are some example uses. The format of the identifier string is very flexible and can be literally anything that would otherwise be recognized by conducting a search *by identifier* using the online search form at *simbad.u-strasbg.fr/simbad*.

```
from slipy import Simbad

# distance in parsecs to Regulus
distance = Simbad.Distance("regulus")
```

```
<Quantity 24.313153415998055 pc>
```

```
# position of Messier 31
ra, dec = Simbad.Position("M31")
```

```
[<Quantity 10.684708 deg>, <Quantity 41.26875 deg>]
```

```
# the prototype for its namesake variable type
sptype = Simbad.Sptype("delta scuti")
```

```
'F2IIIp'
```

```
# search the database for alternative identifiers of the "north star"
idlist = Simbad.IDList("north star")
```

```
['2MASS J02314822+8915503',
 'ADS  1477 AP',
 '** STF   93A',
 'WDS J02318+8916A',
 '** WRH   39',
 'NAME Lodestar',
 'PLX  299',
```

[10] For example, `%C00(d;C)` is the code for retrieving the right ascension and declination of the target in decimal degrees.

[11] The **Quantity** object is meant to represent a value that has some unit associated with the number. (Astropy Collaboration et al., 2013)

```
'SBC9 76',
'*   1 UMi',
'* alf UMi',
'AAVSO 0122+88',
'ADS  1477 A',
'AG+89    4',
'BD+88     8',
'CCDM J02319+8915A',
'CSI+88     8  1',
'FK5  907',
'GC  2243',
'GCRV  1037',
'GEN# +1.00008890A',
'GSC 04628-00237',
'HD   8890',
'HIC  11767',
'HIP  11767',
'HR   424',
'IDS 01226+8846 A',
'IRAS 01490+8901',
'JP11   498',
'N30  381',
'NAME NORTH STAR',
'NAME POLARIS',
'PMC 90-93   640',
'PPM    431',
'ROT  3491',
'SAO    308',
'SBC7    51',
'SKY#  3738',
'TD1   835',
'TYC 4628-237-1',
'UBV   21589',
'UBV M   8201',
'V* alf UMi',
'PLX  299.00',
'WDS J02318+8916Aa,Ab']
```

Taking the names of the available unique targets from the ELODIE archive and doing a search for their right ascension and declination I was able to restrict my list to site lines inside the 40° field of interest. Before downloading the files, I was able to again query SIMBAD on this smaller set of targets and retrieve their spectral types. Inserting the first letter of the spectral type into the path of the *names* given to **Elodie.Download()** (e.g., "/A/") after creating the appropriate directory structure, I was able to easily organize the data by spectral type.

2.4.2 Managing FITS Files

In order to analyze these data we need to get it out of the FITS files. Many software libraries and languages have implementations for doing such. In fact, within Python the Astropy Project [12] already has significant capabilities (Astropy Collaboration et al., 2013). The **Fits** module within SLiPy builds on the functionality provided by **astropy.io.fits**. The most significant item provided by the module is the **Fits.GetData()** function. I'll document its usage here, and in relation to it the usage of some other related functions.

Fits.GetData() can be used in two ways. First, an arbitrary number of *files* can be requested via their relative paths. Second (or in combination with the first), you can provide the name of a *toplevel* directory under which there exists some FITS files. With the first method it will simply extract the data from those listed files. With the second, either the **Fits.Find()** or the **Fits.RFind()** function will be used first to search under or recursively under the *toplevel* directory named. By default it uses **Fits.Find()** but if *recursive* is assigned `True` then it uses **Fits.RFind()**. Both **Fits.Find()** and **Fits.RFind()** take two optional arguments. If no arguments are given it will return a list of the relative paths to *.fits files below the current working directory. If given an argument, it is expected to be the relative path to an alternative directory (under which it will search for *.fits files). The second argument, when provided, is an alternative glob pattern to match (e.g., *.fits.gz).

```
28 # find file path names under `Regulus` directory and import data
29 files   = Fits.Find("Data/Regulus")
30 regulus = Fits.GetData(*files)
31
32 # files organized under `Data/All/{O,B,A,F,G,K,M}` directories
33 spectra = Fits.GetData(toplevel="Data/All", recursive=True)
```

```
Importing data from 284 Fits files ...
[====================>                        ]   47.18 %
```

[12] The Astropy Project is a community effort to develop a single core package for Astronomy in Python and foster interoperability between Python astronomy packages (www.astropy.org).

In the above snippet, we have imported two groups of spectra. First, the six files for Regulus from the previous example are now assumed to be under the "Data/Regulus" directory. Their paths are read in and expanded in the *files* arguments to the **Fits.GetData()** function. Similarly for the remainder of the FITS files, except they are housed under subdirectories according to spectral type. The return type of this function is a Python *list* of **Spectrum** objects.

2.4.3 The Spectrum Datatype

In an effort to reduce complexity and make operations more intuitive, a new data type was created, the **Spectrum**. As of today, I argue that there does not currently exist an implementation of a 1D spectrum object for the Python language as functional and dynamic as this one. It is important to highlight this aspect of SLiPy because throughout the following chapters I will necessarily refer to the arguments of the various functions as being of type **Spectrum**.

A **Spectrum** can be created two distinct ways. The first (as done by the **Fits.GetData()** function) is to provide a *filename*. With a proper FITS file containing both the flux data and also the corresponding wavelength calibration, the **Fits.Header()** function is used to retrieve the "CRPIX1", "CRVAL1", and "CDELT1" values from the header and construct a wavelength array equal in length to that of the flux data. [13] Once created, the flux data and wavelength array can be accessed independently with the **.data** and **.wave** members respectively. Units are bound to the arrays via **astropy.units**. [14] If the units are not specified by the *xunits* and *yunits* keyword arguments then the flux data (*yunits*) will be dimensionless and the wavelength array (*xunits*) will be in Angstroms. If

[13] Although you can request an alternative *key* to be read from the FITS files by specifying it as an argument of the same name (e.g., `Spectrum("myData.fits", crpix1="something else")`).

[14] While units are bound to the arrays, it is assumed unless provided by the *xunits* and *yunits* keywords that they are dimensionless.

wavecal is assigned `False` and no wavelength calibration is read from the header, an array of pixel values will be generated in its place. Alternatively, instead of providing a *filename* you can provide a *data* array directly. If no *wave* array is given as a second argument, a pixel array will be generated as in the previous case. If no units are bound to these arrays, they will be assigned as dimensionless.

```
34 from slipy import Spectrum
35
36 regulus = Spectrum("Data/elodie_19960502_0013.fits")
```

```
[ 1.01038098  1.01141405  1.01213932 ...,  0.9898265   0.99048185
  0.99079037]
[ 5850.    5850.01  5850.02 ...,  5949.98  5949.99  5950.  ] Angstrom
```

```
37 import numpy as np
38
39 data     = np.random.rand(4)
40 spectrum = Spectrum(data)
```

```
[ 0.91460036  0.04310631  0.92479238  0.40179515]
[ 1.  2.  3.  4.] pix
```

In addition to the benefit of simplicity in passing around data as a **Spectrum** such that it brings with it its wavelength information (plus other attributes like header information can be attached), creating such an object allows for operator overloading in a fashion that behaves like you would think a spectrum should. Addition, subtraction, multiplication, division, comparison operators, and shift operators are all defined. [15] If both operands are of type **Spectrum** the operation is applied element wise, but after insuring that both are on the same wavelength domain. If the RHS spectrum is on a different wavelength domain the units will be converted to the units of the LHS if they are different and then the RHS will be resampled onto the wavelength domain of the LHS before the operation is applied. There is a very import behavior to highlight here however. You are allowed to operate one spectrum on another that is defined on a different but overlapping band. For example, if *SpectrumA* is defined on 500-600 *nm* and *SpectrumB* is

[15] Includes +, -, *, /, +=, -=, *=, /=, <, >, <=, >=, ==, &, ^, |, <<, >>

defined on 550-600 *nm*, the operation `SpectrumC = SpectrumA + SpectrumB` would yield a new spectrum with the wavelength domain of *SpectrumA* where all pixels that fall into the domain of *SpectrumB* are operated on. The necessary condition is that the RHS spectrum is equivalent to or entirely contained (inclusively) by the LHS.

```
41 spectrumA = Spectrum(np.array([1, 2, 3, 4, 5]))
42 spectrumB = Spectrum(np.array([2, 3, 4, 5, 6]))
43
44 spectrumA + spectrumB
```

```
[  3.   5.   7.   9.  11.]
[ 1.  2.  3.  4.  5.] pix
```

```
41 from astropy import units as u
42
43 spectrumA.wave = spectrumB.wave.value * u.nm
44 spectrumB.wave = spectrumB.wave.value * u.nm
45
46 spectrumA * spectrumB[4:5]
```

```
[  1.   2.   3.  20.  30.]
[ 1.  2.  3.  4.  5.] nm
```

In the immediately above code snippet, the wavelength arrays are given units of *nm* because you cannot operate on two spectra that have pixel units for wavelength arrays unless they are the same length (the alternative makes no sense!). Only the last two elements were operated on. The `[4:5]` style notation is called *slicing* and is common in several other programming languages in addition to Python. Here, its been implemented in a non-standard way. Despite how it may appear, the command does not say to select elements 4-5, but instead says to retrieve the spectrum defined between those wavelength values (assuming the same units as the wavelength array). The spectra used in this study are defined on 5850-5950 Å. So `regulus[5885:5910]` would return a new spectrum as a segment of the Regulus spectrum, all pixels contained within those two wavelength values. Additionally, a third component to the slice may optionally be given to specify a desired resolution. `regulus[5885:5910:0.1]` says to take the same segment but resample onto a new resolution (the current being 0.01 Å pixel^{-1}). This is simply *syntactic sugar* and is

equivalent to the **.resample()** function [16] (e.g., `regulus.resample(5885 * u.AA, 5910 * u.AA, 250)`, where the third argument is now the number of pixels not the resolution). Spectra can also be operated on by scalars (left hand and right hand operations).

```
47 2 * spectrumA
```

```
[  2.   4.   6.   8.  10.]
[ 1.  2.  3.  4.  5.] nm
```

The comparison operators return a **Spectrum** object but with binary, dimensionless data. In the case where the LHS and RHS don't have identical wavelength domains, the above described behavior holds, with the exception that all pixels in the LHS spectrum that remain unaffected by the domain of the RHS are returned as *false* (i.e., "0"). This can be interesting and useful functionality to have. [17]

The shift operators (<<, >>) do what you might expect. They subtract from or add to (respectively) the **.wave** array (as oppose to the **.data** array for the other operators).

```
48 spectrumA >> 2 * u.nm
```

```
[ 1.  2.  3.  4.  5.]
[ 3.  4.  5.  6.  7.] nm
```

One could then, for example, manually apply a velocity correction as follows.

```
49 from astropy.constants import c
50
51 v = 15 * u.km / u.s
52 regulus << regulus.wave * ( v / c.to(u.km / u.s) )
```

```
[ 1.01038098  1.01141405  1.01213932 ...,  0.9898265   0.99048185
  0.99079037]
[ 5849.70729751  5849.71729701  5849.72729651 ...,  5949.68229505
  5949.69229455  5949.70229405] Angstrom
```

There are several other areas of functionality provided by the **SLiPy.Data** sub-package, the **Fits** module, and the **Spectrum** object; descriptions of these features are resigned to the online documentation.

[16] The **.resample()** method for a spectrum performs linear interpolation by default but can employ any other method of interpolation offered by the backend function, scipy.interpolate.interp1d(), by passing the *kind* keyword argument (e.g., `kind = "cubic"`).

[17] For example, `sum( (regulus > 1).data )` would tell you how many pixels have flux data greater than one.

With these tools, along with the functions and algorithms described in the following chapters, one can more easily process larger spectral data sets with consistency.

CHAPTER 3

CALIBRATIONS

3.1 Removal of Telluric Features

3.1.1 Description

Generally speaking, spectral data of the sort used in this study will have three categories of absorption line features. First, the star (whence the photons arise) is both a source and an absorber of photons. The stellar lines will likely be the largest (greatest equivalent width). Second, one or more interstellar sources of gas and dust along the site line between the star and Earth can and will add more features to the spectrum (often *blending* with stellar features). Last, as the photons propagate through our atmosphere, specific wavelengths will be absorbed from the atomic and molecular species present therein.

One of the first pressing issues to deal with given ground based observatories is the presence of numerous absorption features in the data that arise from the Earth's atmosphere, otherwise referred to as *telluric absorption*. A popular approach taken to address this issue is to remove these lines by dividing them out using a reference or *calibation* spectrum. The idea is that by taking the spectrum of a star, ideally before and after your science target, for which there are no stellar/interstellar absorption features in the domain of interest, one can preferentially select out the features that are telluric in origin. [1] That is to say, if something is present in both spectra, by dividing one spectrum

[1]The reference star of choice will be a spectral type "A" star, though sometimes a "B" star is possible.

by the other, the offending line can be removed. An import recognition here is that this only applies for spectra that have been *continuum normalized*.

Algorithmically, where there is no absorption line in the spectrum the continuum will ideally be very close if not exactly equal to 1. As such, dividing by unity proffers no change. The absorption features come down from the continuum. So in order for the division operation to work, the spectrum needs to be inverted – with the continuum at zero and signal rising above it. So we flip both spectra, perform the division, then again invert the spectrum.

Practically speaking, this is not always a clean procedure. If the two spectra are taken during significantly different times or along differing lines of site through the atmosphere (differing airmasses), the telluric features can and will have conflicting shape and saturations. The relative strength between two corresponding lines both within the same spectrum and between spectra can cause inappropriate artifacts in the resulting corrected spectrum. For this study, there were numerous stellar targets from within the ELODIE archive of early type stars from which to choose. The difficulty is in choosing the right calibration star such that there are also no interstellar features.

A complete treatment of the options and benefits of various procedures for dealing with telluric features could occupy a lengthy paper, if not several. Here, I wish to demonstrate the successful implementation of the chosen approach and comment on its effectiveness and its drawbacks. For our purposes, Regulus (α Leo, HD 87901), a B8IV star, was the best choice. While the time of observation and the position do not necessarily line up well with the targets of interest in this study, it has been demonstrated to high order that Regulus lacks an interstellar Na I D1, D2 component [2] (Peek et al., 2011), an arguably more critical requirement. Because of the differing airmass (extinction levels), time of

These stars make good reference stars because they have little to no *metal* absorption features. That is to say they have little or no features other than broad Hydrogen lines in our region of interest.

[2]The lower limit for an interstellar component is 0.1 mÅ.

observation, and exposure between the Regulus spectra and that of the target spectra, it was necessary to modulate any spectral division algorithm to prefer calibrations that better fit the data. In other words, because the telluric features present in a spectrum of Regulus and that of a particular target could potential differ significantly, we needed to incorporate an auto-correcting feature to the algorithm that would preferentially choose a spectrum of Regulus out of many that best fit the current spectrum and choose *that* calibration.

3.1.2 Implementation

The **Telluric.Correct()** function (see Appendix C.18) takes two or more **Spectrum** arguments. The first is the spectrum to be corrected. The second argument, and any further arguments are to be *candidate* reference spectra. In this particular function, it is necessary to insure that the target spectrum occupies the same domain (not necessarily the same resolution) as the calibration spectra. One of the hallmarks of a telluric line is that it will be at zero velocity, or at least in principle it will be very near zero. As such, no velocity shift needs to be applied to the spectra before division. However, often there will be a wavelength difference of a few pixels (the extent of which depends on the resolution of the spectra). For this reason, a horizontal cross-correlation is performed to determine the optimal shift in pixels left or right on the calibration spectrum using the **Correlate.XCorr()** function (see Appendix C.11). Further, because of differing exposure times, etc., we necessarily must amplify the calibration spectra to best fit one with the other. Our goal is only to trace the continuum and telluric lines. The *best* fit from both of these correlations is quantified by minimizing the RMS (root mean square) difference between the spectra. This procedure is applied to all the provided candidate calibration spectra and the best fit is chosen to use for the division.

Options for this function include the *lag* for the wavelength correlation. This argument by default is 25 pixels but can be overridden; it is passed to **Correlate.XCorr()**

and specifies how many pixels to shift the two spectra apart in searching for the best fit. Also, a *range* can be specified for amplitude correlation. This should take the form of a *tuple* of length three and is the start, end, and number of percentages (e.g., `range=(0.5, 2.0, 151)` is the default and specifies a search between 50% and 200% of the calibration spectrum on a 1% increment). It's important to note here that only on the part of the spectra that overlap is the operation applied; so if the best shift of the chosen calibration spectrum is three pixels, than that many pixels on either side of the target spectrum will not be altered.

```
53 from slipy import Telluric
54 from slipy.Framework import Display
55 display = Display.Monitor()
56
57 # display progress while applying telluric corrections to all targets
58 for a, spectrum in enumerate(spectra):
59     display.progress(a+1, len(spectra))
60     Telluric.Correct(spectrum, *regulus, range=(0.5, 2.0, 1501))
```

```
[======================>                      ]   49.43 %
```

In the above code snippet we are applying the telluric correction procedure to all the spectra using the Regulus spectra from the previous examples. In this case, the density of the amplitude trials has been increased by a factor of ten, resulting in about a one second computation time for each spectrum. Figures 3.1 and 3.2 showcase the results of the **Telluric.Correct()** algorithm for two stars. HD 192640, an A2V variable star at 20^h 14^m 32.0^s +36° 48′ 22.7″ approximately 11° southeast of δ Cygni, is shown to have a broad stellar component and a strong interstellar component for Na I D1 and D2 in Figure 3.1. In Figure 3.2 is Altair (HD 187642), another A type star, which has no interstellar features. Suffice it to say that Altair itself could potentially be a candidate for a reference spectrum for telluric corrections. When the algorithm is given this spectrum it effectively removes all features – behavior we would expect of a successful implementation of a telluric correcting algorithm. [3]

[3]Even better, if you give any one of the Regulus spectra as the first argument to the function you will get

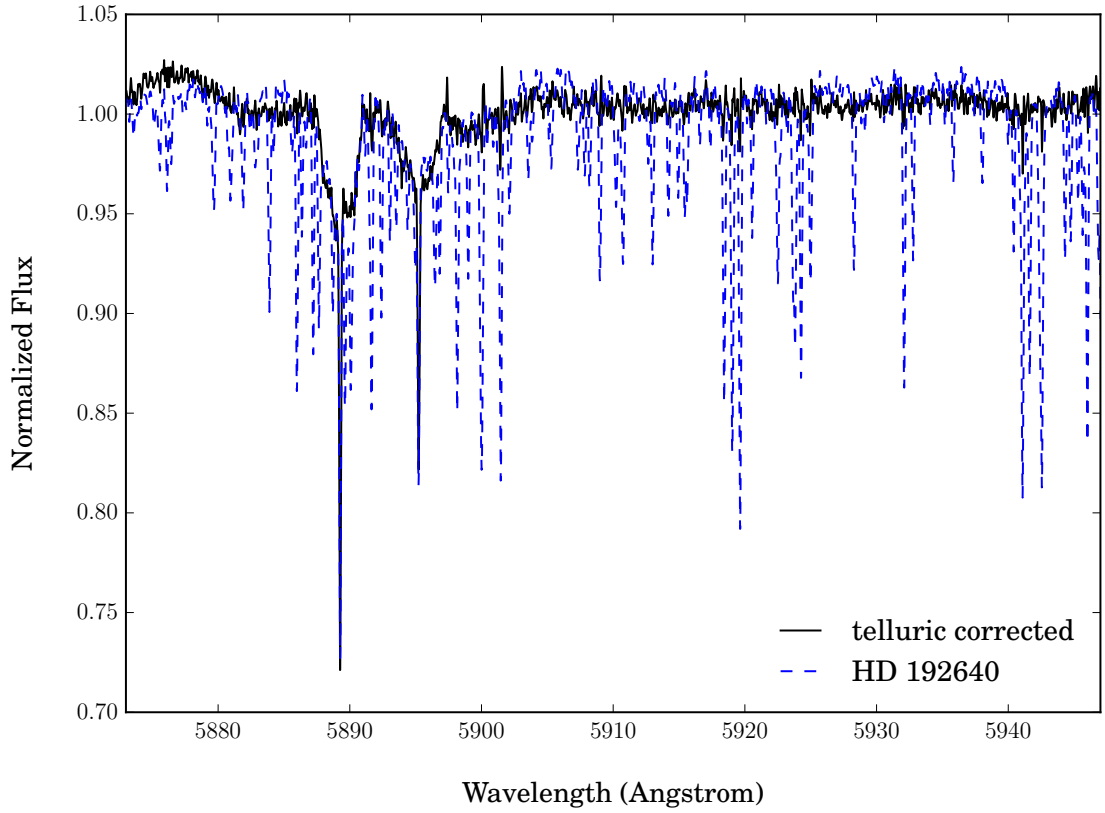

Figure 3.1: The above figure showcases the **Telluric.Correct()** routine. The spectrum is of HD 192460 (b03 Cyg), an A2V variable star. As in the entire analysis for this study, Regulus (α Leo), a B8IV star, was used as the calibration spectrum. The Elodie archive has six spectra; the best fit is automatically chosen by the algorithm.

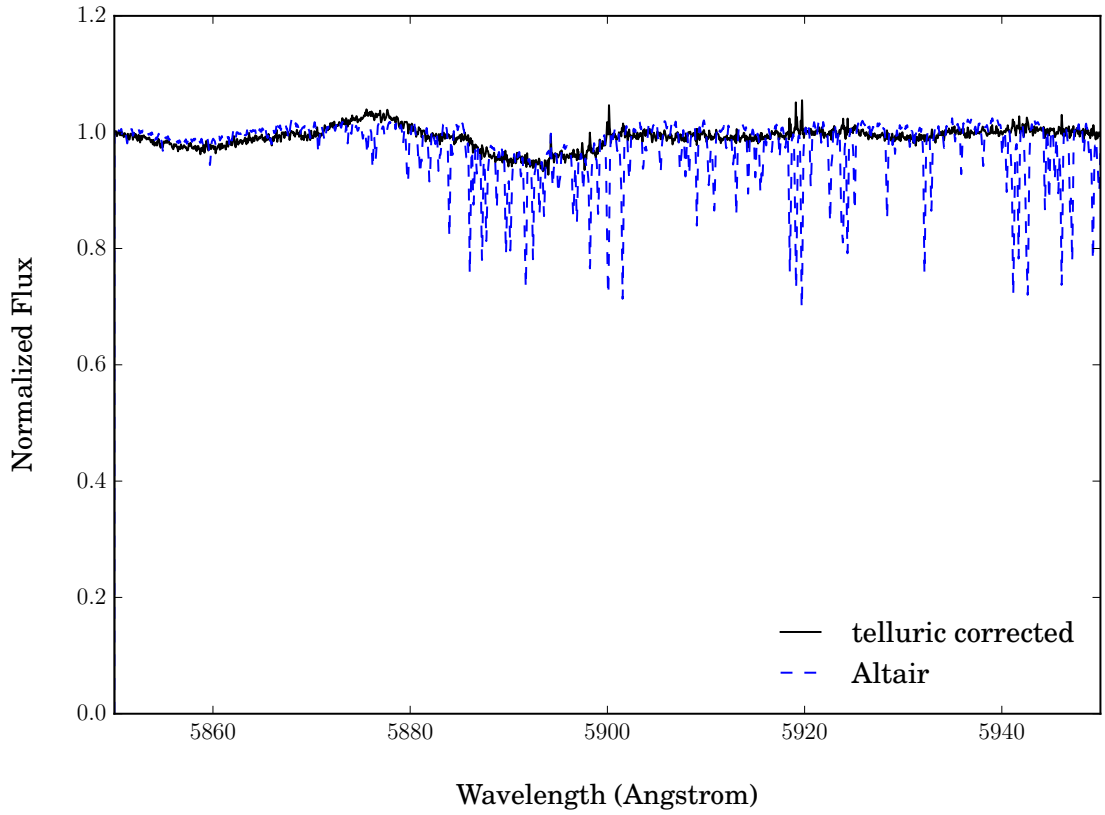

Figure 3.2: The above figure is an empirical test of the **Telluric.Correct()** routine. Here, a spectrum of Altair (HD 187642) was corrected against the same six spectra of Regulus as in Figure 3.1. As an A7V star, Altair has no absorption features in the above wavelength domain (with the exception of absorption due to Earth's atmosphere (telluric) and weak broad stellar absorption). By applying the routine to a star that might otherwise be a suitable candidate for a calibration spectrum itself, we have effectively removed every feature.

3.2 Heliocentric Velocity Corrections

3.2.1 Description

Another significant calibration to be performed is a velocity correction. The Earth is in continual motion, with both rotation and revolution. In order to un-bias the measurement of the velocity of an absorption line in spectra, we must subtract out the motion of the observer along the line of site to the target. The largest influence is from the motion of the Earth around the Sun and can cause a shift up to approximately 30 km s^{-1}. A successful algorithm for computing the velocity correction along the line of site to a target has been implemented since the 80s (Stumpff, 1980). The P. Stumpff algorithm is used in the *rvcorrect* task for IRAF. There already existed an implementation of this algorithm for Python [4] maintained by Dr. Sergey Koposov. [5] The accuracy of this is expected to be no worse than $\sim$ 1 m s^{-1}. [6]

The algorithm computes the velocity vector of the observer due to the Earth orbiting the Sun, the motion from the Earth - Moon system, as well as the rotational velocity of the Earth. This vector is usually either with respect to the Sun, *heliocentric*, or with respect to the gravitational barycenter of the solar system, *barycentric*. The correction is the projection of this vector along the line of sight to the target. The necessary information to computer this correction is the latitude, longitude, and altitude of the observatory; the right ascension and declination of the target; and the Julian Day [7] at the middle of the exposure. For this study, the heliocentric velocity is used.

---

back a spectrum identically one at all wavelengths!

[4] Specifically for Python 2, it was necessary to modify many of the files provided in the library to conform to Python 3.

[5] Institute of Astronomy, University of Cambridge

[6] At the very least, the performance of the system put together within SLiPy was tested against the output of the *rvcorrect* task within IRAF and found to be consistent for the same import parameters.

[7] The Julian Day is a non-relative measure of the exact time an event occurs, and is defined as the count of days from 12 noon on 1 January, 4713 BC. The fraction of the current day is simply expressed as the decimals following the count.

3.2.2 Implementation

The **Velocity.HelioCorrect()** and **Velocity.BaryCorrect()** functions provide this capability. These functions provide a high level interface to the **helcorr()** function maintained by Dr. Koposov. **helcorr()** relies on a number of other modules all housed inside a library, **astrolibpy**. A modified version of this entire library is distributed with **SLiPy** (as a sub-package). The **Velocity.HelioCorrect()** and **Velocity.BaryCorrect()** functions access **helcorr()** in a more abstracted way. All the necessary arguments are condensed into two, an *observatory* and a *spectrum*. The *observatory* is an object that has the *latitude*, *longitude*, and *altitude* information. The *spectrum* argument should be of type **Spectrum** and have the right ascension, declination, and Julian Day of observation attached. [8]

In a similar design strategy as in IRAF, an **Observatory** is a thing that has attributes such as **.latitude**, **.longitude**, **.altitude**, etc. By default, there are 70 observatories defined. The 69 observatories (and attributes thereof) defined within IRAF's observatory database (usually under noao$lib/obsdb.dat) have been adapted to an object oriented pattern within SLiPy. The additional observatory is for the present study. It and any additional observatories can be defined as follows:

```
61 class OHP(Observatory):
62     """
63     The Observatoire de Haute-Provence, France.
64     """
65     def __init__(self):
66         self.name       = 'Observatoire de Haute-Provence'
67         self.longitude  = 356.286670 * u.degree # West
68         self.latitude   = 43.9308334 * u.degree # North
69         self.altitude   = 650 * u.meter
70         self.timezone   = 1 * u.hourangle
71         self.resolution = 42000 * u.dimensionless_unscaled
```

In the above snippet "Observatory" is an abstract base class that can/should not be

[8] In Python, everything is a first class object and is dynamic. I can attach a new data member to anything by simply assigning it (e.g., `spectrum.ra = 20`).

instantiated alone. The velocity correction functions check to see if the observatory the user provided is derived from this. Units are demanded for all the arguments' attributes, such that they can be properly converted if necessary to that needed for the P. Stumpff algorithm. Below is an example usage of the **Velocity.HelioCorrect()** function. As in the **Telluric.Correct()** function, the **Spectrum** argument(s) provided are altered; nothing is returned. As mentioned previously, the metadata for each observation needs to be attached to the spectra.

```
from slipy import Velocity, Observatory

# attach metadata to spectra - 'files' is a list of file paths
for spectrum, filepath in zip(spectra, files):
    spectrum.ra      = Fits.Header(filepath, "POS1") * u.hourangle
    spectrum.dec     = Fits.Header(filepath, "POS2") * u.degree
    spectrum.exptime = Fits.Header(filepath, "EXPTIME") * u.second
    spectrum.mjd     = Fits.Header(filepath, "MJD-OBS") * u.day
    spectrum.jd      = (spectrum.mjd + 2400000.5 * u.day +
                        0.5 * spectrum.exptime)

# create observatory object
OHP = Observatory.OHP()

# calculate and apply all velocity corrections
Velocity.HelioCorrect(OHP, *spectra)
```

In the above snippet, the Modified Julian Day is taken from the header files because it is provided for the ELODIE files. The Julian Day at the middle of the exposure is computed using this. All data used in this study was calibrated in a similar manner.

CHAPTER 4

MEASUREMENTS

The primary purpose of this study was to identify either the definitive presents or absence of the Sodium D lines; and in their presents, measure such quantities as equivalent width, column density, velocity, and the broadening parameter. This chapter covers the several distinct components involved in doing so, including the modeling of the continuum over the line, fitting and/or deblending multiple line components, and calculating these quantities from the extracted line profile.

4.1 Non-parametric Gaussian Kernel Regression

In spectra such as these, the absorption line features come down from a background continuum level and we must model this curve over the span of the line in order to integrate over that line. The continuum will have noise however and is almost never smooth. Generally, we solve for a smooth curve through the noise and interpolate over the line gap. There are of course numerous different methods and techniques for doing both of these. Here, we have chosen to use kernel regression to both simultaneously smooth out the continuum and gain information about the error in our measurements from the noise in the data. Gaussian kernel smoothing is a popular approach and is generally applicable with a balance between performance and complexity.

SLiPy contains a sub-package, **Algorithms**, which currently houses two modules: **Functions** and **KernelFit**. It was convenient to abstract away the interface to a number of

profile functions. The Gaussian, Lorentzian, and Voigt functions are all defined (along with variants and normalized versions thereof) within the **Functions** module. Within the **KernelFit** module is the **KernelFit1D** object. Most of the functionality discussed in the rest of this chapter is defined in the **Profile** module, whose functions use the kernel fitting algorithm.

Here, we use the Nadaraya-Watson kernel regression method. Given a discrete data set of (x, y) elements with sufficient noise, we can estimate its real valued function as

$$f(x) = \frac{\sum_i k(x - x_i)\, y_i}{\sum_i k(x - x_i)} \quad \text{where} \quad k(x) = A \exp\left(-\frac{x^2}{2\sigma^2}\right) . \tag{4.1.1}$$

This is essentially an arithmetic mean of the data set but with weights determined by the proximity of that data point from the point in question. The Gaussian kernel, k, has a length scale (or *bandwidth*), σ. This is an important thing to understand because it must be specified by the user. When trying to model the continuum and extract the line profile from the data, the length scale must be sufficiently large as to not trace the noise yet not so large as to fail to follow the bulk shape of the curve. This is generally true of any application using kernel smoothing. For most of the ELODIE data used in this study a bandwidth of approximately 0.08 Å was optimal. An example usage of **KernelFit1D** is available in Figure 4.1

4.2 Spectral Plots

Before continuing on with this discussion, the **SPlot** (\ϵs plät\) object needs mentioning. The **Profile.Select()**, **Profile.Extract()**, **Profile.AutoFit()**, and **Profile.MultiFit()** functions each offer a graphical interface for the user to select data as a sample of the continuum and the line contained within. The arguments to these functions

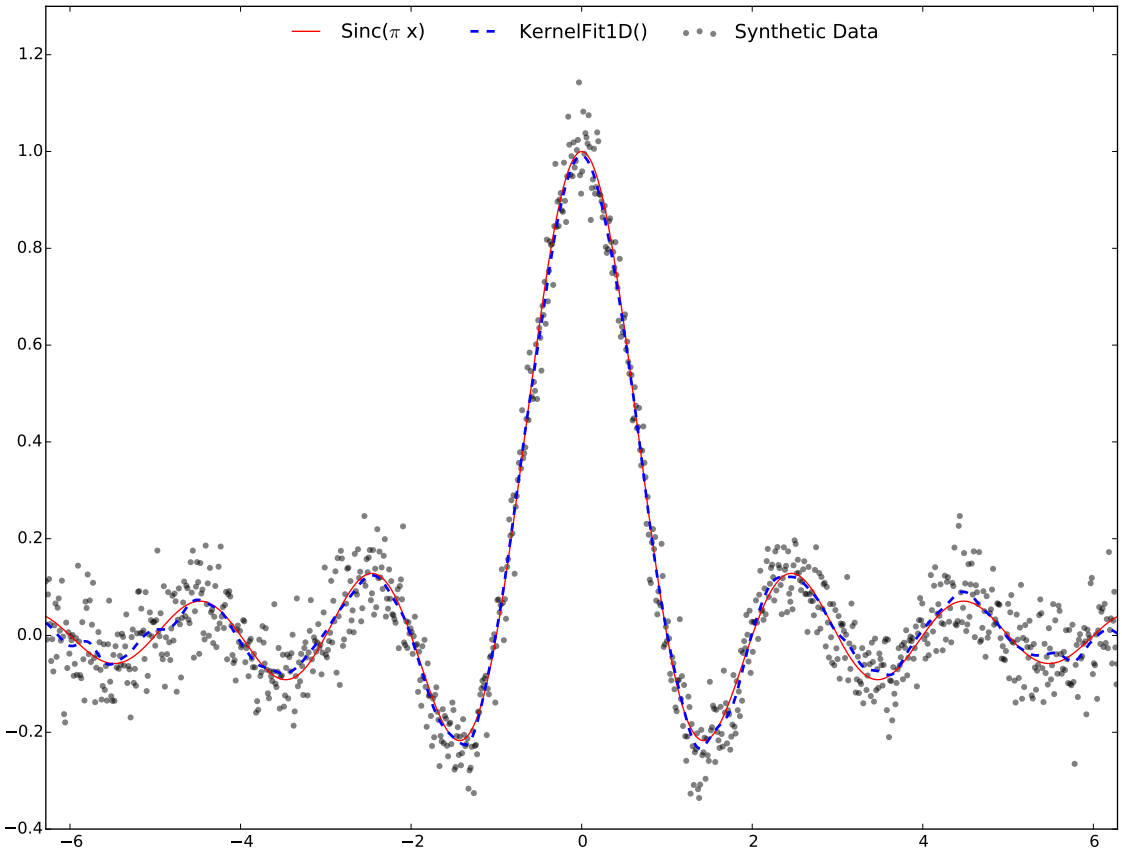

Figure 4.1: The above figure showcases **KernelFit1D** in a best-usage scenario. Here, we have a synthetic data set constructed of a normalized sinc function with added both normal *white* and *red* noise. The analytical curve is shown with a thin solid red line. The bold blue dashed line is the solution using the kernel smoothing algorithm with a *bandwidth* of $\pi/32$.

are all of type **SPlot**. [1]

An **SPlot**, or Spectral-Plot, is a manager class for creating and maintaining figures using **matplotlib** (Hunter, 2007). Generally, using the **matplotlib.pyplot** module allows one to plot points and curves of x-y data. The x and y limits can be set, axis labels and legends (or other text) can be set. What an **SPlot** does is to retain all the calls to the most common methods applied using the **pyplot** module and allow you to pass around figures as objects. The full documentation for this feature will not be provided here (it can be accessed online), but an example should suffice. The results of the below snippet are in Figure 4.2.

```
88  from slipy import Plot
89
90  figure = Plot.SPlot(regulus[0], label="Regulus", marker="k-")
91  figure.legend(loc="lower left")
92  figure.draw()
93  figure.save("regulus.pdf")
94
95  # create many plots and access them individually
96  figure = [ Plot.SPlot(spectrum, label=Fits.Header(filepath, "OBJECT"))
97              for spectrum, filepath in zip(spectra, files) ]
98
99  # draw the first spectrum
100 figure[0].draw()
101
102 # iterate through all the plots and choose ones of interest
103 keepers = Plot.Iterate(*figure, keep="plot")
```

4.3 An Interactive Graphical User Interface

The principle function of the **Profile** module is the **MultiFit()** routine which creates, maintains, then destroys a **FittingGUI** and then takes the measurements for all the lines extracted. The following subsections outline some of the major areas involved with this. The **FittingGUI** uses the API provided by **matplotlib** to add widgets to the **SPlot** figures. The goal here is to be able to interactively fit profiles to data with one or more components.

[1] Although, they may alternatively be of type **Spectrum** for which a generic **SPlot** will be created.

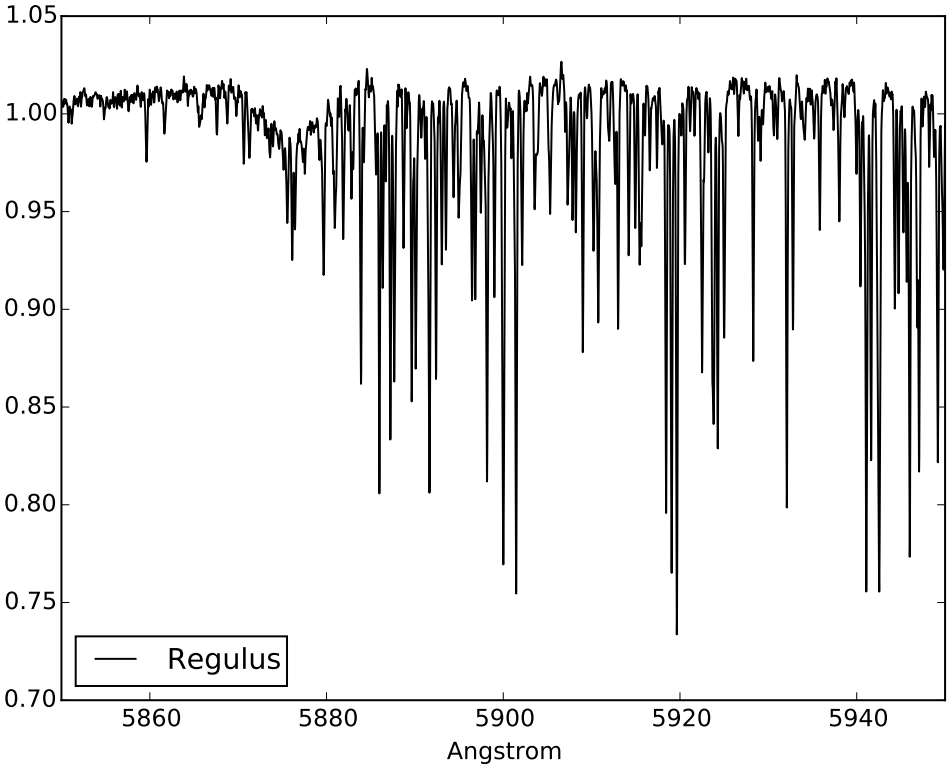

Figure 4.2: The output file from the example usage of the **SPlot** object.

4.3.1 The Voigt Profile

Generally, the **FittingGUI** has been programmed to have the capacity to model Gaussian, Lorentzian, and Voigt line shapes. For absorption lines (barring some more complex scenarios such as rapid rotators, etc.) we expect to see Voigt profiles. This is the result of an intrinsic line profile well approximated by a Lorentzian [2] that is broadened with the convolution of one or more Gaussian profiles. The first Gaussian is physical and due to both thermal (Doppler broadening) and turbulence. The second Gaussian is due to the blurring from instrumental effects. [3] Together, the Voigt profile can be expressed as (Draine, 2011)

$$\phi_{\nu}^{\text{Voigt}} \equiv \frac{1}{\sqrt{2\pi}} \int \frac{dv}{\sigma_v} e^{-v^2/2\sigma_v^2} \frac{4\gamma_{ul}}{16\pi^2 \left[\nu - (1 - v/c)\nu_{ul}\right]^2 + \gamma_{ul}^2} \tag{4.3.1}$$

where the integral is over velocity and $\nu_{ul} \equiv (E_u - E_l)/h$ crosses the transition between energy levels u and l. The width of the Lorentzian reflects the fundamental uncertainty in the finite lifetimes of the energy levels against transitions to all other levels. The intrinsic width of the absorption line is related to γ_{ul} and the Einstein coefficient A_{ul} as

$$(\Delta v)_{\text{FWHM}}^{\text{intr.}} = \frac{\gamma_{ul}}{2\pi} = \frac{\lambda A_{ul}}{2\pi} \, . \tag{4.3.2}$$

The strength of the absorption transition can also be described by the oscillator strength,

$$f_{ul} \equiv \frac{m_e c}{\pi e^2} \int \sigma_{lu}(\nu) d\nu \tag{4.3.3}$$

where the monochromatic absorption cross section $\sigma_{lu}(\nu)$ to a normalized line profile ϕ_ν is

[2] The line profile is better approximated by the Kramers-Heisenberg formula (Draine, 2011).

[3] Though it is not necessarily the case that an instrumental line-spread function (LSF) actually be a Gaussian, this is often sufficient.

$$\sigma_{lu}(\nu) \;=\; \frac{g_u}{g_l}\frac{c^2}{8\pi\nu_{lu}^2}A_{ul}\phi_\nu \;. \tag{4.3.4}$$

These quantities, A_{ul} and f_{ul}, are import to relating our Voigt line profile back to physical properties.

These are analytical expressions; the **Profile** module within SLiPy relies on the relationship between the Voigt function and the Faddeeva function (a complex complementary error function). The Voigt profile can be expressed as the real part of this function, $w(z)$, as

$$V(x;\sigma,\gamma) \;=\; \frac{\mathrm{Re}[w(z)]}{\pi\left(\sigma\sqrt{2\pi}\right)} \;\text{ for }\; z \;=\; \frac{x+i\gamma}{\sigma\sqrt{2}} \;. \tag{4.3.5}$$

Within the SciPy library there is a **scipy.special** module with a numerical implementation provided for **wofz()**. [4] The profiles fit in this study come from this approach. In reality, it is only the absorbers whose profile is Voigt in shape. The actual feature in spectra is an exponential relationship, $I_\nu = I_{0,\nu}\exp(-\tau_\nu)$. For only small absorption features, this is approximately linear. Using something called the curve of growth (Draine, 2011) one can detangle the relationship between the equivalent width and the number of absorbers in a medium. Alternatively, we will employ the so called apparent optical depth method (Savage and Sembach, 1991) for non-optically thin lines; more on this in the following sections.

4.3.2 Atomic Ions

The Na I (D1 and D2) lines have oscillator strengths 0.320 and 0.641, respectively (Morton, 2003) and transition probabilities (Einstein coefficients) 6.14×10^7 s^{-1} and

[4] This is essentially `exp(-z**2)*erfc(-i*z)`. Please see the Steven G. Johnson, Faddeeva W function implementation, http://ab-initio.mit.edu/Faddeeva, for more details.

6.16×10^7 s^{-1}, respectively (Kramida et al., 2014). This relates to the **FittingGUI** in that the user provides the information on both of these quantities as a single *ion* argument. So in fitting a particular absorption line in your data, you only need to specify what *ion* you are fitting and it knows the *wavelength*, *fvalue*, and *A* (in both air and vacuum) for most ions with wavelengths long-ward of the Lyman limit.

Within the **SLiPy.Data** sub-package is an **Atomic** module that not only incorporates the entire data set from Morton (2003) but defines the **Atomic.Ion** object. For the sake of brevity, this functionality will not be documented entirely here, suffice it to say however that it's a very high level aspect of the library to not require the user to look up these values as the most recent published data is already built in to the software. [5] A short example is presented here, and again in the later example using the **Profile.MultiFit()** function.

```
104 from slipy.Data import Atomic
105
106 # find oscillator strengths for C III ions
107 Atomic.IonSearch("C III")
```

```
[(<Quantity 977.0201 Angstrom>, 0.7570010842747638),
 (<Quantity 1908.734 Angstrom>, 1.6875419997146983e-07)]
```

```
108 # find ions within a certain region
109 Atomic.IonSearch((585*u.nm, 590*u.nm), lookup="ion")
```

```
[(<Quantity 5891.5833 Angstrom>, 'Na I'),
 (<Quantity 5895.725 Angstrom>, 'Ti I'),
 (<Quantity 5897.5581 Angstrom>, 'Na I')]
```

```
110 # declare the Na I 'D2' ion
111 D2 = Atomic.Ion("Na I", 589*u.nm, A=6.16e-7*u.second, medium="air")
```

```
Ion:        Na I
Wavelength: 588.9951 nm
fvalue:     0.6408671009122726
A:          6.16e-07 s
```

[5] While the oscillator strengths for these ions is part of the software, as of today the transition probabilities have not been built in and must be specified by the user.

4.3.3 Fitting Absorption Line Parameters

With an *ion* declared we can pass it along with an *splot*, an *observatory*, [6] and a *bandwidth* to **Profile.MultiFit()**. The first thing that happens is you are asked to select four data points along the spectrum. This is to mark the edges of the line in addition to samples of the continuum on either side. **KernelFit1D** models the continuum and the **Profile.Extract()** routine is run. With a separated line and continuum (interpolated [7] over the absorption line gap) you are then asked to select more data points along the line to mark the suspected peaks of one or more components.

The plot moves up in the figure and *sliders* appear for a number of parameters along with radio buttons for each line selected. Above the sliders and below the plot is a preview of the absorption line parameters for each Voigt profile that is now plotted for each component. These steps are illustrated in Figures 4.3 - 4.6. What sliders appear depends on the *function* requested ("voigt" by default) and whether or not you provide an *observatory* and an *ion*. In this context, by providing both of these, *gamma* does not appear as an alterable parameter. This is because the transition probability provided by the *ion* solely determines the expected FWHM of the intrinsic profile and the only parameter to be fit is an observed *sigma*.

The broadening parameter can be extracted from the observed *sigma* as the convolution of one Gaussian on another is simply another Gaussian with

$$\sigma^2_{\text{convolution}} = \sigma_a^2 + \sigma_b^2 \,. \tag{4.3.6}$$

So the width of the thermal/turbulance Gaussian is

[6] This should be either one of the defined **Observatory** objects or something derived thereof. We pass this to **Profile.MultiFit()** with an attached **.R** (resolution) so we know the FWHM of the instrumental Gaussian profile.

[7] The default is to use natural cubic spline interpolation, though alternatives can be requested by passing the *kind* keyword argument (see **scipy.interpolate.interp1d** for details).

$$\sigma_{\text{therm}} = \sqrt{\sigma^2_{\text{observed}} - \sigma^2_{\text{instrument}}} \tag{4.3.7}$$

where the resolution of the instrument gives the width,

$$(\Delta\lambda)_{\text{FWHM}} = \frac{\lambda}{R} \tag{4.3.8}$$

$$= 2\sqrt{2\ln 2}\ \sigma_{\text{instrument}} \tag{4.3.9}$$

and the broadening parameter is,

$$b \equiv \sqrt{2}\,\sigma_{\text{therm}}\,. \tag{4.3.10}$$

While the calculations are performed perpetually during the fitting process, a final calculation of the equivalent width and column density are performed after the fitting process is complete, attached to each returned *line* as **.W** and **.N** respectively of **Measurement** types. [8] Also, the **.parameters** for the Voigt profile and the velocity, **.v**, and the broadening parameter, **.b**, are attached to each line.

The details behind the calculation of the equivalent width and column density is reserved for the following subsections. An example usage of the **Profile.MultiFit()** function is presented here. and Figures 4.3 - 4.6 showcase individual steps.

```
113 from slipy import Profile
114
115 # declare the Na I ions
116 D1 = Atomic.Ion("Na I", 589.5*u.nm, A=6.14e-7*u.second, medium="air")
117 D2 = Atomic.Ion("Na I", 588.9*u.nm, A=6.16e-7*u.second, medium="air")
118
119 # SPlot of HD 200120
120 figure = Plot.SPlot(spectra[31], label="HD 192639", marker="k-")
121 figure.legend(frameon=False, loc="lower right")
122
```

[8] The **Measurement** class is unique to SLiPy and is derived from a **Quantity** but has a **.name**, **.uncertainty**, and **.notes**. It can be operated on the same as a **Quantity** (in fact it decays to one afterwards). This functionality is in the works by contributors at Astropy by is not currently available there.

```
# call MultiFit()
line, continuum = Profile.MultiFit(figure, obs=Observatory.OHP(),
    ion=D2, bandwidth=u.Angstrom/12, boost=20)
```

In the above snippet, the spectrum could have been passed directly to the function and the **SPlot** would have been created automatically. The idea here is that any **SPlot** can be customized and passed to the function and it will retain its attributes (e.g., we could have set the *marker* to "k+" and crosses would have been used for the data points and not an interpolated curve or we could have used **.overlay()** to add more than one **Spectrum** to the same figure and fit them simultaneously!). The extra keyword argument, *boost*, is an option that artificially inflates the number of pixels in order to increase the accuracy of the numerical integration. [9] This is not performed during the fitting process (the preview does not boost the resolution) which is why the attached calculations might be slightly different than the previewed results.

4.3.4 Equivalent Widths

The equivalent width, W, of an absorption line is a good measure of the strength of a line, or power removed by the absorbers (Draine, 2011), and is measurable even in the face of lower resolution data or high thermal broadening. It can be expressed in a few different ways but generally is the width a line would have if it maintained the same integrated area but a height equal to that of the continuum level. It is most accurately calculated by numerically integrating the Voigt profile.

$$W = \int \left[1 - \frac{I(\lambda)}{I_0(\lambda)}\right] d\lambda \tag{4.3.11}$$

$$= \int \left[1 - e^{-\tau(\lambda)}\right] d\lambda \tag{4.3.12}$$

[9] The numerical integration uses the composite Simpson's method (see **scipy.integrate.simps**). This is very accurate even at moderate resolution, but more elements gives more accurate results.

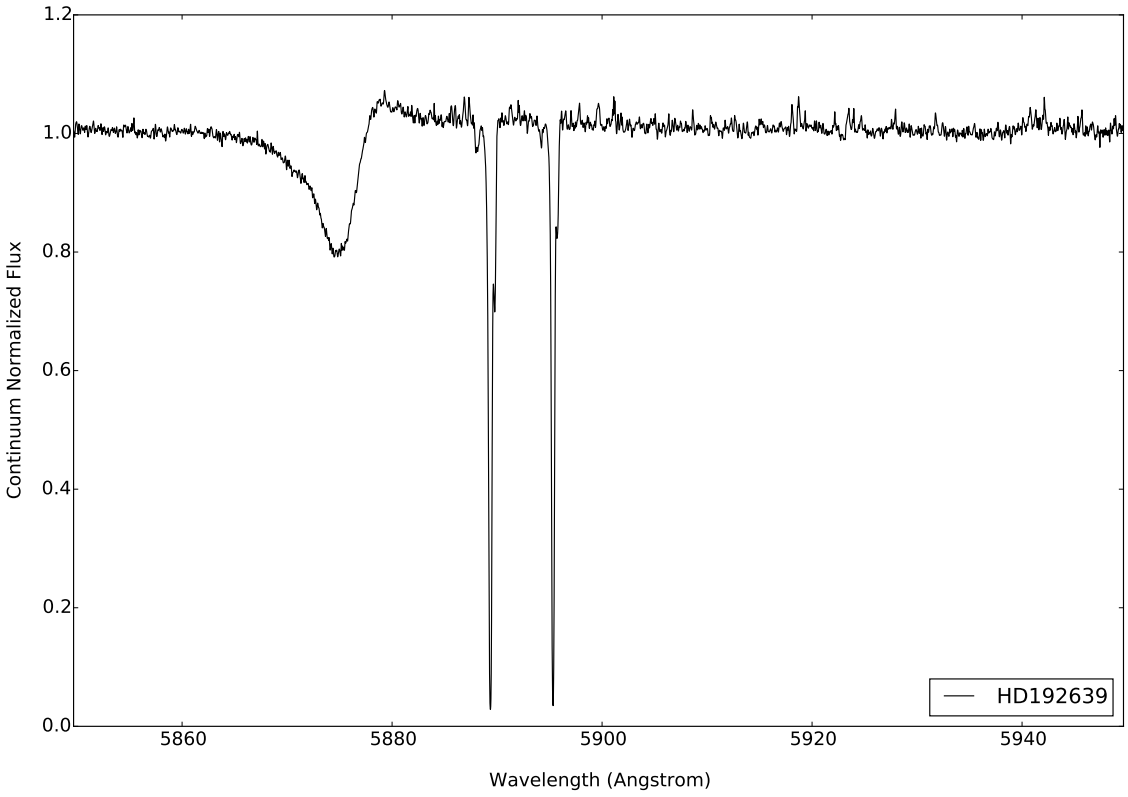

Figure 4.3: The blue supergiant HD 192639 is represented by an **SPlot** using SLiPy. Here we can see more than one interstellar Na I component. The **SPlot** itself can be passed as an argument to **Profile.MultiFit**().

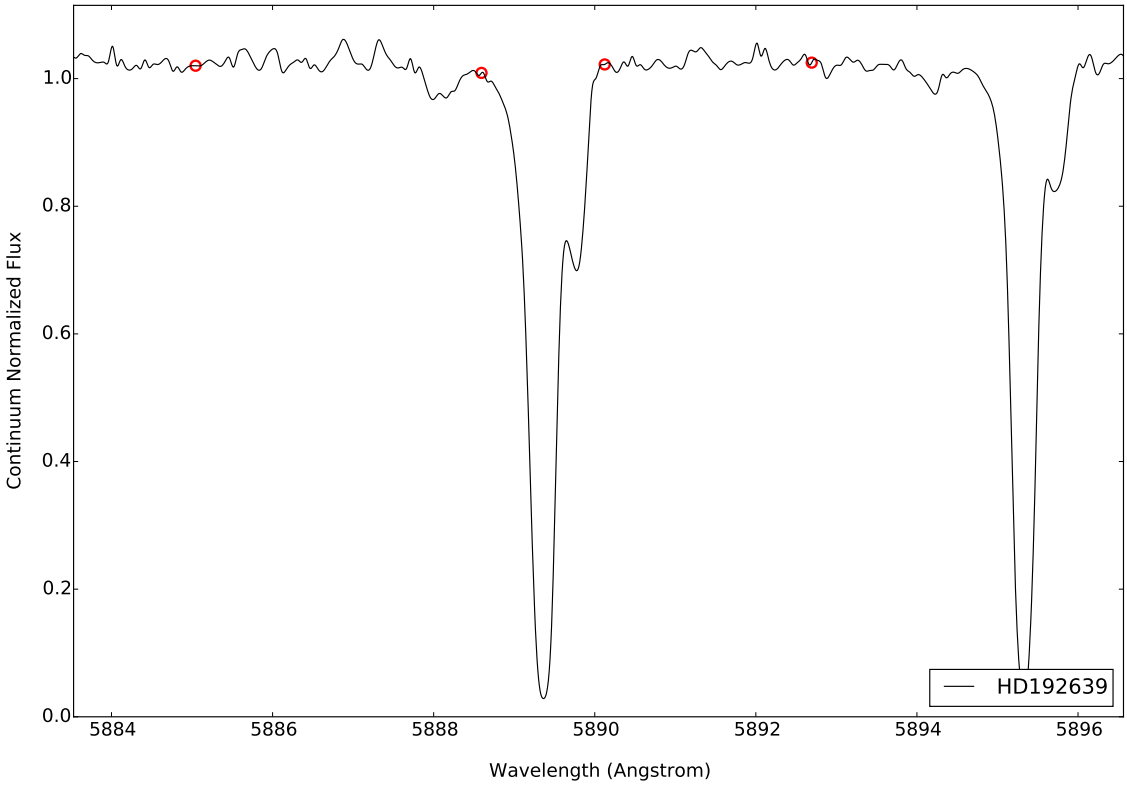

Figure 4.4: After passing the **SPlot** object from Figure 4.3 to **Profile.MultiFit()**, along with several necessary additional arguments, we are prompted to select four points along the spectrum to mark the boundaries of the line(s) and a sample of continuum on either side. The **KernelFitID** routine is used to smooth out the continuum and evaluate an RMS noise error.

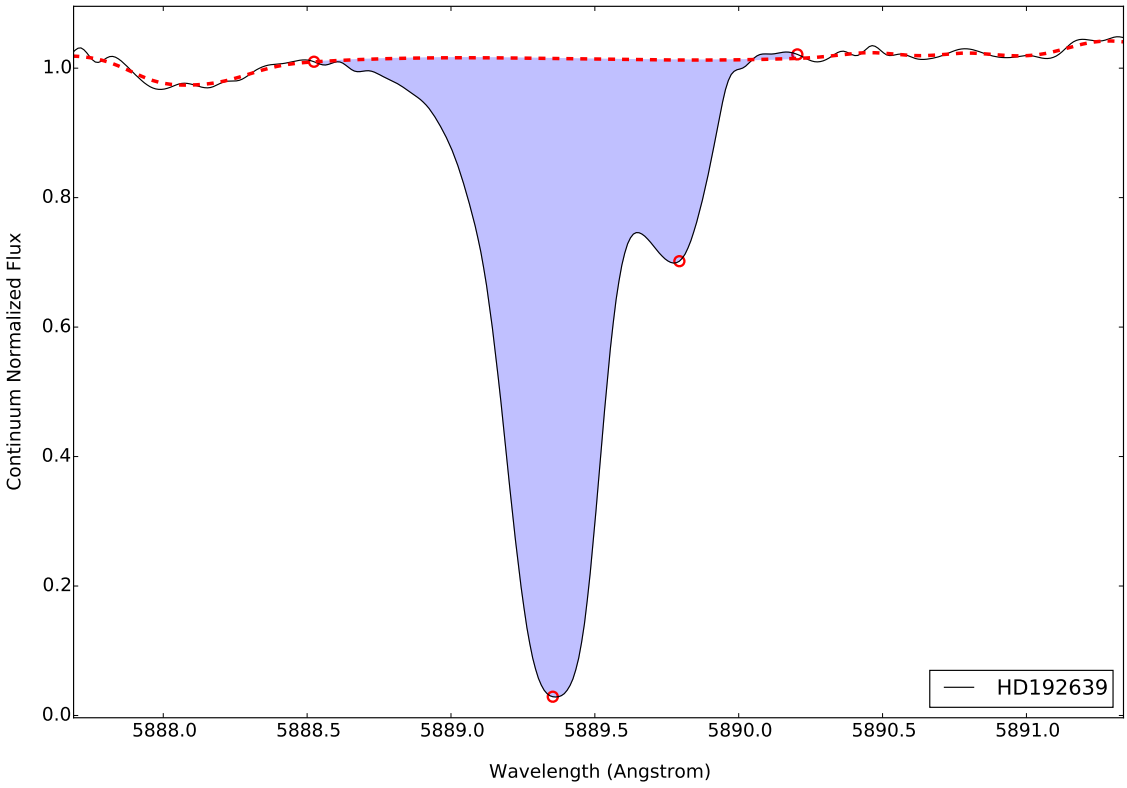

Figure 4.5: After extracting the line(s) from the continuum in Figure 4.4 we select the peaks of the absorption line(s). Buttons not shown in the figure allow the user to pan and zoom on the spectrum. It is unavoidable that the continuum interpolated over the line will be below the wings. This is because the noise in the continuum is assumed to be equally distributed above and below the continuum. By smoothing out the continuum we draw a curve *through* the noise and the interpolated curve between the samples will be continuous, in the mathematical sense. The idea is that we can get the non-linear aspect of the continuum over the line even if it is too low at first. In the next step we can raise the overall continuum level using a *slider*.

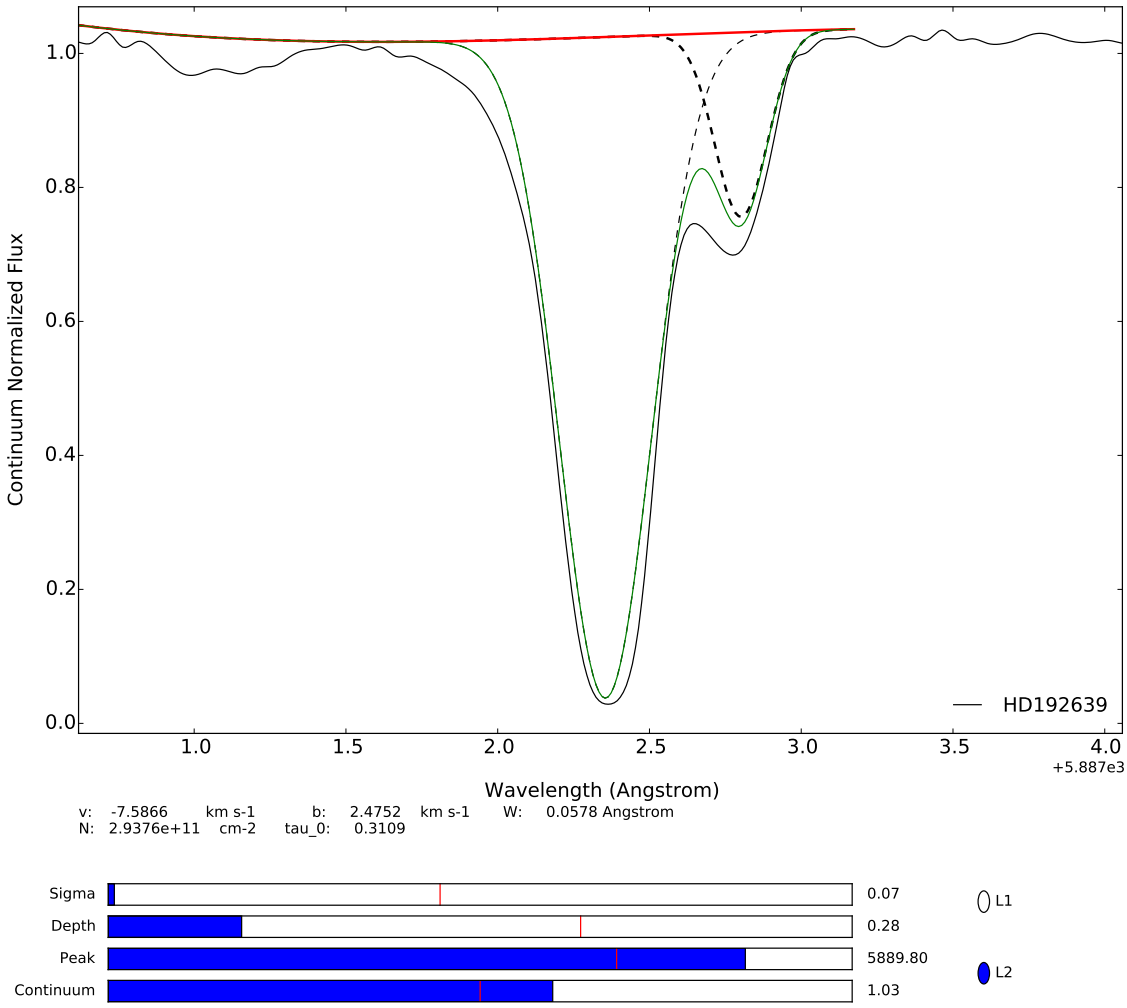

Figure 4.6: The last phase of the **Profile.MultiFit()** routine creates an analytical profile (as a Voigt function by default) for each selection made. Above is a mid-session view during the fitting process. The currently selected line is represented by a bold dashed line. To select a different line, choose the corresponding radio button in the bottom right. The continuum fit in the previous step is represented by a solid red line and the blended (superposition) line is solid green. The sliders created (bottom left) depend on the *function* argument. When an *ion* and *observatory* are given with appropriate member attributes, a Voigt profile is used and it is assumed that we are in fact trying to fit absorption lines in astronomical spectra (generally, this routine can be used outside of this context). In this context the velocity, v, the broadening parameter, b, and the apparent column density, N are previewed between the sliders and the active plot. The sliders allow for a range of possible values that are reasonable for the lines marked.

where I_0 is the continuum and $\tau(\lambda)$ is the optical depth of the line,

$$\tau(\lambda) \equiv \ln\left[I_0(\lambda)/I(\lambda)\right] . \tag{4.3.13}$$

Within SLiPy, both the **Profile.EquivalentWidth()** and **Profile.ColumnDensity()** functions are extensions of the **Profile.OpticalDepth()** function. For a single, well behaved absorption line the **Profile.Extract()** function can be used alone without any fitting procedure and a *line* and *continuum* can be separated by selecting these regions from the *splot* given.

```
126 # extract line from continuum ("fig" is an SPlot)
127 line, continuum = Profile.Extract(fig, bandwidth=u.Angstrom/12)
128
129 # "spectrum" is the spectrum used in "fig" with "error" being
130 # the percent errors from counting statistics.
131 W = Profile.EquivalentWidth(line, continuum, error=spectrum.error)
```

```
Name  = Equivalent Width
Value = 0.024997410090099946 Angstrom
Error = [ 0.01070082 -0.01070067] Angstrom
Notes = Measured using Profile.EquivalentWidth() from SLiPy
```

The above snippet is of HD 186155 and was measured to have an equivalent width for the D2 line of Na I of (25.0 $\pm$ 10.7) mÅ. [10] Doing the same set of commands but instead selecting D1 gives us (17.7 $\pm$ 9.6) mÅ and the ratio $W_2 \,/\, W_1 \simeq 1.41$. This puts it on the flat portion of the curve of growth and the stronger line has become more optically thick. As such, using the equivalent width to infer the column density no longer is accurate.

4.3.5 Column Densities and the Apparent Optical Depth Method

The column density is essentially a measure of the total number of absorbers along a column and is usually expressed as the number of atoms cm^{-2}. With complementary observations, it can be used to understand the length of a cloud. In this case, given the

[10] This value of course depends on what is provided via a selection by the user!

distance to a star for which there is interstellar absorption from the LCCC, the cloud's leading edge is that much closer. [11] In the limit of optically thin gas, $\tau_0 \ll 1$ (where τ_0 is the optical depth at line center), the column density can be related to the equivalent width as (Draine, 2011)

$$N = \frac{m_e c^2}{\pi e^2} \frac{W}{f_{lu} \lambda_{lu}^2} \tag{4.3.14}$$

$$= 1.130 \times 10^{12} \text{ cm}^{-1} \frac{W}{f_{lu} \lambda_{lu}^2} \, . \tag{4.3.15}$$

There are several techniques for computing the column density of a line that might offer better results dependent on the circumstances. Here, I will review the essence of the so called "apparent optical depth method" as described by Savage and Sembach (1991). More directly, the *apparent* column density relates to the *apparent* optical depth as

$$N_a(\lambda) = \frac{m_e c^2}{\pi e^2 f_{lu} \lambda_{lu}^2} \, \tau_a(\lambda) \, . \tag{4.3.16}$$

The usage of the word *apparent* here embodies the understanding that with an instrumentally smeared line we are only measuring the *observed* absorption. In contrast with the equivalent width, the measure of column density is indeed sensitive to instrumental broadening.

In either case, there is not much that can be done for severely saturated lines. This was not the case for the majority of the lines of interest for Non-OB stars investigated in this study; however there were indeed absorption lines for which saturation was near complete. Saturation is when the optical depth of a line begins to approach one at line center, note that this often occurs well before an observed line shows a significant depth due to instrumental broadening. When photons propagate through the medium, as more

[11] Not to insinuate that the LCCC necessarily has a hard edge.

are removed and scattered by the absorbers we cannot remove more photons that were there to begin with. So within the ISM, if the size or density of a cloud is sufficient, the absorption can be near complete before we even get all the way through. This scenerio spells doom to inferring information about the extent of the cloud. As the stronger line (in the context of doublets again) becomes optically thick and as saturation sets in we lose the ability to make accurate measurements. In principle, for doublet lines who differ in the product $f\lambda$ by a factor of two, we expect that the stronger line will have double the equivalent width and the same column density.

So long as the saturation is only modest, we can gain significant empirical information about the extent of saturation by measuring the difference in the observed line strengths (Savage and Sembach, 1991). Integrating the column density per unit wavelength gives the combined apparent column density,

$$N_a = \frac{m_e c^2}{\pi e^2 f_{lu} \lambda_{lu}^2} \int \tau_a(\lambda)\, d\lambda \,. \tag{4.3.17}$$

The *true* column density can be estimated as

$$\log N_a^{\,c} = \log N_a^{\,n-1} + \Delta \log N_a^{\,n-1} \tag{4.3.18}$$

where $\Delta \log N_a^{\,n-1}$ is a correction factor determined from the relationship between $\log N_{\rm true} - \log N_a^{\,n-1}$ and the difference $\log N_a^{\,n-1} - \log N_a^{\,n}$. [12] Given a value for this difference, we can estimate the correction based on published values (Savage and Sembach, 1991).

The **Profile.MultiFit()** function currently approximates the apparent column density during the fitting procedure using the optically thin approximation and the relationship between the column density and the equivalent width. Afterward, the more

[12] The n and $n-1$ refer to the strong and weak line respectively for the doublet.

accurate direct integration is performed using the **Profile.ColumnDensity()** function. The arguments consist of a *line* and a *continuum*, the same as for the **Profile.OpticalDepth()** and **Profile.EquivalentWidth()** functions. Here however we want to also provide a third, *ion*, argument (as described previously). Further, if we have already computed the apparent column density for the weaker of two lines of the same species, we can provide that measurement with the keyword argument *weakline* and the returned measurement will have the **.correction** and **.true** values attached. This uses the published corrections from Savage and Sembach (1991) and interpolates to estimate the correction given the calculated $\log N_a^{n-1} - \log N_a^{n}$.

```
# apparent column density of Na I D1 in HD 186155 (from previous example)
N = Profile.ColumnDensity(line, continuum, D1, error=spectrum.error)
```

```
Name  = Column Density (Apparent)
Value = 181581935136.28503 1 / cm2
Error = [  8.68433738e+10  -8.68433738e+10] 1 / cm2
Notes = Measured using Profile.ColumnDensity() from SLiPy
```

For only slightly saturated lines and/or not optically thin, we can now estimate the *true* column density by passing these results as the keyword argument, *weakline*, to **Profile.ColumnDensity()**.

4.4 Uncertainty Calculations

The uncertainties provided in this study were computed from both the noise in the continuum (RMS) and from counting statistics. Given a "raw" spectrum in counting units, the uncertainty in those counts goes as

$$\delta s_{\text{count}} = \sqrt{s} \,. \tag{4.4.1}$$

The uncertainty from these statistics is added in quadrature with those from the continuum fitting,

$$\delta s_{\text{total}} \;=\; \left[\, \delta s_{\text{count}}^{\,2} \;+\; \delta s_{\text{rms}}^{\,2} \,\right]^{1/2} \tag{4.4.2}$$

Within SLiPy, either both or neither of these may be provided. The **Profile.Extract()** function (so in turn also **Profile.MultiFit()**), attaches an **.rms** value in like units directly to the *continuum* returned. The measurement functions look for this member variable and can handle either absolute or percent units. The uncertainties from counting statistics may be provided via the *error* keyword argument (as demonstrated by the last two examples). These work both in isolated usage and in connection with each other (adding in quadrature).

Preparing an *error* spectrum from an original spectrum of counts might be accomplished as in the following example.

```
135 # build error spectrum from original counts
136 s       = Spectrum("/from/file.fits")
137 s.error = Spectrum(100 * u.percent * np.sqrt(s.data.value) /
138             s.data.value, s.wave)
```

Be sure however to perform the same velocity corrections on both your spectra and corresponding *error* spectra.

CHAPTER 5

RESULTS

In this chapter I provide both numerical data and visual representations of the results of this study. Figures 5.1 - 5.3 all map the stellar targets from ELODIE over a 40° background sky centered on δ Cygni. [1] Figure 5.1 shows all the stellar targets; Figures 5.2 and 5.3 show the definitive detection of interstellar sodium or lack thereof, respectively. The numerical annotations in these latter two plots correspond directly with listings in Tables 5.1 - 5.2 and Tables 5.3 - 5.4, respectively, for Non-OB stars. O and B spectral type stars are typically at a much further distance than other stars. They typically showed multiple interstellar components from unique clouds at different velocities. Tables 5.3 - 5.11 provide a listing of the targets investigated and represent four categories: non-OB absorption, lack of absorption, and inconclusive results and then the OB targets. The tables provide the target name, the right ascension and declination (J2000) position, spectral classification, and distance (from parallax measurements). This data was retrieved directly from the SIMBAD database using the **Simbad** module from SLiPy. The author makes no claim of particular accuracy or uncertainty in these quantities and are provided by publications cited by SIMBAD.

[1] The mosaic used in the background of Figures 5.1 - 5.3 was adapted from data retieved from *sky-map.org*. Originally, the author attempted to construct the mosaics manually using the *Montage* image mosaicing software (montage.ipac.caltech.edu). A high-level wrapper to the *Montage* software was developed to facilitate the creation of large mosaics using DSS, SDSS, and 2MASS data (or user provided data). This functionality is provided within SLiPy under the **Montage** module (see Appendix C.13). Its usage is not documented here. Ultimately, there were serious issues with the state of the data retrieved from the DSS survey. *sky-map.org* has used the same data from DSS and put together a quality mosaic that has overcome many of these challenges. In the interest of efficiency, the *sky-map* data was used instead and converted to FITS format to mark the targets using APLpy (aplpy.github.io).

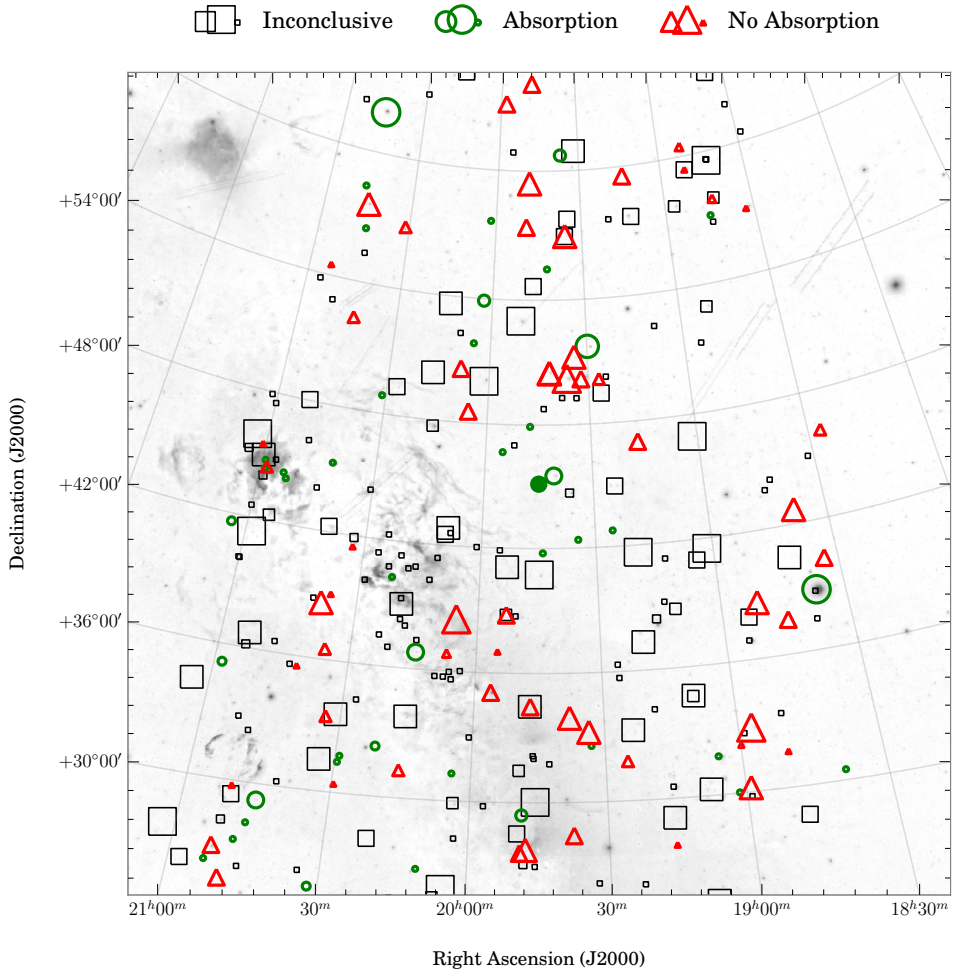

Figure 5.1: The above figure shows the entire 40° square field with every target star in the study represented. There are three different marker styles. The positive identification of interstellar sodium absorption is represented by a green circle. The definitive absence of absorption is given by a red triangle. The remaining data found to be inconclusive is given as a black square. δ Cygni is located in the center of the field and is marked by a filled-in green circle. The sizes of the markers are inversely proportional to the distance to that star. The largest markers show stars within the Local Cavity and the smallest markers are from 100 pc to more than 1 kpc.

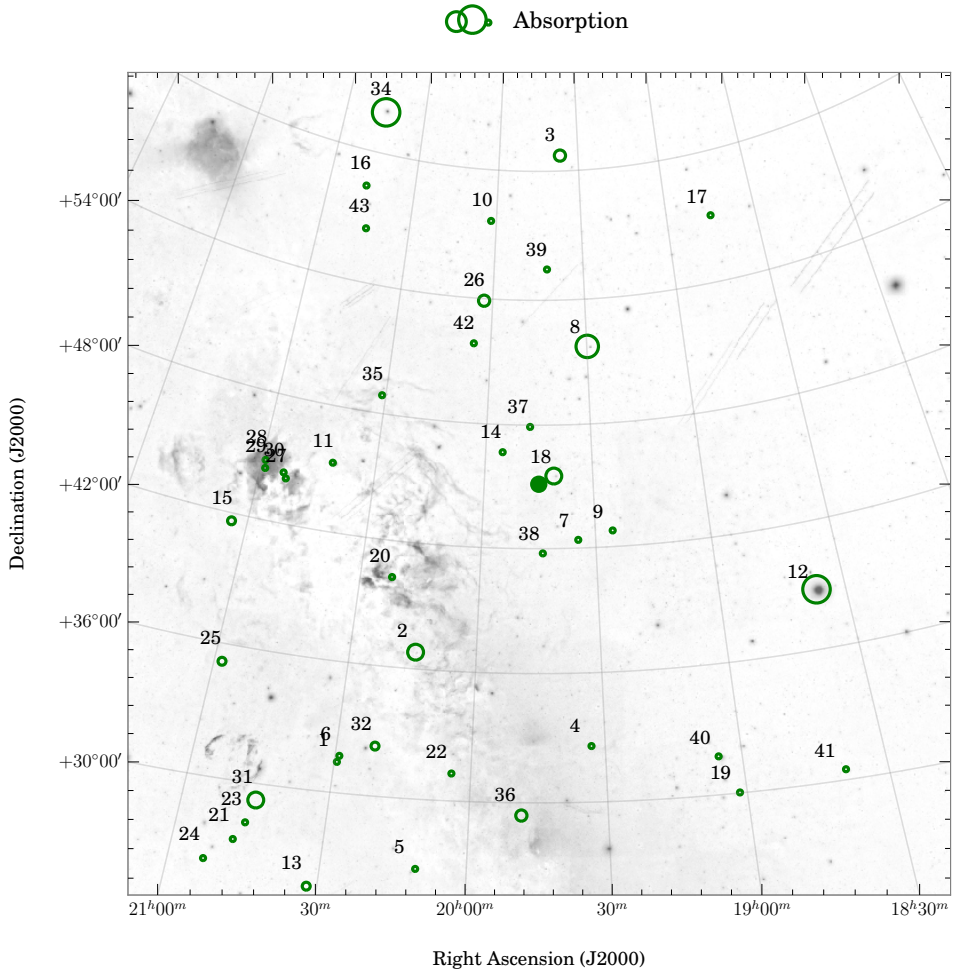

Figure 5.2: The above figure includes those targets from Figure 5.1 that have been identified as containing measurable interstellar Na I absorption. The annotations for the markers correspond to the items in Tables 5.1 and 5.2. Targets 12 and 34 are marked as measurable but have since been rendered *inconclusive*.

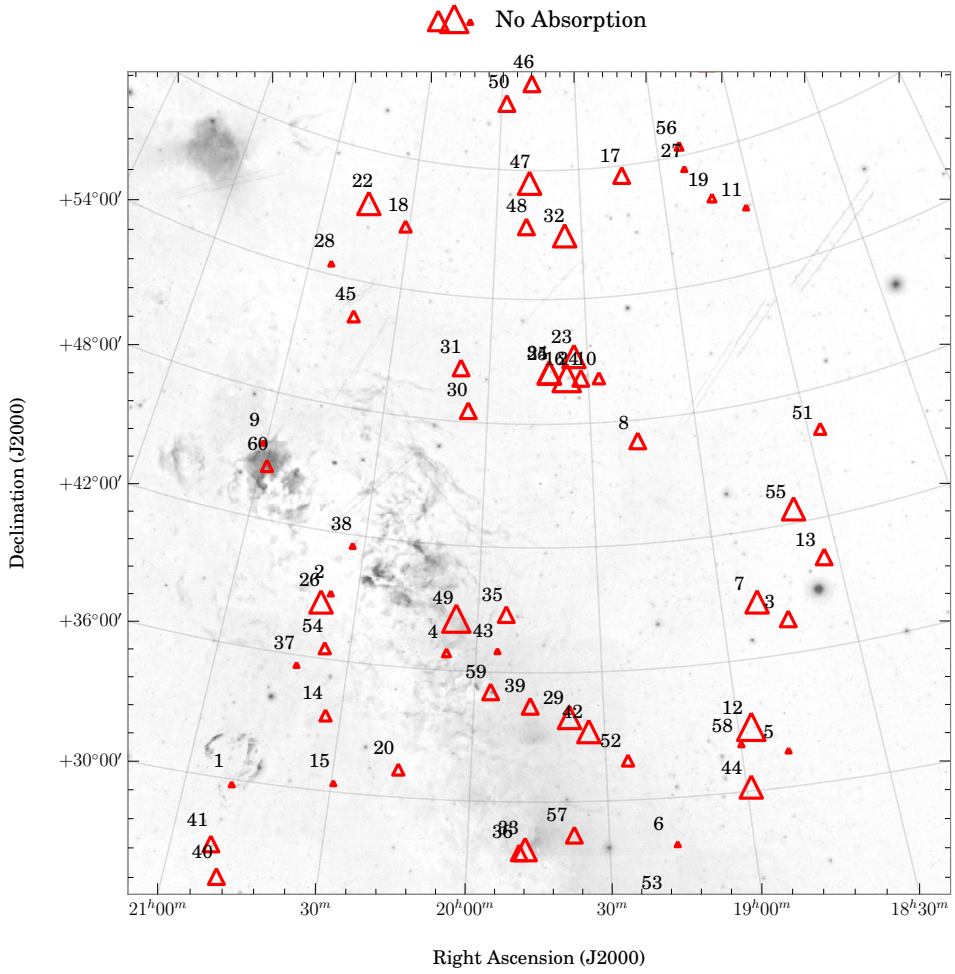

Figure 5.3: The above figure includes those targets from Figure 5.1 that have been identified as containing definitively no interstellar Na I absorption. The annotations for the markers correspond to the items in Tables 5.3 and 5.4.

TABLE 5.1
Non-OB Targets with Measurable Absorption – A

	Target [α]	Sp-Type [β]	RA [β]	Dec [β]	Dist [β]	W [γδ]	N [γϵ]	v [γζ]
			hrs	deg	pcs	mÅ	10^{10} cm^{-2}	km s^{-1}
1	HD 332329	A0	20.49	31.39	793.65	$93.2^{+49.0}_{-2.6}$	$134.36^{+56.2}_{-4.1}$	-23.0
2	HD 192640	A2V	20.24	36.81	42.70	$41.9^{+33.5}_{-0.7}$	$44.8^{+27.0}_{-0.8}$	-40.0
3	HD 185732	A5	19.62	60.71	76.10	$56.8^{+40.3}_{-1.3}$	$42.4^{+30.6}_{-1.8}$	-30.7
4	HD 184471	A8...	19.56	32.58	507.61	$8.6^{+8.3}_{-2.1}$	$4.4^{+4.3}_{-1.1}$	-32.8
5	HD 191747	A3III	20.18	26.90	129.87	$8.5^{+8.1}_{-0.6}$	$7.4^{+5.4}_{-0.6}$	-32.8
η 6	BD+314105	A2	20.49	31.69	...	...	...	...
7	HD 184875	A2V	19.58	42.41	176.68	$34.5^{+28.5}_{-1.2}$	$14.8^{+12.9}_{-0.7}$	-30.2
8	HD 184006	A5V	19.50	51.73	37.20	$12.5^{+11.7}_{-0.8}$	$5.1^{+4.9}_{-0.5}$	-14.9
η 9	HD 182989	A8-F7	19.42	42.78	289.02	...	...	...
10	HD 190397	A0	20.03	57.65	348.43	$134.8^{+38.0}_{-2.8}$	$198.1^{+34.8}_{-2.3}$	-29.4
η 11	HD 197345	A2Ia	20.69	45.28	432.90	...	...	...
η 12	HD 172167	A0Va	18.62	38.78	7.68	...	...	...
η 13	HD 195692	Am	20.53	25.80	87.03	...	...	...
14	HD 188854	A7p...	19.92	46.67	136.24	$83.0^{+56.9}_{-2.2}$	$57.0^{+20.6}_{-1.1}$	-31.8
15	HD 200723	F3IV	21.06	41.63	96.34	$125.6^{+48.9}_{-1.1}$	$146.2^{+39.6}_{-0.9}$	-34.8
16	HD 198343	F8	20.78	58.42	7692.31	...	...	-36.9
17	HD 173977	F2	18.75	57.09	187.62	$20.2^{+18.1}_{-1.2}$	$24.5^{+7.1}_{-0.7}$	-32.7
18	HD 186155	F5II-III	19.68	45.52	49.63	$27.3^{+23.3}_{-0.7}$	$24.8^{+8.5}_{-0.4}$	-32.1
19	BD+293448	F0p...	19.03	30.03	1388.89	$86.4^{+43.4}_{-8.1}$	$86.9^{+23.3}_{-5.2}$	-4.9
η 20	HD 194093	F8Iab	20.37	40.26	561.80	...	...	...
21	HD 198226	F0	20.80	27.46	456.62	$35.1^{+28.8}_{-1.8}$	$23.6^{+9.5}_{-0.9}$	-35.5
22	HD 190537	F0III	20.07	31.24	111.11	$49.6^{+36.5}_{-1.1}$	$153.5^{+18.7}_{-0.7}$	-37.5
23	HD 335198	G0	20.77	28.26	273.97	$74.3^{+47.4}_{-3.3}$	$92.5^{+23.7}_{-2.2}$	-45.9
24	HD 341233	G5	20.88	26.49	...	$32.6^{+27.7}_{-3.4}$	$30.1^{+10.1}_{-1.7}$	-40.1
25	BD+344217	G9.5Ve+K3Ve	20.97	35.17	83.40	$76.1^{+49.4}_{-3.2}$	$80.0^{+29.1}_{-2.2}$	-27.3

[α] Numbers correspond to annotations in Figure 5.2.
[β] Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.
[γ] Measurements and uncertainties computed with **Profile** module.
[δ] Equivalent Width for Na I D2.
[ϵ] *Corrected* Column Density for Na I line, see text (Savage and Sembach, 1991).
[ζ] Velocity of Na I D2 line center, from 5,889.951 Å. Uncertainty is approximately 2.6 km s^{-1}.
[η] Inconclusive identification.

TABLE 5.2
Non-OB Targets with Measurable Absorption – B

Target [α]	Sp-Type [β]	RA [β]	Dec [β]	Dist [β]	W [γδ]	N [γϵ]	v [γζ]
		hrs	deg	pcs	mÅ	10^{10} cm^{-2}	km s^{-1}
26 HD 190508	G5	20.04	53.89	61.96	$89.5^{+47.6}_{-5.5}$	$190.6^{+28.7}_{-3.9}$	-24.8
θ 27 BD+433754	G5V	20.88	44.12	...	$209.9^{+41.9}_{-6.7}$	...	-26.4
θ 28 HD 200019	G5	20.99	44.79	204.50	$113.4^{+59.7}_{-2.4}$	...	-39.8
29 HD 199939	G9III:...	20.98	44.41	306.75	$87.9^{+46.8}_{-4.2}$	$89.3^{+24.6}_{-2.8}$	-38.4
30 HD 199178	G5III-IV	20.90	44.39	101.32	$11.5^{+10.8}_{-1.1}$	$12.9^{+4.0}_{-0.6}$	-4.3
31 HD 335070	K0	20.75	29.27	57.41	$53.8^{+39.4}_{-3.0}$	$34.8^{+13.9}_{-1.6}$	-32.3
η 32 HD 193891	K0	20.36	32.31	89.13	$126.9^{+49.0}_{-2.2}$	$170.1^{+37.5}_{-1.8}$	-40.3
33 HD 170527	K0	18.42	64.84	158.48	$44.0^{+33.6}_{-0.8}$	$56.6^{+14.7}_{-0.5}$	-14.0
η 34 HD 198149	K0IV	20.75	61.84	14.27	...	...	...
θ 35 BD+483149	K0	20.52	48.87	...	$48.5^{+36.1}_{-2.0}$	...	-24.9
θ 36 HD 332698	K0	19.81	29.41	61.20	$67.8^{+43.6}_{-3.7}$	...	-7.3
θ 37 HD 187372	M1III	19.79	47.91	299.40	$48.5^{+36.0}_{-2.3}$	...	-31.6
38 HD 186619	M0III	19.73	41.77	174.83	$62.6^{+42.5}_{-2.8}$	$175.6^{+22.8}_{-1.8}$	-38.1
η 39 HD 186532	M5IIIa	19.70	55.46	316.46	...	...	...
η 40 HD 177808	M0III	19.08	31.74	189.75	...	...	-32.9
θ 41 HD 172381	M2III	18.64	30.45	270.27	$118.1^{+37.7}_{-14.2}$	...	-23.0
42 HD 190964	M1IIIa	20.09	51.84	219.30	$87.8^{+47.0}_{-3.2}$	$233.5^{+29.0}_{-2.3}$	-30.7
43 HD 197939	M3III	20.74	56.49	336.70	$123.7^{+37.9}_{-2.9}$	$175.2^{+32.4}_{-2.2}$	-30.0

[α] Numbers correspond to annotations in Figure 5.2.
[β] Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.
[γ] Measurements and uncertainties computed with **Profile** module.
[δ] Equivalent Width for Na I D2.
[ϵ] *Corrected* Column Density for Na I line, see text (Savage and Sembach, 1991).
[ζ] Velocity of Na I D2 line center, from 5,889.951 Å. Uncertainty is approximately 2.6 km s^{-1}.
[η] Inconclusive identification.
[θ] Too Saturated to apply optical depth method.

TABLE 5.3
Non-OB Targets with No Absorption – A

	Target [α]	Sp-Type [β]	RA [β]	Dec [β]	Dist [β]
			hrs	*deg*	*pcs*
1	HD 335238	A2	20.85	29.80	952.38
2	HD 196542	Ap...	20.61	39.10	...
3	HD 173648	Am	18.75	37.61	47.87
4	HD 191158	Am	20.12	36.83	88.11
5	HD 174567	A0Vs	18.83	31.63	278.55
6	HD 180583	F6Ib-II	19.27	27.93	429.18
7	HD 174912	F8	18.86	38.63	30.56
8	HD 181096	F6IV:	19.28	47.00	42.03
9	BD+443670	F8	21.01	45.50	...
10	HD 183361	F5	19.45	50.15	74.57
11	HD 171635	F7Ib	18.54	57.05	649.35
12	HD 176051	F9V+K1V	18.95	32.90	14.87
13	HD 171485	F5	18.55	40.16	41.70
14	HD 195992	F5	20.56	33.39	62.54
15	HD 195295	F5Iab:	20.49	30.37	235.85
16	HD 185395	F3+V	19.61	50.22	18.34
17	HD 180712	F8	19.23	59.55	44.56
18	HD 195872	F8	20.52	56.89	66.31
19	HD 173700	F8	18.72	57.82	85.40
20	HD 192804	F8V	20.26	31.24	76.22
21	HD 172323	F9V	18.59	63.70	43.65
22	HD 198084	F8IV-V+F9IV-V	20.76	57.58	27.29
23	HD 184960	F7V	19.57	51.24	25.11
24	HD 184448	F8	19.53	50.18	50.63
25	HD 186408	G1.5Vb	19.70	50.53	21.08
26	HD 196850	G1V	20.64	38.64	27.16
27	HD 175306	G9III_Fe-0.5	18.85	59.39	104.82
28	HD 199191	GIII+...	20.89	54.52	151.06
29	HD 185501	G5	19.63	33.89	31.96
30	HD 191022	G0	20.09	48.57	50.33

[α] Numbers correspond to annotations in Figure 5.3.
[β] Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.

TABLE 5.4
Non-OB Targets with No Absorption – B

	Target [α]	Sp-Type [β]	RA [β]	Dec [β]	Dist [β]
			hrs	*deg*	*pcs*
[31]	HD 191649	G0	20.15	50.59	45.27
[32]	HD 185414	G0	19.60	56.98	24.11
[33]	HD 187237	G2IV-V	19.80	27.87	26.24
[34]	HD 186427	G3V	19.70	50.52	21.21
[35]	HD 188326	G8IVv	19.88	38.77	55.93
[36]	HD 187462	G0IV	19.82	27.73	50.38
[37]	HD 197310	G0	20.69	35.55	...
[38]	HD 195988	G0	20.55	41.47	...
[39]	HD 187123	G5	19.78	34.42	48.26
[40]	HD 198483	G0V	20.83	25.77	51.23
[41]	HD 198809	G7III	20.87	27.10	57.80
[42]	HD 184499	G0V	19.56	33.20	31.92
[43]	HD 188650	G1Ib-IIe...	19.91	37.00	369.00
[44]	HD 176377	G1V	18.98	30.18	23.84
[45]	HD 197666	G0	20.72	52.30	79.49
[46]	HD 187792	G0	19.80	63.86	55.22
[47]	HD 187748	G0	19.80	59.42	27.71
[48]	HD 187876	G0	19.82	57.41	46.93
[49]	HD 190771	G2V	20.09	38.48	18.79
[50]	HD 189749	G	19.96	62.96	56.98
[51]	HD 170357	G1V	18.44	46.08	71.99
[52]	HD 182758	G0	19.42	31.80	61.46
[53]	HD 181047	G8V	19.30	25.37	46.82
[54]	HD 196361	G5	20.59	36.47	66.40
[55]	HD 172393	G5	18.63	42.67	28.08
[56]	HD 175441	G0	18.86	60.43	87.41
[57]	HD 185269	G0IV	19.62	28.50	50.28
[58]	HD 176670	K2.5III	19.00	32.15	338.98
[59]	HD 188947	K0III	19.94	35.08	41.37
[60]	HD 199870	K0IIIbCN...	20.97	44.47	79.30

[α] Numbers correspond to annotations in Figure 5.3.
[β] Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.

TABLE 5.5
Non-OB Targets Inconclusive – A

Target	Sp-Type [γ]	RA [γ]	Dec [γ]	Dist [γ]
		hrs	*deg*	*pcs*
BD+293427	A5p...	18.99	29.80	...
HD 340577	A3	20.58	26.49	699.30
HD 200405	A0pSrCr	21.03	47.91	746.27
HD 191742	A7p	20.16	42.54	304.88
HD 192678	A4p	20.23	53.66	198.02
HD 195725	A7III	20.49	62.99	41.84
HD 177196	A6IV	19.02	46.93	37.44
HD 183534	A1V	19.46	52.32	107.53
BD+353616A	F0	19.44	35.71	...
HD 341037	F0	20.78	26.36	3333.33
HD 197572	F7Ib...	20.72	35.59	1538.46
HD 198726	F5Ib+A0.8V	20.86	28.25	369.00
BD+423935	F5.5Ib-G5Ib:	21.00	42.60	-2564.10
HD 238932	F5	18.74	56.78	120.92
BD+423607	F3	20.15	42.87	82.58
HD 194951	F1II	20.45	34.33	1000.00
HD 331319	F3Ib	19.82	31.45	...
HD 195069	F0V:	20.45	49.38	37.57
HD 193370	F6Ib	20.31	34.98	970.87
HD 200805	F5Ib	21.07	45.16	373.13
HD 178593	F8	19.14	25.37	61.35
HD 188307	F8	19.88	41.08	53.30
HD 187253	F8	19.77	54.64	72.41
HD 185239	F8	19.58	57.77	84.53
HD 175225	G9IVa	18.86	52.98	25.67
HD 173701	G8V	18.74	43.83	26.69
HD 177153	G0	19.03	41.49	41.48
HD 185351	G9IIIbCN...	19.61	44.69	40.83
BD+413931	G5	20.92	42.30	67.29
HD 175306	G9III_Fe-0.5	18.85	59.39	104.82

[γ] Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.

TABLE 5.6
Non-OB Targets Inconclusive – B

Target	Sp-Type [γ]	RA [γ]	Dec [γ]	Dist [γ]
		hrs	*deg*	*pcs*
HD 182488	G9+V	19.39	33.22	15.76
HD 175900	G5	18.91	51.31	80.97
HD 193795	G4IV	20.35	28.11	70.67
HD 200391	G0V+G5V	21.04	27.81	52.00
HD 181655	G5V	19.33	37.33	25.39
BD+364301	G0	20.89	36.70	...
HD 179484	G5V	19.19	38.78	52.85
HD 190360	G7IV-V	20.06	29.90	15.86
HD 178911	G1V+K1V	19.15	34.60	52.33
HD 199598	G0V	20.96	26.41	31.63
HD 173605	G5	18.71	57.87	65.70
HD 190228	G5IV	20.05	28.31	61.61
HD 188326	G8IVv	19.88	38.77	55.93
HD 187123	G5	19.78	34.42	48.26
HD 182736	G8IV	19.40	44.93	57.64
HD 171242	G0	18.53	45.01	61.92
HD 185657	G6V	19.63	49.28	140.25
HD 171008	G5	18.47	60.44	79.30
HD 193216	G5	20.28	50.28	30.77
HD 178911B	G5	19.15	34.60	42.59
HD 200102	G1Ib	21.00	45.00	1250.00
HD 190403	G5Ib-II	20.06	29.99	588.24
HD 174104	G0Ib	18.80	28.72	1315.79
HD 191010	G3Ib	20.11	25.69	518.13
HD 180161	G8V	19.20	57.67	20.02
HD 180683	G0	19.26	38.38	64.52
HD 197037	G0	20.66	42.25	32.33
HD 180502	G0IV	19.26	29.12	85.54
HD 197488	G2V	20.71	45.82	54.14
HD 175425	G0	18.89	37.99	53.53

[γ] Retrieved from SIMBAD database (as of June 13, 2015);
uncertainties not listed.

TABLE 5.7
Non-OB Targets Inconclusive – C

Target	Sp-Type [γ]	RA [γ]	Dec [γ]	Dist [γ]
		hrs	*deg*	*pcs*
HD 193215	G5	20.26	64.20	68.97
HD 172557	G5	18.61	63.36	52.63
HD 184601	G0	19.53	60.86	83.68
HD 199100	G5IV-V	20.89	36.13	57.60
HD 195987	G9V	20.55	41.90	22.05
HD 187548	G0V	19.82	28.61	45.11
HD 177780	G3V	19.07	41.00	55.01
HD 190605	G2V	20.08	26.05	44.98
HD 178911	G1V+K1V	19.15	34.60	52.33
HD 178450	G8VSB	19.13	30.25	27.96
HD 173024	G5	18.69	40.48	80.97
HD 173289	G5	18.71	44.27	93.81
HD 197140	G5	20.67	38.78	53.56
HD 197207	G5V	20.69	30.19	55.46
HD 189087	K1V	19.95	29.82	26.72
HD 234677	K4Ve+K7.5Ve	18.57	51.72	16.35
HD 197989	K0III-IV	20.77	33.97	22.29
HD 183255	K2.5V	19.44	49.47	25.01
HD 191026	K0IV	20.11	35.97	23.95
HD 338867	K0	19.81	27.29	...
HD 199956	K0	20.98	44.06	201.61
HD 334514	K0	20.56	31.47	...
HD 199546	K0	20.93	53.81	452.49
HD 189806	K2	20.01	32.95	189.75
HD 181209	K0	19.30	34.14	432.90
HD 188056	K3III	19.84	52.99	62.07
BD+333930	K0V	20.52	33.54	48.85
BD+334140	K0	21.06	34.22	...
HD 192787	K0III	20.26	33.73	97.47
HD 196134	K0III-IV	20.56	41.77	92.08

[γ] Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.

TABLE 5.8
Non-OB Targets Inconclusive – D

Target	Sp-Type [γ]	RA [γ]	Dec [γ]	Dist [γ]
		hrs	*deg*	*pcs*
HD 176408	K1III	18.95	57.81	84.67
HD 198550	K5V	20.84	29.38	20.99
HD 192910	K3Ib+	20.26	47.71	324.68
HD 193469	K5Ib	20.32	39.00	1492.54
HD 198794	K3Ib	20.85	48.03	232.02
HD 200560	K2.5V	21.04	45.88	19.47
HD 190470	K3V	20.07	25.79	21.95
BD+413306	K0V	19.32	41.63	35.68
BD+522815	K7-V	20.84	52.90	22.42
HD 332518	K5V	19.76	30.01	20.32
HD 331093	K0	19.71	31.74	55.07
HD 191806	K0	20.16	52.28	69.40
HD 171607	K0	18.53	61.78	46.75
HD 179094	K1IV	19.14	52.43	71.02
HD 198425	K2.5V	20.82	32.28	24.05
HD 188753	K0	19.92	41.87	46.23
BD+400883	M3.0V	21.00	40.07	15.29
HD 173739	M3.0V	18.71	59.63	3.57
GJ 0747	M3V	19.13	32.54	8.13
GJ 0809	M2.0V	20.89	62.15	7.05
HD 186686	M2-7e	19.73	48.78	-3448.28
GJ 0725	M3.0V	18.71	59.63	3.57
BD+394208	M3Iab:	20.48	39.98	1098.90
BD+364025	M4Iab	20.36	36.93	133.69
HD 182190	M1III	19.34	57.65	242.13
HD 199871	M0III	20.97	41.36	213.68
HD 186776	M4III	19.75	40.72	251.26
HD 175865	M5III	18.92	43.95	91.41
HD 175588	M4II	18.91	36.90	225.73
HD 184786	M4.5III	19.56	49.26	406.50

[γ] Retrieved from SIMBAD database (as of June 13, 2015);
uncertainties not listed.

TABLE 5.9
Non-OB Targets Inconclusive – E

Target	Sp-Type [γ]	RA [γ]	Dec [γ]	Dist [γ]
		hrs	*deg*	*pcs*
GJ 0815	M3.0V	21.00	40.07	15.29
GJ 0766	M4.5V	19.76	27.13	10.31
HD 189063	M0III	19.91	60.82	540.54
HD 178003	M0III	19.10	29.92	446.43
HD 180450	M0III	19.26	30.53	389.11
HD 179869	M3III	19.20	41.24	317.46
HD 187849	M2III	19.84	38.72	212.31
HD 190544	M1III	20.02	64.82	180.18
HD 199305	M2.0V	20.89	62.15	7.05
HD 173740	M3.5V	18.71	59.63	3.45
HD 331161 [ϵ]	K5	19.77	32.02	13.61
HD 190163 [ϵ]	M5.5-6.5e	20.02	50.04	401.61
HD 182917 [ϵ]	M7IIIab+Be	19.41	50.24	242.72
GJ 0802 [ϵ]	M5.0V	20.72	55.35	16.67

[γ] Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.
[ϵ] Inconclusive because the spectrum was bad (see text).

TABLE 5.10
OB Targets – A

Target	Sp-Type [β]	RA [β]	Dec [β]	Dist [β]	W [γδ]	N [γϵ]	v [γζ]
		hrs	*deg*	*pcs*	*mÅ*	$10^{10} cm^{-2}$	kms^{-1}
[θη] HD 188209	O9.5Iab	19.87	47.03	1098.90	$212.0^{+27.9}_{-3.0}$	...	-24.9
[η] HD 198820	B3III	20.87	32.85	531.91	$92.8^{+53.2}_{-1.5}$	$88.4^{+26.1}_{-1.0}$	-0.5
[η] HD 194335	B2IIIe	20.40	37.48	414.94	$101.6^{+46.5}_{-1.2}$	$118.5^{+28.2}_{-0.8}$	-20.8
[η] HD 200120	B1.5Vnne	21.00	47.52	434.78	$86.0^{+46.4}_{-1.0}$	$94.8^{+29.7}_{-0.7}$	-29.9
HD 182255	B6III	19.38	26.26	120.48	$16.6^{+15.4}_{-1.1}$	$11.6^{+4.8}_{-0.6}$	-5.7
[η] HD 338529	B5	19.54	26.39	137.93	$40.2^{+31.6}_{-1.9}$	$64.4^{+14.2}_{-1.1}$	-24.4
[η] HD 183056	B9sp...	19.44	36.32	216.92	$22.8^{+20.0}_{-0.7}$	$13.6^{+6.0}_{-0.4}$	-33.3
HD 174638	B8II-IIIep	18.83	33.36	294.99	$164.2^{+38.0}_{-2.4}$	$643.8^{+48.4}_{-3.5}$	-17.5
[η] HD 176437	B9III	18.98	32.69	190.11	$19.7^{+17.6}_{-2.0}$	$9.9^{+4.7}_{-1.1}$	-34.0
HD 180163	B2.5IV	19.23	39.15	425.53	$96.9^{+46.6}_{-1.6}$	$109.8^{+27.1}_{-1.1}$	-11.3
HD 192685	B3V	20.25	25.59	303.95	$27.8^{+23.7}_{-1.1}$	$14.9^{+6.9}_{-0.6}$	-31.2
HD 198183	B5V	20.79	36.49	235.85	$174.1^{+70.6}_{-3.5}$	$244.2^{+49.3}_{-2.6}$	-28.9
[η] BD+404124	B2Ve	20.34	41.36	79.49	$142.9^{+38.5}_{-3.1}$	$152.7^{+36.3}_{-2.4}$	-41.0
[η] HD 194279	B2Ia	20.39	40.76	1724.14	$120.8^{+41.5}_{-2.2}$	$222.0^{+33.2}_{-1.9}$	-39.9
[η] HD 172324	B9Ib	18.63	37.43	1030.93	$77.5^{+45.6}_{-1.6}$	$80.3^{+22.3}_{-1.0}$	-32.8
[η] BD+413731	B3n	20.40	42.30	2222.22	$131.2^{+36.9}_{-4.6}$	$513.6^{+35.0}_{-4.6}$	-38.5
[θη] HD 193322	O9V	20.30	40.73	595.24	$190.6^{+43.1}_{-3.8}$	...	-33.3
[θη] HD 195592	O9.7Ia	20.51	44.32	909.09	$38.3^{+30.8}_{-1.6}$	...	-27.5
[θη] HD 190864	O6.5III(f)	20.09	35.61	487.80	...	...	...
[θη] HD 191423	ON9IIIn	20.14	42.61	5263.16	$239.1^{+10.6}_{-4.1}$	...	-37.2
[θη] HD 189957	O9.7III	20.02	42.01	-2702.70	$46.6^{+34.8}_{-2.6}$	...	-38.3
[θη] HD 192639	7.5Iabf	20.24	37.35	1886.79	...	...	-30.3
[θη] HD 191978	O8V	20.18	41.35	100000.00	...	...	-31.1
[θη] HD 190429	O4If+O9.5II	20.06	36.03	862.07	...	...	-43.4

[β] Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.
[γ] Measurements and uncertainties computed with **Profile** module.
[δ] Equivalent Width for Na I D2.
[ϵ] *Corrected* Column Density for Na I line, see text (Savage and Sembach, 1991).
[ζ] Velocity of Na I D2 line center, from 5895.924 Å.
[η] Multiple components. Closest velocity listed.
[θ] Too Saturated to apply optical depth method.

TABLE 5.11
OB Targets – B

Target	Sp-Type [β]	RA [β]	Dec [β]	Dist [β]	W [γδ]	N [γϵ]	v [γζ]
		hrs	*deg*	*pcs*	*mÅ*	$10^{10} cm^{-2}$	kms^{-1}
[θη] HD 199579	O6.5V	20.94	44.92	1666.67	...	...	...
[θη] HD 193514	O7Ib	20.32	39.27	-925.93	...	...	...
[θη] HD 190429	O4If	20.06	36.03	...	...	...	...
[θη] HD 186980	O7.5III	19.77	32.12	-2631.58	...	...	...
[θη] HD 193443	O9III	20.31	38.28	1086.96	...	...	...
[θη] HD 191612	O8fpe	20.16	35.73	2040.82	...	...	...
[θη] HD 192281	O4.5V(n)((f))	20.21	40.27	1265.82	...	...	...
[θη] HD 191201	O9III+O9V	20.12	35.72	628.93	...	...	...
[θη] HD 193237	B1-2_Ia-0_ep	20.30	38.03	3125.00	...	...	...
[θη] HD 198478	B4Ia	20.82	46.11	714.29	...	...	...
[θη] HD 228712	B0.5Ia	20.28	40.89	...	...	...	...
[θη] HD 194839	B0.5Iae	20.44	41.38	1041.67	...	...	...

[β] Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.
[γ] Measurements and uncertainties computed with **Profile** module.
[δ] Equivalent Width for Na I D2.
[ϵ] *Corrected* Column Density for Na I line, see text (Savage and Sembach, 1991).
[ζ] Velocity of Na I D2 line center, from 5895.924 Å.
[η] Multiple components. Closest velocity listed.
[θ] Too Saturated to apply optical depth method.

CHAPTER 6

DISCUSSION

6.1 Structure and Target Selection Bias

Looking at Figures 5.2 and 5.3 we can attempt to glean information about the possible structure of the LCCC. Using only the information from the positive absorption however is not appropriate because any apparent pattern in location here is primarily due to the fact that these were just the targets available from the archive (reflecting stars with potential exoplanets as studied by ELODIE) and not a uniform and/or unbiased selection pool of nearby stars. With that said, three stars of particular interest are HD 192640 (b03 Cyg, #2, 42.7 pc), HD 184006 (ι Cyg, #8, 37.2 pc), and HD 186155 (#18, 49.6 pc). These lines-of-sight show interstellar sodium absorption similar to that of δ Cyg (Welty et al., 1994) and are all within 50 parsecs, distances well inside the neutral gas "edge" of the local cavity. If we consider simultaneously the conclusions of the definitive absence of sodium in the spectra of all the stars represented in Figure 5.3, an imaginary line drawn from b03 Cyg to ι Cyg (nearly 15 degrees across the sky) is interrupted by a clear lack of interstellar absorption in the intervening gaps (particularly just south of ι Cyg). This punctuated chain of absorption indicates the possibility of a similar broken ribbon structure to that of the LLCC.

6.2 The Soft X-ray Background

As brought up in the Introduction, there is a soft X-ray background radiation (SXBR) believed to originate from the hot gas that fills the Local Bubble. Studies of the Local Leo Cold Cloud (Peek et al., 2011, Snowden et al., 2015) using the *Rosat* All Sky-Survey and have shown no shadowing effect despite the expectation that such shadowing should be present if the X-ray source were behind the cloud. Snowden et al. (2015) offers a concise synthesis of the constraints this places on a Local Hot Bubble given multi-wavelength observations of the LLCC and the current literature. Very recently, it has been shown that in fact the solar wind charge exchange (SWCX) contributes significantly to the SXBR at both 1/4 and 3/4 keV. A lengthy discussion of the SXBR and the constraints it places on a Local Hot Bubble are not offered here, but indeed the LLCC has provided a near perfect test case for studying the dynamics and content of the Local Bubble. Unfortunately, it offers only a sample of one object. Now, this preliminary study of the LCCC provides needed observations to now begin multi-wavelength observations and extend the current understanding of the complex of local interstellar clouds.

6.3 Stellar vs Interstellar

I would like to make a brief comment in regards to the interpretation rendered on stellar lines versus interstellar lines. It can at times be difficult to immediately distinguish between weak stellar absorption and a real interstellar component. For the most part, the dominant sodium lines can be readily separated given their size. However, we can be fooled into a detection without making velocity comparisons. As an example, in Figure 6.1, I have over-plotted [1] HD 181047 on top of HD 198149. It was originally

[1] Using **.overplot()** on a **SPlot** giving any other **SPlot** as an argument over plots the two. Any number of them can be given. The legend will automatically be updated provided they both have labels.

suspected that HD 198149 might have an interstellar component, given its velocity. By over-plotting a similar spectrum we can see that even though the one was of a strength and velocity that make it a potential detection for the LCCC, it appears to be a weak stellar line. It *moves* with the other stellar lines – an interstellar line would not.

6.4 Future Studies

Follow up and additional observations of Na I for sight-lines in the Cygnus field using the KPNO Coudè Feed are needed to confirm results presented here. Preliminary results from the Coudè Feed have shown absorption of interstellar Na I towards seven stars, including δ Cygni. Six of the targets overlap with those studied from the ELODIE archive. The detections here are consistent with those measured with ELODIE. This data is available in Table 6.1. A similar campaign to that pursued for the Leo cloud over the past decade could add to the weight of current evidence in regards to our understanding of the local cavity. A proposal to use the *Hubble Space Telescope* for some of these much need observations was submitted. The author is a co-inverstigator on this proposal. The full text for this is provided in Appendix A.

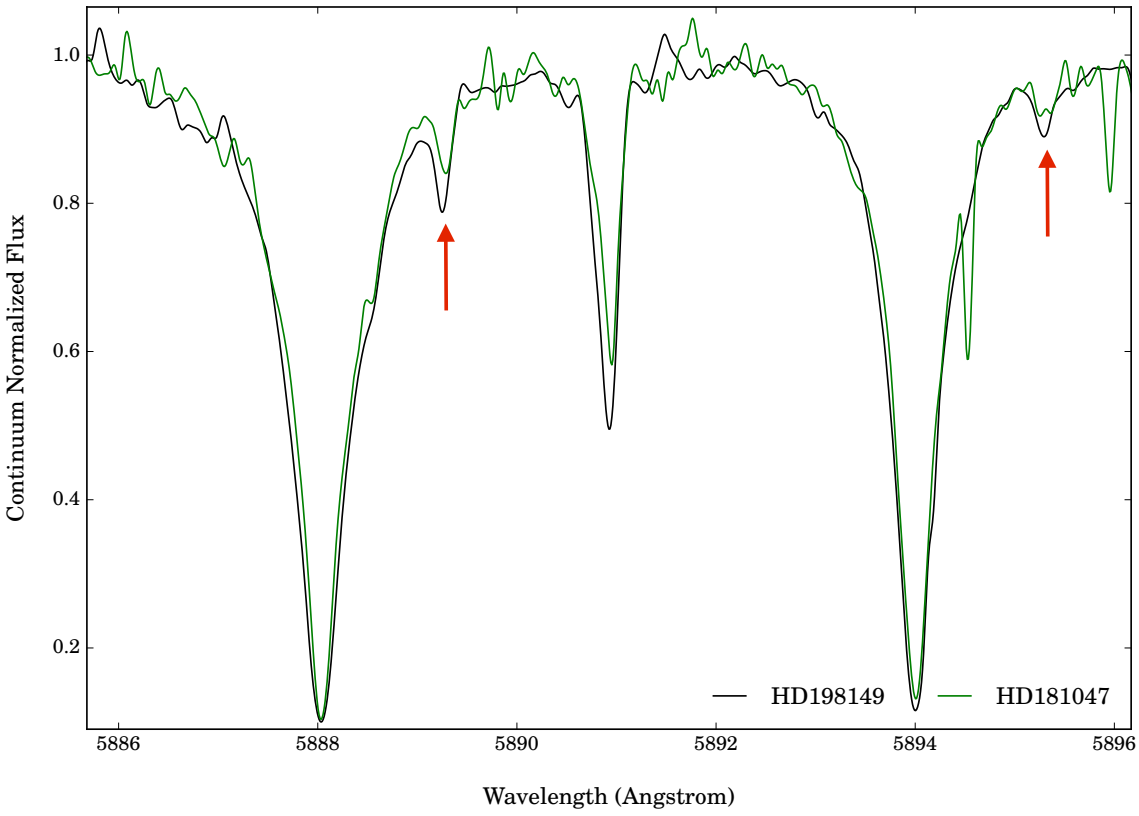

Figure 6.1: Often, narrow stellar lines can mimic interstellar absorption. An indication that a particular line may not be interstellar is when it not only appears in more than one spectrum of stars of the same type, but when after a velocity correction, it appears at the same shift from the larger component. In this figure, the spectra have been artificially shifted to overlay their lines for comparison. So while HD 198149 appears to have a narrow interstellar line and is at a velocity close to those for the cloud, it is more likely than not to in fact be stellar in origin. The D1 and D2 components in question are indicated by red arrows.

TABLE 6.1

Stars Observed in the Region of δ Cygnus using the Coudè Feed

Star	Spectral Type	l [α] (deg)	b [α] (deg)	Distance (parsecs)	Absorption
δ Cyg	B9.5IV	78.71	10.24	52	Y
HD 175824	FIII	78.65	19.56	53	Y
HD 179958	G4V	80.67	17.27	22	N
[β] HD 184006	A5Vn	83.61	15.44	38	N?
[β] HD 184960	F7V	83.46	14.59	26	N
[β] HD 185351	G9III	77.62	11.35	41	Y
[β] HD 185395	F4V	82.67	13.45	19	N
[β] HD 186155	F5II-III	78.72	11.07	49	Y
HD 187160	G0	78.16	9.60	43	Y
HD 192485	F5V	81.92	5.79	35	N
[β] HD 192640	A2V	74.45	1.17	41	Y
HD 192866	A3	81.99	5.99	343	Y

[α] l and b refer to galactic coordinates.

[β] Observed with both ELODIE and KPNO Coudè Feed.

CHAPTER 7

CONCLUSIONS

The goal of this study was to search in the vicinity of δ Cygni for evidence of a new local interstellar cloud. Using public access archival data from ELODIE, we have indeed observed the absorption of interstellar sodium (D2 λ5889.591 and D1 λ5895.924) extensively in the 40 degree square field here. The suspected stellar origin of the D2 line from Lallement et al. (2003) is now in question, given the presents of so much interstellar sodium in the region, particularly HD 186155 (target #18 in Figure 5.2 and Table 5.1) in such close proximity to δ Cygni. These observations provide motivation for further investigation in the region. Our understanding of the Local Hot Bubble has been expanded by the observations and analysis of the Local Leo Cold Cloud and now the Local Cygnus Cold Cloud offers a new testbed for investigation.

In addition to the observations highlighted above, this work has made progress in other areas. First, it illustrates the potential for investigations of the interstellar medium using archival data provided by exoplanet studies. With the multitude of planetary search archives currently on line and coming in the near future, the amount of public access spectral data is growing and will facilitate "big data" absorption line studies for understanding the interstellar medium. With this comes the need for precision auto-calibration tools for larger spectral data sets. Indeed, at the moment one must still manually perform the fitting of individual lines; however, the reduction, preparation, and calibration of such data can be more quickly undertaken using the tools provided by

SLiPy. Using this framework it may be possible to create automatic fitting procedures in the future.

As a platform independent, free, and open source product it is now open for further community development and may in time be rolled into the existing dominant package for computational astronomy with Python. The **Spectrum** class, built using **numpy** arrays, is an efficient structure for manipulating and working with spectral data. The **Profile** module gives the user a fitting and measurement facility for analyzing absorption features in the same environment within which they work with spectra as numerical objects (not just relying on file systems). The automated **Telluric** and **Velocity** corrections offer quick and precise calibrations.

The **Telluric.Correct()** function merits particular emphasis here. As described in Chapter 3, the method of using reference spectra to control for telluric absorption is effective in many instances. [1] The all-in-one approach of the function offered here, to horizontally cross-correlate and vertical amplitude fit all the calibration spectra separately guarantees the best possible fit for the provided candidate reference spectra. With that, the ability to easily provide an arbitrary number of candidate reference spectra offers to automatically choose the best telluric reference out of all the candidates. In the region investigated in this study there exists a veritable forest of telluric lines. In the example control test plotted in Figure 3.2, immediately adjacent to the region shown, $\lambda \simeq 5810$Å, the signal to noise was observed to be $\simeq 320$. The mean S/N in the same region for the six spectra of Regulus was $\simeq 310$. After the telluric correction procedure was applied, the signal to noise measured deep inside the afflicted region was $\simeq 210$. This is roughly a $\sqrt{2}$ reduction assuming the continuum would otherwise maintain a similar S/N up through 6000Å. Therefore, we conclude that the algorithm has retained the expected S/N for

[1] Recently, *TelFit* (Gullikson et al., 2014) was developed for Python and offers a wrapper to the Line-By-Line Radiative Transfer Model (LBLRTM). This is a separate model than that offered by **Telluric.Correct()**.

combining the two spectra and the reduction is a consequence of averaging two stochastic sets of noise.

REFERENCES

L. Aller, G. Berge, R. Bless, S. Czyzak, L. Davis, H. Friedman, J. Greenberg, G. Haro, H. L. Johnson, H. Johnson, F. Kerr, W. Liller, B. Lynds, R. Minkowski, G. Münch, E. Parker, L. Spiterzer, and E. Upton. Nebula and interstellar matter. In B. M. Middlehurst and L. H. Aller, editors, *Stars and Stellar Systems*, volume VII. The University of Chicago Press, Chicago and London, 1968.

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, Oct. 2013. doi: 10.1051/0004-6361/201322068.

A. Baranne, D. Queloz, M. Mayor, G. Adrianzyk, G. Knispel, D. Kohler, D. Lacroix, J.-P. Meunier, G. Rimbaud, and A. Vin. ELODIE: A spectrograph for accurate radial velocity measurements. *Astronomy and Astrophysics Supplement*, 119:373–390, Oct. 1996.

F. Bouchy and Sophie Team. SOPHIE: the successor of the spectrograph ELODIE for extrasolar planet search and characterization. In L. Arnold, F. Bouchy, and C. Moutou,

editors, *Tenth Anniversary of 51 Peg-b: Status of and prospects for hot Jupiter studies*, pages 319–325, Feb. 2006.

B. T. Draine. *Physics of the Interstellar and Intergalactic Medium*. Princeton University Press, Princeton and Oxford, 2011.

P. C. Frisch. The Local Bubble and Interstellar Material Near the Sun. *Space Science Reviews*, 130:355–365, June 2007. doi: 10.1007/s11214-007-9209-z.

K. Gullikson, S. Dodson-Robinson, and A. Kraus. Correcting for Telluric Absorption: Methods, Case Studies, and Release of the TelFit Code. *AJ*, 148:53, Sept. 2014. doi: 10.1088/0004-6256/148/3/53.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`. [Online; accessed 2015-06-07].

A. Kramida, Yu. Ralchenko, J. Reader, and and NIST ASD Team. NIST Atomic Spectra Database (ver. 5.2), [Online]. Available: `http://physics.nist.gov/asd` [2015, July 13]. National Institute of Standards and Technology, Gaithersburg, MD., 2014.

R. Lallement, B. Y. Welsh, J. L. Vergely, F. Crifo, and D. Sfeir. 3D mapping of the dense interstellar gas around the Local Bubble. *A&A*, 411:447–464, Dec. 2003. doi: 10.1051/0004-6361:20031214.

D. M. Meyer, J. T. Lauroesch, C. Heiles, J. E. G. Peek, and K. Engelhorn. A Cold Nearby Cloud inside the Local Bubble. *ApJL*, 650:L67–L70, Oct. 2006. doi: 10.1086/508658.

D. M. Meyer, J. T. Lauroesch, J. E. G. Peek, and C. Heiles. The Remarkable High Pressure of the Local Leo Cold Cloud. *ApJ*, 752:119, June 2012. doi: 10.1088/0004-637X/752/2/119.

D. C. Morton. Atomic Data for Resonance Absorption Lines. III. Wavelengths Longward of the Lyman Limit for the Elements Hydrogen to Gallium. *ApJS*, 149:205–238, Nov. 2003. doi: 10.1086/377639.

J. Moultaka, S. A. Ilovaisky, P. Prugniel, and C. Soubiran. The ELODIE Archive. *The Publications of the Astronomical Society of the Pacific*, 116:693–698, July 2004. doi: 10.1086/422177.

J. E. G. Peek, C. Heiles, K. M. G. Peek, D. M. Meyer, and J. T. Lauroesch. The Local Leo Cold Cloud and New Limits on a Local Hot Bubble. *ApJ*, 735:129, July 2011. doi: 10.1088/0004-637X/735/2/129.

P. Prugniel, C. Soubiran, M. Koleva, and D. Le Borgne. New release of the ELODIE library: Version 3.1. *ArXiv Astrophysics e-prints*, Mar. 2007.

B. D. Savage and K. R. Sembach. The analysis of apparent optical depth profiles for interstellar absorption lines. *ApJ*, 379:245–259, Sept. 1991. doi: 10.1086/170498.

S. L. Snowden, R. Egger, D. P. Finkbeiner, M. J. Freyberg, and P. P. Plucinsky. Progress on Establishing the Spatial Distribution of Material Responsible for the 1 4 keV Soft X-Ray Diffuse Background Local and Halo Components. *ApJ*, 493:715–729, Jan. 1998. doi: 10.1086/305135.

S. L. Snowden, C. Heiles, D. Koutroumpa, K. D. Kuntz, R. Lallement, D. McCammon, and J. E. G. Peek. Revisiting the Local Leo Cold Cloud and Revised Constraints on the Local Hot Bubble. *ApJ*, 806:119, June 2015. doi: 10.1088/0004-637X/806/1/119.

L. Spitzer. *Physical Processes in the Interstellar Medium*. John Wiley & Sons, Inc., 1978.

P. Stumpff. Two Self-Consistent FORTRAN Subroutines for the Computation of the Earth's Motion. *A&AS*, 41:1, June 1980.

S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation, 2011.

G. L. Verschuur. Some Very Cold HI Clouds Found in Emission. *ApL*, 4:85–87, 1969.

S. S. Vogt, S. L. Allen, B. C. Bigelow, L. Bresee, B. Brown, T. Cantrall, A. Conrad, M. Couture, C. Delaney, H. W. Epps, D. Hilyard, D. F. Hilyard, E. Horn, N. Jern, D. Kanto, M. J. Keane, R. I. Kibrick, J. W. Lewis, J. Osborne, G. H. Pardeilhan, T. Pfister, T. Ricketts, L. B. Robinson, R. J. Stover, D. Tucker, J. Ward, and M. Z. Wei. HIRES: the high-resolution echelle spectrometer on the Keck 10-m Telescope. In D. L. Crawford and E. R. Craine, editors, *Instrumentation in Astronomy VIII*, volume 2198 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, page 362, June 1994.

D. E. Welty, L. M. Hobbs, and V. P. Kulkarni. A high-resolution survey of interstellar NA I D1 lines. *ApJ*, 436:152–175, Nov. 1994. doi: 10.1086/174889.

M. Wenger, F. Ochsenbein, D. Egret, P. Dubois, F. Bonnarel, S. Borde, F. Genova, G. Jasniewicz, S. Laloë, S. Lesteven, and R. Monier. The SIMBAD astronomical database. The CDS reference database for astronomical objects. *A&AS*, 143:9–22, Apr. 2000. doi: 10.1051/aas:2000332.

APPENDIX A

HUBBLE SPACE TELESCOPE PROPOSAL

The Local Cygnus Cold Cloud - Testing the Hot Local Bubble

Scientific Category: ISM AND CIRCUMSTELLAR MATTER
Scientific Keywords: Dust, Galactic Structure, Interstellar And Intergalactic Medium
Instruments: STIS

Proprietary Period: 12 Proposal Size: Small UV Initiative: Yes

Orbit Request	Prime	Parallel
Cycle 23	5	0

Abstract

There has recently been a revolution in our knowledge about the nature of clouds within the Local Bubble which is challenging our models for the content and structure of this gas. Over several decades a view had been developed of the Local Bubble as a cavity surrounding the Sun filled with hot (~1E6 K) X-ray emitting gas that was devoid of cooler material. The recent detection of the Local Leo Cold Cloud withing the bubble has upset this view, and we are now trying to understand the formation of extremely cold gas (20 K) within this cavity. In order to further understand the formation and nature of this material, we request observations of two stars behind the Local Cygnus Cold Cloud, the second such pocket of cold material discovered in the Local Bubble. We will search for pressure variations within the cloud, which are expected to arise where turbulent fragmentation in the wake of collisions of warm clouds pushes the localized overpressures above the limiting values implied by the measured velocity constraints on the colliding clouds.

J Lauroesch : The Local Cygnus Cold Cloud - Testing the Hot Local Bubble

Investigators:

	Investigator	Institution	Country
PI&	J Lauroesch	University of Louisville Research Foundation, Inc.	USA/KY
CoI	G Lentner	University of Louisville Research Foundation, Inc.	USA/KY

Number of investigators: 2
& Phase I contacts: 1

Target Summary:

Target	RA	Dec	Magnitude
HR-7444	19 34 41.2593	+42 24 45.04	V = 5.348, 1.7e-11 at 1565 Ang, 2.4e-11 at 1965 Ang, 1.8e-11 at 2365 Ang
-B03-CYG	20 14 32.0333	+36 48 22.69	V = 4.94 +/- 0.02, 3.5e-12 at 1598 Ang, 3e-11 at 2000 Ang

Observing Summary:

Target	Config Mode and Spectral Elements	Flags	Orbits
HR-7444	STIS/FUV-MAMA Spectroscopic E140H (1598)		2
	STIS/NUV-MAMA Spectroscopic E230H (2113)		
-B03-CYG	STIS/FUV-MAMA Spectroscopic E140H (1598)		3
	STIS/NUV-MAMA Spectroscopic E230H (2113)		

Total prime orbits: 5

Scientific Justification

Our knowledge of the interstellar medium in the immediate Solar vicinity has been undergoing a revolution over the past few years. But even as we begin to have a more complete census of the interstellar components near the Sun, we find ourselves facing new difficulties in integrating the multi–wavelegth observations into a coherent whole. The limitations we face in our ability to model the interstellar medium near the Sun clearly reflect limitations in our ability to model the interstellar medium elsewhere in the Galaxy, for if we cannot understand the gas and dust near the Sun where we have all the advantages of proximity we have no hope of understanding what is going on in distant regions of the Galaxy. The physical conditions in the ambient interstellar medium surrounding the Sun also set the boundry conditions for the heliosphere, which is the edge of the Solar System (Frisch 1995).

From a combination of X-ray, UV and optical measurements of the interstellar medium a picture was developed over several decades of the interstellar environment within which the Solar System is embedded (Cox & Reynolds 1987; Frisch 1995). Observations suggested the existence of a "Local Bubble" – a low density cavity surrounded by dense neutral material lying roughly in the galactic plane filled with hot gas with a temperature of more than one million K (Snowden *et al.* 1998) that was the source of the observed diffuse soft X-ray background emission. The neutral gas boundary of this region was marked by a "wall" of dense gas at distances between approximately 65 and 250 pc of the Sun whose presence could be detected from strong Na I absorption detected in background stellar spectra of early-type stars (Sfeir *et al.* 1999; Lallement *et al.* 2003). More recently it was recognized that there were several warm (T$\sim$10,000 K) clouds in the bubble, and that the Solar System is itself embedded in a warm interstellar cloud. It was generally thought that there was no cold interstellar gas within this region of space (Lallement *et al.* 2003).

One interstellar cloud presented a puzzle for almost 3 decades. Originally identified by Verschuur (1969) as two roughly degree scale patches of very narrow H I 21 cm emission in the constellation Leo. This region was subsequently re–observed as part of the Millennium Arecibo 21 cm absorption line survey (Heiles & Troland 2003). Coupling new observations of 21 cm absorption toward 3C 225 and 3C 237 with the Leiden-Dwingeloo 21 cm sky survey (Hartmann & Burton 1997), Heiles & Troland (2003) suggested that this cloud had a very thin, ribbon–like geometry. Motivated by these results, Meyer *et al.* (2006) obtained observations of the the interstellar Na I λ5889, 5895Å absorption lines toward 33 stars at different distances in the direction of the Leo cloud using the Kitt Peak National Observatory 0.9 m coudé feed telescope. An extremely narrow absorption component was detected at the same velocity as the narrow 21 cm absorption detected toward 3C 225 in the spectra of 23 of the stars whose sky positions place them within the outer contour of the H I 21 cm emission. The closest of these stars had distances of $\sim$42 parsecs, placing this cloud well within the neutral gas 'edge" of the Local Bubble mapped by Lallement *et al.* (2003). While there were a number of warm, partially ionized clouds identified within the "Local Bubble" which was presumed to be full of hot gas (Redfield & Linksky 2004; 2008), this was the first indication of cold material within this region.

Peek *et al.* (2011) used optical observations of this Local Leo Cold Cloud (LLCC) to further constrain the distance to this material, placing it between 11 and 24.5 pc. Now such cold, neutral material will absorb X-ray emission, so if the soft X-rays observed around the Sun come from a pervasive hot gas filling the "Local Bubble" this cloud being well within the boundary should absorb the emission from the hot gas behind the cloud. In the direction of the LLCC the edge of the local cavity is between 100 and 150 pc away (Meyer *et al.* 2006). An analysis of *Rosat* All Sky-Survey 1/4 keV (C-band) data shows no shadowing of the X-ray flux in the direction of the LLCC, which suggests there is no diffuse, hot X-ray emitting bubble around the Sun (Peek *et al.* 2011). Instead Peek *et al.* (2011) suggested the source of the diffuse soft X-rays is charge exchange between the Solar Wind and the ambient interstellar medium (Cravens 1997; Cravens 2000; Wargelin *et al.* 2008). This is consistent with the model of Welsh & Shelton (2009), who suggested that much of the observed X-ray flux was due to Solar wind exchange. However Galeazzi *et al.* 2014 suggested that no more than 40% of the diffuse X-ray emission could come from Solar Wind change exchange, and that the remainder must come from the surrounding interstellar medium. However these authors did not address the constraints from the LLCC. Further observations of cold clouds will place tight constraints on the location and origin of the diffuse X-rays.

Meyer *et al.* (2012) used *Hubble Space Telescope* observations of absorption by interstellar C I in the spectra stars behind this cloud to directly measure the pressure in this cloud. The pressures of 40,000–80,000 cm^3K are much greater than that of the warm clouds in the Local Bubble, where Redfield & Linsky (2004) measured a mean thermal pressure to be 2300 cm^3K for 50 warm, partially ionized clouds within the Local Bubble. Measurements of the foreground X-ray emission from Solar Wind charge exchange (Robertson *et al.* 2010) and the Peek *et al.* (2011) finding of weak X-ray shadowing by the LLCC have weakened the case for a higher-pressure hot Local Bubble. Although some hot gas is undoubtedly present, its thermal pressure is most likely in a range (3000-7000 cm^3K) consistent with that of the warm clouds (Frisch *et al.* 2011). Thus it is clear that the LLCC is not in thermal pressure equilibrium with either the hot gas or the warm clouds in the Local Bubble. That the LLCC is significantly overpressured with respect to the Local Bubble is qualitatively consistent with predictions its anomalously low temperature is the result of a warm cloud collision (Vazquez-Semadeni *et al.* 2006; Meyer *et al.* 2006; Redfield & Linsky 2008). In their one-dimensional numerical simulations of the large-scale transonic (Mach number $\sim$1) compression of colliding warm gas flows, Vazquez-Semadeni *et al.* (2006) find that a thin cold layer forms within the shocked interface. A common feature of the colliding warm flow models (Audit & Hennebelle 2005; Gazol *et al.* 2005; Heitsch *et al.* 2006; Vazquez-Semadeni *et al.* 2006) is the turbulent fragmentation of the cold interface layer into clumpy structures. Some of the simulations (Audit & Hennebelle 2005; Gazol *et al.* 2005) have shown that localized regions within these structures can reach pressures up to 10^5 cm^3K even when the collision is transonic (Hennebelle *et al.* 2007). It is quite likely that the different LLCC pressures measured are due in part to real localized pressure variations, where the turbulent fragmentation expected in the wake of such collisions pushes the localized overpressures above the limiting values implied by the measured velocity constraints on the colliding clouds.

In any case, the geometry and physical properties of such clouds provide constraints on theoretical models of their formation.

Currently there is only one published example of very cold material within the Local Bubble, the LLCC, and it is difficult to draw any significant conclusions for the general processes which lead to the formation of cold clouds from a sample of one object. Using the Kitt Peak National Observatory Coudé Feed telescope combined with archival optical and radio observations we have attempted to identify additional local cold clouds. The survey of Lallement *et al.* (2003) identified several anomalous sightlines within the Local Bubble which showed some possible interstellar Na I absorption, though it was typically somewhat below the limits representative of the 'edge" of the Local Bubble. One sightline that drew immediate attention was that toward δ Cyg – while it showed weak Na I absorption (an equivalent width of only $\sim$20 mÅ, or about $\frac{1}{2}$ that seen toward the LLCC), it was notable for the narrowness of the absorption profile which shows hyper–fine splitting indicative of low temperatures and turbulent motions (Welty, Hobbs, & Kulkarni 1989; Lallement *et al.* 2003). Jenkins (2002) used C I observations from *HST*/GHRS to measure a pressure in this cloud of $10^3 < p/k < 10^4$ cm^{-3}K, more similar to that measured in the warm clouds in the Local Bubble. Using archival Elodie spectrograph observations we were able to confirm that there was a large region of Na I absorption in this region (Figure 1). This local Cygnus cold cloud (LCCC) thus represents a second of what may be a population of such cold material in the Solar vicinity (with other candidate clouds already known).

The question we seek to answer is does any of the material in the LCCC have the same high pressure as in the LLCC? The sightlines through the LLCC have a mean pressure of 60,000 cm^{-3}K Meyer *et al.* (2012), well in excess of the mean thermal pressure of 2300 cm^{-3}K for the warm, partially ionized clouds in the Local Bubble (Redfield & Linsky 2004). Such excess pressures are expected in models that involve collisions of warm clouds (see, for example, Vazquez–Semadeni *et al.* 2006). Flow–driven formation of cold atomic and molecular clouds in models have shown that shocked flows of warm interstellar gas can provide a rapid formation of such clouds (Ballesteros–Paredes *et al.* 1999; Hennebelle & Pérault 1999; Koyama & Inutsuka 2000; Hartmann *et al.* 2001). While the timescales of instabilities in flow driven models of cold atomic and molecular cloud formation suggest that thermal instabilities drive their rapid formation (Heitsch, Hartmann & Burkert 2008), and provide seeds for local gravitational collapse. Since this gas is significantly over–pressured one would expect it to have a relatively short lifetime, but a population of high pressure cold clouds argues for either some form of confinement or the rapid production of such clouds in the turbulent warm ISM. The measurent of a lower pressure toward δ Cyg (Jenkins 2002) suggests the gradual equillibration of such clouds with the ambient ISM, and support the idea that such clouds can become the seeds of the cold neutral medium. Pressure and density measurements in additional sightlines in the LCCC will determine if there is any variation in the pressure across the cloud, and the detection of high pressure gas like in the LLCC will point to turbulent fragmentation expected in the models. The measurement of the H I column (inferred from the Zn II column) will enable us to use X–ray shadowing to place limits on the location and amount of hot gas in this direction.

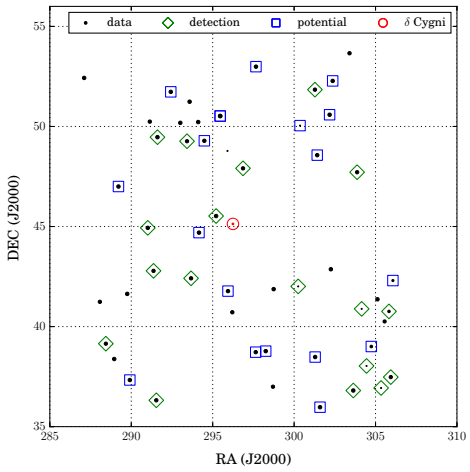

Figure 1: Positions of stars with detected Na I absorption behind the δ Cyg Cloud from observations taken from the Elodie spectrograph archive. The point size indicates the distance to the star, with small points showing distant stars and large points indicating nearby stars. Delta Cyg is marked by a circle.

References

Audit, E. & Hennebelle, P. 2005, A&A, 433, 1

Ballesteros-Paredes, J., Vázquez-Semadeni, E. & Scalo, J. 1999, ApJ, 515, 286

Cox, D. P. & Reynolds, R. J. 1987, ARA&A, 25, 303

Cravens, T. E. 1997, GeoRL, 24, 105

Cravens, T. E. 2000, ApJL, 532, L153

Frisch, P. C. 1995, SSRv, 72, 499

Frisch, P. C., Redfield, S. & Slavin, J. D. 2011, ARA&A, 49, 237

Gazol, A., Vázquez–Semadeni, E. & Kim, J. 2005, ApJ, 630, 911

Galeazzi, M., Chiao, M., Collier, M.R., Cravens, T., Kouttroumpa, D.. Lallement, R., Lepri, S. T., McCammon, D., Morgan, K., Porter, F. S., Robertson, I. P., Snowden, S. L., Thomas, N. E., Uprety, Y., Ursino, E. & Walsh, B. M. 2014, Nature, 512, 171

Hartmann, D. & Burton, W. B. 1997, "Atlas of Galactic Neutral Hydrogen" Cambridge Univ. Press (Cambridge)

Hartmann, L., Ballesteros-Paredes, J, & Bergin, E. A. 2001, ApJ, 562, 852

Heiles, C. & Troland, T. H. 2003, ApJ, 586, 1067

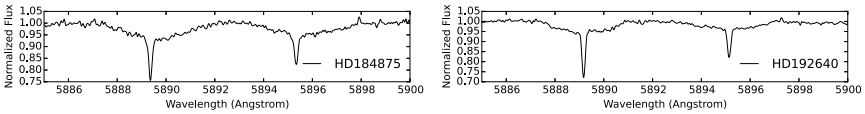

Figure 2: Archival Na I absorption spectra toward our selected stars behind the δ Cyg Cloud taken with the Elodie spectrograph

Heitsch, F., Slyz, A. D., Devriendt, J. E. G., Hartmann, L. W. & Burkert, A. 2006, ApJ, 648, 1052

Heitsch, F., Hartmann, L. W. & Burkert, A. 2008, ApJ, 683, 786

Hennebelle, P. & Pérault, M. 1999, A&A, 351, 309

Hennebelle, P., Audit, E. & Miville-Deschênes, M.-A. 2007, A&A, 465, 445

Jenkins, E. B. & Tripp, T. M. 2001, ApJS, 137, 297

Jenkins, E. B. 2002, ApJ, 580, 938

Lallement, R., Welsh, B. Y., Vergely, J. L., Crifo, F. & Sfeir, D. 2003, A&A, 411, 447

Koyama, H. & Inutsuka, S.-I. 2000, ApJ, 532, 980

Meyer, D. M., Lauroesch, J. T., Heiles, C., Peek, J. E. G. & Engelhorn, K. 2006,ApJ, 650, L67

Meyer, D. M., Lauroesch, J. T., Peek, J. E. G. & Heiles, C. 2012, ApJ, 752, 119

Peek, J. E. G., Heiles, C., Peek, K. M. G., Meyer, D. M., & Lauroesch, J. T. 2011, ApJ, 735, 129

Redfield, S. & Linsky, J. L. 2004, ApJ, 613, 1004

Redfield, S. & Linsky, J. L. 2008, ApJ, 673, 283

Robertson, I. P., Kuntz, K. D., Collier, M. R., Cravens, T. E. & Snowden, S. L. 2010, AIPC, 1216, 532

Sfeir, D. M., Lallement, R., Crifo, F. & Welsh, B. Y. 1999, A&A, 346, 785

Snowden, S. L., Egger, R., Finkbeiner, D. P., Freyberg, M. J., & Plucinsky, P. P. 1998, ApJ, 493, .715

Vázquez–Semadeni, E., Ryu, D., Passot, T., González, R. F. & Gazol, A. 2006, ApJ, 643, 245

Verschurr, G. L. 1969, Astrophys. Letters, 4, 85

Wargelin, B. J., Beiersdorfer, P., & Brown, G. V. 2008, CaJPh, 86, 151

Welsh, B. Y. & Shelton, R. L. 2009, Ap&SS, 323, 1

Welty, D. E., Hobbs, L. M. & Kulkarni, V. P. 1994, ApJ, 436, 152

Description of the Observations

We have selected our targets from a sample of stars observed with the Kitt Peak Coudé Feed telescope or the Elodie spectrograph at OHP in the vicinity of δCyg (Figure 1). Stars with suitably bright UV flux as well as rapid rotation were identified, and their Na I checked to be sure that the LCCC was detected but also that no more distant gas was seen (see Figure 2). For some of these stars we take advantage of one of the ND slits.

We target the C I fine structure lines as well as the Zn II absorption lines. By fitting the C I ground and fine–structure features as well as the (optical) hyper–fine split Na I absorption line one can determine the cloud temperature and turbulent velocity. Then the C I features allow one to determine the pressure and density within the cloud by using the excitation balance (Jenkins & Tripp 2001). The Zn II lines serve as a proxy for H I since in the Cygnus region there is too much background gas to identify H I absorption or emission associated with the LCCC. From the observations we can determine a cloud thickness using the inferred H I column density and the volume density determined from C I. We can then infer a mass for the LCCC.

We will observe the strong 1560Å multiplet of C I we will using E140H centered at 1489Å, and we will observe Zn II 2026, 2062Å using E230H centered at 2113Å. In addition we will cover other lines including Si IV 1393, 1402Å, Si II 1526Å, P II 1532Å, Ti II 1910Å, Cr II 2056, 2062, 2066Åwhich will provide a measure of depletion and sample the warm/hot interstellar medium. We selected the 1560Å multiplet of C I due to the large exposure times needed toe cover the 1656Å mulitplet, and the lack of FUV flux in these objects preventing the use of multiplets such as 1277 or 1328Å.

For HD 184875 we used the fluxes from the TD1 satellite, which are 1.7e-11 at 1565 Å, 2.4e-11 at 1965Å, and 1.8e-11 at 2365Å. We used the highest of these fluxes to check bright object protection, and then the lower fluxes to estimate the exposure time. This results in exposure times of 1,150 seconds for the 1489Å setting using the 0.2x0.09 slit and 900 seconds at 2113Å using the 31X0.05NDA.

For HD 192640 low resolution, large aperture IUE SWP spectra show a flux at 1656Å of 6E-12 erg/cm^2/s/Å, and a flux of no greater than 3E-11 erg/cm^2/s/Å anywhere in the 1763Å observation band. For the longer wavelength setting near 2113Å IUE low resolution, large aperture LWP observations show a flux of 3E-11 erg/cm^2/s/Å across the wavelength region of interest. These fluxes result in an exposure time of 5400 seconds at 1598Å using the 0.2x0.09 slit and at the 2113Å setting we use the 31X0.05NDA and have an exposure time of 715 seconds.

Estimating overheads of 6 minutes for guide acq (and 5 for re-acq), 6 minutes each for target acq and peakup, 8 minutes for 1st MAMA exposure. Thus the set-up time is a minimum of 26 minutes and we have 55 minutes of visibility for our targets. Thus HD 184875 will take a minimum of 2 orbits, and HD 192640 will take 3 orbits.

Special Requirements

None

Coordinated Observations

None

Justify Duplications

None

Past HST Usage

Programs in the last 4 *HST* cycles by Lauroesch:

HST GO 12191 - PI Lauroesch "Prospecting for Rare Elements in the Interstellar Medium", data acquisition completed, data analysis completed, publication in preparation.

HST SNAP 12192 - PI Lauroesch "A SNAPSHOT Survey of Interstellar Absorption Lines", data acquisition complete, publication detailing temporal variations toward HD 210809 in preparation. A second publication detailing limits on temporal variations toward repeated stars in planning.

Publications from previous *HST* programs:

Welty, D.E., Hobbs, L.M., **Lauroesch, J. T.**, Morton, D.C., & York, D.G. 1995, "Interstellar Lead", ApJ, 449, L149

Welty, D. E., **Lauroesch, J. T.**, Blades, J. C., Hobbs, L. M., & York, D. G. 1997, "Interstellar Abundances in the Magellanic Clouds. I. – GHRS Observations of the SMC Star Sk 108", ApJ, 489, 672

Welty, D. E., **Lauroesch, J. T.**, Blades, J. C., Hobbs, L. M., & York, D. G. 1997, "Interstellar Abundances in the Magellanic Clouds. I. – GHRS Observations of the SMC Star Sk 108", ApJ, 489, 672

Vanden Berk, D. E., **Lauroesch, J. T.**, Stoughton, C., Szalay, A. S., Koo, D. C., Crotts, S. P. S., Blades, J. C., Melott, A. L., Boyle, B. J., Broadhurst, T. J., & York, D. G. 1999, "Clustering Properties of Low-Redshift QSO Absorption Systems Towards the Galactic Poles", ApJS, 122, 355

Lauroesch, J. T., & Meyer, D. M. 1999, "Observations of Small-Scale Interstellar Structure in Dense Atomic Gas", ApJ, 519, L181

Welty, D.E., Hobbs, L. M., **Lauroesch, J. T.**, Morton, D. C., Spitzer, L., & York, D. G. 1999, "The Diffuse Interstellar Clouds toward 23 Orionis", ApJS, 124, 465

Lauroesch, J. T., Meyer, D. M., & Blades, J. C. 2000, "Evidence of Interstellar Na I Structure at Scales Down to 15 AU in Low-Density Gas", ApJ, 543, L43

Meyer, D. M., **Lauroesch, J. T.**, Sofia, U. J., Draine, B. T., & Bertoldi, F. 2001, "The Rich Ultraviolet Spectrum of Vibrationally Excited Interstellar H_2 toward HD 37903", ApJ, 553, L59

Welty, D. E., **Lauroesch, J. T.**, Blades, J. C. Hobbs, L. M., & York, D. G. 2001, "Unusual Depletions toward the SMC Star SK 155-Differences in Dust Composition in the SMC Interstellar Medium?", ApJ, 554, L75

Wang, Q. D., Immler, S., Walterbos, R., **Lauroesch, J. T.**, & Breitschwerdt, D. 2001, "Chandra Detection of a Hot Gaseous Corona around the Edge-on Galaxy NGC 4631", ApJ, 555, L99

Cartledge, S. I. B., Meyer, D. M., **Lauroesch, J. T.**, & Sofia, U. J. 2001, "Space Telescope Imaging Spectrograph Observations of Interstellar Oxygen and Krypton in Translucent Clouds", ApJ, 562, 394

Fox, A. J., Savage, B. D., Sembach, K. R., Fabian, D., Richter, P., Meyer, D. M., **Lauroesch, J. T.**, & Howk, J. Christopher 2003, "Origins of the Highly Ionized Gas along the Line of Sight Towards HD 116852", ApJ, 582, 793

Jenkins, E. B., Bowen, D. V., Tripp, T. M., Sembach, K. R., Leighly, K. M., Halpern, J. P., & **Lauroesch, J. T.** 2003, "Absorption-Line Systems and Galaxies in Front of the Second Brightest Quasar, PHL 1811" AJ, 125, 2824

Knauth, D. C., Howk, J. C., Sembach, K. R., **Lauroesch, J. T.**, & Meyer, D. M. 2003, "On the Origin of the High-Ionization Intermediate-Velocity Gas Toward HD 14434", ApJ, 592, 964

Cartledge, S. I. B., Meyer, D. M., & **Lauroesch, J. T.** 2003, "The Homogeneity of Interstellar Krypton in the Galactic Disk", ApJ, 597, 408

Sofia, U. J., **Lauroesch, J. T.**, Meyer, D. M., & Cartledge, S. I. B. 2004, "Interstellar Carbon in Translucent Sightlines", ApJ, 605, 272

Cartledge, S. I. B., **Lauroesch, J. T.**, Meyer, D. M., & Sofia, U. J. 2004, "The Homogeneity of Interstellar Oxygen in the Galactic Disk", ApJ, 613, 1037

Kulkarni, V. P., Fall, S. M., **Lauroesch, J. T.**, York, D. G., Khare, P., Truran, J. W., & Welty, D. E. 2005, "Element Abundances in Low-Redshift Damped Lyman-Alpha Galaxies", ApJ, 618, 68

Cartledge, S. I. B., **Lauroesch, J. T.**, Meyer, D. M., & Sofia, U. J. 2006, "The Homogeneity of Interstellar Elemental Abundances in the Galactic Disk", ApJ, 641, 327

Miller, A., **Lauroesch, J. T.**, Sofia, U. J., Cartledge, S. I. B., & Meyer, D. M. 2007, "Interstellar Iron and Silicon Depletions in Translucent Sight Lines", ApJ, 659, 441

Cartledge, S. I. B., **Lauroesch, J. T.**, Meyer, D. M., Sofia, U. J. & Clayton, G. C. 2008, "Interstellar Krypton Abundances: The Detection of Kiloparsec-scale Differences in Galactic Nucleosynthetic History", ApJ, 687, 1043

Hornbeck, J. B., Grady, C. A., Perrin, M. D., Wisniewski, J. P., Tofflemire, B. M., Brown, A., Holtzman, J. A., Arraki, K., Hamaguchi, K., Woodgate, B., Petre, R., Daly, B., Grogin, N. A., Bonfield, D. G., Williger, G. M. & **Lauroesch, J. T.** 2012, "PDS 144: The First Confirmed Herbig Ae – Herbig Ae Wide Binary", ApJ, 744, 54

Meyer, D. M., **Lauroesch, J. T.**, Peek, J. E. G. & Heiles, C. 2012, "The Remarkable High Pressure of the Local Leo Cold Cloud", ApJ, 752, 119

APPENDIX B

OBSERVATIONAL METADATA

The tables included in this chapter have correspondence with Tables 5.1 - 5.11 in Chapter 5. These tables contain the observational metadata from the Elodie archive spectra. Annotations and table order match those in Chapter 4.

TABLE B.1
Observational Metadata for Non-OB Measurable Targets – A

	Target [α]	File [β]		Date [γδ]	ExpTime [γ]	S/N [γϵ]
					$10^3 s$	
[1]	HD 332329	20040704	0017	2004-07-05 23:54:13	1.800	48.2
[2]	HD 192640	19990604	0032	1999-06-05 01:50:37	1.384	323.9
[3]	HD 185732	19990707	0016	1999-07-07 23:14:31	3.001	101.3
[4]	HD 184471	19940924	0009	1994-09-24 19:12:07	0.702	23.0
[5]	HD 191747	19990813	0024	1999-08-13 22:27:47	1.410	322.6
[6]	BD+314105	19940918	0013	1994-09-18 20:28:02	0.901	19.3
[7]	HD 184875	19960728	0014	1996-07-28 22:15:11	0.458	180.0
[8]	HD 184006	20050826	0013	2005-08-26 21:05:55	0.500	298.6
[9]	HD 182989	19960625	0040	1996-06-26 00:50:34	1.685	116.0
[10]	HD 190397	20060624	0017	2006-06-25 00:11:05	1.800	57.5
[11]	HD 197345	20040823	0040	2004-08-23 22:52:39	0.060	336.8
[12]	HD 172167A	19960502	0040	1996-05-03 03:03:07	0.062	483.8
[13]	HD 195692A	20010807	0022	2001-08-07 22:23:57	0.600	146.2
[14]	HD 188854A	20000721	0024	2000-07-21 22:46:41	1.801	145.2
[15]	HD 200723	19990813	0031	1999-08-14 00:54:29	2.812	297.6
[16]	HD 198343	20001005	0017	2000-10-05 20:54:15	3.001	162.1
[17]	HD 173977	20020920	0007	2002-09-20 18:49:18	7.201	193.2
[18]	HD 186155	19990810	0023	1999-08-10 21:44:50	1.306	298.3
[19]	BD+293448	19940924	0008	1994-09-24 18:56:43	0.701	16.6
[20]	HD 194093	19960823	0020	1996-08-23 22:12:43	0.120	513.5
[21]	HD 198226A	19990817	0020	1999-08-18 01:11:59	1.570	92.5
[22]	HD 190537A	20000720	0021	2000-07-21 00:14:39	0.901	147.0
[23]	HD 335198	20000709	0007	2000-07-09 22:27:17	4.501	46.9
[24]	HD 341233	20000805	0006	2000-08-05 20:15:56	3.601	51.1
[25]	BD+344217	20000908	0010	2000-09-08 21:32:39	4.501	56.5
[26]	HD 190508	20050805	0015	2005-08-05 23:34:58	1.200	32.7
[27]	BD+433754	20010727	0006	2001-07-27 21:20:26	4.501	36.6
[28]	HD 200019	20000804	0013	2000-08-04 22:01:55	1.201	113.8
[29]	HD 199939	20020622	0045	2002-06-23 00:53:25	0.481	78.6
[30]	HD 199178	19981124	0019	1998-11-24 17:49:33	1.501	121.7

[α] Numbers correspond to annotations in Figure 5.2 and Tables 5.1 - 5.2.
[β] Elodie file names given by "elodie:yyyymmdd/nnnn".
[γ] Metadata taken directly from FITS headers.
[δ] At the beginning of the exposure. Format is $yyyy-mm-dd\ hh:mm:ss$
[ϵ] As reported by the archive.

TABLE B.2
Observational Metadata for Non-OB Measurable Targets – B

	Target [α]	File [β]	Date [γδ]	ExpTime [γ]	S/N [γϵ]
				$10^3 s$	
[31]	HD 335070	20010825 0012	2001-08-25 21:57:14	4.501	54.4
[32]	HD 193891	20000706 0007	2000-07-06 20:53:03	1.201	105.9
[33]	HD 170527	20031103 0009	2003-11-03 18:07:57	2.700	166.4
[34]	HD 198149	19950911 0017	1995-09-11 21:23:23	0.300	405.2
[35]	BD+483149	20010731 0030	2001-08-01 00:27:20	1.501	84.0
[36]	HD 332698	20031004 0011	2003-10-04 20:05:47	5.563	78.1
[37]	HD 187372	20000809 0018	2000-08-09 22:12:40	0.601	182.9
[38]	HD 186619	20030703 0023	2003-07-03 23:23:47	0.300	76.5
[39]	HD 186532	20030703 0021	2003-07-03 23:08:41	0.360	61.1
[40]	HD 177808	20030703 0013	2003-07-03 21:39:20	0.360	79.5
[41]	HD 172381	20010802 0035	2001-08-03 00:01:44	0.421	52.2
[42]	HD 190964	20030703 0029	2003-07-04 00:17:46	0.300	54.3
[43]	HD 197939	20030703 0031	2003-07-04 00:45:06	0.300	47.2

[α] Numbers correspond to annotations in Figure 5.2 and Tables 5.1 - 5.2.
[β] Elodie file names given by "elodie:yyyymmdd/nnnn".
[γ] Metadata taken directly from FITS headers.
[δ] At the beginning of the exposure. Format is $yyyy-mm-dd\ hh:mm:ss$
[ϵ] As reported by the archive.

TABLE B.3
Observational Metadata for Non-OB No Absorption Targets – A

	Target [α]	File [β]		Date [γδ]	ExpTime [γ]	S/N [γϵ]
					$10^3 s$	
[1]	HD 335238	19940715	0017	1994-07-15 22:32:22	0.000	104.7
[2]	HD 196542A	19940918	0016	1994-09-18 21:14:04	0.901	27.9
[3]	HD 173648	19990809	0015	1999-08-09 20:56:26	0.785	325.9
[4]	HD 191158	20000722	0034	2000-07-23 01:18:42	0.600	75.6
[5]	HD 174567	20060531	0016	2006-06-01 00:04:02	3.602	92.7
[6]	HD 180583	19950908	0008	1995-09-08 19:35:27	1.801	257.6
[7]	HD 174912A	19961005	0007	1996-10-05 18:55:30	1.503	104.3
[8]	HD 181096	19970827	0019	1997-08-27 21:09:31	0.801	88.8
[9]	BD+443670	20000706	0009	2000-07-06 21:57:26	1.801	71.5
[10]	HD 183361A	20000717	0015	2000-07-17 23:39:45	1.201	137.3
[11]	HD 171635	20030326	0017	2003-03-27 04:01:33	0.301	158.7
[12]	HD 176051	20030709	0009	2003-07-09 21:42:21	0.900	226.8
[13]	HD 171485A	20030715	0011	2003-07-15 22:39:16	1.200	95.7
[14]	HD 195992	20030801	0024	2003-08-02 00:30:11	0.900	82.5
[15]	HD 195295	20031101	0011	2003-11-01 18:40:40	0.240	217.3
[16]	HD 185395	20031105	0021	2003-11-05 17:57:14	0.720	330.7
[17]	HD 180712	20040731	0013	2004-07-31 22:19:37	1.200	70.7
[18]	HD 195872	20040731	0017	2004-07-31 23:27:00	1.200	75.6
[19]	HD 173700	20041124	0011	2004-11-24 17:09:51	1.200	98.1
[20]	HD 192804	20050630	0014	2005-07-01 01:08:19	1.200	31.9
[21]	HD 172323A	20050711	0012	2005-07-12 00:56:40	1.520	55.7
[22]	HD 198084	20050714	0013	2005-07-15 00:31:14	0.500	152.2
[23]	HD 184960	20050716	0033	2005-07-17 01:40:47	0.900	170.5
[24]	HD 184448	20051023	0008	2005-10-23 19:18:51	1.801	67.5
[25]	HD 186408	19960414	0022	1996-04-15 03:18:37	0.900	160.5
[26]	HD 196850	19960830	0010	1996-08-30 23:00:04	0.901	122.2
[27]	HD 175306A	19961004	0008	1996-10-04 18:37:01	0.123	120.3
[28]	HD 199191	19970615	0019	1997-06-16 01:00:09	0.601	111.9
[29]	HD 185501A	19970721	0029	1997-07-22 00:46:27	0.901	120.3
[30]	HD 191022	19970722	0032	1997-07-23 01:32:51	0.901	101.3

[α] Numbers correspond to annotations in Figure 5.3 and Tables 5.3 - 5.4.
[β] Elodie file names given by "elodie:yyyymmdd/nnnn".
[γ] Metadata taken directly from FITS headers.
[δ] At the beginning of the exposure. Format is $yyyy-mm-dd\ hh:mm:ss$
[ϵ] As reported by the archive.

TABLE B.4
Observational Metadata for Non-OB No Absorption Targets – B

Target [α]	File [β]	Date [γδ]	ExpTime [γ]	S/N [γϵ]
			$10^3 s$	
[31] HD 191649	19970817 0032	1997-08-17 23:44:52	0.901	120.3
[32] HD 185414	19970818 0015	1997-08-18 22:21:58	0.901	178.1
[33] HD 187237	19970909 0021	1997-09-09 20:53:22	0.721	126.5
[34] HD 186427	19980808 0010	1998-08-08 21:03:54	0.601	156.2
[35] HD 188326A	19980808 0018	1998-08-08 23:24:01	0.901	119.3
[36] HD 187462	19980810 0015	1998-08-10 22:30:06	0.701	119.9
[37] HD 197310	19980810 0020	1998-08-11 00:12:43	0.900	111.5
[38] HD 195988	19980901 0018	1998-09-01 22:25:17	0.901	109.3
[39] HD 187123A	19990916 0036	1999-09-16 20:51:41	0.901	88.4
[40] HD 198483	19990926 0017	1999-09-26 20:21:45	0.300	46.4
[41] HD 198809	20000706 0015	2000-07-07 00:31:12	0.301	218.1
[42] HD 184499	20030710 0032	2003-07-11 23:56:30	0.900	136.1
[43] HD 188650	20031101 0010	2003-11-01 18:12:46	1.500	261.1
[44] HD 176377	20040706 0007	2004-07-06 23:25:16	1.805	186.2
[45] HD 197666	20040726 0029	2004-07-27 00:41:51	1.200	48.3
[46] HD 187792	20040729 0012	2004-07-29 22:33:34	1.200	79.7
[47] HD 187748	20040731 0014	2004-07-31 22:42:52	0.600	101.6
[48] HD 187876	20040731 0015	2004-07-31 22:58:00	1.200	87.5
[49] HD 190771	20040901 0012	2004-09-01 20:01:19	1.800	273.5
[50] HD 189749	20040927 0010	2004-09-27 20:53:50	1.200	63.7
[51] HD 170357	20041122 0017	2004-11-22 17:10:24	1.201	47.7
[52] HD 182758	20041123 0012	2004-11-23 17:14:01	1.200	57.5
[53] HD 181047	20050528 0021	2005-05-29 02:09:04	1.100	54.4
[54] HD 196361	20050803 0013	2005-08-03 23:25:06	1.200	28.2
[55] HD 172393	20050805 0010	2005-08-05 21:45:23	1.200	48.8
[56] HD 175441	20050913 0007	2005-09-13 19:23:26	1.500	64.6
[57] HD 185269	20060608 0028	2006-06-09 01:25:25	2.400	139.3
[58] HD 176670	19960902 0009	1996-09-02 20:35:23	0.601	227.7
[59] HD 188947	19970828 0014	1997-08-28 22:52:29	0.091	126.5
[60] HD 199870	20021028 0017	2002-10-28 21:03:10	0.902	220.8

[α] Numbers correspond to annotations in Figure 5.3 and Tables 5.3 - 5.4.
[β] Elodie file names given by "elodie:yyyymmdd/nnnn".
[γ] Metadata taken directly from FITS headers.
[δ] At the beginning of the exposure. Format is $yyyy-mm-dd\ hh:mm:ss$
[ϵ] As reported by the archive.

TABLE B.5
Observational Metadata for Non-OB Inconclusive Targets – A

Target	File [β]		Date [γδ]	ExpTime [γ]	S/N [γϵ]
				$10^3 s$	
BD+293427	19940918	0009	1994-09-18 19:16:24	0.801	18.7
HD 340577	19940918	0015	1994-09-18 20:52:00	0.901	27.6
HD 200405	19940918	0023	1994-09-18 23:35:40	0.703	26.6
HD 191742	19940919	0009	1994-09-19 20:01:26	1.203	26.0
HD 192678	19940924	0012	1994-09-24 20:01:23	0.603	51.3
HD 195725	19990809	0021	1999-08-09 22:40:21	1.004	316.4
HD 177196	19990813	0019	1999-08-13 20:54:11	1.600	303.4
HD 183534	20060531	0018	2006-06-01 01:10:57	4.501	154.1
BD+353616A	19940918	0010	1994-09-18 19:36:40	0.602	17.4
HD 341037	19940924	0015	1994-09-24 20:55:23	0.801	27.7
HD 197572	19960822	0011	1996-08-22 21:00:20	0.000	324.6
HD 198726	19960823	0027	1996-08-24 00:01:34	2.001	257.3
BD+423935	19970614	0020	1997-06-15 00:59:00	0.601	19.7
HD 238932A	19970617	0017	1997-06-17 23:51:12	0.301	18.9
BD+423607	19970822	0008	1997-08-22 20:57:07	3.602	59.7
HD 194951	19990706	0018	1999-07-07 02:29:49	0.900	81.7
HD 331319	19990709	0016	1999-07-10 01:06:21	5.401	98.1
HD 195069	20030625	0017	2003-06-26 00:15:07	7.202	560.0
HD 193370	20030725	0014	2003-07-26 00:30:27	0.660	241.1
HD 200805	20030729	0010	2003-07-30 00:13:00	4.501	152.7
HD 178593	20040801	0011	2004-08-01 22:23:03	1.506	103.7
HD 188307	20040802	0016	2004-08-03 23:55:22	1.200	28.6
HD 187253	20050722	0034	2005-07-23 00:41:41	1.200	31.8
HD 185239	20050913	0008	2005-09-13 19:55:05	1.500	75.3
HD 175225	19960503	0033	1996-05-04 02:09:59	0.601	216.8
HD 173701	19961023	0025	1996-10-23 18:00:10	0.901	133.0
HD 177153	19970820	0011	1997-08-20 20:55:14	0.901	132.8
HD 185351	19970823	0006	1997-08-23 20:29:39	0.630	267.0
BD+413931	19970823	0012	1997-08-23 22:30:41	3.601	68.2
HD 175306	19970827	0016	1997-08-27 20:02:16	0.601	24.4

[β] Elodie file names given by "elodie:yyyymmdd/nnnn".
[γ] Metadata taken directly from FITS headers.
[δ] At the beginning of the exposure. Format is $yyyy-mm-dd\ hh:mm:ss$
[ϵ] As reported by the archive.

TABLE B.6
Observational Metadata for Non-OB Inconclusive Targets – B

Target	File [β]	Date [γδ]	ExpTime [γ]	S/N [γϵ]
			$10^3 s$	
HD 182488	19980901 0009	1998-09-01 20:02:18	0.901	163.6
HD 175900	19990814 0014	1999-08-14 22:37:57	0.904	91.8
HD 193795	19990926 0015	1999-09-26 20:05:14	0.300	33.7
HD 200391	20000707 0007	2000-07-07 21:35:17	1.201	95.2
HD 181655	20000731 0015	2000-07-31 23:06:50	0.751	183.9
BD+364301	20000804 0016	2000-08-04 23:21:21	3.601	74.8
HD 179484A	20000813 0020	2000-08-13 22:27:28	1.801	73.8
HD 190360A	20000824 0023	2000-08-24 23:03:32	0.721	206.8
HD 178911A	20010402 0032	2001-04-03 02:59:15	1.201	107.3
HD 199598	20010721 0021	2001-07-22 00:46:46	0.901	151.9
HD 173605	20010728 0005	2001-07-28 20:09:56	1.501	127.5
HD 190228	20010811 0015	2001-08-11 22:12:07	0.901	125.9
HD 188326	20010812 0017	2001-08-12 22:25:24	1.501	112.7
HD 187123	20010814 0018	2001-08-14 22:52:25	1.001	96.4
HD 182736	20011010 0022	2001-10-10 21:42:19	0.901	116.6
HD 171242	20011011 0012	2001-10-11 19:32:37	0.901	75.5
HD 185657	20021028 0015	2002-10-28 20:24:06	1.343	200.9
HD 171007	20030521 0022	2003-05-22 00:49:22	1.801	42.3
HD 193216A	20030715 0013	2003-07-15 23:21:17	0.900	50.1
HD 178911B	20030717 0020	2003-07-18 00:47:04	0.900	117.0
HD 200102	20030725 0019	2003-07-26 02:02:02	3.001	258.1
HD 190403	20030727 0011	2003-07-27 23:26:14	3.601	156.8
HD 174104	20030729 0007	2003-07-29 20:08:57	5.001	122.4
HD 191010	20030729 0009	2003-07-29 22:53:58	4.201	149.7
HD 180161	20040710 0010	2004-07-11 00:43:52	2.401	129.8
HD 180683	20040726 0024	2004-07-26 22:57:15	1.200	50.4
HD 197037	20040726 0027	2004-07-27 23:53:04	1.200	75.0
HD 180502	20040728 0012	2004-07-28 22:20:08	1.200	80.1
HD 197488	20040728 0016	2004-07-29 23:58:53	1.200	79.0
HD 175425	20040926 0012	2004-09-26 19:17:56	1.200	57.1

[β] Elodie file names given by "elodie:yyyymmdd/nnnn".
[γ] Metadata taken directly from FITS headers.
[δ] At the beginning of the exposure. Format is $yyyy-mm-dd\ hh:mm:ss$
[ϵ] As reported by the archive.

TABLE B.7
Observational Metadata for Non-OB Inconclusive Targets – C

Target	File [β]	Date [γδ]	ExpTime [γ]	S/N [γϵ]
			$10^3 s$	
HD 193215	20041001 0010	2004-10-01 19:11:51	1.200	57.1
HD 172557	20041002 0009	2004-10-02 18:44:12	1.200	60.3
HD 184601	20041002 0011	2004-10-02 19:30:53	1.200	69.1
HD 199100	20041002 0013	2004-10-02 20:16:51	1.200	91.8
HD 195987	20041009 0008	2004-10-09 17:53:28	1.802	126.3
HD 187548	20050529 0016	2005-05-30 00:28:37	1.500	43.9
HD 177780	20050530 0015	2005-05-31 01:51:44	1.500	22.8
HD 190605	20050616 0029	2005-06-17 02:09:52	1.200	82.6
HD 178911	20050711 0011	2005-07-12 00:25:49	1.017	110.0
HD 178450	20050714 0009	2005-07-14 22:56:23	2.108	88.6
HD 173024	20050723 0015	2005-07-23 21:41:57	1.200	35.2
HD 173289	20050725 0022	2005-07-26 00:05:31	1.200	38.3
HD 197140	20050805 0016	2005-08-06 00:02:23	1.200	44.3
HD 197207	20050912 0014	2005-09-12 21:36:10	1.500	58.4
HD 189087	19961023 0026	1996-10-23 18:19:44	0.801	103.7
HD 234677	19970521 0013	1997-05-22 00:54:31	3.601	134.7
HD 197989	19970823 0010	1997-08-23 21:16:55	0.021	204.7
HD 183255	19970924 0011	1997-09-24 19:45:41	5.401	215.5
HD 191026	19971108 0009	1997-11-08 18:31:03	1.200	226.7
HD 338867	19980318 0029	1998-03-19 03:24:27	0.602	43.7
HD 199956	20000708 0006	2000-07-08 22:08:06	2.401	140.0
HD 334514	20000709 0006	2000-07-09 21:06:31	4.501	36.1
HD 199546	20010730 0018	2001-07-30 20:39:07	1.201	91.2
HD 189806	20010731 0024	2001-07-31 20:28:22	1.201	76.8
HD 181209	20010731 0025	2001-07-31 20:55:43	1.202	68.2
HD 188056	20010809 0028	2001-08-09 22:28:08	0.603	95.3
BD+333930	20010825 0011	2001-08-25 21:11:28	2.401	124.5
BD+334140	20010826 0009	2001-08-26 21:53:42	3.601	58.8
HD 192787	20021022 0008	2002-10-22 18:07:01	1.641	261.8
HD 196134	20021027 0014	2002-10-27 20:49:15	2.401	209.3

[β] Elodie file names given by "elodie:yyyymmdd/nnnn".
[γ] Metadata taken directly from FITS headers.
[δ] At the beginning of the exposure. Format is $yyyy-mm-dd\ hh:mm:ss$
[ϵ] As reported by the archive.

TABLE B.8
Observational Metadata for Non-OB Inconclusive Targets – D

Target	File [β]	Date [γδ]	ExpTime [γ]	S/N [γϵ]
			$10^3 s$	
HD 176408	20021028 0013	2002-10-28 19:57:56	0.900	274.0
HD 198550	20030711 0031	2003-07-12 00:47:33	0.900	59.4
HD 192910	20030725 0013	2003-07-26 00:14:37	0.360	152.3
HD 193469	20030726 0017	2003-07-27 02:15:11	2.101	188.3
HD 198794	20030727 0012	2003-07-28 00:32:39	3.600	114.6
HD 200560	20031102 0010	2003-11-02 19:16:11	3.602	139.7
HD 190470	20040712 0012	2004-07-13 01:33:42	3.300	70.5
BD+413306	20040819 0017	2004-08-19 21:05:36	3.600	85.2
BD+522815	20040902 0009	2004-09-02 20:41:52	5.401	95.9
HD 332518	20040907 0008	2004-09-07 20:43:30	6.301	84.6
HD 331093	20040926 0014	2004-09-26 20:05:21	1.200	36.8
HD 191806	20040928 0008	2004-09-28 20:18:13	1.200	66.8
HD 171607	20041002 0008	2004-10-02 18:21:04	1.200	70.1
HD 179094	20050625 0022	2005-06-26 02:16:58	0.600	118.5
HD 198425	20050703 0019	2005-07-04 01:25:28	1.200	51.0
HD 188753	20051025 0006	2005-10-25 18:54:47	1.801	111.1
BD+400883	19940928 0015	1994-09-28 19:51:46	3.601	35.1
HD 173739	19970524 0023	1997-05-23 22:20:39	2.701	109.2
GJ 0747	19970812 0018	1997-08-12 21:10:39	2.402	29.2
GJ 0809	19980723 0020	1998-07-24 00:35:53	1.201	75.4
HD 186686	19980805 0018	1998-08-05 23:18:11	1.801	204.8
GJ 0725	19990702 0012	1999-07-02 23:44:29	1.201	63.0
BD+394208	19990821 0014	1999-08-21 21:24:26	5.952	164.2
BD+364025	19990823 0014	1999-08-23 20:26:11	7.201	152.9
HD 182190	20000809 0017	2000-08-09 21:57:55	0.601	190.7
HD 199871	20000809 0021	2000-08-09 22:55:12	0.900	131.3
HD 186776	20000810 0014	2000-08-10 22:09:23	0.601	141.0
HD 175865	20010809 0025	2001-08-09 21:36:41	0.601	255.7
HD 175588	20010810 0013	2001-08-10 21:21:48	0.181	93.1
HD 184786	20010812 0014	2001-08-12 21:46:43	0.361	121.1

[β] Elodie file names given by "elodie:yyyymmdd/nnnn".
[γ] Metadata taken directly from FITS headers.
[δ] At the beginning of the exposure. Format is $yyyy-mm-dd\ hh:mm:ss$
[ϵ] As reported by the archive.

TABLE B.9
Observational Metadata for Non-OB Inconclusive Targets – E

Target	File [β]	Date [γδ]	ExpTime [γ]	S/N [γϵ]
			$10^3 s$	
GJ 0815	20010902 0018	2001-09-02 21:36:39	1.201	29.7
GJ 0766	20011030 0010	2001-10-30 19:09:25	3.601	12.7
HD 189063	20020618 0027	2002-06-19 01:53:06	0.482	79.8
HD 178003	20030703 0014	2003-07-03 21:49:54	0.360	49.3
HD 180450	20030703 0017	2003-07-03 22:23:43	0.360	45.2
HD 179869	20030703 0018	2003-07-03 22:34:21	0.401	52.3
HD 187849	20030703 0024	2003-07-03 23:32:47	0.300	98.5
HD 190544	20030703 0026	2003-07-03 23:52:15	0.300	88.8
HD 199305	20040818 0013	2004-08-18 22:22:13	3.601	81.3
HD 173740	20040822 0011	2004-08-22 21:34:21	7.201	104.9
HD 331161	19940925 0014	1994-09-25 20:14:54	3.601	17.6
HD 190163	20000809 0020	2000-08-09 22:34:45	0.900	67.4
HD 182917	20030703 0019	2003-07-03 22:44:36	0.500	50.6
GJ 0802	20050922 0012	2005-09-22 21:11:31	3.601	4.9

[β] Elodie file names given by "elodie:yyyymmdd/nnnn".
[γ] Metadata taken directly from FITS headers.
[δ] At the beginning of the exposure. Format is $yyyy-mm-dd\ hh:mm:ss$
[ϵ] As reported by the archive.

TABLE B.10
Observational Metadata for OB Targets – A

Target	File [β]	Date [γδ]	ExpTime [γ]	S/N [γϵ]
			$10^3 s$	
HD 188209	19970917 0012	1997-09-17 21:36:34	1.803	417.5
HD 198820	20041110 0019	2004-11-10 19:10:20	3.601	161.1
HD 194335	19970921 0027	1997-09-21 23:04:10	1.201	243.6
HD 200120	20030808 0014	2003-08-08 22:09:33	1.200	376.4
HD 182255	20031106 0018	2003-11-06 17:37:00	3.601	512.2
HD 338529	19970822 0009	1997-08-22 22:04:28	3.601	83.9
HD 183056	20000519 0019	2000-05-20 01:15:20	1.801	268.1
HD 174638	20010906 0025	2001-09-06 21:22:35	0.241	249.7
HD 176437	19960425 0031	1996-04-26 02:27:27	0.301	64.5
HD 180163	19981128 0015	1998-11-28 18:33:38	0.090	114.3
HD 192685	20030819 0020	2003-08-19 22:17:13	0.901	371.6
HD 198183	20040823 0041	2004-08-23 23:05:39	0.600	85.9
BD+404124	19970707 0008	1997-07-07 22:02:19	5.401	62.9
HD 194279	19970919 0021	1997-09-19 22:39:34	3.601	296.8
HD 172324	20001008 0011	2000-10-08 19:51:41	6.001	152.9
BD+413731	19970708 0011	1997-07-08 22:27:27	3.601	93.6
HD 193322A	20050817 0007	2005-08-17 19:33:11	1.200	54.6
HD 193322	20040829 0016	2004-08-29 23:18:09	1.200	189.4
HD 195592	19970918 0029	1997-09-18 20:52:24	3.601	322.8
HD 190864	20010812 0018	2001-08-12 22:56:35	2.402	144.2
HD 191423	20040829 0013	2004-08-29 19:39:37	5.401	150.5
HD 189957	20040827 0014	2004-08-27 19:38:55	2.701	137.0
HD 192639	20010811 0018	2001-08-11 22:49:26	0.721	129.6
HD 191978	20040829 0014	2004-08-29 21:14:58	5.401	125.7
HD 190429	20040827 0015	2004-08-27 20:39:05	1.500	174.4
HD 199579	19970921 0016	1997-09-21 20:49:31	1.508	248.1
HD 193514	20040828 0008	2004-08-29 00:04:36	2.401	129.4
HD 190429A	20051106 0008	2005-11-06 18:38:01	3.601	154.9
HD 186980	20010811 0014	2001-08-11 21:43:19	1.501	136.0
HD 193443	20040828 0007	2004-08-28 23:08:18	1.800	157.5

[β] Elodie file names given by "elodie:yyyymmdd/nnnn".
[γ] Metadata taken directly from FITS headers.
[δ] At the beginning of the exposure. Format is $yyyy-mm-dd\ hh:mm:ss$
[ϵ] As reported by the archive.

TABLE B.11
Observational Metadata for OB Targets – B

Target	File [β]	Date [γδ]	ExpTime [γ]	S/N [γϵ]
			$10^3 s$	
HD 191612	20040828 0003	2004-08-28 19:16:22	3.601	152.4
HD 192281	20040828 0005	2004-08-28 21:33:21	2.701	153.0
HD 191201	20040827 0017	2004-08-27 21:43:10	1.600	142.4
HD 193237	19980614 0031	1998-06-15 01:02:40	1.800	250.5
HD 198478	20040824 0038	2004-08-24 21:44:20	0.900	158.0
HD 228712	19970917 0011	1997-09-17 19:28:57	7.201	226.3
HD 194839	19970919 0008	1997-09-19 18:52:58	5.401	279.9

[β] Elodie file names given by "elodie:yyyymmdd/nnnn".
[γ] Metadata taken directly from FITS headers.
[δ] At the beginning of the exposure. Format is $yyyy-mm-dd\ hh:mm:ss$
[ϵ] As reported by the archive.

APPENDIX C

SOURCE CODE

The following appendix contains most of the relevant source code for the **SLiPy** library. Each *module* is included in its own section for which the name reflects the file structure to that module. Some modules are not included for practical reasons. The code is self documenting using Python's *docstring* functionality (accounting for some of its length). The software can be downloaded from its *GitHub* repository at *http://github.com/glentner/slipy* or from its website at *http://glentner.github.io/slipy*. For interactive, human readable documentation visit the website. The code can be downloaded via several formats including *zip* and *tar.gz* or can be *cloned* directly to your desktop.

The **LICENSE** and **README** file have been omitted. This content can be accessed online. Also, the **AtomicData** module has been omitted intentionally. It would occupy up to a hundred or more pages of raw data in the form of a Python list and serves no benefit to be included here. Neither are any of the files from the **SLiPy.Data.Archives** included.

C.1 .. __init__

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# SLiPy/__init__.py
"""
SLiPy - A Spectroscopy and astrophysics Library for Python 3

This Python package is an expanding code base for doing computational
astronomy, particularly spectroscopy. It contains both a *Spectrum* class
for handling spectra as objects (with +, -, \*, /, etc... operations defined)
and a growing suite of analysis tools.
"""

# base exception class for whole project, module exception classes will
# be derived from here.
class SlipyError(Exception):
    pass

# exposed modules
from .SLiPy import Fits, Simbad, Correlate, Telluric, Velocity, \
    Observatory, Montage, Plot, Spectrum, Measure, Profile

# elevate `Spectrum` to the package level
from .SLiPy.Spectrum import Spectrum
```

C.2 .. Algorithms . Functions

```
# Copyright (c) Geoffrey Lentner 2015. All Right Reserved.
# See LICENSE (GPLv3)
# slipy/Algorithms/Functions.py
"""
Analytical functions used in SLiPy.
"""

import numpy as np
from scipy.special import wofz as w # Faddeeva function

# No FunctionsError implimented yet

def Gaussian(x, *params):
    """
    The Gaussian function. The function template has `*params` to work with
    scipy...curve_fit; the user *must* specify `A, mu, sigma = params`.
    """
    A, mu, sigma = params
    return A * np.exp( -0.5 * (x - mu)**2 / sigma**2 )

def NormalizedGaussian(x, *params):
    """
    The normalized Gaussian function.
    """
    mu, sigma = params
    return np.exp( -0.5 * (x - mu)**2 / sigma**2 ) / (sigma *
        np.sqrt(2 * np.pi))

def InvertedGaussian(x, *params):
    """
    An inverted Gaussian function (i.e., 1 - Guassian()). The function
    template has `*params` to work with scipy...curve_fit; the user *must*
    specify `A, mu, sigma = params`.
    """
    return 1 - Gaussian(x, *params)

def Lorentzian(x, *params):
    """
    The Lorentzian function. Typically used to describe an absorption line profile.
    """
    x0, gamma = params
    return 1 / ((2 * (x - x0) / gamma)**2 + 1)

def NormalizedLorentzian(x, *params):
    """
    The Lorentzian function, normalized.
    """
    x0, gamma = params
    return 2 * Lorentzian(x, *params) / (np.pi * gamma)

def NormalizedVoigt(x, *params):
    """
    The Voigt function. The result of the convolution of a Lorentzian with one or more
    Gaussians. Often used to describe an intrinsically Lorentzian absorption profile blurred
    by Gaussian broadening from thermal motions and turbulance and another Gaussian instrument
    profile. The result is also the real part of the Faddevva function (the complex probability
    function), implemented by Scipy as wofz in scipy.special

    This returns a normalized profile. The name `NormalizedVoigt` is to keep the convention
    for the rest of this module. The amplitude controlling Voigt profile `Voigt` evaluates
    this function first for `x - x0 = 0` to empirically `un-normalize` it before scaling by the
    requested amplitude

    Reference:
        Olivero, J.J; R.L. Longbothum. ``Empirical fits to the Voigt line width:
        A brief review''. Journal of Quantitative Spectroscopy and Radiative Transfer. Vol 17.
        Issue 2. pages 223-236. Feb 1977

    Parameters: `x0, sigma, gamma = params`

        x0    -> center of the profile
        sigma -> the Gaussian width
        gamma -> the Lorentzian width
    """
    x0, sigma, gamma = params
    return w( ((x-x0) + 1j*gamma) / (sigma * np.sqrt(np.pi)) ).real / (
        sigma * np.sqrt(2 * np.pi) )

def Voigt(x, *params):
    """
    The Voigt line profile. See ..Algorithms.Function.NormalizedVoigt for more information.
    This function returns an amplitude controlled Voigt profile.

    Parameters: `A, x0, sigma, gamma = params`

        A     -> amplitude of the profile
        x0    -> center of the profile
        sigma -> the Gaussian width
        gamma -> the Lorentzian width
```

```
90          """
91          return params[0] * NormalizedVoigt(x, *params[1:]) / NormalizedVoigt(0, 0, *params[2:])
92
93
94      def InvertedLorentzian(x, *params):
95          """
96          An inverted Lorentzian (i.e, A - Lorentzian()).
97          """
98          A, x0, gamma = params
99
100         return A - Lorentzian(x, x0, gamma)
```

C.3 .. Algorithms . KernelFit

```
# Copyright (c) Geoffrey Lentner 2015. All Right Reserved.
# See LICENSE (GPLv3)
# slipy/Algorithms/KernelFit.py
"""
Non-parametric Kernel Regression.
"""

import numpy as np

from .. import SlipyError
from ..Framework.Options import Options, OptionsError

from .Functions import Gaussian

class KernelFitError(SlipyError):
    """
    Exception specific to the KernelFit module
    """

class KernelFit1D():
    """
    One dimensional kernel regression.
    """
    def __init__(self, x, y, kernel = Gaussian, **kwargs):
        """
        Keep the input `x` and `y` arrays. Define parameters.
        The default kernel function is the Guassian. Alternatives should
        have the same signature as slipy.Algorithms.Functions.Gaussian
        """
        if not hasattr(x, '__iter__') or not hasattr(y, '__iter__'):
            raise KernelFitError('Both `x` and `y` are expected to '
            'be array-like!')

        if len(np.shape(x)) != 1 or len(np.shape(y)) != 1:
            raise KernelFitError('Both `x` and `y` are expected to '
            'be one-dimensional objects!')

        if len(x) != len(y):
            raise KernelFitError('Both `x` and `y` should be of the '
            'same length!')

        if not hasattr(x, 'copy') or not hasattr(y, 'copy'):
            raise KernelFitError('Both `x` and `y` should have a '
            'copy() method implimented!')

        if type(x) is not type(y):
            raise KernelFitError('Both `x` and `y` are expected to '
            'be of the same type!')

        if hasattr(x, 'unit') and not hasattr(y, 'unit'):
            raise KernelFitError('The `x` array given has units but the `y` '
            'array doesn`t!, maybe give `y` u.dimensionless_unscaled?')

        if hasattr(y, 'unit') and not hasattr(x, 'unit'):
            raise KernelFitError('The `y` array given has units but the `x` '
            'array doesn`t!, maybe give `x` u.dimensionless_unscaled?')

        # the default `bandwidth` is 1/10 the domain of `x`, this
        # is likely to be a bad choice!
        bandwidth = 0.1 * (x.max() - x.min())

        try:

            options = Options( kwargs, {

                'bandwidth' : bandwidth # for the kernel function
            })

            self.kernel    = kernel
            self.bandwidth = options('bandwidth') # not necessarily with units

        except OptionsError as err:
            print(' --> OptionsError:', err)
            raise KernelFitError('Unrecognized option for KernelFit1D()!')

        self.x = x.copy()
        self.y = y.copy()

    def mean(self, x):
        """
        Solve for smooth profile through the data on the new `x` array.
        This is essentially a weighted mean.
        """

        if not hasattr(x, '__iter__'):
            raise KernelFitError('`x` is expected to be array-like!')

        if not hasattr(x, 'copy'):
            raise KernelFitError('`x` is expected to have a copy() method!')
```

```
90
91          if hasattr(x, 'unit') and not hasattr(self.x, 'unit'):
92              raise KernelFitError('The provided array has units but the '
93              'original domain does not!')
94
95          if hasattr(self.x, 'unit') and not hasattr(x, 'unit'):
96              raise KernelFitError('The provided array does not have units '
97              'but the original domain did!')
98
99          # copy `x` such that the return type is the same as the input type
100         y = x.copy()
101
102         # fix units though
103         if hasattr(x, 'unit'):
104             y = y.value * self.y.unit
105
106         for a, point in enumerate(x):
107
108             weights = Gaussian(point, 1, self.x, self.bandwidth)
109
110             # weights should be dimensionless
111             if hasattr(x, 'unit'):
112                 weights = weights.decompose()
113
114             args = np.where(np.isfinite(self.y))
115             y[a] = np.sum(weights[args] * self.y[args]) / np.sum(weights[args])
116
117         return y
```

C.4 .. Data . Atomic

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# slipy/Data/Atomic.py

"""
Access methods for the atomic data published by Donald C. Morton (2003).
See .Archives.AtomicData.MortonTable
"""

import numpy as np
from astropy import units as u

from .. import SlipyError
from ..Framework.Options import Options, OptionsError
from .Archives import AtomicData

class AtomicError(SlipyError):
    """
    Exception specific for the IonManager class.
    """
    pass

class IonManager:
    """
    Managing class for the atomic data (Morton 2003). See SLiPy.Data.Archives.AtomicData.
    """
    def __init__(self):
        """
        Imports the atomic data and creates the dictionary.
        """
        self.data = AtomicData.MortonTable

        # march through the table and apply units, compute oscillator strengths
        for a, entry in enumerate(self.data):

            AirWave = None if not entry[0] else entry[0] * u.Angstrom
            VacWave = None if not entry[1] else entry[1] * u.Angstrom # shouldn't happen?
            Ion     = entry[2]

            # conversion from 'cm-1' (Mohr & Taylor 2000)
            ELow    = entry[3] * 1.23984186 * u.eV

            # solve for oscillator strengths
            Logwf   = entry[4]
            os      = None if not Logwf else 10**(Logwf) / VacWave.value

            self.data[a] = [AirWave, VacWave, Ion, ELow, Logwf, os]

        # build dictionary by ion name
        self.ions = { ion : [] for ion in set([entry[2] for entry in self.data]) }
        for entry in self.data:
            self.ions[ entry[2] ].append( entry[:2] + entry[3:] )

    def __call__(self, key, **kwargs):
        """
        Retrieve data from the Archives.AtomicData table. If the `key` is a string
        type it is expected to be the name of an ion (e.g., 'C III'). If the `key` is
        a number it is expected to be a wavelength value (if not with units Angstroms are
        implied). The default is Vacuum wavelength, but Air can be specified with the
        keyword argument `wavelength='Air'`.

        If the key was the name of an ion, all the lines for that ion are returned. If the
        key was a wavelength, the closest line in the table to that wavelength is returned.
        You can request a wavelength range by giving the `key` as a tuple of two wavelengths
        specifying the range.

        The results default to the f-value (a.k.a. the oscillator strength) but can be
        changed with the keyword argument `entry`. Options include, `Air`, `Vacuum`, `Ion`,
        `ELow`, `LOGWF`, and `fvalue`.

        The if either a single pair or a list of pairs: the first element of each pair is
        always a wavelength value (in Air if wavelength='Air' or in Vacuum otherwise), the
        second being the entries requested. The wavelength type is always that used for
        the look-up. That is, Vacuum by default, but if `wavelength='Air'` is given, the
        returns will be in Air wavelengths. Be aware that `None` might be returned if
        there is not Air wavelength for the line. Further, all results will be returned
        as `None` where no data is available.
        """
        try:

            options = Options( kwargs, {

                    'wavelength' : 'vacuum', # alternative is `air`
                    'lookup'     : 'fvalue'  # others: `air`, `vacuum`, `ion`, `elow`, `logwf`
                })

            wavelength = options('wavelength')
            lookup     = options('lookup')

```

```
90            except OptionsError as err:
91                print(' --> OptionsError:', err)
92                raise AtomicError('Failed keyword assignment from __call__ to IonManager!')
93
94            if isinstance(key, tuple):
95
96                if len(key) != 2:
97                    raise AtomicError('tuples expected to be length 2 on __call__ to '
98                    'IonManager!')
99
100               table = self.Between(*key)
101
102           else:
103
104               table = self.__getitem__(key)
105
106           # map entries to index value
107           lookup_options = { 'air': 0, 'vacuum': 1, 'ion': 2, 'elow': 3, 'logwf': 4, 'fvalue' : 5 }
108
109           if lookup not in lookup_options:
110               raise AtomicError('`{}` is not an available search option!'.format(lookup))
111
112           if wavelength not in ['air', 'vacuum']:
113               raise AtomicError('Only `air` and `vacuum` wavelengths are understood!')
114
115           if isinstance(key, str):
116
117               # alter the index dictionary for column changes
118               lookup_options = { 'air': 0, 'vacuum': 1, 'elow': 2, 'logwf': 3, 'fvalue' : 4 }
119
120               if lookup == 'ion':
121                   # `Ion` column won't be present in the returned `table` !!!
122                   raise AtomicError('You provided the name of an ion but requested the names '
123                   'of the ions as the return value!')
124
125           if not isinstance(key, tuple) and not isinstance(key, str):
126               # assume this was a single wavelength value, `table` is a single line
127               return tuple([ table[ lookup_options[wavelength] ], table[ lookup_options[lookup] ] ])
128
129           return [# return pairs of wavelength and `lookup` for each `line` found
130                   tuple([ line[ lookup_options[wavelength] ], line[ lookup_options[lookup] ] ])
131                   for line in table
132               ]
133
134       def __getitem__(self, index):
135           """
136           Access methods.
137
138           If the `index` is a string value giving the name of a particular ion, this method
139           returns the entries in the data for that ion.
140
141           If the `index` is a numerical value, this method returns the table entry closest
142           in wavelength (vacuum).
143
144           If the `index` is a slice object (e.g., `5850:5950`) it returns the segment of the
145           table data on that range. A `step` is not acceptable (e.g., `::0.5`).
146           """
147           if isinstance(index, slice):
148
149               start, stop, step = index.start, index.stop, index.step
150
151               if step: raise AtomicError('You cannot slice the table with a `step` size!')
152
153               if not start:
154                   # the first vacuum wavelength in the table
155                   start = self.data[0][1]
156
157               if not stop:
158                   # the last vacuum wavelength in the table
159                   stop = self.data[-1][1]
160
161               return self.Between(start, stop)
162
163           elif isinstance(index, str):
164
165               if index not in self.ions:
166                   raise AtomicError('`{}` is not a recognized or available ion in the '
167                   'data!'.format(index))
168
169               return self.ions[index]
170
171           else:
172
173               if hasattr(index, 'unit'):
174                   index = index.to(u.Angstrom).value
175
176               proximity = (np.array([entry[1].value for entry in self.data]) - index)**2
177
178               return self.data[ proximity.argmin() ]
179
180       def Below(self, wavelength):
181           """
182           Return all table entries below the given wavelength. If units are not given,
```

```
183             Angstroms are implied.
184             """
185             if not hasattr(wavelength, 'unit'):
186                 wavelength *= u.Angstrom
187 
188             if wavelength > self.data[-1][1] or wavelength < self.data[0][1]:
189                 raise AtomicError('Cannot access wavelengths outside the data available!')
190 
191             return [ entry for entry in self.data if entry[1] < wavelength ]
192 
193         def Above(self, wavelength):
194             """
195             Return all table entries below the given wavelength. If units are not given,
196             Angstroms are implied.
197             """
198             if not hasattr(wavelength, 'unit'):
199                 wavelength *= u.Angstrom
200 
201             if wavelength > self.data[-1][1] or wavelength < self.data[0][1]:
202                 raise AtomicError('Cannot access wavelengths outside the data available!')
203 
204             return [ entry for entry in self.data if entry[1] > wavelength ]
205 
206         def Between(self, wavelengthA, wavelengthB):
207             """
208             Return all table entries between the given wavelengths. If units are not given,
209             Angstroms are implied.
210             """
211             if not hasattr(wavelengthA, 'unit'):
212                 wavelengthA *= u.Angstrom
213 
214             if not hasattr(wavelengthB, 'unit'):
215                 wavelengthB *= u.Angstrom
216 
217             if wavelengthA > self.data[-1][1] or wavelengthA < self.data[0][1]:
218                 raise AtomicError('Cannot access wavelengths outside the data available!')
219 
220             if wavelengthB > self.data[-1][1] or wavelengthB < self.data[0][1]:
221                 raise AtomicError('Cannot access wavelengths outside the data available!')
222 
223             if wavelengthA > wavelengthB:
224                 # that doesn't make sense, switch the order
225                 wavelengthA, WavelengthB = WavelengthB, WavelengthA
226 
227             return [ entry for entry in self.Below(wavelengthB) if entry[1] > wavelengthA ]
228 
229 
230     # create a static instance of the IonManager class.
231     IonSearch = IonManager()
232 
233     class Ion():
234         """
235         An object for declaring atomic ions.
236 
237         An `Ion` should have a name, wavelength (air or vacuum), osciallator strength,
238         and a transition probability (Einstein coefficient).
239 
240         Much of this data is available from the ..Data.Archives.AtomicData module searchable
241         with the ..Data.Atomic.IonManager class. In the future, we hope to impliment an
242         internal database of transition probabilities for each ion in the AtomicData module
243         from the Morton 2003 publication. Currently, the user must provide this data.
244 
245         The NIST Atomic Spectra Database Lines Data is a good source for atomic data values:
246             http://physics.nist.gov/PhysRefData/ASD/lines_form.html
247         """
248         def __init__(self, name=None, wavelength=None, fvalue=None, A=None, **kwargs):
249             """
250             Create a new `Ion`. If no arguments are given the state remains uninitialized.
251             The `name` (e.g., 'Ca III') is used to connect with the data in the
252             ..Data.Archives.AtomicData module via the IonManager class. The `wavelength` need
253             not necessarily be the exact value of the line; the line closest to that given
254             for the `name`d ion is used. This is by default the wavelength in vacuum, but can
255             be the wavelength in air if the keyword argument `medium='air'` is given.
256             The created member attribute `wavelength` and `fvalue` are automatically assigned
257             based on this procedure, but can be explicitely assigned if the `fvalue` is
258             provided directly. The transition probability (Einstein coefficient) `A` simply
259             attached as `A`. If not units are given for `A`, `s-1` is assigned.
260             """
261             try:
262 
263                 options = Options( kwargs, {
264 
265                         'medium' : 'vacuum' # alternative is 'air'
266                     })
267 
268                 self.medium = options('medium')
269 
270                 if self.medium not in ['vacuum', 'air']:
271                     raise AtomicError('From Ion.__init__(), the only allowed values for `medium` '
272                     'are `vacuum` and `air`!')
273 
274             except OptionsError as err:
```

```
275                 print(' --> OptionsError:', err)
276                 raise AtomicError('From Ion.__init__(), that was not an exceptable keyword assignment '
277                 'for an `Ion`!')
278 
279             if not name or not wavelength or fvalue:
280                 # simply assign whatever is provided.
281                 self.name       = name
282                 self.wavelength = wavelength
283                 self.fvalue     = fvalue
284                 self.A          = A
285 
286             else:
287 
288                 if not hasattr(wavelength, 'unit'):
289                     wavelength *= u.Angstrom
290 
291                 # lookup the ion using IonManager (already created as IonSearch)
292                 found_ions  = IonSearch(name, wavelength=self.medium)
293                 wavelengths = np.array([(wavelength - entry[0]).value for entry in found_ions])**2
294                 closest_ion = wavelengths.argmin()
295 
296                 # assign the closest wavelength to that avaiable
297                 self.wavelength = found_ions[ closest_ion ][0].to(wavelength.unit)
298                 self.fvalue     = found_ions[ closest_ion ][1] * u.dimensionless_unscaled
299 
300                 self.name = name
301 
302                 if A and not hasattr(A, 'unit'):
303                     A /= u.s
304 
305                 self.A    = A
306 
307         def __str__(self):
308             """
309             Show the available information for this Ion.
310             """
311             return ('Ion:        {}\n'
312                     'Wavelength: {}\n'
313                     'fvalue:     {}\n'
314                     'A:          {}\n'.format(self.name, self.wavelength, self.fvalue, self.A))
315 
316         def __repr__(self):
317             """
318             The same as the `str` representation.
319             """
320             return str(self)
```

C.5 .. Data . Elodie

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# AstroPython/Data/Elodie.py
"""
Methods for data retrieval from the Elodie Archive.
"""

import os, shutil, numpy as np
from urllib.request import urlopen

from .. import SlipyError
from ..Framework.Options import Options, OptionsError
from ..Framework.Display import Monitor, DisplayError


class ElodieError(SlipyError):
  """
  Exception specific to Elodie module.
  """
  pass

class Archive:
    """
    Import and parse ascii catalog of Elodie archive files. The complete
    archive is stored in the member `data`. It's organized in a dictionary
    by unique target names. Each target has a list of pairs consisting of the
    name of the file and the signal to noise for that spectrum. The reduced
    archive, accessed with `files`, contains identified `HD`, `HR`, `BD`, `GC`,
    and `GJ` objects, choosing the file pertaining to the spectra with the
    highest signal-to-noise ratio available.
    """
    def __init__(self, **kwargs):
        try:

            # default input file
            default_infile = os.path.join( os.path.dirname(__file__),
                'Archives/Elodie.csv')

            # parameter defaults
            options = Options( kwargs,
                {
                    'infile'  : default_infile            , # path to input file
                    'catalogs': ['HD','BD','HR','GC','GJ']  # catalogues to keep
                })

            # parameter assignments
            infile   = options('infile')
            catalogs = options('catalogs')

        except OptionsError as err:
            print(' --> OptionsError:', err)
            raise ElodieError('Failed keyword assignment from Archive.__init__')

        # import data from archive
        with open(infile, 'r') as archive:
            data = [ line.split(',') for line in archive.readlines() ]

        # strip elements of white space
        data = [ [x.strip() for x in line] for line in data ]

        # build empty dictionary of unique names with empty lists
        targets = { name:[] for name in set([ line[0] for line in data ]) }

        # compile list of spectra files organized by identifier w/ S/N
        for line in data:
            targets[line[0]].append( line[2:] )

        # reject files from spectra not in catalogues
        targets = { k:v for k, v in targets.items() if k[:2] in catalogs }

        files = {}
        for target, options in targets.items():
            # choose best S/N
            idx = np.array( np.array(options)[:,-1], dtype=np.int_ ).argmax()
            files[target] = options[idx][0]

        # members
        self.data  = targets
        self.files = files
        self.names = [
                '-'.join('-'.join(x.split(':')).split('/')) + '.fits'
                for x in files.values()
            ]

def Script(filename, pipeline=''):
    """
    Construct url script for Elodie archive given `filename` and optionally
    `pipeline` instructions (e.g., `&z=wrs|fca[1,nor]`).
    """
```

```
    return ''.join(['http://atlas.obs-hp.fr/elodie/E.cgi?&c=i&o=',
        filename, pipeline, '&a=mime:application/x-fits'])

def Download( *files, **kwargs ):
    """
    Download ‘files‘ from Elodie archive via url scripts. The spectra can be
    further reduced via Elodie‘s pipeline with the following options.

    kwargs = {
            'verbose'  : True           , # display messages, progress
            'resample' : (min, max, res), # resample spectra (no default)
            'normalize': True           , # continuum normalization
            'outpath'  : './'           , # directory for downloaded files
            'names'    : []               # alternative output names for ‘files‘
        }
    """
    try:

        # function parameter defaults
        options = Options( kwargs,
            {
                'verbose'  : True      , # display messages, progress
                'resample' : (-1,-1,-1), # handled by Elodie
                'normalize': True      , # continuum normalization
                'outpath'  : './'      , # directory for downloaded files
                'names'    : []          # alternative output names for ‘files‘
            })

        # function parameter assignments
        verbose   = options('verbose')
        resample  = options('resample')
        normalize = options('normalize')
        outpath   = options('outpath')
        names     = options('names')

        # check for ‘resampled‘ assignment
        if 'resample' not in kwargs:
            resample = None
        elif len(resample) != 3:
            raise ElodieError('Download() expects ‘resample‘ to be of length '
            'three.')

        # set default names
        if not names:
            names = [ '-'.join(fname.split(':')) for fname in files ]
            names = [ '-'.join(fname.split('/')) for fname in names ]
            names = [            fname + '.fits' for fname in names ]

        elif len(names) != len(files):
            raise ElodieError('Download() expects ‘names‘ option to be of '
            'length equal to that of the number of ‘files‘ arguments.')

    except OptionsError as err:
        print(' --> OptionsError:', err)
        raise ElodieError('Failed keyword assignment from Download().')

    if not resample and not normalize:
        pipeline = ''
    else:
        pipeline = '&z=wrs'

    if normalize:
        pipeline += '|fca[1,nor]'

    if resample:
        resample = [ str(x) for x in resample ]
        pipeline += '|wrs[1,' + ','.join(resample) + ']'

    if verbose:
        display = Monitor(ETC=True)
        nfiles  = len(files)
        print('\n Downloading {} files from Elodie ...'.format(nfiles))

    for a, spectra in enumerate(files):

        # show progress
        if verbose: display.progress(a + 1, nfiles)

        # download file
        with urlopen( Script(spectra, pipeline) ) as response, open(
            os.path.join(outpath, names[a]), 'wb') as outfile:
            shutil.copyfileobj(response, outfile)

    if verbose:
        display.complete()
        display.elapsed()
```

C.6 .. Framework . Argument

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# slipy/Framework/Arguments.py
"""
Module contains `Argument` class for handling conversions and type
checking for function/class keyword argument options.
"""

from .. import SlipyError

class ArgumentError(SlipyError):
  """
  Exception specific to Argument module.
  """
  pass

class Argument:
  """
  `Argument` object has type and value management.
  """
  def __init__(self, value, name = 'unspecified', **kwargs ):
    """
    Build Argument `value`, `type`, and set `options`.
    """
    options = {
        'lock' : False # disallow all type conversions
      }
    for arg in kwargs:
      if arg not in options:
        raise ArgumentError('`{}` is not an option for Argument.'
          .format(arg))
      if type(kwarg[arg]) is not type(options[arg]):
        raise ArgumentError('Option `{}` expects {}, given {}'
          .format(arg, type(options[arg]), type(kwargs[arg])) )
      # accept reassignment
      options[arg] = kwargs[arg]

    self.lock  = options['lock']
    self.T     = type(value)
    self.value = value
    self.name  = str(name)
    if type(name) is not str:
      raise ArgumentError('Argument expects type str for name.')

  def __call__(self, value):

    if self.lock and type(value) is not self.T:
      raise ArgumentError('Argument `{}` is locked as {}, '
        'but new value has {}.'
        .format(self.name, self.T, type(value)) )

    if self.T is bool:
      # special rules for `bool` conversions

      if type(value) is str:
        if value != "True" and value != "False":
          raise ArgumentError('Invalid conversion from {} '
            'for Argument `{}`.'.format(self.T, self.name))
        self.value = True if value == 'True' else False

      elif type(value) is int:
        if value != 0 and value != 1:
          raise ArgumentError('Invalid conversion from {} '
            'for Argument `{}`.'.format(self.T, self.name) )
        else:
          self.value = True if value else False

      elif type(value) is not bool:
        raise Error('Invalid conversion from {} '
          'for Argument `{}`.'.format(self.T, self.name))

      else: self.value = value

    else:

      try:
        self.value = self.T(value)

      except ValueError as error:
        raise ArgumentError('Cannot convert {} to {} for `{}`'
          ' Argument.'.format(self.T, type(value), self.name))
```

C.7 .. Framework . Command

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3).
# slipy/Framework/Interface.py
"""
Command line interface tools.
"""

from .. import SlipyError
from .Options import Options, OptionsError

class CommandError(SlipyError):
  """
  Exception specific to Command module.
  """
  pass

def Parse( clargs, **kwargs ):
  """
  Parse command line arguments, `clargs` (i.e., sys.argv).
  """
  if type(clargs) is not list:
    raise CommandError('Parse function expects a list for `clargs`.')
  try:
    options = Options( kwargs,
      {
        'exe' : True # search first argument for '@func' pattern
      })

    if options('exe'):
      function = clargs[0].split('@')
      if len(function) != 2 or function[0]:
        raise CommandError('Incorrect formatting in function call.')
      function = function[1]
      del(clargs[0])

    # args should not have an assignment
    args = [ x for x in clargs if '=' not in x ]

    # remaining clargs should be kwargs
    kwargs = {
        key : value for key, value in [
          arg.split('=') for arg in set(clargs) - set(args)
          ]
      }

    if options('exe'):
      return function, args, kwargs

    else:
      return args, kwargs

  except OptionsError as err:
    print('\n --> OptionsError:', err)
    raise CommandError('from Parse')

  except ValueError as key:
    raise CommandError('Incorrect formatting of keyword arguments.')
```

C.8 .. Framework . Display

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# slipy/Framework/Display.py
"""
Display - Python module for displaying content to the terminal.
"""

import os, sys, math
from time import time
from datetime import datetime, timedelta

from .. import SlipyError
from .Options import Options, OptionsError

class DisplayError(SlipyError):
  """
  Exception specific to the Display module.
  """
  pass

class Monitor:
  """
  Class for displaying a progress bar during iterative tasks.
  """
  def __init__(self, **kwargs ):
    try:
      # available keyword options
      self.options = Options( kwargs,
        {
          'width'    : 45    , # number of characters wide
          'numbers'  : True  , # display numberical percent
          'template' : '[=>]', # template for progress bars
          'freq'     : 0.25  , # refresh rate
          'ETC'      : False , # display estimated time of completion
          'inline'   : True    # vanish after completion
        })

      # give assignments
      self.width   = self.options('width')
      self.numbers = self.options('numbers')
      self.freq    = self.options('freq')
      self.ETC     = self.options('ETC')
      self.inline  = self.options('inline')
      self.left, self.char, self.tip, self.right = self.options('template')

      # start clocks
      self.start = time()
      self.last  = time()

    except OptionsError as err:
      print( '\n --> OptionsError:', err.msg )
      raise DisplayError('Failed to initialize Monitor.')

    except ValueError as err:
      raise DisplayError(
        ''`template` option requires exactly 4 characters.')

  def __EstimatedCompletionTime(self):
    """
    Estimated time of completion, based on percent complete and
    current total elapsed time.
    """
    if self.percent > 0:
      elapsed   = time() - self.start
      remaining = elapsed * (1 / self.percent - 1)
      etc = datetime.today() + timedelta(seconds=remaining)
      return etc.strftime(' ETC: %Y-%m-%d @ %H:%M ')

  def __build(self):
    """
    Build the progress bar
    """
    bars    = self.char * math.floor( self.percent * self.width )
    empty   = ' ' * (self.width - len(bars) - 1)
    display = self.left + bars + self.tip + empty + self.right

    if self.numbers:
      display += '{:>8.2f} % '.format( self.percent * 100 )

    if self.ETC:
      display += self.__EstimatedCompletionTime()

    sys.stdout.write('\r \033[K \r {}'.format(display))
    sys.stdout.flush()

  def progress(self, i, imax):
    """
    Request a progress bar.
    """
```

```
90        if time() - self.last > self.freq:
91          # refresh rate surpassed,
92          # update time of last call and percent complete
93          self.last    = time()
94          self.percent = float(i) / float(imax)
95          # display progress bar
96          self.__build()
97
98      def complete(self):
99        """
100       Call to finalize the progress bar.
101       """
102       if self.inline:
103         sys.stdout.write('\r\033[K\r')
104         sys.stdout.flush()
105
106       else:
107         self.percent = 1
108         self.numbers = False
109         self.ETC     = False
110         self.__build()
111         sys.stdout.write(' complete\n')
112         sys.stdout.flush()
113
114     def elapsed(self):
115       """
116       Display total time elapsed since instantiation.
117       """
118       total = time() - self.start
119       abrv  = [ 'd',         'h',      'm',    's']
120       unit  = { 'd': 86400, 'h':3600, 'm':60 }
121       count = { 'd':0,       'h':0,    'm':0,  's':0 }
122
123       for item in abrv:
124         if item in unit:
125           while total > unit[item]:
126             total -= unit[item]
127             count[item] += 1
128         else: count[item] = math.floor(total)
129
130       total = [
131           '{} {}'.format( v, u )
132           for u, v in zip( abrv, [ count[v] for v in abrv ] )
133           if count[u]
134         ]
135       total = ' Time Elapsed: ' + ' '.join(total)
136       total = ' ' + '-' * (len(total)+5) + '\n ' + total
137
138       sys.stdout.write(total)
139       sys.stdout.flush()
```

C.9 .. Framework . Measurement

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# slipy/Framework/Measurement.py
"""
The `Measurement` is a container for a `value` and `error`.

Astropy already has a very useful object, `Quantity` that is expanded into
a `Constant` class. Something with a value, a name, abbreviation, uncertainty,
and a reference. A `Measurement` is nothing more than a `Constant` by
a different name. It functions just like a Quantity/Constant, only we don't
want to be calling it a "constant" and we want to be able to have many of them.
"""

from astropy.units import Quantity

class Measurement(Quantity):
    """
    A container for a `result` and an `error`. This object is meant to be functionally
    equivalent to the astropy.constant.Constant, without the instance checking (we can
    have many `Measurement`s). There are `notes` instead of `references`.
    """
    def __new__(cls, value, error=None, name=None, notes=None):

        instance = super().__new__(cls, value)

        instance.error = error
        instance.name  = name
        instance.notes = notes

        return instance

    # These operations are `broken` because they would otherwise yield a `Measurement`.
    # The right handed operations are not affected, these are all that is needed I think.
    def __truediv__(self, other):
        return Quantity(super().__truediv__(other))

    def __mul__(self, other):
        return Quantity(super().__mul__(other))

    def __add__(self, other):
        return Quantity(super().__add__(other))

    def __sub__(self, other):
        return Quantity(super().__sub__(other))
    # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

    def __repr__(self):
        return '<' + ' '.join([ label + str(attr) for attr, label in zip(['Measurement',
            self.value * self.unit, self.error, self.name, self.notes],
            ['', '', '| error = ', '| name = ', '| notes = ']) if attr]) + '>'

    def __str__(self):
        attr = [ self.name, self.value*self.unit, self.error, self.notes ]
        name = [' Name  = {}', ' Value = {}', ' Error = {}', ' Notes = {}']
        show = [ a for a in attr if a ]
        return '\n'.join([n for a, n in zip(attr, name) if a]).format(*show)
```

C.10 .. Framework . Options

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# slipy/Framework/Options.py
"""
Class object for handling kwargs in classes and functions.
"""

from .. import SlipyError
from .Argument import Argument as Arg, ArgumentError

class OptionsError(SlipyError):
  """
  Exception specific to Options module.
  """
  pass

class Options:
  """
  Class object for handling kwargs in classes and functions.
  """
  def __init__(self, kwargs, options ):
    """
    Check types and build options dictionary
    """
    try:
      # initial assignment
      self.options = {
          name : Arg(value, name)
          for name, value in options.items()
        }
      # attempted reassignment
      for key, value in kwargs.items():
        self.options[key](value)

    except AttributeError as err:
      raise OptionsError(
        'Options object expects dictionary types.')

    except KeyError as key:
      raise OptionsError(
        '{} was not a recognized option.'.format(key))

    except ArgumentError as err:
      print('\n --> ArgumentError:', err.msg )
      raise OptionsError('Failed assignment.')

  def __call__(self, option):
    """
    Retrieve value of `option`.
    """
    try:
      return self.options[option].value
    except KeyError as key:
      raise OptionsError('{} was not recognized.'.format(key))

  def items(self):
    """
    Access options.items() values.
    """
    return { k:v.value for k,v in self.options.items() }.items()
```

C.11 .. SLiPy . Correlate

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# slipy/SLiPy/Correlate.py
"""
Correlate - Module of correlation functions for astronomical data.
"""
import numpy as np

from .. import SlipyError
from .Spectrum import Spectrum, SpectrumError
from ..Framework.Options import Options, OptionsError

class CorrelateError(SlipyError):
  """
  Exception specific to Correlate module.
  """
  pass

def RMS( array ):
  """
  RMS( array ):

  Return the root mean square of an `array`.
  """
  if type(array) is not np.ndarray or len(np.shape(array)) != 1:
    raise CorrelateError('RMS() expects 1D numpy.ndarray`s.')

  return np.sqrt( ( array**2 ).sum() ) / len(array)

def Xcorr( spectrumA, spectrumB, **kwargs ):
  """
  Xcorr( spectrumA, spectrumB, **kwargs ):

  Cross correlate two spectra of equal pixel length. The function returns
  an integer value representing the best shift within a `lag` based on
  the computed RMS of each configuration.
  """
  try:
    # define `options` for Xcorr()
    options = Options( kwargs,
      {
        'lag' : 25 # pixels to shift for correlation
      })

    # check argument types, values
    if ( type(spectrumA) is not Spectrum or
      type(spectrumB) is not Spectrum ):
      raise CorrelateError('Xcorr() expects `Spectrum` arguments.')

    elif len(spectrumA.data) != len(spectrumB.data):
      raise CorrelateError('Xcorr() expects `Spectrum` arguments to '
      'be of equal length.')

    # assign `lag` and the pixel length of the spectra
    lag  = options('lag')
    npix = len(spectrumA.data)

    # resample `B` to wavelength space of `A`
    spectrumB.resample(spectrumA)

    # arrays to correlate
    A = spectrumA.data.value
    B = spectrumB.data.value

    # shift spectra `left` over each other
    left = np.array([ RMS(diff) for diff in [ A[-shift:] -
      B[:shift] for shift in range(-lag,0) ] ])

    # shift spectra `right` over each other
    right = np.array([ RMS(diff) for diff in [ A[:-shift] -
      B[shift:] for shift in range(1,lag+1) ] ])

    # include `zero` shift in rms vector.
    rms = np.hstack((left, RMS(A - B), right))

    # return the shift corresponding to the minimum RMS
    return rms.argmin() - lag

  except OptionsError as err:
    print(' --> OptionsError:', err)
    raise CorrelateError('Inappropiate keyword arguments passed '
    ' to Xcorr().')
```

C.12 .. SLiPy . Fits

```
#!/usr/bin/env python
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# slipy/SLiPy/Fits.py
"""
Fits - FITS file handling module.
"""
import os, sys, fnmatch
from astropy.io import fits as pyfits
from numbers import Number

from .. import SlipyError
from ..Framework.Command import Parse, CommandError
from ..Framework.Options import Options, OptionsError
from ..Framework.Display import Monitor, DisplayError
from .Spectrum import Spectrum, SpectrumError
from .Simbad import Position, Distance, Sptype, IDList, SimbadError

class FitsError(SlipyError):
    """
    Exception for Fits module.
    """
    pass

def Find(toplevel = './', pattern = '*.fits'):
  """
  Search for file paths below `toplevel` fitting `pattern`.
  """
  if not os.path.isdir(toplevel):
    raise FitsError('`{}` does not name a directory.'.format(toplevel))

  return [ os.path.join(toplevel, filename)
    for filename in fnmatch.filter(os.listdir(toplevel), pattern) ]

def RFind(toplevel = './', pattern = '*.fits'):
  """
  Recursively search for paths below `toplevel` fitting `pattern`.
  """
  if not os.path.isdir(toplevel):
    raise FitsError('`{}` does not name a directory.'.format(toplevel))

  return [ os.path.join(dirpath, filename)
    for dirpath, dirnames, filenames in os.walk(toplevel)
    for filename in fnmatch.filter(filenames, pattern) ]

def GetData( *files, **kwargs ):
  """
  Import data from FITS `files`.

  kwargs = {
      verbose   : True    , # display messages, progress
      toplevel  : ''      , # request import from directory `toplevel`
      pattern   : '*.fits', # pattern matching with `toplevel`
      recursive : False   , # search recursively below `toplevel`
      wavecal   : True    , # fit wavelength vector to data
      crpix1    : 'crpix1', # reference pixel header keyword
      crval1    : 'crval1', # value at reference pixel
      cdelt1    : 'cdelt1', # resolution (delta lambda)
    }
  """
  try:
    # convert `files` to list
    files = list(files)

    # available key word arguments
    options = Options( kwargs,
      {
        'verbose'  : True    , # display messages, progress
        'toplevel' : ''      , # request import from `toplevel` dir
        'pattern'  : '*.fits', # pattern matching with `toplevel`
        'recursive': False   , # search recursively below `toplevel`
        'wavecal'  : True    , # fit wavelength vector to data
        'crpix1'   : 'crpix1', # reference pixel header keyword
        'crval1'   : 'crval1', # value at reference pixel
        'cdelt1'   : 'cdelt1', # resolution (delta lambda)
      })

    # assignment options
    verbose   = options('verbose')
    toplevel  = options('toplevel')
    pattern   = options('pattern')
    recursive = options('recursive')
    wavecal   = options('wavecal')
    crpix1    = options('crpix1')
    crval1    = options('crval1')
    cdelt1    = options('cdelt1')

    if toplevel:
      # search for files matching `pattern`
```

```
90          find   = RFind if recursive else Find
91          files += find( toplevel, pattern )
92
93        if verbose:
94          # import iteratively, displaying progress
95          display = Monitor()
96          nfiles  = len(files)
97          data    = []
98          print(' Importing data from {} Fits files ...'.format(nfiles) )
99          for a, filename in enumerate(files):
100           display.progress(a, nfiles)
101           data.append( Spectrum(filename, wavecal=wavecal,
102             crpix1=crpix1, crval1=crval1, cdelt1=cdelt1) )
103
104         display.complete()
105         return data
106
107       # import spectra 'silently'
108       return [ Spectrum(filename, wavecal=wavecal, crpix1=crpix1,
109         crval1=crval1, cdelt1=cdelt1) for filename in files ]
110
111     except OptionsError as err:
112       print(' --> OptionsError:', err)
113       raise FitsError('Data retrieval failure.')
114
115     except SpectrumError as err:
116       print(' --> SpectrumError:', err)
117       raise FitsError('Failed to construct spectrum.')
118
119   def Header( filename, keyword = None, **kwargs ):
120       """
121       Retrieve `keyword` from FITS header in `filename`. If not provided a
122       keyword, the entire header object is returned.
123       """
124       try:
125
126           options = Options( kwargs,
127               {
128                   'is_main' : False # reserved for calls from Main()
129               })
130
131           is_main = options('is_main')
132
133           with pyfits.open(filename) as hdulist:
134               header = hdulist[0].header
135               if keyword: header = header[keyword]
136
137           if is_main:
138               print( header )
139               return
140
141           else:
142               return header
143
144       except OptionsError as err:
145           print(' --> OptionsError:', err)
146           raise FitsError('Failed keyword assignment in Header().')
147
148       except KeyError as key:
149           raise FitsError('Header element `{}` was not accessible '
150           'from `{}`'.format(keyword, filename))
151
152   def Search( *files, **kwargs ):
153     """
154     Exctract object names from Fits `files` and use Simbad.py
155     to resolve the `attribute` (a required keyword argument)
156     from the SIMBAD astronomical database.
157
158     kwargs = {
159         verbose   : True    , # display messages, progress
160         toplevel  : ''      , # search under `toplevel` directory
161         pattern   : '*.fits', # for files under `toplevel`
162         recursive : False   , # search recusively under `toplevel`
163         attribute : ''      , # attribute to search for (no default)
164       }
165     """
166     try:
167
168       # convert `files` to list
169       files = list(files)
170
171       # available keyword arguments
172       options = Options( kwargs,
173         {
174           'verbose'   : True    , # display messages, progress
175           'toplevel'  : ''      , # search under `toplevel` directory
176           'pattern'   : '*.fits', # for files under `toplevel`
177           'recursive' : False   , # search recusively under `toplevel`
178           'attribute' : ''      , # attribute to search for (no default)
179           'is_main'   : False     # reserved for calls from Main()
180         })
181
182       # assign parameters
```

```
183       verbose   = options('verbose')
184       toplevel  = options('toplevel')
185       pattern   = options('pattern')
186       recursive = options('recursive')
187       attribute = options('attribute')
188       is_main   = options('is_main')
189
190       # Available search functions from Simbad.py
191       SimbadSearch = {
192           'Position': Position, # ra, dec (degrees)
193           'Distance': Distance, # in parsecs
194           'Sptype'  : Sptype  , # spectral type
195           'IDList'  : IDList    # alternate IDs
196         }
197
198       if not attribute:
199         raise FitsError('An `attribute` must be specified for Search().')
200
201       if attribute not in SimbadSearch:
202         raise FitsError('`{}` is not an available search criteria.'
203             .format(attribute))
204
205       if toplevel:
206         # search for files in `toplevel` directory
207         find   = RFind if recursive else Find
208         files += find(toplevel, pattern)
209
210       nfiles = len(files)
211       display = Monitor()
212
213       if verbose:
214         # read object names iteratively
215         print(' Reading object names for {} Fits files ...'.format(nfiles))
216         obj_ids = []
217         for a, name in enumerate(files):
218           display.progress(a, nfiles)
219           obj_ids.append( Header(name, 'object') )
220
221         display.complete()
222
223       else: obj_ids = [ Header(name, 'object') for name in files ]
224
225       if verbose:
226         # query for `attribute` iteratively
227         print(' Searching for `{}`s with SIMBAD ...'.format(attribute))
228         results = []
229         for a, obj in enumerate(obj_ids):
230           display.progress(a, nfiles)
231           results.append( SimbadSearch[attribute](obj) )
232
233         display.complete()
234
235       else: results = [ SimbadSearch[attribute](obj) for obj in obj_ids ]
236
237       if is_main:
238         formatted = {
239             'Position':'{1:.2f} {1:.2f}',
240             'Distance':'{0:.2f}',
241             'Sptype'  : '{}'
242           }
243         for item in results:
244           if type(item) is list:
245             print( formatted[attribute].format(*item))
246           else:
247             print( formatted[attribute].format(item))
248
249       else: return results
250
251     except OptionsError as err:
252       print(' --> OptionsError:', err)
253       raise FitsError('Failed assignment for Search().')
254
255     except SimbadError as err:
256       print(' --> SimbadError:', err)
257       raise FitsError('Simbad failed.')
258
259   def PositionSort( center, radius, *files, **kwargs ):
260       """
261       Return a list of files from `files` that lie in a `radius` (in degrees)
262       from `center`, based on the `ra` (right ascension) and `dec` (declination).
263
264       kwargs = {
265               'ra'       : 'pos1'  , # header element for right ascension
266               'dec'      : 'pos2'  , # header element for declination
267               'obj'      : 'object', # header element for object id
268               'raconvert': True    , # convert decimal hours to decimal degrees
269               'verbose'  : True    , # display messages, progress
270               'toplevel' : ''      , # `toplevel` directory to look for files in
271               'recursive': False   , # search `recursive`ly below `toplevel`
272               'pattern'  : '*.fits', # glob `pattern` for file search
273               'useSimbad': False     # use Simbad instead of header elements
274           }
```

```
275         """
276         try:
277             # function parameter defaults
278             options = Options( kwargs,
279                 {
280                     'ra'       : 'pos1'   , # header element for right ascension
281                     'dec'      : 'pos2'   , # header element for declination
282                     'obj'      : 'object', # header element for object id
283                     'raconvert': True     , # convert decimal hours to decimal degrees
284                     'verbose'  : True     , # display messages, progress
285                     'toplevel' : ''       , # `toplevel` directory for file search
286                     'recursive': False    , # search `recursive`ly below `toplevel`
287                     'pattern'  : '*.fits', # glob `pattern` for file search
288                     'useSimbad': False      # use Simbad instead of header elements
289                 })
290 
291             # function parameter assignments
292             ra        = options('ra')
293             dec       = options('dec')
294             obj       = options('obj')
295             raconvert = options('raconvert')
296             verbose   = options('verbose')
297             toplevel  = options('toplevel')
298             recursive = options('recursive')
299             pattern   = options('pattern')
300             useSimbad = options('useSimbad')
301 
302         except OptionsError as err:
303             print(' --> OptionsError:', err)
304             raise FitsError('Failed keyword assignment in PositionSort().')
305 
306         # check arguments
307         if not hasattr( center, '__iter__'):
308             raise FitsError('PositionSort() expects `center` argument to be '
309             '`iterable` and have two elements.')
310         if len(center) != 2:
311             raise FitsError('PositionSort() expects `center` argument to have '
312             'exactly two elements.')
313         if not isinstance( radius, Number ):
314             raise FitsError('PositionSort() expects `radius` argument to '
315             'be a `Number`.')
316         for a, f in enumerate(files):
317             if not isinstance(f, str):
318                 raise FitsError('PositionSort() expects `str` like arguments '
319                 'for all `files` (from argument {})'.format(a))
320 
321         # convert `files` to list type
322         files = list(files)
323 
324     # look under `toplevel` if requested
325         if toplevel:
326             find   = RFind if recursive else Find
327             files += find(toplevel, pattern)
328 
329         if verbose:
330             # create display object
331             display = Monitor()
332             nfiles  = len(files)
333 
334         # initialize blank lists
335         pos1, pos2 = [], []
336 
337         if not useSimbad:
338             if verbose: print(' Retrieving position information from {} files ... '.format(nfiles))
339             # check file headers for requested information
340             for a, fitsfile in enumerate(files):
341                 try:
342                     alpha = Header(fitsfile, ra)
343                     delta = Header(fitsfile, dec)
344                     if raconvert: alpha *= 180 / 12
345                     pos1.append(alpha)
346                     pos2.append(delta)
347                 except FitsError as err:
348                     # attempt to get info from SIMBAD instead
349                     print('\n Failed to retrieve position from file {}, contacted SIMBAD ...'.format(a))
350                     pos = Position( Header(fitsfile, obj) )
351                     print('\033[A\r\033[K\033[2A')
352                     pos1.append( pos[0] )
353                     pos2.append( pos[1] )
354 
355                 if verbose: display.progress(a + 1, nfiles)
356 
357         else:
358             # use the Simbad module to search for positions
359             if verbose: print(' Retrieving {} positions from SIMBAD ... '.format(nfiles))
360             for a, fitsfile in enumerate(files):
361                 pos = Position( Header(fitsfile, obj) )
362                 pos1.append( pos[0] )
363                 pos2.append( pos[1] )
364                 if verbose: display.progress(a, nfiles)
365 
366         # erase progress bar
```

```
367       if verbose:
368           display.complete()
369           print('\r\033[K Compiling list of files ... ')
370
371       # keep files for targets within range
372       keepers = [ f for p1, p2, f in zip(pos1, pos2, files)
373           if abs(p1 - center[0]) < radius and abs(p2 - center[1]) < radius ]
374
375       # account for p1 ~ 0 && center ~ 360 like comparisons
376       keepers += [ f for p1, p2, f in zip(pos1, pos2, files)
377           if abs(p1 + 360 - center[0]) < radius and abs(p2 - center[1]) < radius ]
378
379       # account for p1 ~ 360  && center ~ 0 like comparisons
380       keepers += [ f for p1, p2, f in zip(pos1, pos2, files)
381           if abs(p1 - center[0] - 360) < radius and abs(p2 - center[1]) < radius ]
382
383       if verbose: print('\033[A\r\033[K Compiling list of files ... done')
384
385       # exclude any potential double countings
386       return list(set(keepers))
387
388   def Main( clargs ):
389     """
390     Main function. See __doc__ for details.
391     """
392
393     if len(clargs) < 2:
394       # show usage
395       print( __doc__ )
396       return 0
397
398     # Available functions for execution
399     executable = {
400         'Header' : Header, # Header function
401         'Search' : Search  # Search function
402       }
403
404     try:
405
406       # Parse command line arguments
407       function, args, kwargs = Parse( clargs[1:] )
408       if not args and not kwargs:
409         # show function usage
410         print( executable[function].__doc__ )
411         return 0
412
413       # run execution
414       executable[function]( *args, is_main=True, **kwargs )
415       return 0
416
417     except CommandError as err:
418       print(' --> CommandError:', err)
419       return 1
420
421     except KeyError as key:
422       print(' --> {} was not a recognized function.'.format(key))
423       return 1
424
425     except FitsError as err:
426       # don't let uncaught self exception pass if from main.
427       print(' --> FitsError:', err.msg)
428       return 1
429
430     except Exception as err:
431       print(' --> Unrecognized error from Fits module.')
432       print(' --> Exception: `{}``'.format(err))
433       return 1
434
435   if __name__ == '__main__':
436     # call Main function, exit 0 or 1
437     sys.exit( Main(sys.argv) )
```

C.13 .. SLiPy . Montage

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# slipy/SLiPy/Montage.py
"""
This module makes use of the `Montage` mosaic tools from caltech, see:
http://montage.ipac.caltech.edu/

The user should have Montage`s executables available on their path.
"""

import os, shutil as sh, numpy as np
from subprocess import check_output as call, CalledProcessError
from sys import stdout
from numbers import Number

from .. import SlipyError
from ..Framework.Options import Options, OptionsError
from ..Framework.Display import Monitor, DisplayError


class MontageError(SlipyError):
  """
  Exception specific to the Montage module
  """
  pass

def SolveGrid( sides, grid ):
  """
  SolveGrid( sides, grid ):

  Helper function for the Field and SubField classes. Both `sides` and `grid`
  need to be array-like and of length two. `sides` is the side length of the
  field in decimal degrees in right ascension and declination respectively.
  `grid` specifies the subdivision along these axis (e.g., (2,2) says 2x2).

  The user should mindful of their choices. If the side lengths cannot be
  subdivided into well-behaved (rational) segments, higher decimal places
  will be lost in the SubField.ArchiveList() task resulting in small
  gaps in the mosaic.
  """
  # check arguments
  if not hasattr(sides, '__iter__') or not hasattr(grid, '__iter__'):
    raise MontageError('Grid() expects both arguments to be array-like.')
  if len(sides) != 2 or len(grid) != 2:
    raise MontageError('Grid() expects both arguments to have length two.')

  # grid `site` centers in the horizontal axis
  ra_left_site    = -sides[0] / 2 + 0.5 * sides[0] / grid[0]
  ra_right_site   =  sides[0] / 2 - 0.5 * sides[0] / grid[0]
  ra_site_centers = np.linspace( ra_left_site, ra_right_site, grid[0] )

  # grid `site` centers in the vertical axis
  dec_bottom_site  = -sides[1] / 2 + 0.5 * sides[1] / grid[1]
  dec_top_site     =  sides[1] / 2 - 0.5 * sides[1] / grid[1]
  dec_site_centers = np.linspace( dec_bottom_site, dec_top_site, grid[1] )

  return ra_site_centers, dec_site_centers

def Mosaic(resolution, *folders, **kwargs):
  """
  Mosaic(resolution, *folders, **kwargs):

  Conduct standard build procedures for all `folders`. `resolution` is the
  number of pixels per degree for the output image. Note: `folders` should
  be absolute paths.

  kwargs = {
      verbose : True, # display messages, progress
      bkmodel : True  # model and correct for background effects
    }
  """
  try:
    # function parameter defaults
    options = Options( kwargs,
      {
        'verbose': True, # display messages, progress
        'bkmodel': True  # model andd correct for background effects
      })

    # function parameter assignments
    verbose = options('verbose')
    bkmodel = options('bkmodel')

    # build mosaics at all sites
    for a, folder in enumerate(folders):

      # change directories
      os.chdir(folder)

```

```
90          # display message
91          if verbose:
92            stdout.write('\n Building mosaic for {}: {} / {} ... \n'
93              .format(os.path.basename(folder), a + 1, len(folders)))
94            stdout.write( ' ' + '-' * 70 + '\n' )
95            stdout.write(' Generating image meta-data table ... ')
96            stdout.flush()
97
98          # generate image meta-data table
99          output = call(['mImgtbl','images','images.tbl']).decode('utf-8')
100         if 'ERROR' in output: raise MontageError('Failed `mImgtbl` from `{}`'
101           .format(folder))
102
103         if verbose:
104           stdout.write('done.\n Generating FITS header template ... ')
105           stdout.flush()
106
107         # create mosaic FITS header template
108         output = call(['mMakeHdr','-p','{}'.format(1 / resolution),
109           '-n','images.tbl','template.hdr']).decode('utf-8')
110         if 'ERROR' in output: raise MontageError('Failed `mMakeHdr` from '
111           '`{}`'.format(folder))
112
113         if verbose:
114           stdout.write('done\n Reprojecting images ... ')
115           stdout.flush()
116
117         # reproject images
118         output = call(['mProjExec','-p','images','images.tbl',
119           'template.hdr','projected','stats.tbl']).decode('utf-8')
120         if 'ERROR' in output: raise MontageError('Failed `mProjExec` in '
121           '`{}`'.format(folder))
122
123         if verbose:
124           stdout.write('done\n Generating new image meta-data table '
125             'for projected images ... ')
126           stdout.flush()
127
128         # create new meta-data table for reprojected images
129         output = call(['mImgtbl','projected','proj-images.tbl'
130           ]).decode('utf-8')
131         if 'ERROR' in output: raise MontageError('Failed `mImgtbl` in '
132           '`{}`'.format(folder))
133
134         if not bkmodel:
135           # simply co-add images
136           if verbose:
137             stdout.write('done\n Co-adding images ... ')
138             stdout.flush()
139           output = call(['mAdd','-p','projected','proj-images.tbl',
140             'template.hdr','final/mosaic.fits']).decode('utf-8')
141           if 'ERROR' in output: raise MontageError('Failed `mAdd` in '
142             '`{}`'.format(folder))
143
144         else:
145           # Fit overlaps for background corrections
146           if verbose:
147             stdout.write('done\n Fitting overlaps for background '
148               'corrections ... ')
149             stdout.flush()
150           output = call(['mOverlaps','proj-images.tbl','diffs.tbl'
151             ]).decode('utf-8')
152           if 'ERROR' in output: raise MontageError('Failed `mOverlaps` in '
153             '`{}`'.format(folder))
154
155           # perform background subtractions on overlaps
156           if verbose:
157             stdout.write('done\n Performing background subtractions '
158               'on overlaps ... ')
159             stdout.flush()
160           output = call(['mDiffExec','-p','projected','diffs.tbl',
161             'template.hdr','differences']).decode('utf-8')
162           if 'ERROR' in output: raise MontageError('Failed `mDiffExec` in '
163             '`{}`'.format(folder))
164
165           # computing plane-fitting coefficients
166           if verbose:
167             stdout.write('done\n Computing plane-fitting coefficients ... ')
168             stdout.flush()
169           output = call(['mFitExec','diffs.tbl','fits.tbl',
170             'differences']).decode('utf-8')
171           if 'ERROR' in output: raise MontageError('Failed `mFitExec` in '
172             '`{}`'.format(folder))
173
174           # create table of background corrections
175           if verbose:
176             stdout.write('done\n Creating table of background '
177               'corrections ... ')
178             stdout.flush()
179           output = call(['mBgModel','proj-images.tbl','fits.tbl',
180             'corrections.tbl']).decode('utf-8')
181           if 'ERROR' in output: raise MontageError('Failed `mBgModel` in '
```

```
182             ''{}''.format(folder))
183
184           # apply background matching to reprojected images
185           if verbose:
186             stdout.write('done\n Applying background matching to '
187               'reprojected images ... ')
188             stdout.flush()
189           output = call(['mBgExec','-p','projected','proj-images.tbl',
190             'corrections.tbl','corrected']).decode('utf-8')
191           if 'ERROR' in output: raise MontageError('Failed `mBgExec` in '
192             ''{}''.format(folder))
193
194           # co-add images for final mosaic
195           if verbose:
196             stdout.write('done\n Co-adding corrected images ... ')
197             stdout.flush()
198           output = call(['mAdd','-p','corrected','proj-images.tbl',
199             'template.hdr','final/mosaic.fits']).decode('utf-8')
200           if 'ERROR' in output: raise MontageError('Failed `mAdd` in '
201             ''{}''.format(folder))
202
203         # finished mosaic
204         if verbose:
205           stdout.write('done\n')
206           stdout.flush()
207
208     except CalledProcessError as err:
209       print(' --> CalledProcessError:', err)
210       raise MontageError('Failed process from Mosaic().')
211
212   class SubField:
213     """
214     SubField( center, sides, grid, **kwargs ):
215     """
216     def __init__(self, center, sides, grid, **kwargs ):
217       """
218       Create `site` grid for SubField.
219       """
220       try:
221         # function parameter defaults
222         options = Options( kwargs,
223           {
224             'survey': 'DSS'  , # DSS, SDSS, 2MASS
225             'band'  : 'DSS2B', # filter for `survey`, see `bands` dict
226             'pad'   : 0.0      # amount to add (degrees) around `sites`
227           })
228
229         # function parameter assigments
230         survey = options('survey').upper()
231         band   = options('band').upper()
232         pad    = options('pad')
233
234         # available bands for each survey
235         bands = {
236             # Two-Micron All-Sky Survey
237             '2MASS': ['J', 'H', 'K'],
238
239             # Sloan Digital Sky Survey
240             'SDSS': ['U','G','R','I','Z'],
241
242             # STScI Digitized Sky Survey
243             'DSS': ['DSS1B','DSS1R','DSS2B','DSS2R','DSS2IR','Quick-V']
244           }
245
246         # check for appropriate survey, band
247         if survey not in bands:
248           raise MontageError(''{}` was not a recognized survey for '
249           'the Montage Python module.'.format(survey))
250         if band not in bands[survey]:
251           raise MontageError(''{}` was not a recognized filter band '
252           'for the `{}` survey.'.format(band, survey))
253
254         # check arguments
255         if ( not hasattr(center, '__iter__') or
256           not hasattr(sides, '__iter__') or not hasattr(grid, '__iter__') ):
257           raise MontageError('SubField() expects array-like arguments for '
258           ''center', 'sides', and `grid` arguments.')
259         if len(center) != 2 or len(sides) != 2 or len(grid) != 2:
260           raise MontageError('SubField() expects `center`, `sides` and '
261           ''grid` arguments to have length two.')
262
263         # SolveGrid()
264         ra_site_centers, dec_site_centers = SolveGrid(sides, grid)
265         # relative to SubField `center`:
266         ra_site_centers  += center[0]
267         dec_site_centers += center[1]
268
269         # record number of `site`s along each axis
270         self.num_ra_sites  = grid[0]
271         self.num_dec_sites = grid[1]
272
273         # build arguments for subprocess call
```

```
274         self.archive_command_list = [
275             ['mArchiveList', survey, band, '{:.2f} {:.2f}'.format(ra_site,
276               dec_site), str(pad + sides[0]/grid[0]),
277               str(pad + sides[1]/grid[1]), 'remote.tbl']
278             for ra_site in ra_site_centers for dec_site in dec_site_centers
279           ]
280
281         # make current directory `home`
282         self.location = os.path.abspath('.')
283
284         # new tree structure
285         self.folders = [os.path.join(self.location,'Site-{:03d}'.format(a+1))
286           for a in range(len(self.archive_command_list)) ]
287
288         # initialize folder structure with `images` directory
289         for folder in self.folders:
290           abspath = os.path.join(folder,'images')
291           if not os.path.exists(abspath):
292             os.makedirs(abspath)
293
294       except OptionsError as err:
295         print(' --> OptionsError:', err)
296         raise MontageError('Failed keyword assignment in SubField().')
297
298     def ArchiveList(self, **kwargs):
299       """
300       Run the `mArchiveList` command on the `site` grid.
301       """
302       try:
303         # function parameter defaults
304         options = Options( kwargs,
305           {
306             'verbose': True # display messages, progress
307           })
308
309         # function parameter assignments
310         verbose = options('verbose')
311
312         for a, command in enumerate(self.archive_command_list):
313
314           # navigate to `images` directory
315           os.chdir( os.path.join(self.folders[a],'images') )
316
317           if verbose:
318             stdout.write('\n Running `mArchiveList` on {} ... '.format(
319               os.path.basename(self.folders[a])))
320             stdout.flush()
321
322           # run `mArchiveList`
323           output = call(command).decode('utf-8')
324           if 'ERROR' in output or 'count="0"' in output:
325             raise MontageError('Failed archive list from archive() '
326             '(command: {}), (output: {}).'.format(command, output))
327
328           if verbose: stdout.write('done')
329
330         if verbose:
331           stdout.write('\n')
332           stdout.flush()
333
334       except OptionsError as err:
335         print(' --> OptionsError:', err)
336         raise MontageError('Failed keyword assigned from SubField.exec().')
337
338       except CalledProcessError as err:
339         print(' --> CalledProcessError:', err)
340         raise MontageError('`mArchiveList` returned exit status 1.')
341
342     def ArchiveExec(self, **kwargs):
343       """
344       Run `mArchiveExec` on each `site` in the SubField.
345       """
346       try:
347         # function parameter defaults
348         options = Options( kwargs,
349           {
350             'verbose': True # display messages, progress
351           })
352
353         # function parameter assignments
354         verbose = options('verbose')
355
356         for a, folder in enumerate(self.folders):
357
358           # navigate to site folder
359           os.chdir( os.path.join(folder,'images') )
360
361           if verbose:
362             stdout.write('\n Running `mArchiveExec` on {} ... '.format(
363               os.path.basename(folder)))
364             stdout.flush()
365
366           # run `mArchiveExec`
```

```
367           output = call(['mArchiveExec','remote.tbl']).decode('utf-8')
368           if 'ERROR' in output: raise MontageError('Failed `mArchiveExec` '
369             'in folder `{}`. Output: {}'.format(folder, output))
370
371           if verbose: stdout.write('done')
372
373         if verbose:
374           stdout.write('\n')
375           stdout.flush()
376
377       except OptionsError as err:
378         print(' --> OptionsError:', err)
379         raise MontageError('Failed keyword assigment in ArchiveExec().')
380
381       except CalledProcessError as err:
382         print(' --> CalledProcessError:', err)
383         raise MontageError('`mArchiveExec` returned exit status 1.')
384
385     def Build(self, resolution, **kwargs):
386       """
387       Run the build process for the `sites` in this SubField. See the
388       Montage.Mosaic() function documentation.
389       """
390       try:
391         # function parameter options
392         options = Options( kwargs,
393           {
394             'verbose':True, # display message, progress
395             'bkmodel':True  # run background modelling procedure
396           })
397
398         # function parameter assignments
399         verbose = options('verbose')
400         bkmodel = options('bkmodel')
401
402       except OptionsError as err:
403         print(' --> OptionsError:', err)
404         raise MontageError('Failed keyword assignment in SubField.Build().')
405
406       if verbose:
407         stdout.write(' Setting up folder structure ... ')
408         stdout.flush()
409
410       # setup folder structure
411       for subdir in ['corrected','projected','differences','final']:
412         for folder in self.folders:
413           abspath = os.path.join(folder,subdir)
414           if not os.path.exists(abspath):
415             os.makedirs(abspath)
416
417       if verbose:
418         stdout.write('done\n')
419         stdout.flush()
420
421       # run Mosaic() on all `site`s
422       Mosaic( resolution, *self.folders, verbose=verbose, bkmodel=bkmodel )
423
424     def Collect(self, **kwargs):
425       """
426       Collect( **kwargs ):
427
428       Collect the mosaics from all `site` locations into a master `images`
429       folder. The only `kwarg` is `verbose` (default: True).
430       """
431       try:
432
433         # function parameter defaults
434         options = Options( kwargs,
435           {
436             'verbose': True # display messages, progress
437           })
438
439         # function parameter assignments
440         verbose = options('verbose')
441
442       except OptionsError as err:
443         print(' --> OptionsError:', err)
444         raise MontageError('Failed keyword assignment in SubField.Collect().')
445
446       # change to SubField `location` directory
447       os.chdir(self.location)
448
449       # check that we have a finished mosaic at each `site`
450       for folder in self.folders:
451         if not os.path.exists( os.path.join(folder, 'final/mosaic.fits') ):
452           raise MontageError('No `mosaic.fits` file in `{}`.'.format(folder))
453
454       # ensure path to master/images/
455       master_images = os.path.join(self.location, 'master/images')
456       if not os.path.exists(master_images):
457         os.makedirs(master_images)
458
459       for a, folder in enumerate(self.folders):
```

```
460
461         if verbose:
462           stdout.write('\n Copying image from {} ... '.format(
463             os.path.basename(folder)))
464           stdout.flush()
465
466         # Copy over image
467         sh.copy( os.path.join(folder,'final/mosaic.fits'), os.path.join(
468           master_images, 'mosaic-{}.fits'.format(a + 1)) )
469
470         if verbose: stdout.write('done')
471
472       if verbose:
473         stdout.write('\n')
474         stdout.flush()
475
476     def Merge(self, resolution, **kwargs):
477       """
478       Merge(resolution, **kwargs ):
479
480       Merge all `site` mosaics into a single master SubField mosaic. The only
481       keyword options are `verbose` (default: True) and `bkmodel` (default:
482       True). See Montage.Mosaic().
483       """
484       try:
485
486         # function parameter defaults
487         options = Options( kwargs,
488           {
489             'verbose': True, # display messages, progress
490             'bkmodel': True  # model and correct for background effects
491           })
492
493         # function parameter assignments
494         verbose = options('verbose')
495         bkmodel = options('bkmodel')
496
497       except OptionsError as err:
498         print(' --> OptionsError:', err)
499         raise MontageError('Failed keyword assigment in SubField.Merge().')
500
501       # check for master directory
502       master_dir = os.path.join(self.location,'master')
503       if not os.path.exists(master_dir):
504         raise MontageError('No `master` directory detected for SubField '
505           '`{}`.'.format(self.location))
506
507       # change directories
508       os.chdir(master_dir)
509
510       # create folder structure
511       for subdir in ['corrected','projected','differences','final']:
512         path = os.path.join(master_dir,subdir)
513         if not os.path.exists(path):
514           os.makedirs(path)
515
516       # run Mosaic on `site` mosaics
517       Mosaic(resolution, '.', verbose=verbose, bkmodel=bkmodel)
518
519   class Field:
520     """
521     Mosaic manager for `Montage`.
522     """
523     def __init__(self, center, sides, grid, subgrid, **kwargs):
524       """
525       Initialize a Field centered on `center` (array-like of length two with
526       right ascension and declination in decimal degrees) and side lengths
527       `sides` (array-like of length two in units of decimal degrees in
528       right ascension and declination respectively). `grid` (array-like of
529       length two) specifieds the grid layout (e.g., (4,4) says 4x4) to
530       subdivide the Field. The `subgrid` is the same but pertaining to the
531       further subdivision of each SubField into a grid layout of `sites`.
532       See SubField class.
533
534       kwargs = {
535           'verbose': True   , # display message, progress
536           'survey' : 'DSS'  , # 2MASS, SDSS, DSS
537           'band'   : 'DSS2B'  # filter `band` pertaining to `survey`
538         }
539       """
540       try:
541         # function parameter defaults
542         options = Options( kwargs,
543           {
544             'verbose': True   , # display message, progress
545             'survey' : 'DSS'  , # 2MASS, SDSS, DSS
546             'band'   : 'DSS2B'  # filter `band` pertaining to `survey`
547           })
548
549         # function parameter assignments
550         verbose = options('verbose')
551         survey  = options('survey')
552         band    = options('band')
```

```
553
554         except OptionsError as err:
555           print(' --> OptionsError:', err)
556           raise MontageError('Failed keyword assignment in Field().')
557
558         # available bands for each survey
559         bands = {
560             # Two-Micron All-Sky Survey
561             '2MASS': ['J', 'H', 'K'],
562
563             # Sloan Digital Sky Survey
564             'SDSS': ['U','G','R','I','Z'],
565
566             # STScI Digitized Sky Survey
567             'DSS': ['DSS1B','DSS1R','DSS2B','DSS2R','DSS2IR','Quick-V']
568           }
569
570         # check for appropriate survey, band
571         if survey not in bands:
572           raise MontageError("'{}' was not a recognized survey for "
573           'the Montage Python module.'.format(survey))
574         if band not in bands[survey]:
575           raise MontageError("'{}' was not a recognized filter band "
576           'for the '{}' survey.'.format(band, survey))
577
578         # check arguments
579         if ( not hasattr(center, '__iter__') or
580           not hasattr(sides, '__iter__') or not hasattr(grid, '__iter__') or
581           not hasattr(subgrid, '__iter__')):
582           raise MontageError('Field() expects array-like arguments.')
583         if ( len(center) != 2 or len(sides) != 2 or len(grid) != 2 or
584             len(subgrid) != 2 ):
585           raise MontageError('Field() expects arguments to be length two.')
586
587         if verbose:
588           stdout.write('\n Setting up {:d}x{:d} Field around ({:.2f}, '
589             '{:.2f}) ... '.format(grid[0], grid[1], center[0], center[1]))
590           stdout.flush()
591
592         # SolveGrid()
593         self.ra_centers, self.dec_centers = SolveGrid(sides, grid)
594         # relative to Field 'center':
595         self.ra_centers  += center[0]
596         self.dec_centers += center[1]
597
598         # side lengths for SubField's
599         self.sub_sides = ( sides[0] / grid[0], sides[1] / grid[1] )
600
601         # set current directory to 'Field directory'.
602         self.location = os.path.abspath('.')
603
604         # name SubField directories
605         self.folders = [os.path.join(self.location,'SubField-{:03d}'.format(a+1))
606           for a in range(len(self.ra_centers) * len(self.dec_centers)) ]
607
608         # create SubField directories
609         for folder in self.folders:
610           if not os.path.exists(folder):
611             os.makedirs(folder)
612
613         # zip together all ra, dec pairs
614         self.sub_centers = [ (ra, dec)
615           for ra in self.ra_centers for dec in self.dec_centers ]
616
617         # initialize empty list of SubField's
618         self.subfields = []
619
620         # initalize all SubField's
621         for a, folder, sub_center in zip(range(len(self.folders)), self.folders,
622           self.sub_centers):
623
624           if verbose:
625             stdout.write('\n Initializing {} ... '.format(
626               os.path.basename(folder)))
627             stdout.flush()
628
629           # change directories
630           os.chdir(folder)
631
632           # create SubField
633           self.subfields.append( SubField(sub_center, self.sub_sides,
634             subgrid, survey = survey, band = band) )
635
636           if verbose: stdout.write('done')
637
638         if verbose:
639           stdout.write('\n')
640           stdout.flush()
641
642       def ArchiveList(self, **kwargs):
643         """
644         Run 'ArchiveList()' on all SubFields. The only keyword option is
```

```
645         `verbose` (default: True).
646         """
647         try:
648           # function parameter defaults
649           options = Options( kwargs,
650             {
651               'verbose': True # display messages, progress
652             })
653
654           # function parameter assignments
655           verbose = options('verbose')
656
657         except OptionsError as err:
658           print(' --> OptionsError:', err)
659           raise MontageError('Failed keyword assignment in '
660           'Field.ArchiveList().')
661
662         if verbose:
663           stdout.write('\n Running ArchiveList() on all SubFields ... ')
664           stdout.write('\n ' + '-' * 70 + '\n')
665           stdout.flush()
666
667         # Run ArchiveList() on all SubField's
668         for a, subfield in enumerate(self.subfields):
669
670           if verbose:
671             stdout.write('\n Running ArchiveList() on {} ... '
672               .format(os.path.basename(self.folders[a])))
673             stdout.flush()
674
675           # run ArchiveList()
676           subfield.ArchiveList(verbose = False)
677
678           if verbose: stdout.write('done')
679
680         if verbose:
681           stdout.write('\n')
682           stdout.flush()
683
684       def ArchiveExec(self, **kwargs):
685         """
686         Run `ArchiveExec()` on all SubFields. The only keyword option is
687         `verbose` (default: True).
688         """
689         try:
690           # function parameter defaults
691           options = Options( kwargs,
692             {
693               'verbose': True # display messages, progress
694             })
695
696           # function parameter assignments
697           verbose = options('verbose')
698
699         except OptionsError as err:
700           print(' --> OptionsError:', err)
701           raise MontageError('Failed keyword assignment in '
702           'Field.ArchiveExec().')
703
704         if verbose:
705           stdout.write('\n Running ArchiveExec() on all SubFields ... ')
706           stdout.write('\n' + '-' * 70 + '\n')
707           stdout.flush()
708
709         # Run ArchiveExec() on all SubField's
710         for a, subfield in enumerate(self.subfields):
711
712           if verbose:
713             stdout.write('\n Running ArchiveExec() on `{}`: {} / {} ... '
714               .format(os.path.basename(self.folders[a]), a + 1,
715               len(self.folders)))
716             stdout.flush()
717
718           # run ArchiveList()
719           subfield.ArchiveExec(verbose = False)
720
721           if verbose: stdout.write('done')
722
723         if verbose:
724           stdout.write('\n')
725           stdout.flush()
726
727       def Build(self, resolution, **kwargs):
728         """
729         Build(resolution, **kwargs):
730
731         Run the build process for all SubFields in this Field. See the
732         documentation for Montage.Mosaic() and SubField.Build().
733
734         kwargs = {
735             'verbose': True, # display messages, progress
736             'bkmodel': True  # run background modelling procedures.
737           }
```

```
738         """
739         try:
740           # function parameter defaults
741           options = Options( kwargs,
742             {
743               'verbose': True, # display messages, progress
744               'bkmodel': True  # run background modelling procedure
745             })
746 
747           # function parameter assignments
748           verbose = options('verbose')
749           bkmodel = options('bkmodel')
750 
751         except OptionsError as err:
752           print(' --> OptionsError:', err)
753           raise MontageError('Failed keyword assignment in '
754           'Field.ArchiveExec().')
755 
756         if verbose:
757           stdout.write('\n Running Build() on all SubField's ... ')
758           stdout.write('\n' + '=' * 70 + '\n')
759           stdout.write(       '=' * 70 + '\n')
760           stdout.flush()
761 
762         # Run ArchiveExec() on all SubField's
763         for a, subfield in enumerate(self.subfields):
764 
765           if verbose:
766             stdout.write('\n Running Build() on '{}': {} / {} ... '
767               .format(os.path.basename(self.folders[a]), a + 1,
768               len(self.folders)))
769             stdout.write('\n' + '=' * 70 + '\n')
770             stdout.flush()
771 
772           # run ArchiveList()
773           subfield.Build(verbose = verbose, bkmodel = bkmodel)
774 
775         if verbose:
776           stdout.write('\n')
777           stdout.flush()
778 
779       def Collect(self, **kwargs):
780         """
781         Run Collect() on all SubFields of this Field.
782         """
783         try:
784           # function parameter defaults
785           options = Options( kwargs,
786             {
787               'verbose': True, # display messages, progress
788             })
789 
790           # function parameter assignments
791           verbose = options('verbose')
792 
793         except OptionsError as err:
794           print(' --> OptionsError:', err)
795           raise MontageError('Failed keyword assignment in '
796           'Field.ArchiveExec().')
797 
798         if verbose:
799           stdout.write('\n Running Collect() on all SubField's ... ')
800           stdout.write('\n' + '=' * 70 + '\n')
801           stdout.write(       '=' * 70 + '\n')
802           stdout.flush()
803 
804         # Run ArchiveExec() on all SubField's
805         for a, subfield in enumerate(self.subfields):
806 
807           if verbose:
808             stdout.write('\n Running Collect() on '{}': {} / {} ... '
809               .format(os.path.basename(self.folders[a]), a + 1,
810               len(self.folders)))
811             stdout.write('\n' + '=' * 70 + '\n')
812             stdout.flush()
813 
814           # run ArchiveList()
815           subfield.Collect( verbose=verbose )
816 
817         if verbose:
818           stdout.write('\n')
819           stdout.flush()
820 
821       def Merge(self, resolution, **kwargs):
822         """
823         Merge(resolution, **kwargs):
824 
825         Run Merge() on all SubFields of this Field. See SubField.Merge().
826         """
827         try:
828           # function parameter defaults
829           options = Options( kwargs,
830             {
```

```
            'verbose': True, # display messages, progress
          })

        # function parameter assignments
        verbose = options('verbose')

      except OptionsError as err:
        print(' --> OptionsError:', err)
        raise MontageError('Failed keyword assignment in '
        'Field.ArchiveExec().')

      if verbose:
        stdout.write('\n Running Merge() on all SubField's ... ')
        stdout.write('\n' + '=' * 70 + '\n')
        stdout.write(       '=' * 70 + '\n')
        stdout.flush()

      # Run ArchiveExec() on all SubField's
      for a, subfield in enumerate(self.subfields):

        if verbose:
          stdout.write('\n Running Merge() on '{}': {} / {} ... '
            .format(os.path.basename(self.folders[a]), a + 1,
            len(self.folders)))
          stdout.write('\n' + '=' * 70 + '\n')
          stdout.flush()

        # run ArchiveList()
        subfield.Merge( verbose=verbose )

      if verbose:
        stdout.write('\n')
        stdout.flush()

    def Finalize(self, resolution, **kwargs):
      """
      Finalize(resolution, **kwargs):

      Collect all SubField/master mosaics into a single folder and
      run Mosaic() on them for a single final image.
      """
      try:

        # function parameter defaults
        options = Options( kwargs,
          {
            'verbose': True # display messages, progress
          })

        # function parameter assignments
        verbose = options('verbose')

      except OptionsError as err:
        print(' --> OptionsError:', err)
        raise MontageError('Failed keyword assignment on Field.Finalize().')

      # relocate to Field directory
      os.chdir(self.location)

      # create master directory
      master_dir = os.path.join(self.location, 'master')
      if not os.path.exists(master_dir):
        os.makedirs(master_dir)

      # create master directory sub-structure
      for subdir in ['images','projected','differences','corrected','final']:
        path = os.path.join(master_dir,subdir)
        if not os.path.exists(path):
          os.makedirs(path)


      # collect master mosaics
      for a, folder in enumerate(self.folders):

        if verbose:
          stdout.write('\n Copying image from {} ... '.format(
            os.path.basename(folder)))
          stdout.flush()

        # SubField mosaic
        image = os.path.join(folder,'master/final/mosaic.fits')

        # ensure Merge() was run
        if not os.path.exists(image):
          raise MontageError('Missing 'master' mosaic image in '{}'.'
          .format(os.path.basename(folder)))

        # copy image to Field 'master' image directory
        sh.copy(image, os.path.join(master_dir, 'images/mosaic-{}.fits'
          .format(a + 1)))

        if verbose: stdout.write('done')
```

```
924       if verbose:
925         stdout.write('\n')
926         stdout.flush()
927
928       # change directories to `master`
929       os.chdir(master_dir)
930
931       # run Mosaic() on all SubField `master`s
932       Mosaic(resolution, '.', verbose=verbose, bkmodel=bkmodel)
```

C.14 .. SLiPy . Observatory

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# slipy/SLiPy/Observatory.py
"""
Classes for defining observatory parameters (similar to the IRAF task).
"""

from astropy import units as u

# there is no need for an `ObservatoryError` class ...

class Observatory:
    """
    The Abstract base class for Observatory types.
    """
    def __init__(self):
        raise TypeError('The Observatory base class should not be '
        'instantiated on its own.')

    def __repr__(self):
        return str(self)

    def __str__(self):
        res = '' if not hasattr(self, 'resolution') else 'resolution = ' + str(self.resolution)
        return  ('name       = {}\nlongitude  = {}\nlatitude   = {}\naltitude   = {}\n'
                'timezone   = {}\n'.format(self.name, self.longitude, self.latitude,
                self.altitude, self.timezone)) + res + '\n'

class OHP(Observatory):
    """
    The Observatoire de Haute-Provence, France.
    """
    def __init__(self):
        self.name       = 'Observatoire de Haute-Provence'
        self.longitude  = 356.28667 * u.degree # West
        self.latitude   = 43.9308334 * u.degree # North
        self.altitude   = 650 * u.meter
        self.timezone   = 1 * u.hourangle
        self.resolution = 42000 * u.dimensionless_unscaled

#
# All of the below observatory parameters have been taken directly from IRAF!
#

class KPNO(Observatory):
  """
  Kitt Peak National Observatory
  """
  def __init__(self):
    self.name      = "Kitt Peak National Observatory"
    self.longitude = 111.6 * u.degree # West
    self.latitude  = 31.9633333333 * u.degree # North
    self.altitude  = 2120. * u.meter
    self.timezone  = 7 * u.hourangle

class WIYN(Observatory):
  """
  WIYN Observatory
  """
  def __init__(self):
    self.name      = "WIYN Observatory"
    self.longitude = 111.6 * u.degree # West
    self.latitude  = 31.9633333333 * u.degree # North
    self.altitude  = 2120. * u.meter
    self.timezone  = 7 * u.hourangle

class CTIO(Observatory):
  """
  Cerro Tololo Interamerican Observatory
  """
  def __init__(self):
    self.name      = "Cerro Tololo Interamerican Observatory"
    self.longitude = 70.815 * u.degree # West
    self.latitude  = -30.16527778 * u.degree # North
    self.altitude  = 2215. * u.meter
    self.timezone  = 4 * u.hourangle

class LASILLA(Observatory):
  """
  European Southern Observatory: La Silla
  """
  def __init__(self):
    self.name      = "European Southern Observatory: La Silla."
    self.longitude = 70.73 * u.degree # West
    self.latitude  = -28.7433333333 * u.degree # North
    self.altitude  = 2347 * u.meter
    self.timezone  = 4 * u.hourangle

```

```
89    class PARANAL(Observatory):
90      """
91      European Southern Observatory: Paranal
92      """
93      def __init__(self):
94        self.name      = "European Southern Observatory: Paranal"
95        self.longitude = 70.4033333333 * u.degree # West
96        self.latitude  = -23.375 * u.degree # North
97        self.altitude  = 2635 * u.meter
98        self.timezone  = 4 * u.hourangle
99
100   class LICK(Observatory):
101     """
102     Lick Observatory
103     """
104     def __init__(self):
105       self.name      = "Lick Observatory"
106       self.longitude = 121.636666667 * u.degree # West
107       self.latitude  = 37.3433333333 * u.degree # North
108       self.altitude  = 1290 * u.meter
109       self.timezone  = 8 * u.hourangle
110
111   # Observatory entry from a conversation with Craig Foltz 8/20/97.
112   # Name was changed and the "mmt" entry was removed.
113   class MMTO(Observatory):
114     """
115     MMT Observatory
116     """
117     def __init__(self):
118       self.name      = "MMT Observatory"
119       self.longitude = 110.885 * u.degree # West
120       self.latitude  = 31.6883333333 * u.degree # North
121       self.altitude  = 2600 * u.meter
122       self.timezone  = 7 * u.hourangle
123
124   class CFHT(Observatory):
125     """
126     Canada-France-Hawaii Telescope
127     """
128     def __init__(self):
129       self.name      = "Canada-France-Hawaii Telescope"
130       self.longitude = 155.471666667 * u.degree # West
131       self.latitude  = 19.8266666667 * u.degree # North
132       self.altitude  = 4215 * u.meter
133       self.timezone  = 10 * u.hourangle
134
135   class LAPALMA(Observatory):
136     """
137     Roque de los Muchachos, La Palma
138     """
139     def __init__(self):
140       self.name      = "Roque de los Muchachos, La Palma."
141       self.longitude = 17.88 * u.degree # West
142       self.latitude  = 28.7583333333 * u.degree # North
143       self.altitude  = 2327 * u.meter
144       self.timezone  = 0 * u.hourangle
145
146   class MSO(Observatory):
147     """
148     Mt. Stromlo Observatory
149     """
150     def __init__(self):
151       self.name      = "Mt. Stromlo Observatory"
152       self.longitude = 210.975666667 * u.degree # West
153       self.latitude  = -34.67935 * u.degree # North
154       self.altitude  = 767 * u.meter
155       self.timezone  = -10 * u.hourangle
156
157   class SSO(Observatory):
158     """
159     Siding Spring Observatory
160     """
161     def __init__(self):
162       self.name      = "Siding Spring Observatory"
163       self.longitude = 210.938805556 * u.degree # West
164       self.latitude  = -30.7266388889 * u.degree # North
165       self.altitude  = 1149 * u.meter
166       self.timezone  = -10 * u.hourangle
167
168   class AAO(Observatory):
169     """
170     Anglo-Australian Observatory
171     """
172     def __init__(self):
173       self.name      = "Anglo-Australian Observatory"
174       self.longitude = 210.933913889 * u.degree # West
175       self.latitude  = -30.7229611111 * u.degree # North
176       self.altitude  = 1164 * u.meter
177       self.timezone  = -10 * u.hourangle
178
179   class MCDONALD(Observatory):
180     """
```

```
    McDonald Observatory
    """
    def __init__(self):
      self.name      = "McDonald Observatory"
      self.longitude = 104.0216667 * u.degree # West
      self.latitude  = 30.6716667 * u.degree # North
      self.altitude  = 2075 * u.meter
      self.timezone  = 6 * u.hourangle

  class LCO(Observatory):
    """
    Las Campanas Observatory
    """
    def __init__(self):
      self.name      = "Las Campanas Observatory"
      self.longitude = 70.7016666667 * u.degree # West
      self.latitude  = -28.9966666667 * u.degree # North
      self.altitude  = 2282 * u.meter
      self.timezone  = 4 * u.hourangle

  # Submitted by Alan Koski 1/13/93
  class MTBIGELOW(Observatory):
    """
    Catalina Observatory: 61 inch telescope
    """
    def __init__(self):
      self.name      = "Catalina Observatory: 61 inch telescope"
      self.longitude = 110.731666667 * u.degree # West
      self.latitude  = 32.4166666667 * u.degree # North
      self.altitude  = 2510. * u.meter
      self.timezone  = 7 * u.hourangle

  # Revised by Daniel Durand 2/23/93
  class DAO(Observatory):
    """
    Dominion Astrophysical Observatory
    """
    def __init__(self):
      self.name      = "Dominion Astrophysical Observatory"
      self.longitude = 123.416666667 * u.degree # West
      self.latitude  = 48.5216666667 * u.degree # North
      self.altitude  = 229 * u.meter
      self.timezone  = 8 * u.hourangle

  # Submitted by Patrick Vielle 5/4/93
  class SPM(Observatory):
    """
    Observatorio Astronomico Nacional, San Pedro Martir
    """
    def __init__(self):
      self.name      = "Observatorio Astronomico Nacional, San Pedro Martir."
      self.longitude = 115.486944444 * u.degree # West
      self.latitude  = 31.0291666667 * u.degree # North
      self.altitude  = 2830 * u.meter
      self.timezone  = 7 * u.hourangle

  # Submitted by Patrick Vielle 5/4/93
  class TONA(Observatory):
    """
    Observatorio Astronomico Nacional, Tonantzintla
    """
    def __init__(self):
      self.name      = "Observatorio Astronomico Nacional, Tonantzintla."
      self.longitude = 98.3138888889 * u.degree # West
      self.latitude  = 19.0327777778 * u.degree # North
      self.altitude  = 8 * u.hourangle

  # Submitted by Don Hamilton 8/18/93
  class PALOMAR(Observatory):
    """
    The Hale Telescope
    """
    def __init__(self):
      self.name      = "The Hale Telescope"
      self.longitude = 116.863 * u.degree # West
      self.latitude  = 33.356 * u.degree # North
      self.altitude  = 1706. * u.meter
      self.timezone  = 8 * u.hourangle

  # Submitted by Pat Seitzer 10/31/93
  class MDM(Observatory):
    """
    Michigan-Dartmouth-MIT Observatory
    """
    def __init__(self):
      self.name      = "Michigan-Dartmouth-MIT Observatory"
      self.longitude = 111.616666667 * u.degree # West
      self.latitude  = 31.95 * u.degree # North
      self.altitude  = 1938.5 * u.meter
      self.timezone  = 7 * u.hourangle

  # Submitted by Ignacio Ferrin 9/1/94
```

```
class NOV(Observatory):
  """
  National Observatory of Venezuela
  """
  def __init__(self):
    self.name      = "National Observatory of Venezuela"
    self.longitude = 70.8666666667 * u.degree # West
    self.latitude  = 8.79 * u.degree # North
    self.altitude  = 3610 * u.meter
    self.timezone  = 4 * u.hourangle

# Submitted by Alan Welty 10/28/94
class BMO(Observatory):
  """
  Black Moshannon Observatory
  """
  def __init__(self):
    self.name      = "Black Moshannon Observatory"
    self.longitude = 78.005 * u.degree # West
    self.latitude  = 40.9216666667 * u.degree # North
    self.altitude  = 738 * u.meter
    self.timezone  = 5 * u.hourangle

# Submitted by Biwei JIANG 11/28/95
class BAO(Observatory):
  """
  Beijing XingLong Observatory
  """
  def __init__(self):
    self.name      = "Beijing XingLong Observatory"
    self.longitude = 242.425 * u.degree # West
    self.latitude  = 40.3933333333 * u.degree # North
    self.altitude  = 950. * u.meter
    self.timezone  = -8 * u.hourangle

# From Astronomical Almanac 1996
class KECK(Observatory):
  """
  W. M. Keck Observatory
  """
  def __init__(self):
    self.name      = "W. M. Keck Observatory"
    self.longitude = 155.478333333 * u.degree # West
    self.latitude  = 19.8283333333 * u.degree # North
    self.altitude  = 4160 * u.meter
    self.timezone  = 10 * u.hourangle

# Submitted by Lina Tomasella 6/11/96:  Padova Astronomical Obs., Asiago, Italy.
class EKAR(Observatory):
  """
  Mt. Ekar 182 cm. Telescope
  """
  def __init__(self):
    self.name      = "Mt. Ekar 182 cm. Telescope"
    self.longitude = 348.418866667 * u.degree # West
    self.latitude  = 45.8485888889 * u.degree # North
    self.altitude  = 1413.69 * u.meter
    self.timezone  = -1 * u.hourangle

# Submitted by Michael Ledlow 8/8/96
class APO(Observatory):
  """
  Apache Point Observatory
  """
  def __init__(self):
    self.name      = "Apache Point Observatory"
    self.longitude = 105.82 * u.degree # West
    self.latitude  = 32.78 * u.degree # North
    self.altitude  = 2798. * u.meter
    self.timezone  = 7 * u.hourangle

# Submitted by Michael Ledlow 8/8/96
class LOWELL(Observatory):
  """
  Lowell Observatory
  """
  def __init__(self):
    self.name      = "Lowell Observatory"
    self.longitude = 111.535 * u.degree # West
    self.latitude  = 35.0966666667 * u.degree # North
    self.altitude  = 2198. * u.meter
    self.timezone  = 7 * u.hourangle

# Submitted by S.G. Bhargavi 8/12/96
class VBO(Observatory):
  """
  Vainu Bappu Observatory
  """
  def __init__(self):
    self.name      = "Vainu Bappu Observatory"
    self.longitude = 281.1734 * u.degree # West
    self.latitude  = 12.57666 * u.degree # North
```

```
      self.altitude  = 725. * u.meter
      self.timezone  = -5.5 * u.hourangle

# Submitted by S. Giridhar 6/28/03
class IAO(Observatory):
  """
  Indian Astronomical Observatory, Hanle
  """
  def __init__(self):
    self.name      = "Indian Astronomical Observatory, Hanle"
    self.longitude = 281.03583 * u.degree # West
    self.latitude  = 32.7794 * u.degree # North
    self.altitude  = 4500 * u.meter
    self.timezone  = -5.5 * u.hourangle

# Submitted by Doug Mink 1/6/97
class FLWO(Observatory):
  """
  Whipple Observatory
  """
  def __init__(self):
    self.name      = "Whipple Observatory"
    self.longitude = 110.8775 * u.degree # West
    self.latitude  = 31.6809444444 * u.degree # North
    self.altitude  = 2320 * u.meter
    self.timezone  = 7 * u.hourangle

class FLWO1(Observatory):
  """
  Whipple Observatory
  """
  def __init__(self):
    self.name      = "Whipple Observatory"
    self.longitude = 110.8775 * u.degree # West
    self.latitude  = 31.6809444444 * u.degree # North
    self.altitude  = 2320 * u.meter
    self.timezone  = 7 * u.hourangle

# Submitted by Doug Mink 1/6/97
class ORO(Observatory):
  """
  Oak Ridge Observatory
  """
  def __init__(self):
    self.name      = "Oak Ridge Observatory"
    self.longitude = 71.5581444444 * u.degree # West
    self.latitude  = 42.5052611111 * u.degree # North
    self.altitude  = 184 * u.meter
    self.timezone  = 5 * u.hourangle

# Submitted by Claudia Vilega Rodriques 12/12/97
class LNA(Observatory):
  """
  Laboratorio Nacional de Astrofisica - Brazil
  """
  def __init__(self):
    self.name      = "Laboratorio Nacional de Astrofisica - Brazil"
    self.longitude = 45.5825 * u.degree # West
    self.latitude  = -21.4655555556 * u.degree # North
    self.altitude  = 1864 * u.meter
    self.timezone  = 3 * u.hourangle

# Submitted by John Memzies 12/31/99
class SAAO(Observatory):
  """
  South African Astronomical Observatory
  """
  def __init__(self):
    self.name      = "South African Astronomical Observatory"
    self.longitude = 339.189305556 * u.degree # West
    self.latitude  = -31.6205555556 * u.degree # North
    self.altitude  = 1798 * u.meter
    self.timezone  = -2 * u.hourangle

# Submitted by Jorge Federico Gonzalez 12/10/98
class CASLEO(Observatory):
  """
  Complejo Astronomico El Leoncito, San Juan
  """
  def __init__(self):
    self.name      = "Complejo Astronomico El Leoncito, San Juan."
    self.longitude = 69.3 * u.degree # West
    self.latitude  = -30.2008333333 * u.degree # North
    self.altitude  = 2552 * u.meter
    self.timezone  = 3 * u.hourangle

# Submitted by Jorge Federico Gonzalez 12/10/98
class BOSQUE(Observatory):
  """
  Estacion Astrofisica Bosque Alegre, Cordoba
  """
  def __init__(self):
```

```
      self.name      = "Estacion Astrofisica Bosque Alegre, Cordoba."
      self.longitude = 64.5458333333 * u.degree # West
      self.latitude  = -30.4016666667 * u.degree # North
      self.altitude  = 1250 * u.meter
      self.timezone  = 3 * u.hourangle

  # Submitted by Ilian Iliev 1/19/99
  class ROZHEN(Observatory):
    """
    National Astronomical Observatory Rozhen - Bulgaria
    """
    def __init__(self):
      self.name      = "National Astronomical Observatory Rozhen - Bulgaria."
      self.longitude = 335.256111111 * u.degree # West
      self.latitude  = 41.6930555556 * u.degree # North
      self.altitude  = 1759 * u.meter
      self.timezone  = -2 * u.hourangle

  # Submitted by Bill Vacca 7/14/99
  class IRTF(Observatory):
    """
    NASA Infrared Telescope Facility
    """
    def __init__(self):
      self.name      = "NASA Infrared Telescope Facility"
      self.longitude   = 155.471999 * u.degree # West
      self.latitude  = 19.826218 * u.degree # North
      self.altitude  = 4168 * u.meter
      self.timezone  = 10 * u.hourangle

  # Submitted by Andy Layden 7/16/99
  class BGSUO(Observatory):
    """
    Bowling Green State Univ Observatory
    """
    def __init__(self):
      self.name      = "Bowling Green State Univ Observatory."
      self.longitude = 83.6591666667 * u.degree # West
      self.latitude  = 41.3783333333 * u.degree # North
      self.altitude  = 225. * u.meter
      self.timezone  = 5 * u.hourangle

  # Submitted by Oliver-Mark Cordes 8/5/99
  class DSAZ(Observatory):
    """
    Deutsch-Spanisches Observatorium Calar Alto - Spain
    """
    def __init__(self):
      self.name      = "Deutsch-Spanisches Observatorium Calar Alto - Spain."
      self.longitude = 2.54625 * u.degree # West
      self.latitude  = 37.2236111111 * u.degree # North
      self.altitude  = 2168 * u.meter
      self.timezone  = -1 * u.hourangle

  # Submitted by Matilde Fernandez 2/2/99
  class CA(Observatory):
    """
    Calar Alto Observatory
    """
    def __init__(self):
      self.name      = "Calar Alto Observatory"
      self.longitude = 2.54625 * u.degree # West
      self.latitude  = 37.2236111111 * u.degree # North
      self.altitude  = 2168 * u.meter
      self.timezone  = -1 * u.hourangle

  # Submitted by Oliver-Mark Cordes 8/5/99
  class HOLI(Observatory):
    """
    Observatorium Hoher List (Universitaet Bonn) - Germany
    """
    def __init__(self):
      self.name      = "Observatorium Hoher List (Universitaet Bonn) - Germany."
      self.longitude = 6.85 * u.degree # West
      self.latitude  = 50.16276 * u.degree # North
      self.altitude  = 541 * u.meter
      self.timezone  = -1 * u.hourangle

  # Submitted by Steven Majewski 8/27/99
  class LMO(Observatory):
    """
    Leander McCormick Observatory
    """
    def __init__(self):
      self.name      = "Leander McCormick Observatory"
      self.longitude = 78.5233333333 * u.degree # West
      self.latitude  = 38.0333333333 * u.degree # North
      self.altitude  = 264 * u.meter
      self.timezone  = 5 * u.hourangle

  # Submitted by Steven Majewski 8/27/99
  class FMO(Observatory):
```

```
    """
    Fan Mountain Observatory
    """
    def __init__(self):
      self.name      = "Fan Mountain Observatory"
      self.longitude = 78.6933333333 * u.degree # West
      self.latitude  = 37.8783333333 * u.degree # North
      self.altitude  = 566 * u.meter
      self.timezone  = 5 * u.hourangle

  # Submitted by Kim K. McLeod 10/13/1999
  class WHITIN(Observatory):
    """
    Whitin Observatory, Wellesley College
    """
    def __init__(self):
      self.name      = "Whitin Observatory,Wellesley College"
      self.longitude = 71.305833 * u.degree # West
      self.latitude  = 42.295 * u.degree # North
      self.altitude  = 32 * u.meter
      self.timezone  = 5 * u.hourangle

  # Submitted by Nuno Peixinho 6/7/2000
  # Parameters for the Sierra Nevada Observatory (Spain)
  class OSN(Observatory):
    """
    Observatorio de Sierra Nevada
    """
    def __init__(self):
      self.name      = "Observatorio de Sierra Nevada"
      self.longitude = 3.38472222222 * u.degree # West
      self.latitude  = 37.0641666667 * u.degree # North
      self.altitude  = 2896 * u.meter
      self.timezone  = -1 * u.hourangle

  # Gemini Observatory - Submitted by Inger Jorgensen
  #
  # Gemini-North
  # These values are from the aerial survey made Sept 25, 1996
  #    http://www.ifa.hawaii.edu/mko/coordinates.html

  class GEMININORTH(Observatory):
    """
    Gemini North Observatory
    """
    def __init__(self):
      self.name      = "Gemini North Observatory"
      self.longitude = 155.46904675 * u.degree # West
      self.latitude  = 19.8238015 * u.degree # North
      self.altitude  = 4213.4 * u.meter
      self.timezone  = 10 * u.hourangle

  # Gemini-South
  # The coordinates are from GPS measurements and the elevation
  # is from the Gemini Web pages http://www.gemini.edu/public

  # class GEMINI-SOUTH(Observatory):
    # """
    # Gemini South Observatory
    # """
    #def __init__(self):
      #   self.name      = "Gemini South Observatory"
      #   self.longitude = 70.7233333333 * u.degree # West
      #   self.latitude  = -29.7716666667 * u.degree # North
      #   self.altitude  = 2737. * u.meter
      #   self.timezone  = 4 * u.hourangle

  # Corrected coords from Bryan Miller, 5/18/2006
  class GEMINISOUTH(Observatory):
    """
    Gemini South Observatory
    """
    def __init__(self):
      self.name      = "Gemini South Observatory"
      self.longitude = 70.7366933333 * u.degree # West
      self.latitude  = -29.75925 * u.degree # North
      self.altitude  = 2722. * u.meter
      self.timezone  = 4 * u.hourangle

  # ESO

  class LASILLA(Observatory):
    """
    European Southern Observatory: La Silla
    """
    def __init__(self):
      self.name      = "European Southern Observatory: La Silla."
      self.longitude = 70.73 * u.degree # West
      self.latitude  = -28.7433333333 * u.degree # North
      self.altitude  = 2347 * u.meter
      self.timezone  = 4 * u.hourangle
```

```
641   class PARANAL(Observatory):
642     """
643     European Southern Observatory: Paranal
644     """
645     def __init__(self):
646       self.name      = "European Southern Observatory: Paranal."
647       self.longitude = 70.4033333333 * u.degree # West
648       self.latitude  = -23.375 * u.degree # North
649       self.altitude  = 2635 * u.meter
650       self.timezone  = 4 * u.hourangle
651
652   # The following additional entries were suppied by Peter Weilbacher
653   # weilbach@uni-sw.gwdg.de on 28 Jan 2002 13:12:59 -0700.
654
655   class ESONTT(Observatory):
656     """
657     European Southern Observatory, NTT, La Silla
658     """
659     def __init__(self):
660       self.name      = "European Southern Observatory, NTT, La Silla."
661       self.longitude = 70.7317422222 * u.degree # West
662       self.latitude  = -28.7448777778 * u.degree # North
663       self.altitude  = 2375 * u.meter
664       self.timezone  = 4 * u.hourangle
665
666   class ESO36M(Observatory):
667     """
668     European Southern Observatory, 3.6m Telescope, La Silla
669     """
670     def __init__(self):
671       self.name      = "European Southern Observatory, 3.6m Telescope, La Silla."
672       self.longitude = 70.7296127778 * u.degree # West
673       self.latitude  = -28.7428294444 * u.degree # North
674       self.altitude  = 2400 * u.meter
675       self.timezone  = 4 * u.hourangle
676
677   class ESOVLT(Observatory):
678     """
679     European Southern Observatory, VLT, Paranal
680     """
681     def __init__(self):
682       self.name      = "European Southern Observatory, VLT, Paranal."
683       self.longitude = 70.4022 * u.degree # West
684       self.latitude  = -24.6253 * u.degree # North
685       self.altitude  = 2648 * u.meter
686       self.timezone  = 4 * u.hourangle
687
688
689   # Submited by Giovanni Catanzaro, 7/17/03.
690   class SLN(Observatory):
691     """
692     SLN - Catania Astrophysical Observatory
693     """
694     def __init__(self):
695       self.name      = "SLN - Catania Astrophysical Observatory."
696       self.longitude = 345.026666667 * u.degree # West
697       self.latitude  = 37.6916666667 * u.degree # North
698       self.altitude  = 1725. * u.meter
699       self.timezone  = -1 * u.hourangle
700
701
702   # Submited by Ahmet Devlen, 4/21/04
703   class EUO(Observatory):
704     """
705     Ege University Observatory
706     """
707     def __init__(self):
708       self.name      = "Ege University Observatory"
709       self.longitude = -26.725 * u.degree # West
710       self.latitude  = 38.3983333333 * u.degree # North
711       self.altitude  = 795. * u.meter
712       self.timezone  = 2 * u.hourangle
713
714   # Submitted by Zeki Aslan 8/15/05 who said the "tno" entry was wrong.
715   class TNO(Observatory):
716     """
717     Turkiye National Observatory
718     """
719     def __init__(self):
720       self.name      = "Turkiye National Observatory"
721       self.longitude = -29.6469444444 * u.degree # West
722       self.latitude  = 36.8244444444 * u.degree # North
723       self.altitude  = 2555. * u.meter
724       self.timezone  = 2 * u.hourangle
725
726   class TUG(Observatory):
727     """
728     TUBITAK National Observatory, Turkey
729     """
730     def __init__(self):
731       self.name      = "TUBITAK National Observatory, Turkey."
732       self.longitude = -29.6666666667 * u.degree # West
```

```
      self.latitude  = 36.825 * u.degree # North
      self.altitude  = 2547. * u.meter
      self.timezone  = -2 * u.hourangle


# Submited by Ricky Patterson for Vatican Obs. Research Group, 6/15/04
class MGO(Observatory):
  """
  Mount Graham Observatory
  """
  def __init__(self):
    self.name      = "Mount Graham Observatory"
    self.longitude = 109.891666667 * u.degree # West
    self.latitude  = 32.7016666667 * u.degree # North
    self.altitude  = 3181 * u.meter
    self.timezone  = 7 * u.hourangle

# Submited by Jeewan C. Bandey 7/28/05
# Changed to W longitude MJF 4/1/06)
#   (E) longitude = 79.45639
class ARIES(Observatory):
  """
  Aryabhatta Research Institute of Observational Sciences
  """
  def __init__(self):
    self.name      = "Aryabhatta Research Institute of Observational Sciences."
    self.longitude = 280.54361 * u.degree # West
    self.latitude  = 29.36 * u.degree # North
    self.altitude  = 1950. * u.meter
    self.timezone  = -5.5 * u.hourangle

# Submitted by Eduardo Fern?ndez Laj?s 10/28/05
class OALP(Observatory):
  """
  Observatorio Astronomico de La Plata
  """
  def __init__(self):
    self.name      = "Observatorio Astronomico de La Plata"
    self.longitude = 57.9322995 * u.degree # West
    self.latitude  = -33.0932488889 * u.degree # North
    self.altitude  = 20. * u.meter
    self.timezone  = 3. * u.hourangle

# Submitted by Leslie F. Brown 7/29/06
class OLIN(Observatory):
  """
  Connecticut College - Olin Observatory
  """
  def __init__(self):
    self.name      = "Connecticut College - Olin Observatory"
    self.longitude = 72.1052777778 * u.degree # West
    self.latitude  = 41.3788888889 * u.degree # North
    self.altitude  = 85 * u.meter
    self.timezone  = 5 * u.hourangle

# Submitted by Pat van Heerden 11/20/06
class BOYDEN(Observatory):
  """
  Boyden Observatory
  """
  def __init__(self):
    self.name      = "Boyden Observatory"
    self.longitude = 332.594444444 * u.degree # West
    self.latitude  = -28.9611111111 * u.degree # North
    self.altitude  = 1387 * u.meter
    self.timezone  = -2 * u.hourangle

# Submitted by Mike Yang 8/19/09
class LULIN(Observatory):
  """
  Lulin Observatory
  """
  def __init__(self):
    self.name      = "Lulin Observatory"
    self.longitude = 240.873333333 * u.degree # West
    self.latitude  = 23.4683333333 * u.degree # North
    self.altitude  = 2862. * u.meter
    self.timezone  = -8 * u.hourangle

# Submitted by Mairan Teodoro 1/27/10
class SOAR(Observatory):
  """
  Southern Astrophysical Research Telescope
  """
  def __init__(self):
    self.name      = "Southern Astrophysical Research Telescope."
    self.longitude = 70.7337222222 * u.degree # West
    self.latitude  = -29.762 * u.degree # North
    self.altitude  = 2738 * u.meter
    self.timezone  = 4 * u.hourangle

# Submitted from iraf.net 4/12/10
```

```
class BAKER(Observatory):
  """
  Baker Observatory
  """
  def __init__(self):
    self.name      = "Baker Observatory"
    self.longitude = 93.0417472222 * u.degree # West
    self.latitude  = 37.398625 * u.degree # North
    self.altitude  = 418.2 * u.meter
    self.timezone  = 6 * u.hourangle

# Added MJF 6/1/2010
class HET(Observatory):
  """
  McDonald Observatory - Hobby-Eberly Telescope
  """
  def __init__(self):
    self.name      = "McDonald Observatory - Hobby-Eberly Telescope."
    self.longitude = 104.014722222 * u.degree # West
    self.latitude  = 30.68144444 * u.degree # North
    self.altitude  = 2026 * u.meter
    self.timezone  = 6 * u.hourangle

# Submitted by Robert D. Collier 9/1/10
class JCDO(Observatory):
  """
  Jack C. Davis Observatory, Western Nevada College
  """
  def __init__(self):
    self.name      = "Jack C. Davis Observatory, Western Nevada College"
    self.longitude = 119.790666667 * u.degree # West
    self.latitude  = 39.1857222222 * u.degree # North
    self.altitude  = 1534 * u.meter
    self.timezone  = 8 * u.hourangle

# Submitted by mas_nomi1711@yahoo.com 3/16/12
class LNO(Observatory):
  """
  Langkawi National Observatory
  """
  def __init__(self):
    self.name      = "Langkawi National Observatory"
    self.longitude = 260.218888889 * u.degree # West
    self.latitude  = 6.30694444444 * u.degree # North
    self.altitude  = 111 * u.meter
    self.timezone  = -8 * u.hourangle
```

C.15 .. SLiPy . Plot

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# AstroPython/Astro/Plot.py
"""
Plotting facility for astronomy.
"""
import matplotlib as mpl
from matplotlib import pyplot as plt

from .. import SlipyError
from ..Framework.Options import Options, OptionsError
from . import Fits, Spectrum

mpl.rcParams['figure.facecolor'] = 'w'
plt.ion()

class PlotError(SlipyError):
  """
  Exception specific to Plot module.
  """
  pass

class SPlot:
    """
    SPlot( spectrum, **kwargs )

    Spectrum Plot - Plot the data in `spectrum`.
    """
    def __init__(self, spectrum, **kwargs):
        """
        Assign `options` in `kwargs` and initialize the figure.
        """
        try:
            # available options
            self.options = Options( kwargs,
              {
                'marker': 'b-'          , # marker for plot
                'label' : 'unspecified' , # label for data
                'usetex': False           # pdflatex setting
              })

            # assign options
            self.usetex   = self.options('usetex')
            self.ylimits  = []
            self.gridv    = None
            self.yargs    = []
            self.xargs    = []
            self.largs    = []
            self.targs    = []
            self.xkwargs  = {}
            self.ykwargs  = {}
            self.tkwargs  = {}
            self.lkwargs  = {}
            self.txargs   = []
            self.txkwargs = []

            if type(spectrum) is not Spectrum.Spectrum:
                raise PlotError('Splot expects type `Spectrum`!')

            # data in `list` allows for overplotting
            self.data = [ spectrum.data ]
            self.wave = [ spectrum.wave ]

            if spectrum.data.unit:
              self.yargs = [str(spectrum.data.unit)]

            if spectrum.wave.unit:
              self.xargs = [str(spectrum.wave.unit)]

            # `name` always retains `label`
            self.name   = self.options('label')

            # `label` and `marker`s same as data
            self.label  = [ self.options('label')  ]
            self.marker = [ self.options('marker') ]

            # set x limits to the data
            self.xlimits = [ spectrum.wave.min().value,
              spectrum.wave.max().value ]

        except OptionsError as err:
            print(' --> OptionsError:', err.msg)
            raise PlotError('Failed to construct Splot.')

        # initialize the figure
        self.fig = plt.figure("Spectral-Plot (SLiPy)")
        self.ax  = self.fig.add_subplot(111)
        self.draw()

```

```
90        def xlim(self, xmin, xmax ):
91            """
92            Handle to pyplot.xlim
93            """
94            self.xlimits = [ xmin, xmax ]
95            self.ax.set_xlim(xmin, xmax)
96
97        def ylim(self, ymin, ymax ):
98            """
99            Handle to pyplot.ylim
100           """
101           self.ylimits = [ ymin, ymax ]
102           self.ax.set_ylim(ymin, ymax)
103
104       def xlabel(self, *args, **kwargs ):
105           """
106           x axis label.
107           """
108           self.xargs = args
109           self.xkwargs = kwargs
110           self.ax.set_xlabel( *args, **kwargs )
111
112       def ylabel(self, *args, **kwargs ):
113           """
114           y axis label.
115           """
116           self.yargs = args
117           self.ykwargs = kwargs
118           self.ax.set_ylabel( *args, **kwargs )
119
120       def title(self, *args, **kwargs ):
121           """
122           title for plot.
123           """
124           self.targs = args
125           self.tkwargs = kwargs
126           self.ax.set_title( *args, **kwargs )
127
128       def legend(self, *args, **kwargs):
129           """
130           legend for plot.
131           """
132           self.largs = args
133           self.lkwargs = kwargs
134           plt.legend( *args, **kwargs )
135
136       def text(self, *args, **kwargs):
137           """
138           display text over plot.
139           """
140           self.txargs.append( args )
141           self.txkwargs.append( kwargs )
142           self.ax.text( *args, **kwargs )
143
144       def txtclear(self):
145           """
146           Clear all ‘text‘ from figure.
147           """
148           self.txargs = []
149           self.txkwargs = []
150           self.draw()
151
152       def markers(self, *args):
153           """
154           Reassign the values for the ‘marker‘s in the figure. The number
155           of arguments must equal the number of spectra in the figure. This
156           starts out as one, but will increase for ever SPlot.overlay().
157           """
158           if len(args) != len(self.data):
159               raise PlotError('{} arguments were given but there are {} '
160               'spectra plotted in this figure!'.format(len(args), len(self.data)))
161
162           for a, mark in enumerate(args):
163               if type(mark) is not str:
164                   raise PlotError('Arguments given to SPlot.markers() must be '
165                   '{} but argument #{} was {}'.format(type(''), a+1, type(mark)))
166
167           self.marker = list(args)
168
169       def __build(self, picker = False):
170           """
171           Make the plot.
172           """
173
174           if picker:
175               self.restore()
176               self.ax.plot(self.wave[0], self.data[0], self.marker[0],
177                   label = self.label[0], picker = True)
178
179           else:
180               for x, y, m, l in zip(self.wave, self.data, self.marker, self.label):
181                 self.ax.plot(x, y, m, label=l)
```

```
182
183             if self.xargs or self.xkwargs:
184                 self.xlabel( *self.xargs, **self.xkwargs )
185
186             if self.yargs or self.ykwargs:
187                 self.ylabel( *self.yargs, **self.ykwargs )
188
189             if self.targs or self.tkwargs:
190                 self.title( *self.targs, **self.tkwargs )
191
192             if self.largs or self.lkwargs:
193                 self.legend( *self.largs, **self.lkwargs )
194
195             if self.txargs or self.txkwargs:
196                 for args, kwargs in zip(self.txargs, self.txkwargs):
197                   self.ax.text( *args, **kwargs )
198
199             if self.xlimits:
200                 self.xlim( *self.xlimits )
201
202             if self.ylimits:
203                 self.ylim( *self.ylimits )
204
205             if self.gridv:
206                 self.grid(self.gridv)
207
208             if self.usetex:
209                 plt.rc('text', usetex=True)
210                 plt.rc('font', family='serif')
211
212         def refresh(self):
213             """
214             pyplot.draw()
215             """
216             plt.draw()
217
218         def draw(self, picker = False):
219           """
220           Re-build the plot
221           """
222           self.ax.clear()
223           self.__build(picker = picker)
224           plt.draw()
225
226         def close(self):
227             """
228             Close the plot.
229             """
230             plt.close("Spectral-Plot (SLiPy)")
231
232         def grid(self, value):
233             """
234             Show grid on plot.
235             """
236             self.gridv = value
237             plt.grid(value)
238
239         def save(self, filename):
240             """
241             Save plot to `filename`. Must have extension for formatting.
242             """
243             if type(filename) is not str:
244                 raise PlotError('`filename` should be of type str.')
245             if len(filename.split('.')) < 2:
246                 raise PlotError('`filename` needs an extension.')
247
248             plt.savefig(filename, format=filename.split('.')[-1])
249
250         def xoffset(self, value):
251             """
252             Toggle the offset for the x axis
253             """
254             plt.gca().get_xaxis().get_major_formatter().set_useOffset(value)
255
256         def yoffset(self, value):
257             """
258             Toggle the offset for the y axis
259             """
260             plt.gca().get_yaxis().get_major_formatter().set_useOffset(value)
261
262         def tight_layout(self):
263             """
264             pyplot.tight_layout()
265             """
266             plt.tight_layout()
267
268         def overlay(self, *splots ):
269             """
270             Overlay (add) spectra to this plot from other `splots`.
271             """
272
273             for a, plot in enumerate(splots):
```

```
274
275                 # check data type
276                 if type(plot) is not SPlot:
277                     raise PlotError('Splot.overlay expects '
278                     'type Splot! (from argument {})'.format(a))
279
280                 # add data
281                 self.data   += plot.data
282                 self.wave   += plot.wave
283                 self.marker += plot.marker
284                 self.label  += plot.label
285
286       def restore(self):
287           """
288           Restore self.data and self.wave from possible `overlay`s.
289           """
290           self.data   = [ self.data[0]   ]
291           self.wave   = [ self.wave[0]   ]
292           self.marker = [ self.marker[0] ]
293           self.label  = [ self.label[0]  ]
294
295   def desired( plot ):
296     """
297     Helper function for Iterate. Prompts user to keep `plot`;
298     returns True or False.
299     """
300     # draw the plot
301     plot.draw()
302     # prompt the user for input
303     prompt = input('\r\033[K keep -> `{}` (y/[n]/x)?: '
304         .format(plot.name)).strip()
305     # insure valid response
306     while True:
307       if prompt not in ['y','n','','x']:
308         # invalid input, prompt again
309         print('\r\033[K `{}` was not a recognized response.'.format(prompt))
310         prompt = input('\033[2A\r\033[K keep -> `{}` (y/[n]/x)?: '
311             .format(plot.name)).strip()
312       else:
313         # clear the error message
314         print('\r\033[K\033[1A')
315         break
316
317     if prompt in ['n', '']:
318       return False
319
320     elif prompt in ['y']:
321       return True
322
323     else:
324       # the user input `x`
325       raise KeyboardInterrupt('\r\033[K User exitted early, saving results.')
326
327
328   def Iterate( *plots, **kwargs ):
329     """
330     Iterate( *plots, **kwargs ):
331
332     Iterate thru `plots` to inspect data, the user marks `plots` of
333     interest. The function returns a list of `names` marked.
334     """
335     try:
336       options = Options( kwargs,
337         {
338           'keep' : 'name' # alternatively, `plot`
339         })
340
341       keep = options('keep')
342
343       if keep not in ['name', 'plot']:
344         raise PlotError('Iterate expects either `name` or `plot` for '
345           'keyword argument `keep`.')
346
347       # check input arguments
348       for plot in plots:
349         if not hasattr(plot, 'draw'):
350           raise PlotError('Iterate expects objects to '
351             'have a `draw` method.')
352         if not hasattr(plot, 'name'):
353           raise PlotError('Iterate expects objects to '
354             'have a `name` method.')
355
356       # clear some space
357       print('\n')
358
359       keepers = []
360       for a, plot in enumerate(plots):
361         print('\033[2A\r\033[K Showing plot {} of {} ... '
362             .format(a, len(plots)) )
363         if desired( plot ):
364           if keep == 'name':
365             keepers.append( plot.name )
```

```
366           elif keep == 'plot':
367             keepers.append( plot )
368
369       return keepers
370
371     except OptionsError as err:
372       print(' --> OptionsError:', err.msg)
373       raise PlotError('Failed to initialize Iterate.')
374
375     except KeyboardInterrupt as x:
376       print(x)
377       return keepers
```

C.16 .. SLiPy . Profile

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# slipy/SLiPy/Profile.py

"""
Profile fitting tasks for spectra.
"""
import numpy as np
from scipy.optimize import curve_fit
from scipy.interpolate.interpolate import interp1d
from scipy.special import wofz as w
from scipy.integrate import simps as Integrate

from matplotlib import pyplot as plt
from matplotlib.lines import Line2D
from matplotlib import widgets

from astropy import units as u
from astropy.constants import m_e, e, c

from .. import SlipyError

from .Observatory import Observatory
from .Spectrum import Spectrum, SpectrumError
from .Plot import SPlot, PlotError

from ..Framework.Options import Options, OptionsError
from ..Framework.Measurement import Measurement
from ..Algorithms.Functions import Gaussian, Lorentzian, Voigt, InvertedLorentzian
from ..Algorithms.KernelFit import KernelFit1D
from ..Data import Atomic

class ProfileError(SlipyError):
  """
  Exception specific to Profile module.
  """
  pass

def Pick(event):
    """
    Used to hand selection events.
    """
    if isinstance(event.artist, Line2D):
        # get the x, y data from the pick event
        thisline = event.artist
        xdata = thisline.get_xdata()
        ydata = thisline.get_ydata()
        ind = event.ind
        # update the selection dictionary
        global selected
        selected['wave'].append(np.take(xdata,ind)[0])
        selected['data'].append(np.take(ydata,ind)[0])
        # display points as a visual aid
        plt.scatter(np.take(xdata,ind)[0], np.take(ydata,ind)[0],
            marker = 'o', s=75, edgecolor='red', facecolor='None', lw=2)
        plt.draw()

# empty selection dictionary is filled with the Select() function
selected = {'wave': [], 'data': []}

def Select(splot):
    """
    Select points from the `splot`. This should be of type SPlot
    (or it can optionally be a Spectrum type, for which a SPlot will be
    created). The splot will be rendered and the user clicks on the
    figure. When finished, return to the terminal prompt. A dictionary is
    returned with two entries, `wave` and `data`, representing the x-y
    locations selected by the user. This can always be retrieved later by
    accessing the module member `Profile.selected`.
    """
    if type(splot) is Spectrum:
        splot = SPlot(splot)

    elif type(splot) is not SPlot:
        raise ProfileError('Select() requires either a Spectrum or SPlot '
        'object as an argument')

    # reset the selection dictionary
    global selected
    selected = { 'wave': [], 'data': []}

    splot.draw(picker = True)

    splot.fig.canvas.mpl_connect('pick_event', Pick)

    input(' Press <Return> after making your selections ... ')
    return selected

def AutoFit(splot, function = InvertedLorentzian, params = None):
```

```
90        """
91        Given `splot` of type SPlot, the user selects four points on the
92        spectrum and a parameterized function is fit (an inverted Lorentzian by
93        default). Optionally, `splot` can be of type spectrum and a basic SPlot
94        will be created for you. If the user gives an alternative `function`,
95        `params` (parameters) must be provided. `params` is to be the first guess,
96        `p0` given to scipy...curve_fit; the user can provide them expicitely,
97        or in the form of functions with the template `function(xarray, yarray)`
98        where `xarray` and `yarray` are the `wave` and `data` arrays extracted
99        between the two points selected by the user.
100       """
101
102       print(' Please select four points identifying the spectral line.')
103       print(' Outer points mark the domain of the line.')
104       print(' Inner points mark the sample of the line to fit.')
105
106       # make selections
107       selected = Select(splot)
108
109       if len( selected['wave'] ) != 4:
110           raise ProfileError('Exactly 4 locations should be selected for '
111           'the Profile.AutoFit() routine!')
112
113       # order the selected wavelength locations
114       points = selected['wave']
115       points.sort()
116
117       # get data and wavelength arrays
118       if type(splot) is SPlot:
119           wave, data = splot.wave[0], splot.data[0]
120       else:
121           wave, data = splot.wave, splot.data
122
123       # extract the domains `selected` (i.e., the sample interval)
124       x_inner = wave[np.where(np.logical_and(points[1] < wave, wave < points[2]))]
125       x_outer = wave[np.where(np.logical_and(points[0] < wave, wave < points[3]))]
126       y_inner = data[np.where(np.logical_and(points[1] < wave, wave < points[2]))]
127       y_outer = data[np.where(np.logical_and(points[0] < wave, wave < points[3]))]
128
129       if function.__name__ == 'InvertedLorentzian':
130           # First guess for default behavior
131           params = [ y_inner.min().value, x_inner[ y_inner.argmin() ].value,
132               (x_inner[-1] - x_inner[0]).value / 2]
133
134       elif not params:
135           # the user gave a different function but not parameters!
136           raise ProfileError('The user must provide `params` when giving an '
137           'alternative `function` in Profile.Fit()!')
138
139       else:
140
141           if not hasattr(params, '__iter__'):
142               raise ProfileError('`params` must be an iterable type in '
143               'Profile.AutoFit()!')
144
145           try:
146               for a, parameter in enumerate(params):
147                   if type(parameter) is type(InvertedLorentzian):
148                       # replace parameter with function evaluation
149                       params[a] = parameter(x_inner, y_inner)
150
151           except TypeError as err:
152               print(' --> TypeError:', err)
153               raise ProfileError('Profile.AutoFit() failed to call user functions '
154               'correctly in `params`!')
155
156       # fit a parameterized curve
157       coeff, var_matrix = curve_fit(
158               function,      # function to call, default is InvertedLorentzian
159               x_inner.value, # domain (without units)
160               y_inner.value, # data (without units)
161               p0 = params    # list of parameters to try as a first guess
162           )
163
164       # display visual aids ...
165       # evaluation of the fit profile over larger domain
166       plt.plot(x_outer, function(x_outer.value, *coeff) * y_outer.unit,
167           'b--', linewidth = 4)
168       plt.plot(x_inner, function(x_inner.value, *coeff) * y_inner.unit,
169           'r-', linewidth = 4)
170
171       # return the larger domain, evaluated and of type Spectrum
172       return Spectrum(function(x_outer.value, *coeff) * y_outer.unit, x_outer)
173
174
175   def Extract(splot, kernel = Gaussian, **kwargs):
176     """
177     Select locations in the `splot` figure, expected to be of type SPlot.
178     Exactly four points should be selected. These are used to extract a
179     line profile from the spectrum plotted in the splot figure. The inner
180     section is used for the line, and the outer selection is used to model
181     the continuum; these, respectively, and both returned as Spectrum objects.
```

```
    The gap is jumped using 1D interpolation (scipy...interp1d).
    """
    try:

      options = Options( kwargs, {
        'kind'      : 'cubic' , # given to scipy...interp1d for continuum
        'bandwidth' : 0.1*u.nm  # user should provide this!
        # 'rms'       : False     # measure the RMS of the line, continuum
        })

      kind      = options('kind')
      bandwidth = options('bandwidth')
      # rms       = options('rms')

    except OptionsError as err:
      print(' --> OptionsError:', err)
      raise ProfileError('Unrecognized option given to Extract()!')

    print(' Please select four points identifying the spectral line.')
    print(' Outer intervals sample the continuum.')
    print(' Center interval contains the line.')

    # make selections
    selected = Select(splot)

    if len( selected['wave'] ) != 4:
      raise ProfileError('Exactly 4 locations should be selected for '
      'the profile modeling to work!')

    # order the selected wavelength locations
    wave = selected['wave']
    wave.sort()

    # create `line` profile
    xl = splot.wave[0].copy()
    yl = splot.data[0].copy()
    yl = yl[ xl[ xl < wave[2] ] > wave[1] ]
    xl = xl[ xl[ xl < wave[2] ] > wave[1] ]
    line = Spectrum(yl, xl)

    # extract continuum arrays
    xc = splot.wave[0].copy()
    yc = splot.data[0].copy()
    # inside outer-most selections
    yc = yc[ xc[ xc < wave[3] ] > wave[0] ]
    xc = xc[ xc[ xc < wave[3] ] > wave[0] ]
    # keep wavelengths whole domain for later
    xx = xc.copy()
    yy = yc.copy()
    # but not the line
    yc = yc[np.where(np.logical_or(xc < wave[1], xc > wave[2]))]
    xc = xc[np.where(np.logical_or(xc < wave[1], xc > wave[2]))]

    # use `kernel smoothing` to model the continuum
    model = KernelFit1D(xc, yc, kernel = kernel, bandwidth = bandwidth)

    # interpolate to cross `the gap`
    interp = interp1d(xc, model.mean(xc), kind = kind)

    # continuum model outside the line
    cont_outside = interp(xc)

    # continuum inside the line
    cont_inside = interp(xl)

    # continuum model for whole domain
    cont_domain = interp(xx)

    # build a spectrum from the arrays
    continuum = Spectrum(cont_domain, xx)

    # display visual aid
    plt.plot(xx, cont_domain, 'r--', linewidth = 3)
    plt.fill_between(xl, yl, cont_inside, color = 'blue', alpha = 0.25)
    plt.draw()

    # if not rms:
    #     return line, continuum

    continuum.rms  = np.sqrt( np.sum( (cont_outside * yc.unit - yc)**2 ) / len(yc) )
    continuum.rms *= continuum.data.unit

    return line, continuum


  def OpticalDepth(line, continuum, error=None, boost=None):
    """
    Given an absorption `line` and its background `continuum`, compute the
    apparent optical depth Spectrum. Both the line and the continuum must
    be of Spectrum type. The continuum will be resampled to the pixel space of the
    line if it is not currently.

    If provided an `error` spectrum, it should either have the same units as the `line`
```

```
    or be in percent units (they should have identical wavelength arrays). An upper and
    lower uncertainty will be computed by adding and subtracting before the calculation.
    Further, if an `rms` value is attached to the `continuum` giving the percent error in
    the continuum fit, this will be added in quadrature to the error spectrum before hand.

    `boost` allows for artificially increasing the resolution by resampling to more pixels.
    If the spectrum is not of sufficiently high resolution, the integration could suffer
    numerical errors. If provided, this should be a percentage to increase the resolution
    (e.g., `boost=2` would double the resolution by interpolating in between the pixels).

    This function returns a Spectrum with `upper` and `lower` bounds attached.
    """
    if not isinstance(line, Spectrum):
      raise ProfileError('OpticalDepth() expects the first argument to be of '
      '`Spectrum` type!')

    if not isinstance(continuum, Spectrum):
      raise ProfileError('OpticalDepth() expects the second argument to be of '
      '`Spectrum` type!')

    line      = line.copy()
    continuum = continuum.copy()

    if error:
      if not isinstance(error, Spectrum):
        raise ProfileError('OpticalDepth() expects the `error` to be of `Spectrum` type!')
      if error.data.unit not in [line.data.unit, u.percent]:
        raise ProfileError('OpticalDepth() expects the `error` spectrum to have either '
        'the same units as the `line` or units of `percent`.')

      error = error.copy()

      if error.data.unit == u.percent:
        if np.logical_and(error.data.value < 0, error.data.value > 100).any():
          raise ProfileError('OpticalDepth() was given an `error` spectrum in '
          'units of `percent` with values outside of 0 and 100!')

        error.resample(line)
        error.data = (error.data * line.data).to(line.data.unit)

    if hasattr(continuum, 'rms'):
      if not hasattr(continuum.rms, 'unit') or (continuum.rms.unit not in
        [continuum.data.unit, u.percent]):
        raise ProfileError('OpticalDepth() expects a `continuum` with an `rms` '
        'attribute to have the same units or `percent` units.')

      rms = continuum.rms
      if rms.unit == u.percent:
        rms = rms * continuum.data
        rms = rms.to(continuum.data.unit)

      if not error:
        error = rms

      else:
        # add the two error components in quadrature
        error.resample(line)
        error = Spectrum(np.sqrt(rms**2 + error.data**2) * line.data.unit, line.wave)

    # boost the resolution of the specrum if requested
    if boost: line.resample( line.wave[0], line.wave[-1], len(line) * boost )

    # resample the continuum spectrum, nothing happens if they are already the same
    continuum.resample(line)

    # compute the apparent optical depth
    tau = Spectrum(np.log((continuum / line).data.decompose()), line.wave)

    if error:
      tau.lower = Spectrum(np.log((continuum / (line - error)).data.decompose()), line.wave)
      tau.upper = Spectrum(np.log((continuum / (line + error)).data.decompose()), line.wave)

    return tau

  def EquivalentWidth(line, continuum, error=None, boost=None):
    """
    Given an absorption `line` and its background `continuum`, compute the
    equivalent width, `W`, of the feature. Both the line and the continuum must
    be of Spectrum type. The continuum will be resampled to the pixel space of the
    line if it is not currently.

    If provided an `error` spectrum, it should either have the same units as the `line`
    or be in percent units (they should have identical wavelength arrays). An upper and
    lower uncertainty will be computed by adding and subtracting before the calculation.
    Further, if an `rms` value is attached to the `continuum` giving the percent error in
    the continuum fit, this will be added in quadrature to the error spectrum before hand.

    `boost` allows for artificially increasing the resolution by resampling to more pixels.
    If the spectrum is not of sufficiently high resolution, the integration could suffer
    numerical errors. If provided, this should be a percentage to increase the resolution
    (e.g., `boost=2` would double the resolution by interpolating in between the pixels).

```

```
    Integration is performed using the composite Simpson's rule (scipy.integrate.simps)
    This function returns a 'Measurement' object (..Framework.Measurement.Measurement).
    """
    tau = OpticalDepth(line, continuum, error=error, boost=boost)
    W   = Integrate(1 - np.exp(-tau.data.decompose()), tau.wave) * line.wave.unit

    if error:

      upper = Integrate(1 - np.exp(-tau.lower.data.decompose()), tau.wave) * line.wave.unit
      lower = Integrate(1 - np.exp(-tau.upper.data.decompose()), tau.wave) * line.wave.unit

      # join upper/lower into +/- set
      uncertainty = np.array([(upper - W).value, (lower - W).value]) * tau.wave.unit

    else: uncertainty = None

    return Measurement(W, error = uncertainty, name = 'Equivalent Width',
      notes = 'Measured using Profile.EquivalentWidth() from SLiPy')


  def ColumnDensity(line, continuum, ion, error=None, boost=None, weakline=None, integrate=True):
    """
    Given an absorption 'line' and its background 'continuum', compute the
    apparent column density, 'N', of the feature. Both the line and the continuum must
    be of Spectrum type. The continuum will be resampled to the pixel space of the
    line if it is not currently. An 'ion' must be provided (type Atomic.Ion) with
    members, 'fvalue' (oscillator strength) and 'wavelength' (will be converted to
    Angstroms).

    If provided an 'error' spectrum, it should either have the same units as the 'line'
    or be in percent units (they should have identical wavelength arrays). An upper and
    lower uncertainty will be computed by adding and subtracting before the calculation.
    Further, if an 'rms' value is attached to the 'continuum' giving the percent error in
    the continuum fit, this will be added in quadrature to the error spectrum before hand.

    With 'integrate' set to True (the default) the integrated apparent column density will
    be returned (as type Measurement and in units of cm-2). Otherwise, a Spectrum is
    returned as a function of wavelength and the '.upper' and '.lower' bounds are attached.

    'boost' allows for artificially increasing the resolution by resampling to more pixels.
    If the spectrum is not of sufficiently high resolution, the integration could suffer
    numerical errors. If provided, this should be a percentage to increase the resolution
    (e.g., 'boost=2' would double the resolution by interpolating in between the pixels).

    With a 'weakline' provided (the results of this function on the weaker absorption
    line for a doublet) a correction gives an approximate 'true' column density (attached
    to the returned Measurement as '.true'), see Savage & Sembach, 1991.

    Integration is performed using the composite Simpson's rule (scipy.integrate.simps)
    This function returns a 'Measurement' object (..Framework.Measurement.Measurement).
    """
    if (not isinstance(ion, Atomic.Ion) or not hasattr(ion, 'fvalue') or
      not hasattr(ion, 'wavelength')): raise ProfileError("From ColumnDensity(), 'ion' is "
      "expected to be of type Atomic.Ion and have members 'fvalue' and 'wavelength'!")

    if weakline and not hasattr(weakline, 'unit'):
      raise ProfileError("From ColumnDensity(), 'weakline' is expected to have units!")

    const   = (1.13e12 / u.cm) / (ion.fvalue * ion.wavelength.to(u.cm))
    tau     = OpticalDepth(line, continuum, error=error, boost=boost)
    N       = tau       * const / ion.wavelength.to(u.AA).value
    N.upper = tau.upper * const / ion.wavelength.to(u.AA).value
    N.lower = tau.lower * const / ion.wavelength.to(u.AA).value

    if not integrate and not weakline:
      return N

    elif weakline and not integrate:
      raise ProfileError("From ColumnDensity(), you gave 'integrate' as False but we "
      "need to integrate to solve for the correct N using the 'weakline'!")

    upper = Integrate(N.upper.data, N.wave) / u.cm**2
    lower = Integrate(N.lower.data, N.wave) / u.cm**2
    N     = Integrate(N.data, N.wave) / u.cm**2
    uncertainty = np.array([(N - upper).value, (lower - N).value]) / u.cm**2

    N = Measurement(N, name="Column Density (Apparent)",
      error=uncertainty, notes="Measured using Profile.ColumnDensity() from SLiPy")

    if not weakline:
      return N

    # attach the 'correction' given the already computed 'weakline' column density.
    diff = np.log10(weakline.to(1 / u.cm**2).value) - np.log10(N.value)
    if diff < 0.0 or diff > 0.24:
      raise ProfileError("From ColumnDensity(), the difference between the doublet "
      "lines is too great to solve for a correction using published results!")

    # Table 4: Column Density Corrections for an Isolated Gaussian Component
    # Savage & Sembach (1991)
    table = np.array([  [0.000, 0.000], [0.010, 0.010], [0.020, 0.020], [0.030, 0.030],
      [0.040, 0.040], [0.050, 0.051], [0.060, 0.061], [0.070, 0.073], [0.080, 0.085],
      [0.090, 0.097], [0.100, 0.111], [0.110, 0.125], [0.120, 0.140], [0.130, 0.157],
```

```
460       [0.140, 0.175], [0.150, 0.195], [0.160, 0.217], [0.170, 0.243], [0.180, 0.273],
461       [0.190, 0.307], [0.200, 0.348], [0.210, 0.396], [0.220, 0.453], [0.230, 0.520],
462       [0.240, 0.600]])
463
464     N.correction = interp1d(table[:,0], table[:,1], kind='cubic')(diff)
465     N.true       = 10**(np.log10(weakline.to(1/u.cm**2).value) + N.correction) / u.cm**2
466     return N
467
468
469   class FittingGUI:
470     """
471     Graphical Interface (keeps references alive) for fitting analytical
472     profiles to spectral data.
473     """
474     def __init__(self, splot, obs=None, ion=None, function='Voigt', **kwargs):
475       """
476       Build widget elements based on a spectrum plotted in `splot` (type SPlot).
477       `splot` may also be a Spectrum object (for which a SPlot will be created).
478
479       Make initial guesses at parameters, build widgets, render the figure.
480       """
481       try:
482
483         options = Options( kwargs, {
484
485             'kind'      : 'cubic' , # given to interp1d for continuum
486             'bandwidth' : 0.1*u.nm, # user should provide this!
487           })
488
489         kind      = options('kind')
490         bandwidth = options('bandwidth')
491
492       except OptionsError as err:
493         print(' --> OptionsError:', err)
494         raise ProfileError('Unrecognized option given to FittingGUI.__init__()!')
495
496       if function not in ['Voigt', 'Lorentzian', 'Gaussian']:
497         raise ProfileError('The only currently implemented functions '
498         'for fitting profiles are `Voigt`, `Lorentzian` and `Gaussian`!')
499
500       # grab and/or create the SPlot
501       if isinstance(splot, Spectrum):
502         splot = SPlot(splot, marker='k-', label='spectrum')
503
504       elif not isinstance(splot, SPlot):
505         raise ProfileError('FittingGUI() expects the `splot` argument to '
506         'either be a Spectrum or a SPlot!')
507
508       # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
509       # if the profile `function` is `Voigt` and we have necessary parameters,
510       # show a preview for `b` and `N` and `v`.
511       self.any_line_parameters = True if obs or  ion else False
512       self.has_line_parameters = True if obs and ion else False
513       if self.any_line_parameters and not self.has_line_parameters:
514         raise ProfileError('From FittingGUI.__init__(), if the absoption line '
515         'parameters `obs` or `ion` are provided, both must be given!')
516
517       if self.has_line_parameters:
518
519         if function != 'Voigt':
520           raise ProfileError('From FittingGUI.__init__(), in order to compute the '
521           'broadening from the instrument profile of the `obs`ervatory and the '
522           'column density for the `ion`, it is expected that we try to fit a `Voigt` '
523           'profile `function`!')
524
525         if not isinstance(obs, Observatory):
526           raise ProfileError('From FittingGUI.__init__(), if `obs` is provided it should '
527           'be of type Observatory!')
528
529         if not hasattr(obs, 'resolution'):
530           raise ProfileError('From FittingGUI.__init__(), if an observatory `obs` is '
531           'provided, it needs to have a member `resolution`!')
532
533         if not hasattr(ion, 'wavelength') or not hasattr(ion.wavelength, 'unit'):
534           raise ProfileError('From FittingGUI.__init__(), the provided `ion` either '
535           'did not have a `wavelength` attribute or it has one without units!')
536
537         if not hasattr(ion, 'fvalue'):
538           raise ProfileError('From FittingGUI.__init__(), the provide `ion` does not '
539           'have an `fvalue` (oscillator strength) attribute!')
540
541         if hasattr(ion.fvalue, 'unit') and ion.fvalue.unit != u.dimensionless_unscaled:
542           raise ProfileError('From FittingGUI.__init__(), the osciallator strength, '
543           '`fvalue` for an ion must be a dimensionless Quantity!')
544
545         if not hasattr(ion, 'A') or not ion.A:
546           raise ProfileError('From FittingGUI.__init__(), the provided `ion` does not '
547           'have an `A` (transition probability) attribute!')
548
549         if hasattr(ion.A, 'unit') and ion.A.unit != u.Unit('s-1'):
550           raise ProfileError('From FittingGUI.__init__(), the transition probability, '
551           '`A`, from the `ion` must be in units of `s-1` if units are present!')
```

```
552
553           # the instrument broadening is from R = lambda / delta_lambda
554           # FWHM = 2 sqrt( 2 log 2) sigma for the Gaussian instrument profile
555           self.R = obs.resolution
556           self.sigma_instrument = (ion.wavelength / self.R) / (2 * np.sqrt(2 * np.log(2)))
557           self.sigma_instrument_squared = self.sigma_instrument.value ** 2
558
559           # save parameters for later
560           self.wavelength = ion.wavelength
561           self.fvalue     = ion.fvalue
562           self.A          = ion.A
563
564           # the FWHM of the intrinsic line profile (Lorentzian) is proportional to the
565           # transition probability (Einstein coeff.) `A`...
566           # convert from km s-1 to wavelength units
567           self.gamma = (ion.wavelength * (ion.wavelength * ion.A / (2 * np.pi)).to(u.km / u.s) /
568             c.si).to(ion.wavelength.unit).value
569
570           # the leading constant in the computation of `N` (per Angstrom per cm-2)
571           # self.leading_constant = (m_e.si * c.si / (np.sqrt(np.pi) * e.si**2 * ion.fvalue *
572           #   ion.wavelength.to(u.Angstrom))).value
573           self.leading_constant = 1 / (ion.fvalue * ion.wavelength.to(u.AA)**2 * np.pi *
574             e.emu**2 / (m_e.si * c.to(u.km/u.s)**2)).value
575
576           # setting `function` to `ModifiedVoigt` only makes a change in how the `Voigt`
577           # profile is evaluated by using `self.gamma` instead of self.Params[...]['Gamma']
578           function = 'ModifiedVoigt'
579
580         # - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
581
582         print('\n We need to extract the lines from the continuum before we '
583         'begin the fitting process.')
584
585         # extract the line, continuum, rms from the spectrum
586         self.line, self.continuum = Extract(splot, bandwidth=bandwidth, kind=kind)
587         self.rms = self.continuum.rms
588
589         print('\n Now select the peaks of each line to be fit.')
590         print(' Initial guesses will be made for each line markered.')
591         input(' Press <Return> after making your selections ... ')
592
593         # grab all the selected points
594         global selected
595         points = np.array([
596               [ entry.value for entry in selected['wave'] ],
597               [ entry.value for entry in selected['data'] ]
598           ])
599
600         # point pairs in ascending order by wavelength
601         points = points[:, points[0].argsort()]
602
603         # domain size of the line and the number of components
604         self.domainsize = self.line.wave.value[-1] - self.line.wave.value[0]
605         self.numlines   = np.shape(points)[1] - 4
606
607         if self.numlines < 1:
608           raise ProfileError('FittingGUI() expects at least one line to '
609           'be selected for fitting!')
610
611         # final spectral line profile is convolution of the LSP and the
612         # line profile function. Gaussian on Gaussian, Gaussian on Lorentzian,
613         # etc ... Set the functional form given requested profile function
614         self.Evaluate = self.SetFunction(function)
615
616         # initial guesses for parameters given the line locations,
617         # containing `L1`, `L2`, etc ... the values of which are themselves
618         # dictionaries of parameters (e.g., `FWHM`, `Depth`, etc...) whose values
619         # are the initial guesses for those parameters given the location of
620         # it`s peak and the number of peaks within the `line`s domain.
621         self.Params = {
622
623             'L' + str(line + 1) : self.Parameterize(function, loc)
624             for line, loc in enumerate(points[:, 2:-2].T)
625
626           }
627
628         # grab the actual Figure object and it's axis to keep references
629         self.splot = splot
630         self.fig   = splot.fig
631         self.ax    = splot.ax
632
633         # refresh image, but keep any changes in axis limits
634         splot.xlim( *splot.ax.get_xlim() )
635         splot.ylim( *splot.ax.get_ylim() )
636         splot.draw()
637
638         # bring up plot to make room for sliders
639         plt.subplots_adjust(bottom = 0.30)
640
641         # resample continuum onto line domain
642         self.continuum = self.continuum.copy()
643         self.continuum.resample(self.line)
```

```
644
645         # common domain for plotting, strip units, wavelengths from continuum
646         self.x         = self.continuum.wave.value
647         self.continuum = self.continuum.data.value
648
649         # copy of continuum allows for adjustments
650         self.y = self.continuum.copy()
651
652         # save the mid-level of the continuum to avoid over-calculation
653         self.continuum_level = self.continuum.mean()
654
655         # add plots of each line, keep dictionary of handles
656         self.Component = {
657
658           line : plt.plot(self.x,
659               self.y - self.Evaluate(self.x, **parameters), 'k--')[0]
660
661           for line, parameters in self.Params.items()
662         }
663
664         # add plot of the continuum
665         self.Continuum, = plt.plot(self.x, self.y, 'r-', lw=2)
666
667         # add plot of superposition of each component
668         self.Combination, = plt.plot(self.x, self.y - self.SuperPosition(), 'g-')
669
670         # fix the limits on plot
671         xmin, xmax = splot.ax.get_xlim()
672         ymin, ymax = splot.ax.get_ylim()
673         plt.axis([xmin, xmax, ymin, ymax])
674
675         # axes for parameter sliders
676         self.Axis = {
677
678           # key : axis     xpos , ypos + dy        , xsize, ysize
679           line  : plt.axes([0.10, 0.05 + (k+1) * 0.03, 0.65, 0.02], axisbg = 'white')
680           for k, line in enumerate( self.Params['L1'].keys() )
681         }
682
683         # add an axis to make adjustments to the continuum
684         self.Axis['Continuum'] = plt.axes([0.10, 0.05, 0.65, 0.02], axisbg='white')
685
686         # create the sliders for the parameters
687         self.Slider = {
688
689           # Slider `key` and widget
690           param : widgets.Slider(
691
692             self.Axis[param],    # which axis to put slider on
693             param,               # name of parameter (and the slider)
694             self.Minimum(param), # set the minimum of the slider
695             self.Maximum(param), # set the maximum of the slider
696             valinit = self.Params['L1'][param] # initial value
697           )
698
699           # create a slider for each parameter
700           for param in self.Params['L1'].keys()
701         }
702
703         # create the slider for the continuum (10% adjustments)
704         self.Slider['Continuum'] = widgets.Slider(self.Axis['Continuum'], 'Continuum',
705           0.90 * self.continuum_level, 1.10 * self.continuum_level,
706           valinit = self.continuum_level)
707
708         # connect sliders to update function
709         for slider in self.Slider.values():
710           slider.on_changed(self.Update)
711
712         # create axis for radio buttons
713         self.RadioAxis = plt.axes([0.85, 0.01, 0.1, 0.2],
714           axisbg = 'white', frameon = False)
715
716         # create the radio button widget
717         self.Radio = widgets.RadioButtons(self.RadioAxis,
718           tuple(['L' + str(i+1) for i in range(self.numlines)]), active = 0)
719
720         # connect the radio button to it's update function
721         self.Radio.on_clicked(self.ToggleComponent)
722
723         # set current component as the first line
724         self.current_component = 'L1'
725
726         # make currently selected line bold
727         self.Component['L1'].set_linewidth(2)
728
729         # variable allows for the radio buttons to not screw-up the sliders/graphs
730         # when switching between lines
731         self.stall = False
732
733         # add the initial text for `N` and `b` if applicable.
734         # text is along 20% below the y-axis
735         if self.has_line_parameters:
```

```
736         # self.b, self.N, self.v = self.solve_b(), self.solve_N(), self.solve_v()
737         self.preview = self.ax.text(xmin, ymin - 0.1 * (ymax - ymin),
738           'v: {:>10.4f}        km s-1          b: {:>10.4f}    km s-1'
739           '       W: {:>10.4f}\nN:   {:>10.4e}    cm-2      tau_0: {:>10.4f}'.format(
740           self.solve_v().value, self.solve_b().value, self.solve_W(), self.solve_N().value,
741           self.solve_tau0()), va = 'top')
742
743       # display the text
744       self.fig.canvas.draw_idle()
745
746     def Parameterize(self, function, loc):
747       """
748       Choose the initial parameters of ‘function‘ given the peak ‘loc‘.
749       """
750       if function == 'Voigt':
751         return {'Gamma': 0.10 * self.domainsize / self.numlines,
752           'Sigma': 0.20 * self.domainsize / self.numlines, 'Peak' : loc[0],
753           'Depth': self.continuum[ loc[0] ].value - loc[1]}
754
755       if function == 'Lorentzian':
756         return {'FWHM': 0.5 * self.domainsize / self.numlines, 'Peak': loc[0],
757           'Depth': self.continuum[ loc[0] ].value - loc[1]}
758
759       elif function == 'Gaussian':
760         return {'FWHM': 0.5 * self.domainsize / self.numlines, 'Peak': loc[0],
761           'Depth': self.continuum[ loc[0] ].value - loc[1]}
762
763       elif function == 'ModifiedVoigt':
764         return {'Sigma': (self.Maximum('Sigma') - self.Minimum('Sigma'))/2,
765           'Peak' : loc[0], 'Depth': self.continuum[ loc[0] ].value - loc[1]}
766
767       else:
768         raise ProfileError('From FittingGUI.Parameterize(), the only '
769         'currently implemented functions are the ‘Gaussian‘, ‘Lorentzian‘, and '
770         '‘Voigt‘ profiles!')
771
772     def SetFunction(self, function):
773       """
774       Return how to ‘Evaluate‘ the profile given the ‘function‘.
775       """
776       options = {'Voigt': self.__Voigt, 'Lorentzian': self.__Lorentzian,
777         'Gaussian': self.__Gaussian, 'ModifiedVoigt': self.__ModifiedVoigt}
778
779       if function not in options:
780         raise ProfileError('From FittingGUI.SetFunction(), the only '
781         'currently implemented functions are the ‘Voigt‘, ‘Lorentzian‘, and '
782         'the ‘Gaussian‘ profiles!!')
783
784       return options[function]
785
786     def __Gaussian(self, x, **params):
787       """
788       A Gaussian profile. See ..Algorithms.Functions.Gaussian
789       """
790       return Gaussian(x, params['Depth'], params['Peak'], params['FWHM'] / 2.3548200450309493)
791
792     def __Lorentzian(self, x, **params):
793       """
794       A Lorentzian profile. See ..Algorithms.Functions.Lorentzian
795       """
796       return params['Depth'] * Lorentzian(x, params['Peak'], params['FWHM'])
797
798     def __Voigt(self, x, **params):
799       """
800       The Voigt profile. See ..Algorithms.Functions.Voigt
801       """
802       return Voigt(x, params['Depth'], params['Peak'], params['Sigma'], params['Gamma'])
803
804     def __ModifiedVoigt(self, x, **params):
805       """
806       This is the Voigt profile, but ‘gamma‘ was already set.
807       """
808       return Voigt(x, params['Depth'], params['Peak'], params['Sigma'], self.gamma)
809
810     def SuperPosition(self):
811       """
812       Superposition of each line component (blended line)
813       """
814       combined = np.zeros(np.shape(self.x))
815       for parameters in self.Params.values():
816         combined += self.Evaluate(self.x, **parameters)
817       return combined
818
819     def Minimum(self, param):
820       """
821       Set the lower bound on the ‘param‘eter for its slider.
822       """
823       if param == 'Peak':
824         return self.x[0]
825
826       elif param == 'FWHM':
827         return 1e-6
```

```
828
829         elif param == 'Depth':
830           return 1e-6
831
832         elif param == 'Sigma':
833           if self.Evaluate != self.__ModifiedVoigt:
834             return 1e-6
835           else:
836             return self.sigma_instrument.value
837
838         elif param == 'Gamma':
839           return 1e-6
840
841         else:
842           raise ProfileError('From FittingGUI.Minimum(), `{}` is not '
843           'currently implemented as a parameter to set the minumum '
844           'for!'.format(param))
845
846       def Maximum(self, param):
847         """
848         Set the upper bound on the `param`eter for its slider.
849         """
850         if param == 'Peak':
851           return self.x[-1]
852
853         elif param == 'FWHM':
854           return 0.9 * self.domainsize
855
856         elif param == 'Depth':
857           return 1.5 * self.continuum.max()
858
859         elif param == 'Sigma':
860           return 0.9 * self.domainsize / self.numlines
861
862         elif param == 'Gamma':
863           return 0.9 * self.domainsize / self.numlines
864
865         else:
866           raise ProfileError('From FittingGUI.Maximum(), `{}` is not '
867           'currently implemented as a parameter to set the maximum '
868           'for!'.format(param))
869
870       def Update(self, val):
871         """
872         Cycle thru Sliders and update Parameter dictionary. Re-draw graphs.
873         """
874         # `self.stall` suppresses this procedure while inside `ToggleComponent`
875         if not self.stall:
876
877           # the currently selected line component
878           line = self.current_component
879
880           # update the appropriate parameters in the dictionary
881           for parameter, slider in self.Slider.items():
882             if parameter == 'Continuum':
883               self.y = self.continuum + (slider.val - self.continuum_level)
884             else:
885               self.Params[line][parameter] = slider.val
886
887           # update the appropriate graph data, based on new parameters
888           for line, parameters in self.Params.items():
889             self.Component[line].set_ydata(self.y - self.Evaluate(self.x, **parameters))
890
891           # update the continuum graph
892           self.Continuum.set_ydata(self.y)
893
894           # update the combined graph
895           self.Combination.set_ydata(self.y - self.SuperPosition())
896
897           # update the displayed `N` and `b` if present
898           if self.has_line_parameters:
899             self.Update_Preview()
900
901           # push updates to graph
902           self.fig.canvas.draw_idle()
903
904       def ToggleComponent(self, label):
905         """
906         Toggle function for the radio buttons. Switch between line components
907         `L1`, `L2`, etc. Update the sliders to reflect changing parameters.
908         """
909         # reassign the current component that is selected
910         self.current_component = label
911
912         # don't update the parameter dictionary or draw the graph!!!!!!
913         self.stall = True
914
915         # make current feature bold and the rest thin
916         for line in self.Component.keys():
917           if line == label:
918             self.Component[line].set_linewidth(2)
919           else:
```

```
920             self.Component[line].set_linewidth(1)
921
922         # update the sliders to reflect the current component
923         for parameter, slider in self.Slider.items():
924           if parameter != 'Continuum':
925             slider.set_val(self.Params[label][parameter])
926
927         # update the displayed ‘b‘, ‘N‘ and ‘z‘ if present
928         if self.has_line_parameters:
929           self.Update_Preview()
930
931         # give control back to sliders
932         self.stall = False
933
934         # push updates to graph
935         self.fig.canvas.draw_idle()
936
937       def solve_b(self):
938         """
939         Given the current line component, solve for the broadening parameter, ‘b‘,
940         given the instrument profile and the observed broadening.
941         """
942         # b = sqrt(2) * sigma_v
943         sigma_obs = self.Params[self.current_component]['Sigma']
944         sigma_v   = np.sqrt(sigma_obs**2 - self.sigma_instrument_squared) * self.wavelength.unit
945         b = np.sqrt(2) * (c * sigma_v / self.wavelength)
946         return b.to(u.km / u.second)
947         # return (c.si * (1.4142135623730951 *
948         #   np.sqrt(self.Params[self.current_component]['Sigma']**2
949         #   - self.sigma_instrument_squared) * self.wavelength.unit) / self.wavelength).to(u.km /
950         #   u.second)
951
952       def solve_tau0(self):
953         """
954         Solve for the apparent optical depth at line center.
955         """
956         line_data = self.Component[self.current_component].get_ydata()
957         line_wave = self.Component[self.current_component].get_xdata()
958         line_center = line_data.argmin()
959
960         return np.log(self.y[line_center] / line_data[line_center])
961
962       def solve_W(self):
963         """
964         Solve for the equivalent width of the currently selected line.
965         """
966         line_data = self.Component[self.current_component].get_ydata()
967         line_wave = self.Component[self.current_component].get_xdata()
968
969         # apparent optical depth
970         tau = np.log(self.y / line_data)
971         return Integrate(1 - np.exp(-tau), line_wave) * self.wavelength.unit
972
973       def solve_N(self):
974         """
975         Solve for the column density of the currently selected line component.
976         """
977         # equivalent width (dimensionless)
978         W = self.solve_W() / self.wavelength
979
980         # m_e c^2 / pi e^2 \approx 1.13e12 cm-1 ???? How is that?
981         const = (1.13e12 / u.cm) / (self.fvalue * self.wavelength.to(u.cm))
982         return const * W
983
984       def solve_v(self):
985         """
986         Solve for the velocity of the line given the expected wavelength.
987         The result is returned in km s-1.
988         """
989         line_center = self.Params[self.current_component]['Peak'] * self.wavelength.unit
990         return c.to(u.km / u.second) * (line_center - self.wavelength) / self.wavelength
991
992       def Update_Preview(self):
993         """
994         Re-compute the ‘b‘, ‘N‘, and ‘v‘ values, update the text in the plot.
995         """
996         # self.b, self.N, self.v = self.solve_b(), self.solve_N(), self.solve_v()
997         self.preview.set_text('v: {:>10.4f}        km s-1            b: {:>10.4f}    km s-1'
998         '       W: {:>10.4f}\nN:   {:>10.4e}    cm-2      tau_0: {:>10.4f}'.format(
999         self.solve_v().value, self.solve_b().value, self.solve_W(), self.solve_N().value,
1000        self.solve_tau0()))
1001
1002      def GetSpectra(self):
1003        """
1004        Return the current graphs as Spectrum objects.
1005        """
1006        return [Spectrum(self.Component[line].get_ydata() * self.line.data.unit,
1007          self.x * self.line.wave.unit) for line in sorted(self.Component.keys())]
1008
1009      def GetContinuum(self):
1010        """
1011        Return the current graph of the continuum.
```

```
1012        """
1013        continuum = Spectrum(self.y * self.line.data.unit, self.x * self.line.wave.unit)
1014        continuum.rms = self.rms
1015        return continuum
1016
1017      def kill(self):
1018        """
1019        Close the plot to destroy widgets and restore the `splot` to its original state.
1020        """
1021        plt.close(self.fig)
1022        self.splot.fig = plt.figure("Spectral-Plot (SLiPy)")
1023        self.splot.ax  = self.splot.fig.add_subplot(111)
1024        self.splot.draw()
1025        #del(self)
1026
1027    def MultiFit(splot, error=None, obs=None, ion=None, function='Voigt', measure=True,
1028      boost=None, **kwargs):
1029      """
1030      The MultiFit routine takes a `splot` figure (type SPlot) and allows the
1031      user to interactively fit line profiles. `splot` may optionally be of type
1032      Spectrum, in which case a SPlot figure will be created for you. This function
1033      creates a *FittingGUI* object (not documented here) which uses the Profile.Extract()
1034      routine first (`kwargs` are passed to this function). As in the Extract routine, the user
1035      will select four points that mark the boundaries of the line (blended or otherwise)
1036      and some surrounding continuum to sample. The KernelFit1D (..Algorithms.KernelFit) routine
1037      is applied with the user specified `bandwidth` to smooth the noise out of the continuum
1038      and interpolate, of type `kind`, across the line gap. After this, the user is asked to
1039      further select points marking the peaks of the line(s). The number of selection points
1040      indicates the number of lines to be fit. If you wish to deblend two or more lines, select
1041      all suspected locations. These are not only used to determine the number of lines to fit
1042      but to make initial guesses at the parameters for each line and determine reasonable scales
1043      for the sliders.
1044
1045      After these selections, a slider for each parameter appears along with radio buttons for
1046      each line. The user can interactively adjusts the parameter(s) for a given line by
1047      selecting it (the radio buttons) and dragging a slider. The parameters available to
1048      adjust depend on the `function` argument. There are currently three available line shapes:
1049      'Gaussian', 'Lorentzian', and 'Voigt'. See ..Algorithms.Functions for information on
1050      these.
1051
1052      Each line is represented by a black, dashed curve in the plot. The currently selected line
1053      is bold. The combined (i.e., blended) line is plotted as a solid green curve.
1054
1055      If an Observatory with a `.resolution` is provided via the `obs` argument then the
1056      thermal broadening parameter `b` can be computed (and displayed). This is only applicable
1057      with either a `Gaussian` or `Voigt` profile. Further, an `ion` needs to be provided in
1058      this scenario (type ..Data.Atomic.Ion) with an oscillator strength (fvalue), transition
1059      probability (A) and wavelength as members. With these data, we can interactively show
1060      the broadening prameter, the velocity, and the apparent column density during the fitting
1061      process. Note: the `gamma` slider will disappear in this context because it has already
1062      been determined via the transition probability.
1063
1064      Each slider is labeled on the left side with the name of the parameter it controls. At
1065      the right of each slider is the current value of that parameter. The radio buttons are
1066      labeled "L1", "L2", etc. for each line.
1067
1068      This routine returns both a list of Spectrum `lines` as well as the `continuum` that was
1069      fit using the FittingGUI. This functions acts as both a single line fitting tool as well
1070      as a deblending tool. By default, the final parameterizations are attached to each spectrum
1071      as a dictionary, `.parameters`.
1072
1073      If `measure` is passed as True (the default), the equivalent width is computed and attached
1074      to each spectrum (`L1`, `L2`, etc...) and can be accessed with `.W`. If `b` and/or `N` are
1075      available for each line these will be attached as well, accessible as `.b` and `.N`
1076      respectively. This functionality can be suppressed by setting `measure` to False. `boost`
1077      and `error` are passed to EquivalentWidth() and ColumnDensity().
1078      """
1079
1080      # running the user interface
1081      gui = FittingGUI(splot, obs=obs, ion=ion, function=function, **kwargs)
1082      input('\n Press <Return> when you are finished fitting lines ...')
1083
1084      # attach the parameter dictionaries to each spectrum
1085      lines = gui.GetSpectra()
1086      for a, parameterization in enumerate( sorted(gui.Params.keys()) ):
1087        lines[a].parameters = gui.Params[parameterization]
1088
1089      # attach the continuum to the line list
1090      continuum = gui.GetContinuum()
1091
1092      if not measure:
1093        return lines, continuum
1094
1095      if gui.has_line_parameters:
1096        for line, key in zip(lines, sorted(gui.Params.keys())):
1097          # set the line to `L1`, `L2`, etc...
1098          gui.current_componenet = key
1099          # attach the measured quantities
1100          notes  = 'Measurement made using Profile.MultiFit() from SLiPy'
1101
1102          line.W = EquivalentWidth(line, continuum, error=error, boost=boost)
1103          line.N = ColumnDensity(line, continuum, ion, error=error, boost=boost)
```

```
1104          line.b = Measurement(gui.solve_b(), name='Doppler Broadening Paramater', notes=notes)
1105          line.v = Measurement(gui.solve_v(), name='Velocity', notes=notes)
1106
1107      gui.kill()
1108
1109      return lines, continuum
```

C.17 .. SLiPy . Simbad

```
#!/usr/bin/env python
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# slipy/SLiPy/Simbad.py
"""
usage: Simbad.py @Attribute <identifier> [**kwargs]

This module allows the user to query the SIMBAD astronomical database from
inside Python or shell commands/scripts.

The 'Attribute' points to a function within this module and indicates
what is to be run. Execute 'Simbad.py @Attribute help' for usage details of
a specific function. Currently available attributes are: `Position`,
`Distance`, and `Sptype`.

The identifier names can be anything recognized by SIMBAD (e.g., Regulus,
"alpha leo", "HD 121475", "del cyg", etc ...) if the name is two parts make
sure to use quotation to enclose it.

The **kwargs is the conventional reference to Python keyword arguments.
These should be specific to the 'Attribute' being pointed to.
"""
from sys import argv, exit  # , version_info
from urllib.request import urlopen

from astropy import units as u

from .. import SlipyError
from ..Framework.Command import Parse, CommandError
from ..Framework.Options import Options, OptionsError
from ..Framework.Measurement import Measurement


class SimbadError(SlipyError):
    """
    Exception specific to the Simbad module.
    """
pass


def URLEncoded(url):
    """
    URLEncoded( string ):
    Return a string with reserved url characters encoded.
    """
    # check argument type
    if type(url) is not str:
        raise SimbadError('URLEncoded function expects type str!')

    # percent-encoded pair dictionary
    encoded = {
        ' ': '%20',
        '%': '%25',
        '#': '%23',
        '(': '%28',
        ')': '%29',
        '|': '%7c',
        '+': '%2b'
    }

    return ''.join([
        encoded[character] if character in encoded else character
        for character in list(url)])


def Script(identifier, criteria):
    """
    Script( criteria ):

    URL script for the SIMBAD astronomical database with added
    URLEncoded(criteria).
    """
    script = [
        'http://simbad.u-strasbg.fr/simbad/sim-script?',
        'script=format%20object%20%22', URLEncoded(criteria),
        '%22%0a', URLEncoded(identifier)
        ]

    return ''.join(script)


class Query:
    """
    Query( identifier, criteria, **kwargs ):

    Class for querying the SIMBAD astronomical database for 'citeria'
    of 'identifier'.

    kwargs = {
        'parse' : True,  # extract relavent data from SIMBAD return file
```

```
90          'dtype' : float, # output datatype
91      }
92      """
93      def __init__(self, identifier, criteria, default=float, **kwargs):
94          """
95          Initiate query to SIMBAD database.
96          """
97          # check argument types
98          if type(identifier) is not str or type(criteria) is not str:
99              raise SimbadError('Simbad.Query function expects str'
100             'types for arguments.')
101
102         try:
103             # keyword argument options for Query
104             self.options = Options( kwargs,
105                 {
106                     'parse'  : True    , # parse SIMBAD return file
107                     'full'   : False   , # return full line of info
108                     'dtype'  : default , # convert return data
109                     'is_main': False     # called from Main()
110                 })
111
112             # assignments
113             self.parse   = self.options('parse')
114             self.full    = self.options('full')
115             self.dtype   = self.options('dtype')
116             self.is_main = self.options('is_main')
117
118             # query SIMBAD database
119             with urlopen( Script(identifier, criteria) ) as response:
120                 self.data = str( response.read().decode('utf-8') ).strip()
121
122         except OptionsError as err:
123             print('\n --> OptionsError:', err.msg )
124             raise SimbadError('Simbad.Query was not constructed '
125                 'for `{}`'.format(identifier))
126
127         except URLError as error:
128             raise SimbadError('Failed to contact SIMBAD database for'
129             ' `{}`'.format(identifier) )
130
131         if 'not found' in self.data or 'error' in self.data:
132             raise SimbadError('`{}` could not be resolved by SIMBAD.'
133                 .format(identifier))
134
135         if self.parse:
136             # pre-parse operation common to all criteria
137             self.data = self.data.split('data')[-1]
138
139     def __call__(self):
140         """
141         Retrieve data from Query
142         """
143         return self.data
144
145 def Position( identifier, **kwargs ):
146     """
147     Position( identifier, **kwargs ):
148
149     Handle to the Query class with criteria='%C00(d;C)'.
150     """
151     query = Query( identifier, '%COO(d;C)', **kwargs )
152
153     if query.full:
154         query.data = query.data.split('\n')[-1]
155
156     elif query.parse:
157         # extract relavent data
158         query.data = query.data.split()[-1].split('+')
159         if len( query.data ) == 1:
160             # dec had '-' not '+'
161             query.data    = query.data[0].split('-')
162             query.data[1] = '-' + query.data[1]
163         # return formatted data type
164         query.data = [ query.dtype(pos) * u.degree for pos in query.data ]
165
166     if query.is_main:
167         if query.full or not query.parse:
168             print( query() )
169         else:
170             print('{0:.2f} {1:.2f}'.format(*query()))
171
172     else: return query.data
173
174 def Distance( identifier, **kwargs ):
175     """
176     Distance( identifier, **kwargs ):
177
178     Handle to the Query class with criteria='%PLX'
179     """
180     query =  Query( identifier, '%PLX', **kwargs )
181
```

```
182         if query.full:
183             data = query.data.split('\n')[-1]
184 
185         elif query.parse:
186 
187             # extract relavent data
188             data = query.data.split()
189 
190             if data[1] == '~':
191                 # nothing found!
192                 raise SimbadError('No distance found for `{}`'.format(identifier))
193             try:
194 
195                 # convert milli-arcseconds to parsecs
196                 result = u.pc / ( query.dtype(data[1]) / 1000.0 )
197 
198             except ValueError as err:
199                 raise SimbadError('Use a numeric type for Simbad.Distance!')
200 
201             if data[2][0] == '[' and data[2][-1] == ']':
202                 # TODO: understand SIMBAD error format
203                 # an uncertainty was given by SIMBAD
204                 # uncertainty = u.pc * 0.001 * query.dtype(data[1]) * query.dtype(data[2][1:-1]) / (
205                 #   0.001 * query.dtype(data[1]) )**2
206                 # uncertainty = result * query.dtype(data[2][1:-1])
207                 uncertainty = None
208 
209             else: uncertainty = None
210 
211         else: data = query.data
212 
213         if query.is_main:
214             if query.full or not query.parse:
215                 print( data )
216             else:
217                 print( '{0:.2f}'.format( data ) )
218 
219         elif not query.parse:
220             return data
221 
222         else: return result
223 
224         # Measurement(result, error=uncertainty, name='Distance',
225         #     notes='Retrieved from SIMBAD database for `{}`'.format(identifier))
226 
227     def Sptype(identifier, **kwargs):
228         """
229         Handle to the Query class with criteria='%SP'.
230         """
231         query = Query(identifier, '%SP', **kwargs)
232 
233         if query.full:
234             # return last full line of query
235             query.data = query.data.split('\n')[-1]
236 
237         elif query.parse:
238             # extract relavent data
239             query.data = query.data.split()[1]
240 
241         if query.is_main:
242             print( query() )
243 
244         else: return query()
245 
246     def IDList(identifier, **kwargs):
247         """
248         Handle to the Query class with criteria='%IDLIST'.
249         With `parse` = True, return a list of alternate IDs for
250         the `identifier` provided.
251         """
252         query = Query(identifier, '%IDLIST', **kwargs)
253 
254         if query.parse:
255             # extract relavent data
256             query.data = query.data.split(':')[-1].strip().split('\n')
257 
258         if query.is_main:
259             for line in query.data:
260                 print(line)
261 
262         else: return query()
263 
264     def Main( clargs ):
265         """
266         Main function. See __doc__ for details.
267         """
268 
269         if len(clargs) < 2:
270             # show usage
271             print( __doc__ )
272             return 0
273 
274         # Available functions for execution
```

```
275       executable = {
276               'Distance' : Distance, # search for parsecs
277               'Position' : Position, # search for ra, dec
278               'Sptype'   : Sptype  , # search for spectral types
279               'IDList'   : IDList    # search for IDs
280           }
281
282
283       try:
284
285           # Parse command line arguments
286           function, args, kwargs = Parse( clargs[1:] )
287
288           if not args and not kwargs or args[0] == 'help':
289               # show function usage
290               print( executable[function].__doc__ )
291               return 0
292
293           # run execution
294           for identifier in args:
295               executable[function]( identifier, is_main=True, **kwargs )
296
297           return 0
298
299       except CommandError as err:
300           print(' --> CommandError:', err.msg)
301           return 1
302
303       except KeyError as key:
304           print(' --> {} was not a recognized function.'.format(key))
305           return 1
306
307       except SimbadError as err:
308           # don't let uncaught self exception pass if from main.
309           print(' --> SimbadError:', err.msg)
310           return 1
311
312       except Exception as err:
313           print(' --> Unrecognized error with query for `{}``
314                   .format(args[0]))
315           print(' --> Exception: `{}``.format(err))
316           return 1
317
318   if __name__ == '__main__':
319       # Main return 0 or 1
320       exit( Main( argv ) )
```

C.18 .. SLiPy . Telluric

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# slipy/SLiPy/Tellucic.Py
"""
Telluric - Corrections for atmospheric absoption lines.
"""
import numpy as np
from astropy import units as u

from .. import SlipyError
from ..Framework.Options import Options, OptionsError
from .Correlate import Xcorr, CorrelateError
from .Spectrum import Spectrum, SpectrumError

class TelluricError(SlipyError):
    """
    Exception specific to the Telluric module.
    """
    pass

def Correct(spectrum, *calibration, **kwargs):
    """
    Correct(spectrum, *calibration, **kwargs):

    Perform a telluric correction on `spectrum` with one or more
    `calibration` spectra. If more than one calibration spectrum is
    provided, the one with the best fit after performing both a
    horizontal cross correlation and a vertical amplitude fit is used.
    The spectrum and all the calibration spectra must have the same
    number of pixels (elements). If a horizontal shift in the calibration
    spectra is appropriate, only the corresponding range of the spectrum
    is divided out!

    kwargs = {
        lag  : 25            , # pixel shift limit for XCorr()
        range:(0.5, 2.0, 151), # numpy.linspace for amplitude trials
    }
    """
    try:
        # default keyword arguments
        options = Options( kwargs,
            {
                'lag'  : 25            , # pixel shift limit for XCorr()
                'range':(0.5, 2.0, 151)  # numpy.linspace for amplitude trials
            })

        # check arguments
        if not calibration:
            raise TelluricError('At least one `calibration` spectrum '
            'needs to be provided for Telluric.Correction().')

        if type(spectrum) is not Spectrum:
            raise TelluricError('Telluric.Correct() expects all arguments '
            'of type Spectrum.')

        for cal in calibration:
            if type(cal) is not Spectrum:
                raise TelluricError('Telluric.Correct() expects all '
                'arguments to be of type Spectrum.')
            if spectrum not in cal:
                raise TelluricError('Telluric.Correct() expects the '
                '`spectrum` domain to be equivalent to or at least '
                'contained within each `calibration` spectrum.')

        if len(options('range')) != 3:
            raise OptionsError('`range` expects a tuple of length 3.')

        # assign parameters
        lag    = options('lag')
        amp    = np.linspace( *options('range') )
        trials = len(amp)
        npix   = len(spectrum)

    except OptionsError as err:
        print(' --> OptionsError:', err)
        raise TelluricError('Inappropriate keyword arguments in '
        'Telluric.Correct().')

    # quit if too big of a task
    if trials*npix > 1e8:
        raise TelluricError('Telluric.Correct() is programmed to quit '
        'if it detects a request to operate on matrices with more '
        'than 10**8 elements.')

    # resample all the calibration spectra
    for cal in calibration:
        cal.resample(spectrum)

    # find best XCorr and amplitude adjustment
```

```
90      best = None
91      for cal in calibration:
92
93          # best horizontal pixel shift
94          shift = Xcorr( spectrum, cal, lag=lag)
95
96          # build matrices with identical rows (len=npix-shift)
97          if shift < 0:
98
99              calmatrix = np.tile(cal.data[:shift], (trials, 1))
100             objmatrix = np.tile(spectrum.data[-shift:], (trials, 1))
101
102         elif shift > 0:
103
104             calmatrix = np.tile(cal.data[shift:], (trials, 1))
105             objmatrix = np.tile(spectrum.data[:-shift], (trials, 1))
106
107         else:
108
109             calmatrix = np.tile(cal.data, (trials, 1))
110             objmatrix = np.tile(spectrum.data, (trials, 1))
111
112         # amplitude matrix has identical columns
113         size = np.shape(calmatrix)[1]
114         ampmatrix = np.tile(amp, (size,1)).T
115
116         # remove units for dimensionless operations
117         calmatrix = calmatrix.value
118         objmatrix = objmatrix.value
119
120         # flip arrays for amplification
121         diff = objmatrix - (1 - (1 - calmatrix) * ampmatrix)
122
123         # compute the RMS for each trial
124         rmsvector = np.sqrt(( diff**2 ).sum(axis=1) / size)
125
126         if not best:
127
128             # if first pass, assign to `best`
129             best = ( rmsvector.min(), rmsvector.argmin(), shift, cal )
130
131         elif rmsvector.min() < best[0]:
132
133             # this fit is better, save it to `best`
134             best = ( rmsvector.min(), rmsvector.argmin(), shift, cal )
135
136     # results of calibration fitting
137     index = best[1] # amplitude
138     shift = best[2] # XCorr
139     cal   = best[3] # which calibration spectrum
140
141     # we can't update an attribute...
142     update = spectrum.data.value
143
144     # divide spectrum
145     if shift < 0:
146         update[-shift:] /= 1 - (1 - cal.data[:shift].value) * amp[index]
147     elif shift > 0:
148         update[:-shift] /= 1 - (1 - cal.data[shift:].value) * amp[index]
149     else:
150         update /= 1 - (1 - cal.data.value) * amp[index]
151
152     # re-incorporate units
153     spectrum.data = update * u.Unit(spectrum.yunits)
```

C.19 .. SLiPy . Velocity

```
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# slipy/SLiPy/Velocity.Py
"""
Radial velocity corrections for 1D spectra.
"""

from astropy.io import fits as pyfits
from astropy.constants import c
from astropy import units as u

from .. import SlipyError
from ..astrolibpy.astrolib.helcorr import helcorr
from .Fits import Find, RFind
from .Observatory import Observatory
from .Spectrum import Spectrum
from ..Framework.Options import Options, OptionsError
from ..Framework.Display import Monitor, DisplayError

class VelocityError(SlipyError):
  """
  Exception specific to the Velocity module
  """
  pass

def HeaderInfo( fpath ):
  """
  HeaderInfo( fpath ):

  Helper function of IrafInput.

  Return a formatted string containing the year, month, and day of
  observation from the FITS file with name 'fpath', as well as the
  univeral time of observation and the right ascension and declination
  of the target.
  """
  try:
    with pyfits.open(fpath) as hdulist:
      ra       = ':'.join( hdulist[0].header['alpha'].split() )
      dec      = ':'.join( hdulist[0].header['delta'].split() )
      date_obs = hdulist[0].header['date-obs']
      date, UT = date_obs.split('T')
      year, month, day = [ x.lstrip('0') for x in date.split('-') ]

      info = '{:>5} {:>2} {:>2} {:>8} {:>11} {:>12}\n'.format(
        year, month, day, UT, ra, dec )

  except IOError as error:
    raise VelocityError('Failure in `{}` from HeaderInfo()'.format(fpath))

  return info

def IrafInput( *files, **kwargs):
  """
  IrafInput( *args, **kwargs ):

  Build an input file for IRAF's rvcorrect task.

  `files` should be a list of FITS file names to build the output table for.
  The user can optionally specify a 'toplevel' directory to search
  (recursively!) under fitting the 'pattern' (default=*.fits). This results
  of this pattern search will be added to the list of file names in 'args'
  (if any given).

  kwargs = {
    'toplevel' : ''      , # search `toplevel` directory for files
    'pattern'  : '*.fits', # files under `toplevel` fitting `pattern`
    'recursive': False   , # search recusively under `toplevel`
    'outfile'  : ''        # write lines to file named `outfile`
  }
  """
  try:
    # dictionary of options
    options = Options( kwargs,
      {
      'toplevel' : ''      , # search `toplevel` directory for files
      'pattern'  : '*.fits', # files under `toplevel` fitting `pattern`
      'recursive': False   , # search recursively under `toplevel`
      'outfile'  : ''        # write lines to file named `outfile`
      })

    # convert options types
    toplevel  = options('toplevel')
    pattern   = options('pattern')
    recursive = options('recursive')
    outfile   = options('outfile')

    files = list(files)

```

```
      if toplevel:
        # search for files matching `pattern`
        find   = RFind if recursive else Find
        files += find( toplevel, pattern )

      # get info from files
      info = [ HeaderInfo(fpath) for fpath in files ]

      if outfile:
        with open( outfile, 'w' ) as fp:
          fp.writelines( info )

      return info

    except OptionsError as err:
      print(' --> OptionsError:', err)
      raise VelocityError('Failed to construct table information in '
      'IrafInput().')

  def HelioCorrect( obs, *spectra, **kwargs ):
      """
      Perform heliocentric velocity corrects on `spectra` based on
      `obs`ervatory information (longitude, latitude, altitude) and the
      member attributes, ra (right ascension), dec (declination), and jd
      (julian date) from the `spectra`.
      """
      try:

          # define function parameters
          options = Options( kwargs,
              {
                  'verbose': False # display messages, progress
              })

          # assign parameters
          verbose = options('verbose')

          # check `obs` type
          if not issubclass( type(obs), Observatory):
              raise VelocityError('HelioCorrect() expects its first argument to '
              'be derived from the Observatory class.')
          elif ( not hasattr(obs, 'latitude') or not hasattr(obs,'longitude') or
              not hasattr(obs, 'altitude') ):
              raise VelocityError('HelioCorrect expects `obs`ervatory to have '
              'all three: latitude, longitude, and altitude attributes.')

          # check `spectra` arguments
          for a, spectrum in enumerate(spectra):
              if type(spectrum) is not Spectrum:
                  raise VelocityError('HelioCorrect() expects all `spectrum` '
                  'arguments to be of type Spectrum.')
              if not spectrum.ra or not spectrum.dec or not spectrum.jd:
                  raise VelocityError('Spectrum {} lacks one or all of `ra`, '
                  '`dec`, and `jd`; from HelioCorrect().'.format(a))
              if not hasattr(spectrum.ra, 'unit'):
                  raise VelocityError('From HelioCorrect(), in spectrum {}, '
                  '`ra` doesn`t have units!'.format(a))
              if not hasattr(spectrum.dec, 'unit'):
                  raise VelocityError('From HelioCorrect(), in spectrum {}, '
                  '`dec` doesn`t have units!'.format(a))
              if not hasattr(spectrum.jd, 'unit'):
                  raise VelocityError('From HelioCorrect(), in spectrum {}, '
                  '`jd` doesn`t have units!'.format(a))

          if verbose:
              display = Monitor()
              print(' Running HelioCorrect on {} spectra ...'
                  .format(len(spectra)))

          for a, spectrum in enumerate(spectra):

              # heliocentric velocity correction in km s^-1,
              # the `astrolibpy...helcorr` function doesn't work with units,
              # so I convert to appropriate units and strip them.
              hcorr = helcorr(
                      obs.longitude.to(u.degree).value,
                      obs.latitude.to(u.degree).value,
                      obs.altitude.to(u.meter).value,
                      spectrum.ra.to(u.hourangle).value,
                      spectrum.dec.to(u.degree).value,
                      spectrum.jd.to(u.day).value
                  )[1] * u.km / u.second

              # apply correction to wave vector,
              # del[Lambda] / Lambda = del[V] / c
              spectrum.wave -= spectrum.wave * hcorr / c.to(u.km / u.second)

              # show progress if desired
              if verbose: display.progress(a, len(spectra))

          # finalize progress bar (erase)
          if verbose: display.complete()
```

```
182
183       except OptionsError as err:
184           print(' --> OptionsError:', err)
185           raise VelocityError('Failed to perform HelioCorrect() task.')
186
187       except DisplayError as err:
188           print(' --> DisplayError:', err)
189           raise VelocityError('Exception from Display.Monitor in HelioCorrect().')
190
191
192   def BaryCorrect( obs, *spectra, **kwargs ):
193     """
194     Perform barycentric velocity corrects on `spectra` based on
195     `obs`ervatory information (longitude, latitude, altitude) and the
196     member attributes, ra (right ascension), dec (declination), and jd
197     (julian date) from the `spectra`.
198     """
199     try:
200
201       # define function parameters
202       options = Options( kwargs,
203         {
204           'verbose': False # display messages, progress
205         })
206
207       # assign parameters
208       verbose = options('verbose')
209
210       # check `obs` type
211       if not issubclass( type(obs), Observatory):
212         raise VelocityError('HelioCorrect() expects its first argument to '
213         'be derived from the Observatory class.')
214       elif ( not hasattr(obs, 'latitude') or not hasattr(obs,'longitude') or
215         not hasattr(obs, 'altitude') ):
216         raise VelocityError('HelioCorrect expects `obs`ervatory to have '
217         'all three: latitude, longitude, and altitude attributes.')
218
219       # check `spectra` arguments
220       for a, spectrum in enumerate(spectra):
221         if type(spectrum) is not Spectrum:
222           raise VelocityError('HelioCorrect() expects all `spectrum` '
223           'arguments to be of type Spectrum.')
224         if not spectrum.ra or not spectrum.dec or not spectrum.jd:
225           raise VelocityError('Spectrum {} lacks one or all of `ra`, '
226           '`dec`, and `jd`; from HelioCorrect().'.format(a))
227
228       if verbose:
229           display = Monitor()
230           print(' Running HelioCorrect on {} spectra ...'
231           .format(len(spectra)))
232
233       for a, spectrum in enumerate(spectra):
234
235               # heliocentric velocity correction in km s^-1,
236               # the `astrolibpy...helcorr` function doesn't work with units,
237               # so I convert to appropriate units and strip them.
238         hcorr = helcorr(
239                       obs.longitude.to(u.degree).value,
240                       obs.latitude.to(u.degree).value,
241                       obs.altitude.to(u.meter).value,
242                       spectrum.ra.to(u.hourangle).value,
243                       spectrum.dec.to(u.degree).value,
244                       spectrum.jd.to(u.second).value
245                   )[0] * u.Unit('km s-1')
246
247         # apply correction to wave vector,
248               # del[Lambda] / Lambda = del[V] / c
249         spectrum.wave -= spectrum.wave * hcorr / c
250
251         # show progress if desired
252         if verbose: display.progress(a, len(spectra))
253
254       # finalize progress bar (erase)
255       if verbose: display.complete()
256
257     except OptionsError as err:
258       print(' --> OptionsError:', err)
259       raise VelocityError('Failed to perform HelioCorrect() task.')
260
261     except DisplayError as err:
262       print(' --> DisplayError:', err)
263       raise VelocityError('Exception from Display.Monitor in HelioCorrect().')
```

CURRICULUM VITAE

NAME: Geoffrey Robert Lentner

ADDRESS: Department of Physics
University of Louisville
Louisville, KY 40292

DOB: October 23, 1989

EDUCATION: B.S. Physics and Astronomy
Purdue University
2013

WORK EXPERIENCE

University of Louisville

Graduate Teaching Assistant, 2014-2015

Astronomy & Astrophysics

Purdue University, PRIME Lab

Undergraduate Controller, 2012-2013

Accelerator Mass Spectrometry

Dr. Mark Caffee (Professor, Director of PRIME Lab)

Purdue University

Undergraduate Research Assistant, 2011-2012

Experimental Physics

Dr. Rafael Lang (Assistant Professor)

Purdue University

Undergraduate Research Assistant, 2011

Astronomy & Astrophysics

Dr. Jaehyon Rhee (Research Professor)

TEACHING EXPERIENCE

University of Louisville

Fundamentals of Physics I (Physics 223)

Elementary Astronomy Laboratory (Physics 108)