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

Geoffrey Robert Lentner

University of Louisville

Follow this and additional works at: https://ir.library.louisville.edu/etd

Part of the Physics Commons

Recommended Citation
https://doi.org/10.18297/etd/2213
THE LOCAL CYGNUS COLD CLOUD AND FURTHER 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
Copyright 2015 by Geoffrey Robert Lentner

All rights reserved
THE LOCAL CYGNUS COLD CLOUD AND FURTHER
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 FURTHER 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 \(\lambda 5889.591\) and D1 \(\lambda 5895.924\)) was found towards \(\delta\) 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

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEDICATION</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xiv</td>
</tr>
<tr>
<td><strong>CHAPTER</strong></td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td></td>
</tr>
<tr>
<td>1.1 The Interstellar Medium</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 Background</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2 The Local Bubble</td>
<td>2</td>
</tr>
<tr>
<td>1.1.3 Searching for Interstellar Clouds</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Thesis Overview</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1 Content</td>
<td>4</td>
</tr>
<tr>
<td>1.2.2 SLiPy</td>
<td>5</td>
</tr>
<tr>
<td>2. DATA</td>
<td>8</td>
</tr>
<tr>
<td>2.1 Archival Data</td>
<td>8</td>
</tr>
<tr>
<td>2.2 The Elodie Spectrograph</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Acquisition Methods</td>
<td>9</td>
</tr>
<tr>
<td>2.4 Sorting and Data Management</td>
<td>13</td>
</tr>
<tr>
<td>2.4.1 The Simbad Database</td>
<td>13</td>
</tr>
</tbody>
</table>
6.4 Future Studies ............................................. 68

7 CONCLUSIONS ............................................. 71

REFERENCES ............................................... 74

APPENDICES ................................................. 78

A HUBBLE SPACE TELESCOPE PROPOSAL .............. 78

B OBSERVATIONAL METADATA .............................. 89

C SOURCE CODE ............................................. 101

C.1 .. __init__ .................................................. 102
C.2 .. Algorithms . Functions .............................. 103
C.3 .. Algorithms . KernelFit ............................... 105
C.4 .. Data . Atomic .......................................... 107
C.5 .. Data . Elodie .......................................... 111
C.6 .. Framework . Argument .............................. 113
C.7 .. Framework . Command ............................... 114
C.8 .. Framework . Display ................................. 115
C.9 .. Framework . Measurement ......................... 117
C.10 .. Framework . Options .................... .......................... 118
C.11 .. SLiPy . Correlate ................................. 119
C.12 .. SLiPy . Fits .......................................... 120
C.13 .. SLiPy . Montage ..................................... 125
C.14 .. SLiPy . Observatory .............................. 136
C.15 .. SLiPy . Plot .......................................... 146
C.16 .. SLiPy . Profile ..................................... 151
C.17 .. SLiPy . Simbad ..................................... 164
## LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Non-OB Targets with Measurable Absorption – A</td>
<td>55</td>
</tr>
<tr>
<td>5.2</td>
<td>Non-OB Targets with Measurable Absorption – B</td>
<td>56</td>
</tr>
<tr>
<td>5.3</td>
<td>Non-OB Targets with No Absorption – A</td>
<td>57</td>
</tr>
<tr>
<td>5.4</td>
<td>Non-OB Targets with No Absorption – B</td>
<td>58</td>
</tr>
<tr>
<td>5.5</td>
<td>Non-OB Targets Inconclusive – A</td>
<td>59</td>
</tr>
<tr>
<td>5.6</td>
<td>Non-OB Targets Inconclusive – B</td>
<td>60</td>
</tr>
<tr>
<td>5.7</td>
<td>Non-OB Targets Inconclusive – C</td>
<td>61</td>
</tr>
<tr>
<td>5.8</td>
<td>Non-OB Targets Inconclusive – D</td>
<td>62</td>
</tr>
<tr>
<td>5.9</td>
<td>Non-OB Targets Inconclusive – E</td>
<td>63</td>
</tr>
<tr>
<td>5.10</td>
<td>OB Targets – A</td>
<td>64</td>
</tr>
<tr>
<td>5.11</td>
<td>OB Targets – B</td>
<td>65</td>
</tr>
<tr>
<td>6.1</td>
<td>Stars Observed in the Region of $\delta$ Cygnus using the Coudé Feed</td>
<td>70</td>
</tr>
<tr>
<td>B.1</td>
<td>Observational Metadata for Non-OB Measurable Targets – A</td>
<td>90</td>
</tr>
<tr>
<td>B.2</td>
<td>Observational Metadata for Non-OB Measurable Targets – B</td>
<td>91</td>
</tr>
<tr>
<td>B.3</td>
<td>Observational Metadata for Non-OB No Absorption Targets – A</td>
<td>92</td>
</tr>
<tr>
<td>B.4</td>
<td>Observational Metadata for Non-OB No Absorption Targets – B</td>
<td>93</td>
</tr>
<tr>
<td>B.5</td>
<td>Observational Metadata for Non-OB Inconclusive Targets – A</td>
<td>94</td>
</tr>
<tr>
<td>B.6</td>
<td>Observational Metadata for Non-OB Inconclusive Targets – B</td>
<td>95</td>
</tr>
<tr>
<td>B.7</td>
<td>Observational Metadata for Non-OB Inconclusive Targets – C</td>
<td>96</td>
</tr>
<tr>
<td>B.8</td>
<td>Observational Metadata for Non-OB Inconclusive Targets – D</td>
<td>97</td>
</tr>
</tbody>
</table>
B.9 Observational Metadata for Non-OB Inconclusive Targets – E ................................ 98
B.10 Observational Metadata for OB Targets – A .................................................... 99
B.11 Observational Metadata for OB Targets – B .................................................... 100
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Telluric Correction - HD 192640</td>
<td>26</td>
</tr>
<tr>
<td>3.2 Telluric Correction - Altair</td>
<td>27</td>
</tr>
<tr>
<td>4.1 KernelFit1D Example</td>
<td>33</td>
</tr>
<tr>
<td>4.2 Example SPlot of Regulus</td>
<td>35</td>
</tr>
<tr>
<td>4.3 SPlot of HD 192639</td>
<td>42</td>
</tr>
<tr>
<td>4.4 SPlot of HD 192639 - Profile.MultiFit() A</td>
<td>43</td>
</tr>
<tr>
<td>4.5 SPlot of HD 192639 - Profile.MultiFit() B</td>
<td>44</td>
</tr>
<tr>
<td>4.6 SPlot of HD 192639 - Profile.MultiFit() C</td>
<td>45</td>
</tr>
<tr>
<td>5.1 Cygnus Field - All Data</td>
<td>52</td>
</tr>
<tr>
<td>5.2 Cygnus Field - Absorption</td>
<td>53</td>
</tr>
<tr>
<td>5.3 Cygnus Field - No Absorption</td>
<td>54</td>
</tr>
<tr>
<td>6.1 Stellar vs ISM Lines</td>
<td>69</td>
</tr>
</tbody>
</table>
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), Spitzer (1978), and the earlier Aller et al. (1968).

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 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 $\lambda$5889.591 and D1 $\lambda$5895.924) and found cold interstellar absorption towards 23 stars behind the cloud. The distance to the closest of these stars ($\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 conducting 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 ~ 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 \textit{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. \texttt{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, \texttt{Algorithms.Functions.Lorentzian()} is referencing the Lorentzian \textit{function} inside the Functions \textit{module} contained within the Algorithms \textit{sub-package} of the SLiPy package. Leading punctuation will be used to indicate a function that is a \textit{method} for a \textit{class} (e.g., \texttt{.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:

```python
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, Mouttaka et al., 2004, Prugniet 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[1,5850,5950,0.01] 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.

²In astronomy, resolution is defined as $R = \lambda / \Delta \lambda$.
³Previously, 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 towards $\delta$ 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 $\delta$ 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. 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 ElodieArchive is an object that once instantiated reads in the raw ascii catalog mentioned previously (from ../Data/Archives/Elodie.csv by default) and parses the

\[ W_\lambda = 20.2 \text{ mÅ}, \quad v_\odot = -18.54 \text{ km s}^{-1}, \quad N = 29.6 \times 10^{10} \text{ cm}^{-2}, \quad b = 0.42 \text{ km s}^{-1}. \] (Welty et al., 1994)

\[ W_\lambda = 20.2 \text{ mÅ}, \quad v_\odot = -18.54 \text{ km s}^{-1}, \quad N = 29.6 \times 10^{10} \text{ cm}^{-2}, \quad b = 0.42 \text{ km s}^{-1}. \] (Welty et al., 1994)

\[ W_\lambda = 20.2 \text{ mÅ}, \quad v_\odot = -18.54 \text{ km s}^{-1}, \quad N = 29.6 \times 10^{10} \text{ cm}^{-2}, \quad b = 0.42 \text{ km s}^{-1}. \] (Welty et al., 1994)
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, choosing the file pertaining to the spectra with the highest signal-to-noise ratio available.

```python
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"]
```

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 toconcisely crawl and sort the archive using Simbad (a module

---

6Henry 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.

11
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-yyyyymmdd-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 specific to the attribute of interest and retrieves the file from SIMBAD. The final step is to extract the relevant information from the file and return the data with an appropriate type (either a string or a `Quantity`).

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.

```python
from slipy import Simbad

distance = Simbad.Distance("regulus")
<Quantity 24.31315341598055 pc>

distance = Simbad.Distance("regulus")
<Quantity 24.31315341598055 pc>

ra, dec = Simbad.Position("M31")
[<Quantity 10.684708 deg>, <Quantity 41.26875 deg>]

sptype = Simbad.Sptype("delta scuti")
'F2IIIp'

idlist = Simbad.IDList("north star")
['"2MASS J02314822+8915503",
'ADS 1477 AP",
'** STF 93A",
'WDS J02318+8916A",
'** WRH 39",
'NAME Lodestar",
'PLX 299']
```

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

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

```
# find file path names under 'Regulus' directory and import data
files = Fits.Find("Data/Regulus")
regulus = Fits.GetData(*files)

# files organized under 'Data/All/{0, B, A, F, G, K, M}' directories
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.

```python
from slipy import Spectrum
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

import numpy as np
data = np.random.rand(4)
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. 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

---

15Includes +, -, *, /, +=, -=, *=, /=, <, <=, >, >=, ==, &^, |, <<, >>
defined on 550-600 \( nm \), the operation \( \text{SpectrumC} = \text{SpectrumA} + \text{SpectrumB} \) would yield a new spectrum with the wavelength domain of \( \text{SpectrumA} \) where all pixels that fall into the domain of \( \text{SpectrumB} \) are operated on. The necessary condition is that the RHS spectrum is equivalent to or entirely contained (inclusively) by the LHS.

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

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

```python
from astropy import units as u
spectrumA.wave = spectrumB.wave.value * u.nm
spectrumB.wave = spectrumB.wave.value * u.nm
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 \( \text{regulus}[5885:5910] \) 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. \( \text{regulus}[5885:5910:0.1] \) 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\(^\text{16}\) (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).

\[
2 \ast \text{spectrumA}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\text{nm}
\]

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. \(^\text{17}\)

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).

\[
\text{spectrumA} \gg 2 \ast \text{u.nm}
\]

\[
\begin{bmatrix}
1. & 2. & 3. & 4. & 5. \\
\end{bmatrix}
\text{nm}
\]

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

\[
\text{v} = 15 \ast \text{u.km / u.s}
\]

\[
\text{regulus} \ll \text{regulus.wave} \ast (\text{v} / \text{c.to(u.km / u.s)})
\]

\[
\begin{bmatrix}
1.01038098 & 1.01141405 & 1.01213932 & \ldots, & 0.9898265 & 0.99048185 \\
0.99079037
\end{bmatrix}
\text{Angstrom}
\]

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.

\(^\text{16}\)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"`).

\(^\text{17}\)For example, `\text{sum}\left(\text{regulus data} \right) > 1\)` 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 calibration 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 (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.

2The 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 \textit{range} can be specified for amplitude correlation. This should take the form of a \textit{tuple} of length three and is the start, end, and number of percentages (e.g., \texttt{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.

```python
from slipy import Telluric
from slipy.Framework import Display
display = Display.Monitor()

# display progress while applying telluric corrections to all targets
for a, spectrum in enumerate(spectra):
    display.progress(a+1, len(spectra))
    Telluric.Correct(spectrum, *regulus, range=(0.5, 2.0, 1501))
```

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 \texttt{Telluric.Correct()} algorithm for two stars. HD 192640, an A2V variable star at $20^h 14^m 32.0^s +36^\circ 48^\prime 22.7^\prime\prime$ approximately $11^\circ$ southeast of $\delta$ 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.  

\footnote{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.

---

\(^4\)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.  

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:

```python
class OHP(Observatory):
    """
The Observatoire de Haute-Provence, France.
    """
    def __init__(self):
        self.name = 'Observatoire de Haute-Provence'
        self.longitude = 356.286670 * 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
```

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

---

8In 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.

```python
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

\[
f(x) = \frac{\sum_i k(x-x_i) y_i}{\sum_i k(x-x_i)} \quad \text{where} \quad k(x) = A \exp\left(-\frac{x^2}{2\sigma^2}\right).
\] (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), \(\sigma\). 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 (\(\ell\es p\ät\)) 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 \textbf{KernelFit1D} in a best-usage scenario. Here, we have a synthetic data set constructed of a normalized sinc function with added both normal \textit{white} and \textit{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 \textit{bandwidth} of $\pi/32$. 
are all of type **SPlot**.  

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.

```python
from slipy import Plot

figure = Plot.SPlot(regulus[0], label="Regulus", marker="k-")
figure.legend(loc="lower left")
figure.draw()
figure.save("regulus.pdf")

# create many plots and access them individually
figure = [ Plot.SPlot(spectrum, label=Fits.Header(filepath, "OBJECT"))
          for spectrum, filepath in zip(spectra, files) ]

# draw the first spectrum
figure[0].draw()

# iterate through all the plots and choose ones of interest
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 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. Together, the Voigt profile can be expressed as (Draine, 2011)

$$\phi^\text{Voigt}_\nu \equiv \frac{1}{\sqrt{2\pi}} \int \frac{dv}{\sigma_v} e^{-v^2/2\sigma_v^2} \frac{4\gamma_{ul}}{16\pi^2 \left[ \nu - (1-v/c)\nu_{ul} \right]^2 + \gamma_{ul}^2}$$ (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 $\gamma_{ul}$ and the Einstein coefficient $A_{ul}$ as

$$\left(\Delta\nu\right)_\text{FWHM}^{\text{intr.}} = \frac{\gamma_{ul}}{2\pi} = \frac{\lambda A_{ul}}{2\pi}.$$ (4.3.2)

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

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

where the monochromatic absorption cross section $\sigma_{ul}(\nu)$ to a normalized line profile $\phi_\nu$ 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.
\[ \sigma_{lu}(\nu) = \frac{g_u}{g_l} \frac{c^2}{8\pi^2 \nu_i^2} A_{ul} \phi_{ul}. \]  

(4.3.4)

These quantities, \( A_{ul} \) and \( f_{ul} \), are important in 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

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

(4.3.5)

Within the SciPy library there is a \texttt{scipy.special} module with a numerical implementation provided for \texttt{wofz()}. 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) \( 6.14 \times 10^7 \text{ s}^{-1} \) and

\[ \text{This is essentially } \exp(-z^2) \text{erfc}(-iz) \text{. Please see the Steven G. Johnson, Faddeeva W function implementation, http://ab-initio.mit.edu/Faddeeva, for more details.} \]
6.16 \times 10^7 \text{ s}^{-1}, \text{ respectively} (\text{Kramida et al., 2014}). \text{ 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. A short example is presented here, and again in the later example using the Profile.MultiFit() function.

```python
from slipy.Data import Atomic

# find oscillator strengths for C III ions
Atomic.IonSearch("C III")

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

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

[(<Quantity 5891.5033 Angstrom>, 'Na I'),
 (<Quantity 5895.725 Angstrom>, 'Ti I'),
 (<Quantity 5897.5581 Angstrom>, 'Na I')]

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

Ion: Na I
Wavelength: 588.9951 \text{ nm}
fee: 0.6408671009122726
A: 6.16e-07 \text{ s}
```

5While 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

\[
\sigma_{\text{convolution}}^2 = \sigma_a^2 + \sigma_b^2. \tag{4.3.6}
\]

So the width of the thermal/turbulence 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).
\[ \sigma_{\text{therm}} = \sqrt{\sigma^2_{\text{observed}} - \sigma^2_{\text{instrument}}} \]  

where the resolution of the instrument gives the width,

\[ (\Delta \lambda)_{\text{FWHM}} = \frac{\lambda}{R} \]

\[ = 2 \sqrt{2 \ln 2} \sigma_{\text{instrument}} \]

and the broadening parameter is,

\[ b \equiv \sqrt{2} \sigma_{\text{therm}}. \]

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. 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.

```python
from slipy import Profile

# declare the Na I ions
D1 = Atomic.Ion("Na I", 589.5*u.nm, A=6.14e-7*u.second, medium="air")
D2 = Atomic.Ion("Na I", 588.9*u.nm, A=6.16e-7*u.second, medium="air")

# SPlot of HD 200120
figure = Plot.SPlot(spectra[31], label="HD 192639", marker="k-")
figure.legend(frameon=False, loc="lower right")
```

---

8The 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.
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.  

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.

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

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

\[9\] 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,

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

(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.

```python
# extract line from continuum ("fig" is an SPlot)
line, continuum = Profile.Extract(fig, bandwidth=u.Angstrom/12)

# "spectrum" is the spectrum used in "fig" with "error" being 
# the percent errors from counting statistics.
W = Profile.EquivalentWidth(line, continuum, error=spectrum.error)
```


<table>
<thead>
<tr>
<th>Name</th>
<th>Equivalent Width</th>
<th>Value</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.024997410090099946 Angstrom</td>
<td>[ 0.01070082, -0.01070067] Angstrom</td>
</tr>
<tr>
<td>Notes</td>
<td>Measured using Profile.EquivalentWidth() from SLiPy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.\(^\text{10}\) Doing the same set of commands but instead selecting D1 gives us $(17.7 \pm 9.6)$ mÅ and the ratio $W_2 / W_1 \approx 1.41$. 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.\footnote{Not to insinuate that the LCCC necessarily has a hard edge.} In the limit of optically thin gas, $\tau_0 \ll 1$ (where $\tau_0$ is the optical depth at line center), the column density can be related to the equivalent width as (Draine, 2011)

$$N = \frac{m_e c^2}{\pi e^2} \frac{W}{f_{lu} \lambda_{lu}^2} \tau_0 ,$$

(4.3.14)

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

(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 \textit{apparent} column density relates to the \textit{apparent} optical depth as

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

(4.3.16)

The usage of the word \textit{apparent} here embodies the understanding that with an instrumentally smeared line we are only measuring the \textit{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
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 scenario 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,

$$N_a = \frac{m_e c^2}{\pi e^2 f_{lu} \lambda^2} \int \tau_a(\lambda) \, d\lambda.$$  \hspace{1cm} (4.3.17)

The true column density can be estimated as

$$\log N_a^c = \log N_a^{n-1} + \Delta \log N_a^{n-1}$$  \hspace{1cm} (4.3.18)

where $\Delta \log N_a^{n-1}$ is a correction factor determined from the relationship between $
\log N_{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 \texttt{Profile.ColumnDensity()} function. The arguments consist of a \textit{line} and a \textit{continuum}, the same as for the \texttt{Profile.OpticalDepth()} and \texttt{Profile.EquivalentWidth()} functions. Here however we want to also provide a third, \textit{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 \textit{weakline} and the returned measurement will have the .\texttt{correction} and .\texttt{true} values attached. This uses the published corrections from Savage and Sembach (1991) and interpolates to estimate the correction given the calculated $\log N_{a}^{n-1} - \log N_{a}^{n}$.

For only slightly saturated lines and/or not optically thin, we can now estimate the \textit{true} column density by passing these results as the keyword argument, \textit{weakline}, to \texttt{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

$$\delta s_{\text{count}} = \sqrt{s}.$$  \hspace{1cm} (4.4.1)

The uncertainty from these statistics is added in quadrature with those from the continuum fitting,
\[ \delta s_{\text{total}} = \left[ \delta s_{\text{count}}^2 + \delta s_{\text{rms}}^2 \right]^{1/2} \] (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.

```python
# build error spectrum from original counts
s = Spectrum("/from/file.fits")
s.error = Spectrum(100 * u.percent * np.sqrt(s.data.value) / 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. 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.

The mosaic used in the background of Figures 5.1 - 5.3 was adapted from data retrieved 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.
Measurements and uncertainties computed with $\beta$ retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed. Numbers correspond to annotations in Figure 5.2.

Velocity of Na I D2 line center, from 5,889.951 Å. Uncertainty is approximately 2.6 km s$^{-1}$.

Inconclusive identification.

<table>
<thead>
<tr>
<th>Target $^a$</th>
<th>Sp-Type $^\beta$</th>
<th>RA $^\gamma$</th>
<th>Dec $^\delta$</th>
<th>Dist $^\epsilon$</th>
<th>W $\gamma$</th>
<th>N $\gamma$</th>
<th>v $\gamma$ $^\zeta$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>hrs</td>
<td>deg</td>
<td>pcs</td>
<td>mA</td>
<td>$10^{10}$ cm$^{-2}$</td>
<td>km s$^{-1}$</td>
</tr>
<tr>
<td>1</td>
<td>HD 332329 A0</td>
<td>20.49</td>
<td>31.39</td>
<td>793.65</td>
<td>93.2 $^{+49.0}_{-2.6}$</td>
<td>134.36 $^{+56.2}_{-4.1}$</td>
<td>-23.0</td>
</tr>
<tr>
<td>2</td>
<td>HD 192640 A2V</td>
<td>20.24</td>
<td>36.81</td>
<td>42.70</td>
<td>41.9 $^{+0.7}_{-0.7}$</td>
<td>44.8 $^{+2.0}_{-0.8}$</td>
<td>-40.0</td>
</tr>
<tr>
<td>3</td>
<td>HD 185732 A5</td>
<td>19.62</td>
<td>60.71</td>
<td>76.10</td>
<td>56.8 $^{+40.3}_{-1.3}$</td>
<td>42.4 $^{+30.6}_{-1.8}$</td>
<td>-30.7</td>
</tr>
<tr>
<td>4</td>
<td>HD 184471 A8...</td>
<td>19.56</td>
<td>32.58</td>
<td>507.61</td>
<td>8.6 $^{+8.3}_{-2.1}$</td>
<td>4.4 $^{+4.3}_{-1.1}$</td>
<td>-32.8</td>
</tr>
<tr>
<td>5</td>
<td>HD 191747 A3III</td>
<td>20.18</td>
<td>26.90</td>
<td>129.87</td>
<td>8.5 $^{+8.1}_{-0.6}$</td>
<td>7.4 $^{+5.4}_{-0.6}$</td>
<td>-32.8</td>
</tr>
<tr>
<td></td>
<td>BD+314105 A2</td>
<td>20.49</td>
<td>31.69</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>7</td>
<td>HD 184875 A2V</td>
<td>19.58</td>
<td>42.41</td>
<td>176.68</td>
<td>34.5 $^{+28.5}_{-1.2}$</td>
<td>14.8 $^{+12.9}_{-0.7}$</td>
<td>-30.2</td>
</tr>
<tr>
<td>8</td>
<td>HD 184006 A5V</td>
<td>19.50</td>
<td>51.73</td>
<td>37.20</td>
<td>12.5 $^{+11.7}_{-0.8}$</td>
<td>5.1 $^{+6.9}_{-0.5}$</td>
<td>-14.9</td>
</tr>
<tr>
<td></td>
<td>BD+314105 A2</td>
<td>20.49</td>
<td>31.69</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>9</td>
<td>HD 182989 A8-F7</td>
<td>19.42</td>
<td>42.78</td>
<td>289.02</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>HD 190397 A0</td>
<td>20.03</td>
<td>57.65</td>
<td>348.43</td>
<td>134.8 $^{+38.0}_{-2.8}$</td>
<td>198.1 $^{+34.8}_{-2.3}$</td>
<td>-29.4</td>
</tr>
<tr>
<td></td>
<td>BD+293448 F0p...</td>
<td>19.03</td>
<td>30.03</td>
<td>1388.89</td>
<td>86.4 $^{+43.4}_{-8.1}$</td>
<td>86.9 $^{+23.3}_{-5.2}$</td>
<td>-4.9</td>
</tr>
<tr>
<td></td>
<td>BD+344217 G9.5Ve+K3Ve</td>
<td>20.97</td>
<td>35.17</td>
<td>83.40</td>
<td>76.1 $^{+49.4}_{-3.2}$</td>
<td>80.0 $^{+29.1}_{-2.2}$</td>
<td>-27.3</td>
</tr>
</tbody>
</table>

$^a$ Numbers correspond to annotations in Figure 5.2.

$^\beta$ Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.

$^\gamma$ Measurements and uncertainties computed with Profile module.

$^\epsilon$ Equivalent Width for Na I D2.

$^\zeta$ Corrected Column Density for Na I line, see text (Savage and Sembach, 1991).

$^\zeta$ Velocity of Na I D2 line center, from 5,889.951 Å. Uncertainty is approximately 2.6 km s$^{-1}$.

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

$^a$ Numbers correspond to annotations in Figure 5.2.

$^b$ Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.

$^\gamma$ Measurements and uncertainties computed with Profile module.

$^\delta$ Equivalent Width for Na I D2.

$^\epsilon$ Corrected Column Density for Na I line, see text (Savage and Sembach, 1991).

$^\zeta$ Velocity of Na I D2 line center, from 5,889.951 Å. Uncertainty is approximately 2.6 km s$^{-1}$.

$^\eta$ Inconclusive identification.

$^\theta$ Too Saturated to apply optical depth method.
**TABLE 5.3**
Non-OB Targets with No Absorption – A

<table>
<thead>
<tr>
<th>Target $^\alpha$</th>
<th>Sp-Type $^\beta$</th>
<th>RA $^\beta$</th>
<th>Dec $^\beta$</th>
<th>Dist $^\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>hrs</td>
<td>deg</td>
<td>pcs</td>
</tr>
<tr>
<td>1 HD 335238</td>
<td>A2</td>
<td>20.85</td>
<td>29.80</td>
<td>952.38</td>
</tr>
<tr>
<td>2 HD 196542</td>
<td>Ap...</td>
<td>20.61</td>
<td>39.10</td>
<td>...</td>
</tr>
<tr>
<td>3 HD 173648</td>
<td>Am</td>
<td>18.75</td>
<td>37.61</td>
<td>47.87</td>
</tr>
<tr>
<td>4 HD 191158</td>
<td>Am</td>
<td>20.12</td>
<td>36.83</td>
<td>88.11</td>
</tr>
<tr>
<td>5 HD 174567</td>
<td>A0Vs</td>
<td>18.83</td>
<td>31.63</td>
<td>278.55</td>
</tr>
<tr>
<td>6 HD 180583</td>
<td>F6lb-II</td>
<td>19.27</td>
<td>27.93</td>
<td>429.18</td>
</tr>
<tr>
<td>7 HD 174912</td>
<td>F8</td>
<td>18.86</td>
<td>38.63</td>
<td>30.56</td>
</tr>
<tr>
<td>8 HD 181096</td>
<td>F6IV:</td>
<td>19.28</td>
<td>47.00</td>
<td>42.03</td>
</tr>
<tr>
<td>9 BD+443670</td>
<td>F8</td>
<td>21.01</td>
<td>45.50</td>
<td>...</td>
</tr>
<tr>
<td>10 HD 183361</td>
<td>F5</td>
<td>19.45</td>
<td>50.15</td>
<td>74.57</td>
</tr>
<tr>
<td>11 HD 171635</td>
<td>F7Ib</td>
<td>18.54</td>
<td>57.05</td>
<td>649.35</td>
</tr>
<tr>
<td>12 HD 176051</td>
<td>F9V+K1V</td>
<td>18.95</td>
<td>32.90</td>
<td>14.87</td>
</tr>
<tr>
<td>13 HD 171485</td>
<td>F5</td>
<td>18.55</td>
<td>40.16</td>
<td>41.70</td>
</tr>
<tr>
<td>14 HD 195992</td>
<td>F5</td>
<td>20.56</td>
<td>33.39</td>
<td>62.54</td>
</tr>
<tr>
<td>15 HD 195295</td>
<td>F5Iab:</td>
<td>20.49</td>
<td>30.37</td>
<td>235.85</td>
</tr>
<tr>
<td>16 HD 185395</td>
<td>F3+V</td>
<td>19.61</td>
<td>50.22</td>
<td>18.34</td>
</tr>
<tr>
<td>17 HD 180712</td>
<td>F8</td>
<td>19.23</td>
<td>59.55</td>
<td>44.56</td>
</tr>
<tr>
<td>18 HD 195872</td>
<td>F8</td>
<td>20.52</td>
<td>56.89</td>
<td>66.31</td>
</tr>
<tr>
<td>19 HD 173700</td>
<td>F8</td>
<td>18.72</td>
<td>57.82</td>
<td>85.40</td>
</tr>
<tr>
<td>20 HD 192804</td>
<td>F8V</td>
<td>20.26</td>
<td>31.24</td>
<td>76.22</td>
</tr>
<tr>
<td>21 HD 172323</td>
<td>F9V</td>
<td>18.59</td>
<td>63.70</td>
<td>43.65</td>
</tr>
<tr>
<td>22 HD 198084</td>
<td>F8IV-V+F9IV-V</td>
<td>20.76</td>
<td>57.58</td>
<td>27.29</td>
</tr>
<tr>
<td>23 HD 184960</td>
<td>F7V</td>
<td>19.57</td>
<td>51.24</td>
<td>25.11</td>
</tr>
<tr>
<td>24 HD 184448</td>
<td>F8</td>
<td>19.53</td>
<td>50.18</td>
<td>50.63</td>
</tr>
<tr>
<td>25 HD 186408</td>
<td>G1.5Vb</td>
<td>19.70</td>
<td>50.53</td>
<td>21.08</td>
</tr>
<tr>
<td>26 HD 196850</td>
<td>G1V</td>
<td>20.64</td>
<td>38.64</td>
<td>27.16</td>
</tr>
<tr>
<td>27 HD 175306</td>
<td>G9III_Fe-0.5</td>
<td>18.85</td>
<td>59.39</td>
<td>104.82</td>
</tr>
<tr>
<td>28 HD 199191</td>
<td>GIII+...</td>
<td>20.89</td>
<td>54.52</td>
<td>151.06</td>
</tr>
<tr>
<td>29 HD 185501</td>
<td>G5</td>
<td>19.63</td>
<td>33.89</td>
<td>31.96</td>
</tr>
<tr>
<td>30 HD 191022</td>
<td>G0</td>
<td>20.09</td>
<td>48.57</td>
<td>50.33</td>
</tr>
</tbody>
</table>

$^\alpha$ Numbers correspond to annotations in Figure 5.3.

$^\beta$ Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.
<table>
<thead>
<tr>
<th>Target $^\alpha$</th>
<th>Sp-Type $^\beta$</th>
<th>RA $^\beta$</th>
<th>Dec $^\beta$</th>
<th>Dist $^\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD 191649</td>
<td>G0</td>
<td>20.15</td>
<td>50.59</td>
<td>45.27</td>
</tr>
<tr>
<td>HD 185414</td>
<td>G0</td>
<td>19.60</td>
<td>56.98</td>
<td>24.11</td>
</tr>
<tr>
<td>HD 187237</td>
<td>G2IV-V</td>
<td>19.80</td>
<td>27.87</td>
<td>26.24</td>
</tr>
<tr>
<td>HD 186427</td>
<td>G3V</td>
<td>19.70</td>
<td>50.52</td>
<td>21.21</td>
</tr>
<tr>
<td>HD 188326</td>
<td>G8IVv</td>
<td>19.88</td>
<td>38.77</td>
<td>55.93</td>
</tr>
<tr>
<td>HD 187462</td>
<td>G0IV</td>
<td>19.82</td>
<td>27.73</td>
<td>50.38</td>
</tr>
<tr>
<td>HD 197310</td>
<td>G0</td>
<td>20.69</td>
<td>35.55</td>
<td>...</td>
</tr>
<tr>
<td>HD 195988</td>
<td>G0</td>
<td>20.55</td>
<td>41.47</td>
<td>...</td>
</tr>
<tr>
<td>HD 187123</td>
<td>G5</td>
<td>19.78</td>
<td>34.42</td>
<td>48.26</td>
</tr>
<tr>
<td>HD 198483</td>
<td>G0V</td>
<td>20.83</td>
<td>25.77</td>
<td>51.23</td>
</tr>
<tr>
<td>HD 198809</td>
<td>G7III</td>
<td>20.87</td>
<td>27.10</td>
<td>57.80</td>
</tr>
<tr>
<td>HD 184499</td>
<td>G0V</td>
<td>19.56</td>
<td>33.20</td>
<td>31.92</td>
</tr>
<tr>
<td>HD 188650</td>
<td>G1Ib-IIe...</td>
<td>19.91</td>
<td>37.00</td>
<td>369.00</td>
</tr>
<tr>
<td>HD 176377</td>
<td>G1V</td>
<td>18.98</td>
<td>30.18</td>
<td>23.84</td>
</tr>
<tr>
<td>HD 197666</td>
<td>G0</td>
<td>20.72</td>
<td>52.30</td>
<td>79.49</td>
</tr>
<tr>
<td>HD 187792</td>
<td>G0</td>
<td>19.80</td>
<td>63.86</td>
<td>55.22</td>
</tr>
<tr>
<td>HD 187748</td>
<td>G0</td>
<td>19.80</td>
<td>59.42</td>
<td>27.71</td>
</tr>
<tr>
<td>HD 187876</td>
<td>G0</td>
<td>19.82</td>
<td>25.77</td>
<td>46.93</td>
</tr>
<tr>
<td>HD 190771</td>
<td>G2V</td>
<td>20.09</td>
<td>38.48</td>
<td>18.79</td>
</tr>
<tr>
<td>HD 189749</td>
<td>G</td>
<td>19.96</td>
<td>62.96</td>
<td>56.98</td>
</tr>
<tr>
<td>HD 170357</td>
<td>G1V</td>
<td>18.44</td>
<td>46.08</td>
<td>71.99</td>
</tr>
<tr>
<td>HD 182758</td>
<td>G0</td>
<td>19.42</td>
<td>31.80</td>
<td>61.46</td>
</tr>
<tr>
<td>HD 181047</td>
<td>G8V</td>
<td>19.30</td>
<td>25.37</td>
<td>46.82</td>
</tr>
<tr>
<td>HD 196361</td>
<td>G5</td>
<td>20.59</td>
<td>36.47</td>
<td>66.40</td>
</tr>
<tr>
<td>HD 172393</td>
<td>G5</td>
<td>18.63</td>
<td>42.67</td>
<td>28.08</td>
</tr>
<tr>
<td>HD 175441</td>
<td>G0</td>
<td>18.86</td>
<td>60.43</td>
<td>87.41</td>
</tr>
<tr>
<td>HD 185269</td>
<td>G0IV</td>
<td>19.62</td>
<td>28.50</td>
<td>50.28</td>
</tr>
<tr>
<td>HD 176670</td>
<td>K2.5III</td>
<td>19.00</td>
<td>32.15</td>
<td>338.98</td>
</tr>
<tr>
<td>HD 188947</td>
<td>K0III</td>
<td>19.94</td>
<td>35.08</td>
<td>41.37</td>
</tr>
<tr>
<td>HD 199870</td>
<td>K0IIIbCN...</td>
<td>20.97</td>
<td>44.47</td>
<td>79.30</td>
</tr>
</tbody>
</table>

$^\alpha$ Numbers correspond to annotations in Figure 5.3.

$^\beta$ Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.
<table>
<thead>
<tr>
<th>Target</th>
<th>Sp-Type</th>
<th>RA (%)</th>
<th>Dec (%)</th>
<th>Dist (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BD+293427</td>
<td>A5p...</td>
<td>18.99</td>
<td>29.80</td>
<td>...</td>
</tr>
<tr>
<td>HD 340577</td>
<td>A3</td>
<td>20.58</td>
<td>26.49</td>
<td>699.30</td>
</tr>
<tr>
<td>HD 200405</td>
<td>A0pSrCr</td>
<td>21.03</td>
<td>47.91</td>
<td>746.27</td>
</tr>
<tr>
<td>HD 191742</td>
<td>A7p</td>
<td>20.16</td>
<td>42.54</td>
<td>304.88</td>
</tr>
<tr>
<td>HD 192678</td>
<td>A4p</td>
<td>20.23</td>
<td>53.66</td>
<td>198.02</td>
</tr>
<tr>
<td>HD 195725</td>
<td>A7III</td>
<td>20.49</td>
<td>62.99</td>
<td>41.84</td>
</tr>
<tr>
<td>HD 177196</td>
<td>A6IV</td>
<td>19.02</td>
<td>46.93</td>
<td>37.44</td>
</tr>
<tr>
<td>HD 183534</td>
<td>A1V</td>
<td>19.46</td>
<td>52.32</td>
<td>107.53</td>
</tr>
<tr>
<td>BD+353616A</td>
<td>F0</td>
<td>19.44</td>
<td>35.71</td>
<td>...</td>
</tr>
<tr>
<td>HD 341037</td>
<td>F0</td>
<td>20.78</td>
<td>26.36</td>
<td>3333.33</td>
</tr>
<tr>
<td>HD 197572</td>
<td>F7Ib...</td>
<td>20.72</td>
<td>35.59</td>
<td>1538.46</td>
</tr>
<tr>
<td>HD 198726</td>
<td>F5Ib+A0.8V</td>
<td>20.86</td>
<td>28.25</td>
<td>369.00</td>
</tr>
<tr>
<td>BD+423935</td>
<td>F5.5Ib-G5Ib:</td>
<td>21.00</td>
<td>42.60</td>
<td>-2564.10</td>
</tr>
<tr>
<td>HD 238932</td>
<td>F5</td>
<td>18.74</td>
<td>56.78</td>
<td>120.92</td>
</tr>
<tr>
<td>BD+423607</td>
<td>F3</td>
<td>20.15</td>
<td>42.87</td>
<td>82.58</td>
</tr>
<tr>
<td>HD 194951</td>
<td>F1II</td>
<td>20.45</td>
<td>34.33</td>
<td>1000.00</td>
</tr>
<tr>
<td>HD 331319</td>
<td>F3Ib</td>
<td>19.82</td>
<td>31.45</td>
<td>...</td>
</tr>
<tr>
<td>HD 195069</td>
<td>F0V:</td>
<td>20.45</td>
<td>49.38</td>
<td>37.57</td>
</tr>
<tr>
<td>HD 193370</td>
<td>F6Ib</td>
<td>20.31</td>
<td>34.98</td>
<td>970.87</td>
</tr>
<tr>
<td>HD 200805</td>
<td>F5Ib</td>
<td>21.07</td>
<td>45.16</td>
<td>373.13</td>
</tr>
<tr>
<td>HD 178593</td>
<td>F8</td>
<td>19.14</td>
<td>25.37</td>
<td>61.35</td>
</tr>
<tr>
<td>HD 188307</td>
<td>F8</td>
<td>19.88</td>
<td>41.08</td>
<td>53.30</td>
</tr>
<tr>
<td>HD 187253</td>
<td>F8</td>
<td>19.77</td>
<td>54.64</td>
<td>72.41</td>
</tr>
<tr>
<td>HD 185239</td>
<td>F8</td>
<td>19.58</td>
<td>57.77</td>
<td>84.53</td>
</tr>
<tr>
<td>HD 175225</td>
<td>G9IVa</td>
<td>18.86</td>
<td>52.98</td>
<td>25.67</td>
</tr>
<tr>
<td>HD 173701</td>
<td>G8V</td>
<td>18.74</td>
<td>43.83</td>
<td>26.69</td>
</tr>
<tr>
<td>HD 177153</td>
<td>G0</td>
<td>19.03</td>
<td>41.49</td>
<td>41.48</td>
</tr>
<tr>
<td>HD 185351</td>
<td>G9IIIbCN...</td>
<td>19.61</td>
<td>44.69</td>
<td>40.83</td>
</tr>
<tr>
<td>BD+413931</td>
<td>G5</td>
<td>20.92</td>
<td>42.30</td>
<td>67.29</td>
</tr>
<tr>
<td>HD 175306</td>
<td>G9III_Fe-0.5</td>
<td>18.85</td>
<td>59.39</td>
<td>104.82</td>
</tr>
</tbody>
</table>

* Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.
TABLE 5.6
Non-OB Targets Inconclusive – B

<table>
<thead>
<tr>
<th>Target</th>
<th>Sp-Type</th>
<th>RA γ</th>
<th>Dec γ</th>
<th>Dist γ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>hrs</td>
<td>deg</td>
<td>pcs</td>
</tr>
<tr>
<td>HD 182488</td>
<td>G9+V</td>
<td>19.39</td>
<td>33.22</td>
<td>15.76</td>
</tr>
<tr>
<td>HD 175900</td>
<td>G5</td>
<td>18.91</td>
<td>51.31</td>
<td>80.97</td>
</tr>
<tr>
<td>HD 193795</td>
<td>G4IV</td>
<td>20.35</td>
<td>28.11</td>
<td>70.67</td>
</tr>
<tr>
<td>HD 200391</td>
<td>G0V+G5V</td>
<td>21.04</td>
<td>27.81</td>
<td>52.00</td>
</tr>
<tr>
<td>HD 181655</td>
<td>G5V</td>
<td>19.33</td>
<td>37.33</td>
<td>25.39</td>
</tr>
<tr>
<td>BD+364301</td>
<td>G0</td>
<td>20.89</td>
<td>36.70</td>
<td>...</td>
</tr>
<tr>
<td>HD 179484</td>
<td>G5V</td>
<td>19.19</td>
<td>38.78</td>
<td>52.85</td>
</tr>
<tr>
<td>HD 190360</td>
<td>G7IV-V</td>
<td>20.06</td>
<td>29.90</td>
<td>15.86</td>
</tr>
<tr>
<td>HD 178911</td>
<td>G1V+K1V</td>
<td>19.15</td>
<td>34.60</td>
<td>52.33</td>
</tr>
<tr>
<td>HD 199598</td>
<td>G0V</td>
<td>20.96</td>
<td>26.41</td>
<td>31.63</td>
</tr>
<tr>
<td>HD 173605</td>
<td>G5</td>
<td>18.71</td>
<td>57.87</td>
<td>65.70</td>
</tr>
<tr>
<td>HD 190228</td>
<td>G5IV</td>
<td>20.05</td>
<td>28.31</td>
<td>61.61</td>
</tr>
<tr>
<td>HD 188326</td>
<td>G8IVv</td>
<td>19.88</td>
<td>38.77</td>
<td>55.93</td>
</tr>
<tr>
<td>HD 187123</td>
<td>G5</td>
<td>19.78</td>
<td>34.42</td>
<td>48.26</td>
</tr>
<tr>
<td>HD 182736</td>
<td>G8IV</td>
<td>19.40</td>
<td>44.93</td>
<td>57.64</td>
</tr>
<tr>
<td>HD 171242</td>
<td>G0</td>
<td>18.53</td>
<td>45.01</td>
<td>61.92</td>
</tr>
<tr>
<td>HD 185657</td>
<td>G6V</td>
<td>19.63</td>
<td>49.28</td>
<td>140.25</td>
</tr>
<tr>
<td>HD 171008</td>
<td>G5</td>
<td>18.47</td>
<td>60.44</td>
<td>79.30</td>
</tr>
<tr>
<td>HD 193216</td>
<td>G5</td>
<td>20.28</td>
<td>50.28</td>
<td>30.77</td>
</tr>
<tr>
<td>HD 178911B</td>
<td>G5</td>
<td>19.15</td>
<td>34.60</td>
<td>42.59</td>
</tr>
<tr>
<td>HD 200102</td>
<td>G1Ib</td>
<td>21.00</td>
<td>45.00</td>
<td>1250.00</td>
</tr>
<tr>
<td>HD 190403</td>
<td>G5II-II</td>
<td>20.06</td>
<td>29.99</td>
<td>588.24</td>
</tr>
<tr>
<td>HD 174104</td>
<td>G0Ib</td>
<td>18.80</td>
<td>28.72</td>
<td>1315.79</td>
</tr>
<tr>
<td>HD 191010</td>
<td>G3Ib</td>
<td>20.11</td>
<td>25.69</td>
<td>518.13</td>
</tr>
<tr>
<td>HD 180161</td>
<td>G8V</td>
<td>19.20</td>
<td>57.67</td>
<td>20.02</td>
</tr>
<tr>
<td>HD 180683</td>
<td>G0</td>
<td>19.26</td>
<td>38.38</td>
<td>64.52</td>
</tr>
<tr>
<td>HD 197037</td>
<td>G0</td>
<td>20.66</td>
<td>42.25</td>
<td>32.33</td>
</tr>
<tr>
<td>HD 180502</td>
<td>G0IV</td>
<td>19.26</td>
<td>29.12</td>
<td>85.54</td>
</tr>
<tr>
<td>HD 197488</td>
<td>G2V</td>
<td>20.71</td>
<td>45.82</td>
<td>54.14</td>
</tr>
<tr>
<td>HD 175425</td>
<td>G0</td>
<td>18.89</td>
<td>37.99</td>
<td>53.53</td>
</tr>
</tbody>
</table>

γ Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.
### TABLE 5.7
Non-OB Targets Inconclusive – C

<table>
<thead>
<tr>
<th>Target</th>
<th>Sp-Type</th>
<th>RA $^\gamma$</th>
<th>Dec $^\gamma$</th>
<th>Dist $^\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>hrs</td>
<td>deg</td>
<td>pcs</td>
</tr>
<tr>
<td>HD 193215</td>
<td>G5</td>
<td>20.26</td>
<td>64.20</td>
<td>68.97</td>
</tr>
<tr>
<td>HD 172557</td>
<td>G5</td>
<td>18.61</td>
<td>63.36</td>
<td>52.63</td>
</tr>
<tr>
<td>HD 184601</td>
<td>G0</td>
<td>19.53</td>
<td>60.86</td>
<td>83.68</td>
</tr>
<tr>
<td>HD 199100</td>
<td>G5IV-V</td>
<td>20.89</td>
<td>36.13</td>
<td>57.60</td>
</tr>
<tr>
<td>HD 195987</td>
<td>G9V</td>
<td>20.55</td>
<td>41.90</td>
<td>22.05</td>
</tr>
<tr>
<td>HD 187548</td>
<td>G0V</td>
<td>19.82</td>
<td>28.61</td>
<td>45.11</td>
</tr>
<tr>
<td>HD 177780</td>
<td>G3V</td>
<td>19.07</td>
<td>41.00</td>
<td>55.01</td>
</tr>
<tr>
<td>HD 190605</td>
<td>G2V</td>
<td>20.08</td>
<td>26.05</td>
<td>44.98</td>
</tr>
<tr>
<td>HD 178911</td>
<td>G1V+K1V</td>
<td>19.15</td>
<td>34.60</td>
<td>52.33</td>
</tr>
<tr>
<td>HD 178450</td>
<td>G8VSBe</td>
<td>19.13</td>
<td>30.25</td>
<td>27.96</td>
</tr>
<tr>
<td>HD 173024</td>
<td>G5</td>
<td>18.69</td>
<td>40.48</td>
<td>80.97</td>
</tr>
<tr>
<td>HD 173289</td>
<td>G5</td>
<td>18.71</td>
<td>44.27</td>
<td>93.81</td>
</tr>
<tr>
<td>HD 197140</td>
<td>G5</td>
<td>20.67</td>
<td>38.78</td>
<td>53.56</td>
</tr>
<tr>
<td>HD 197207</td>
<td>G5V</td>
<td>20.69</td>
<td>30.19</td>
<td>55.46</td>
</tr>
<tr>
<td>HD 189087</td>
<td>K1V</td>
<td>19.95</td>
<td>29.82</td>
<td>26.72</td>
</tr>
<tr>
<td>HD 234677</td>
<td>K4Ve+K7.5Ve</td>
<td>18.57</td>
<td>51.72</td>
<td>16.35</td>
</tr>
<tr>
<td>HD 197989</td>
<td>K0III-IV</td>
<td>20.77</td>
<td>33.97</td>
<td>22.29</td>
</tr>
<tr>
<td>HD 183255</td>
<td>K2.5V</td>
<td>19.44</td>
<td>49.47</td>
<td>25.01</td>
</tr>
<tr>
<td>HD 191026</td>
<td>K0IV</td>
<td>20.11</td>
<td>35.97</td>
<td>23.95</td>
</tr>
<tr>
<td>HD 33867</td>
<td>K0</td>
<td>19.81</td>
<td>27.29</td>
<td></td>
</tr>
<tr>
<td>HD 199956</td>
<td>K0</td>
<td>20.98</td>
<td>44.06</td>
<td>201.61</td>
</tr>
<tr>
<td>HD 334514</td>
<td>K0</td>
<td>20.56</td>
<td>31.47</td>
<td></td>
</tr>
<tr>
<td>HD 199546</td>
<td>K0</td>
<td>20.93</td>
<td>53.81</td>
<td>452.49</td>
</tr>
<tr>
<td>HD 189806</td>
<td>K2</td>
<td>20.01</td>
<td>32.95</td>
<td>189.75</td>
</tr>
<tr>
<td>HD 181209</td>
<td>K0</td>
<td>19.30</td>
<td>34.14</td>
<td>432.90</td>
</tr>
<tr>
<td>HD 188056</td>
<td>K3III</td>
<td>19.84</td>
<td>52.99</td>
<td>62.07</td>
</tr>
<tr>
<td>BD+333930</td>
<td>K0V</td>
<td>20.52</td>
<td>33.54</td>
<td>48.85</td>
</tr>
<tr>
<td>BD+334140</td>
<td>K0</td>
<td>21.06</td>
<td>34.22</td>
<td></td>
</tr>
<tr>
<td>HD 192787</td>
<td>K0III</td>
<td>20.26</td>
<td>33.73</td>
<td>97.47</td>
</tr>
<tr>
<td>HD 196134</td>
<td>K0III-IV</td>
<td>20.56</td>
<td>41.77</td>
<td>92.08</td>
</tr>
</tbody>
</table>

$^\gamma$ Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.
### TABLE 5.8
Non-OB Targets Inconclusive – D

<table>
<thead>
<tr>
<th>Target</th>
<th>Sp-Type</th>
<th>RA</th>
<th>Dec</th>
<th>Dist</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>hrs</td>
<td>deg</td>
<td>pcs</td>
</tr>
<tr>
<td>HD 176408</td>
<td>K1III</td>
<td>18.95</td>
<td>57.81</td>
<td>84.67</td>
</tr>
<tr>
<td>HD 198550</td>
<td>K5V</td>
<td>20.84</td>
<td>29.38</td>
<td>20.99</td>
</tr>
<tr>
<td>HD 192910</td>
<td>K3b+</td>
<td>20.26</td>
<td>47.71</td>
<td>324.68</td>
</tr>
<tr>
<td>HD 193469</td>
<td>K3b</td>
<td>20.32</td>
<td>39.00</td>
<td>1492.54</td>
</tr>
<tr>
<td>HD 198794</td>
<td>K3b</td>
<td>20.85</td>
<td>48.03</td>
<td>232.02</td>
</tr>
<tr>
<td>HD 200560</td>
<td>K2.5V</td>
<td>21.04</td>
<td>45.88</td>
<td>19.47</td>
</tr>
<tr>
<td>HD 190470</td>
<td>K3V</td>
<td>20.07</td>
<td>25.79</td>
<td>21.95</td>
</tr>
<tr>
<td>BD+413306</td>
<td>K0V</td>
<td>19.32</td>
<td>41.63</td>
<td>35.68</td>
</tr>
<tr>
<td>BD+522815</td>
<td>K7-V</td>
<td>20.84</td>
<td>52.90</td>
<td>22.42</td>
</tr>
<tr>
<td>HD 332518</td>
<td>K5V</td>
<td>19.76</td>
<td>30.01</td>
<td>20.32</td>
</tr>
<tr>
<td>HD 331093</td>
<td>K0</td>
<td>19.71</td>
<td>31.74</td>
<td>55.07</td>
</tr>
<tr>
<td>HD 191806</td>
<td>K0</td>
<td>20.16</td>
<td>52.28</td>
<td>69.40</td>
</tr>
<tr>
<td>HD 171607</td>
<td>K0</td>
<td>18.53</td>
<td>61.78</td>
<td>46.75</td>
</tr>
<tr>
<td>HD 179094</td>
<td>K1IV</td>
<td>19.14</td>
<td>52.43</td>
<td>71.02</td>
</tr>
<tr>
<td>HD 198425</td>
<td>K2.5V</td>
<td>20.82</td>
<td>32.28</td>
<td>24.05</td>
</tr>
<tr>
<td>HD 188753</td>
<td>K0</td>
<td>19.92</td>
<td>41.87</td>
<td>46.23</td>
</tr>
<tr>
<td>BD+400883</td>
<td>M3.0V</td>
<td>21.00</td>
<td>40.07</td>
<td>15.29</td>
</tr>
<tr>
<td>HD 173739</td>
<td>M3.0V</td>
<td>18.71</td>
<td>59.63</td>
<td>3.57</td>
</tr>
<tr>
<td>GJ 0747</td>
<td>M3V</td>
<td>19.13</td>
<td>32.54</td>
<td>8.13</td>
</tr>
<tr>
<td>GJ 0809</td>
<td>M2.0V</td>
<td>20.89</td>
<td>62.15</td>
<td>7.05</td>
</tr>
<tr>
<td>HD 186686</td>
<td>M2-7e</td>
<td>19.73</td>
<td>48.78</td>
<td>-3448.28</td>
</tr>
<tr>
<td>GJ 0725</td>
<td>M3.0V</td>
<td>18.71</td>
<td>59.63</td>
<td>3.57</td>
</tr>
<tr>
<td>BD+394208</td>
<td>M3lab:</td>
<td>20.48</td>
<td>39.98</td>
<td>1098.90</td>
</tr>
<tr>
<td>BD+364025</td>
<td>M4lab</td>
<td>20.36</td>
<td>36.93</td>
<td>133.69</td>
</tr>
<tr>
<td>HD 182190</td>
<td>M1III</td>
<td>19.34</td>
<td>57.65</td>
<td>242.13</td>
</tr>
<tr>
<td>HD 199871</td>
<td>M0III</td>
<td>20.97</td>
<td>41.36</td>
<td>213.68</td>
</tr>
<tr>
<td>HD 186776</td>
<td>M4III</td>
<td>19.75</td>
<td>40.72</td>
<td>251.26</td>
</tr>
<tr>
<td>HD 175865</td>
<td>M5III</td>
<td>18.92</td>
<td>43.95</td>
<td>91.41</td>
</tr>
<tr>
<td>HD 175588</td>
<td>M4II</td>
<td>18.91</td>
<td>36.90</td>
<td>225.73</td>
</tr>
<tr>
<td>HD 184786</td>
<td>M4.5III</td>
<td>19.56</td>
<td>49.26</td>
<td>406.50</td>
</tr>
</tbody>
</table>

γ Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.
<table>
<thead>
<tr>
<th>Target</th>
<th>Sp-Type</th>
<th>RA £</th>
<th>Dec £</th>
<th>Dist £</th>
</tr>
</thead>
<tbody>
<tr>
<td>GJ 0815</td>
<td>M3.0V</td>
<td>21.00</td>
<td>40.07</td>
<td>15.29</td>
</tr>
<tr>
<td>GJ 0766</td>
<td>M4.5V</td>
<td>19.76</td>
<td>27.13</td>
<td>10.31</td>
</tr>
<tr>
<td>HD 189063</td>
<td>M0III</td>
<td>19.91</td>
<td>60.82</td>
<td>540.54</td>
</tr>
<tr>
<td>HD 178003</td>
<td>M0III</td>
<td>19.10</td>
<td>29.92</td>
<td>446.43</td>
</tr>
<tr>
<td>HD 180450</td>
<td>M0III</td>
<td>19.26</td>
<td>30.53</td>
<td>389.11</td>
</tr>
<tr>
<td>HD 179869</td>
<td>M3III</td>
<td>19.20</td>
<td>41.24</td>
<td>317.46</td>
</tr>
<tr>
<td>HD 187849</td>
<td>M2III</td>
<td>19.84</td>
<td>38.72</td>
<td>212.31</td>
</tr>
<tr>
<td>HD 190544</td>
<td>M1III</td>
<td>20.02</td>
<td>64.82</td>
<td>180.18</td>
</tr>
<tr>
<td>HD 199305</td>
<td>M2.0V</td>
<td>20.89</td>
<td>62.15</td>
<td>7.05</td>
</tr>
<tr>
<td>HD 173740</td>
<td>M3.5V</td>
<td>18.71</td>
<td>59.63</td>
<td>3.45</td>
</tr>
<tr>
<td>HD 331161 £</td>
<td>K5</td>
<td>19.77</td>
<td>32.02</td>
<td>13.61</td>
</tr>
<tr>
<td>HD 190163 £</td>
<td>M5.5-6.5e</td>
<td>20.02</td>
<td>50.04</td>
<td>401.61</td>
</tr>
<tr>
<td>HD 182917 £</td>
<td>M7IIIab+Be</td>
<td>19.41</td>
<td>50.24</td>
<td>242.72</td>
</tr>
<tr>
<td>GJ 0802 £</td>
<td>M5.0V</td>
<td>20.72</td>
<td>55.35</td>
<td>16.67</td>
</tr>
</tbody>
</table>

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

£ Inconclusive because the spectrum was bad (see text).
### TABLE 5.10
**OB Targets – A**

<table>
<thead>
<tr>
<th>Target</th>
<th>Sp-Type</th>
<th>RA $^\beta$ (hrs)</th>
<th>Dec $^\beta$ (deg)</th>
<th>Dist $^\beta$ (pcs)</th>
<th>W $^\gamma$ (mA)</th>
<th>N $^\gamma$ (10$^{18}$ cm$^{-2}$)</th>
<th>$\gamma$ $^\zeta$ (kms$^{-1}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD 188209</td>
<td>O9.5Iab</td>
<td>19.87</td>
<td>47.03</td>
<td>1098.90</td>
<td>212.0 $^{+27.9}_{-3.0}$</td>
<td>88.4 $^{+26.1}_{-10.0}$</td>
<td>24.9</td>
</tr>
<tr>
<td>HD 198820</td>
<td>B3III</td>
<td>20.87</td>
<td>32.85</td>
<td>531.91</td>
<td>92.8 $^{+53.2}_{-1.5}$</td>
<td>44.8 $^{+29.7}_{-0.7}$</td>
<td>-0.5</td>
</tr>
<tr>
<td>HD 194335</td>
<td>B2IIIe</td>
<td>20.40</td>
<td>37.48</td>
<td>414.94</td>
<td>101.6 $^{+46.5}_{-1.2}$</td>
<td>118.5 $^{+28.2}_{-0.8}$</td>
<td>-20.8</td>
</tr>
<tr>
<td>HD 200120</td>
<td>B1.5Vnne</td>
<td>21.00</td>
<td>47.52</td>
<td>434.78</td>
<td>86.0 $^{+46.4}_{-1.0}$</td>
<td>94.8 $^{+29.7}_{-0.7}$</td>
<td>-29.9</td>
</tr>
<tr>
<td>HD 182255</td>
<td>B6III</td>
<td>19.38</td>
<td>26.26</td>
<td>120.48</td>
<td>16.6 $^{+15.4}_{-1.1}$</td>
<td>11.6 $^{+4.8}_{-0.6}$</td>
<td>-5.7</td>
</tr>
<tr>
<td>HD 338529</td>
<td>B5</td>
<td>19.54</td>
<td>26.39</td>
<td>137.93</td>
<td>40.2 $^{+31.6}_{-1.9}$</td>
<td>64.4 $^{+14.2}_{-1.1}$</td>
<td>-24.4</td>
</tr>
<tr>
<td>HD 183056</td>
<td>B9sp...</td>
<td>19.44</td>
<td>36.32</td>
<td>216.92</td>
<td>22.8 $^{+20.0}_{-0.7}$</td>
<td>13.6 $^{+6.0}_{-0.4}$</td>
<td>-33.3</td>
</tr>
<tr>
<td>HD 174638</td>
<td>B8III-IIIep</td>
<td>18.83</td>
<td>33.36</td>
<td>294.99</td>
<td>164.2 $^{+38.0}_{-2.4}$</td>
<td>643.8 $^{+48.4}_{-3.5}$</td>
<td>-17.5</td>
</tr>
<tr>
<td>HD 176437</td>
<td>B9III</td>
<td>18.98</td>
<td>32.69</td>
<td>190.11</td>
<td>19.7 $^{+17.6}_{-2.0}$</td>
<td>9.9 $^{+4.7}_{-1.1}$</td>
<td>-34.0</td>
</tr>
<tr>
<td>HD 180163</td>
<td>B2.5IV</td>
<td>19.23</td>
<td>39.15</td>
<td>425.53</td>
<td>96.9 $^{+46.6}_{-1.6}$</td>
<td>109.8 $^{+27.1}_{-1.1}$</td>
<td>-34.0</td>
</tr>
<tr>
<td>HD 192685</td>
<td>B3V</td>
<td>20.25</td>
<td>25.59</td>
<td>303.95</td>
<td>27.8 $^{+23.7}_{-1.1}$</td>
<td>14.9 $^{+6.9}_{-0.6}$</td>
<td>-31.2</td>
</tr>
<tr>
<td>HD 198183</td>
<td>B5V</td>
<td>20.79</td>
<td>36.49</td>
<td>235.85</td>
<td>174.1 $^{+70.6}_{-3.5}$</td>
<td>244.2 $^{+49.3}_{-2.6}$</td>
<td>-28.9</td>
</tr>
<tr>
<td>BD+404124</td>
<td>B2Ve</td>
<td>20.34</td>
<td>41.36</td>
<td>79.94</td>
<td>142.9 $^{+38.5}_{-3.1}$</td>
<td>152.7 $^{+36.3}_{-2.4}$</td>
<td>-41.0</td>
</tr>
<tr>
<td>HD 194279</td>
<td>B2Ia</td>
<td>20.39</td>
<td>40.76</td>
<td>1724.14</td>
<td>120.8 $^{+41.2}_{-2.2}$</td>
<td>222.0 $^{+33.2}_{-1.9}$</td>
<td>-39.9</td>
</tr>
<tr>
<td>HD 172324</td>
<td>B9Ib</td>
<td>18.63</td>
<td>37.43</td>
<td>1030.93</td>
<td>77.5 $^{+45.6}_{-1.6}$</td>
<td>80.3 $^{+22.3}_{-1.0}$</td>
<td>-32.8</td>
</tr>
<tr>
<td>BD+413731</td>
<td>B3n</td>
<td>20.40</td>
<td>42.30</td>
<td>2222.22</td>
<td>131.2 $^{+36.9}_{-4.6}$</td>
<td>513.6 $^{+35.0}_{-4.6}$</td>
<td>-38.5</td>
</tr>
<tr>
<td>HD 193322</td>
<td>O9V</td>
<td>20.30</td>
<td>40.73</td>
<td>595.24</td>
<td>190.6 $^{+43.1}_{-3.8}$</td>
<td>222.0 $^{+33.2}_{-1.9}$</td>
<td>-39.9</td>
</tr>
<tr>
<td>HD 195592</td>
<td>O9.7Ia</td>
<td>20.51</td>
<td>44.32</td>
<td>909.09</td>
<td>38.3 $^{+30.8}_{-1.6}$</td>
<td>38.3 $^{+30.8}_{-1.6}$</td>
<td>-27.5</td>
</tr>
<tr>
<td>HD 190864</td>
<td>O6.5III(f)</td>
<td>20.09</td>
<td>35.61</td>
<td>487.80</td>
<td>199.6 $^{+38.1}_{-3.8}$</td>
<td>222.0 $^{+33.2}_{-1.9}$</td>
<td>-39.9</td>
</tr>
<tr>
<td>HD 191423</td>
<td>ON9IIIIn</td>
<td>20.14</td>
<td>42.61</td>
<td>5263.16</td>
<td>239.1 $^{+10.6}_{-4.1}$</td>
<td>38.3 $^{+30.8}_{-1.6}$</td>
<td>-37.2</td>
</tr>
<tr>
<td>HD 189957</td>
<td>O9.7III</td>
<td>20.02</td>
<td>42.01</td>
<td>-2702.70</td>
<td>46.6 $^{+34.8}_{-2.6}$</td>
<td>199.6 $^{+38.1}_{-3.8}$</td>
<td>-38.3</td>
</tr>
<tr>
<td>HD 192639</td>
<td>7.5Iabf</td>
<td>20.24</td>
<td>37.35</td>
<td>1886.79</td>
<td>199.6 $^{+38.1}_{-3.8}$</td>
<td>222.0 $^{+33.2}_{-1.9}$</td>
<td>-39.9</td>
</tr>
<tr>
<td>HD 191978</td>
<td>O8V</td>
<td>20.18</td>
<td>41.35</td>
<td>1000000.00</td>
<td>38.3 $^{+30.8}_{-1.6}$</td>
<td>38.3 $^{+30.8}_{-1.6}$</td>
<td>-31.1</td>
</tr>
<tr>
<td>HD 190429</td>
<td>O4If+O9.5II</td>
<td>20.06</td>
<td>36.03</td>
<td>862.07</td>
<td>199.6 $^{+38.1}_{-3.8}$</td>
<td>222.0 $^{+33.2}_{-1.9}$</td>
<td>-39.9</td>
</tr>
</tbody>
</table>

$^\beta$ Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.

$^\gamma$ Measurements and uncertainties computed with Profile module.

$^\zeta$ Equivalent Width for Na I D2.

$^\zeta$ Corrected Column Density for Na I line, see text (Savage and Sembach, 1991).

$^\xi$ Velocity of Na I D2 line center, from 5895.924 Å.

$^\eta$ Multiple components. Closest velocity listed.

$^\theta$ Too Saturated to apply optical depth method.
TABLE 5.11
OB Targets – B

<table>
<thead>
<tr>
<th>Target</th>
<th>Sp-Type</th>
<th>RA $^\beta$</th>
<th>Dec $^\beta$</th>
<th>Dist $^\beta$</th>
<th>W $^\gamma$</th>
<th>N $^\epsilon$</th>
<th>$v$ $^\zeta$</th>
<th>hrs</th>
<th>deg</th>
<th>pcs</th>
<th>mA</th>
<th>$10^{10}$cm$^{-2}$</th>
<th>kms$^{-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta\eta$ HD 199579</td>
<td>O6.5V</td>
<td>20.94</td>
<td>44.92</td>
<td>1666.67</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$\delta\eta$ HD 193514</td>
<td>O7Ib</td>
<td>20.32</td>
<td>39.27</td>
<td>-925.93</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$\delta\eta$ HD 190429</td>
<td>O4If</td>
<td>20.06</td>
<td>36.03</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$\delta\eta$ HD 186980</td>
<td>O7.5III</td>
<td>19.77</td>
<td>32.12</td>
<td>-2631.58</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$\delta\eta$ HD 193443</td>
<td>O9III</td>
<td>20.31</td>
<td>38.28</td>
<td>1086.96</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$\delta\eta$ HD 191612</td>
<td>O8fpe</td>
<td>20.16</td>
<td>35.73</td>
<td>2040.82</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$\delta\eta$ HD 192281</td>
<td>O4.5Ve(){(f)}</td>
<td>20.21</td>
<td>40.27</td>
<td>1265.82</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$\delta\eta$ HD 191201</td>
<td>O9III+O9V</td>
<td>20.12</td>
<td>35.72</td>
<td>628.93</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$\delta\eta$ HD 193237</td>
<td>B1-2_Ia-0_ep</td>
<td>20.30</td>
<td>38.03</td>
<td>3125.00</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$\delta\eta$ HD 198478</td>
<td>B4I</td>
<td>20.82</td>
<td>46.11</td>
<td>714.29</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$\delta\eta$ HD 228712</td>
<td>B0.5Ia</td>
<td>20.28</td>
<td>40.89</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>$\delta\eta$ HD 194839</td>
<td>B0.5Iae</td>
<td>20.44</td>
<td>41.38</td>
<td>1041.67</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

$^\beta$ Retrieved from SIMBAD database (as of June 13, 2015); uncertainties not listed.
$^\gamma$ Measurements and uncertainties computed with Profile module.
$^\delta$ Equivalent Width for Na I D2.
$^\epsilon$ Corrected Column Density for Na I line, see text (Savage and Sembach, 1991).
$^\zeta$ Velocity of Na I D2 line center, from 5895.924 Å.
$^\eta$ Multiple components. Closest velocity listed.
$^\theta$ 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-investigator 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>
<thead>
<tr>
<th>Star</th>
<th>Spectral Type</th>
<th>$l^\alpha$ (deg)</th>
<th>$b^\alpha$ (deg)</th>
<th>Distance (parsecs)</th>
<th>Absorption</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta$ Cyg</td>
<td>B9.5IV</td>
<td>78.71</td>
<td>10.24</td>
<td>52</td>
<td>Y</td>
</tr>
<tr>
<td>HD 175824</td>
<td>FIII</td>
<td>78.65</td>
<td>19.56</td>
<td>53</td>
<td>Y</td>
</tr>
<tr>
<td>HD 179958</td>
<td>G4V</td>
<td>80.67</td>
<td>17.27</td>
<td>22</td>
<td>N</td>
</tr>
<tr>
<td>$\beta$ HD 184006</td>
<td>A5Vn</td>
<td>83.61</td>
<td>15.44</td>
<td>38</td>
<td>N?</td>
</tr>
<tr>
<td>$\beta$ HD 184960</td>
<td>F7V</td>
<td>83.46</td>
<td>14.59</td>
<td>26</td>
<td>N</td>
</tr>
<tr>
<td>$\beta$ HD 185351</td>
<td>G9III</td>
<td>77.62</td>
<td>11.35</td>
<td>41</td>
<td>Y</td>
</tr>
<tr>
<td>$\beta$ HD 185395</td>
<td>F4V</td>
<td>82.67</td>
<td>13.45</td>
<td>19</td>
<td>N</td>
</tr>
<tr>
<td>$\beta$ HD 186155</td>
<td>F5II-III</td>
<td>78.72</td>
<td>11.07</td>
<td>49</td>
<td>Y</td>
</tr>
<tr>
<td>HD 187160</td>
<td>G0</td>
<td>78.16</td>
<td>9.60</td>
<td>43</td>
<td>Y</td>
</tr>
<tr>
<td>HD 192485</td>
<td>F5V</td>
<td>81.92</td>
<td>5.79</td>
<td>35</td>
<td>N</td>
</tr>
<tr>
<td>$\beta$ HD 192640</td>
<td>A2V</td>
<td>74.45</td>
<td>1.17</td>
<td>41</td>
<td>Y</td>
</tr>
<tr>
<td>HD 192866</td>
<td>A3</td>
<td>81.99</td>
<td>5.99</td>
<td>343</td>
<td>Y</td>
</tr>
</tbody>
</table>

$^\alpha l$ and $b$ refer to galactic coordinates.

$^\beta$ 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 $\lambda$5889.591 and D1 $\lambda$5895.924) 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, \(\lambda \simeq 5810\text{Å}\), 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, 
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 

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. 
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 
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,


APPENDIX A

HUBBLE SPACE TELESCOPE PROPOSAL
The Local Cygnus Cold Cloud - Testing the Hot Local Bubble

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 within 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.
Investigators:

<table>
<thead>
<tr>
<th>Investigator</th>
<th>Institution</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI&amp; J Lauroesch</td>
<td>University of Louisville Research Foundation, Inc.</td>
<td>USA/KY</td>
</tr>
<tr>
<td>CoI G Lentner</td>
<td>University of Louisville Research Foundation, Inc.</td>
<td>USA/KY</td>
</tr>
</tbody>
</table>

Number of investigators: 2
& Phase I contacts: 1

Target Summary:

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

Observing Summary:

<table>
<thead>
<tr>
<th>Target</th>
<th>Config Mode and Spectral Elements</th>
<th>Flags</th>
<th>Orbits</th>
</tr>
</thead>
<tbody>
<tr>
<td>HR-7444</td>
<td>STIS/FUV-MAMA Spectroscopic E140H (1598)</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>STIS/NUV-MAMA Spectroscopic E230H (2113)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-B03-CYG</td>
<td>STIS/FUV-MAMA Spectroscopic E140H (1598)</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>STIS/NUV-MAMA Spectroscopic E230H (2113)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 boundary 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∼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 could 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 ∼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 & Linlisky 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$^3$K 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$^3$K 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$^3$K) 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$^3$K 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 ~20 mA, or about 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 & Péralt 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
Cravens, T. E. 1997, GeoRL, 24, 105
Frisch, P. C. 1995, SSRv, 72, 499
Figure 2: Archival Na I absorption spectra toward our selected stars behind the δ Cyg Cloud taken with the Elodie spectrograph.

Wargelin, B. J., Beiersdorfer, P., & Brown, G. V. 2008, CaJPh, 86, 151
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 $\delta$Cyg (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 1560 Å multiplet of C I we will using E140H centered at 1489 Å, and we will observe Zn II 2026, 2062 Å using E230H centered at 2113 Å. In addition we will cover other lines including Si IV 1393, 1402 Å, Si II 1526 Å, P II 1532 Å, Ti II 1910 Å, Cr II 2056, 2062, 2066 Å which will provide a measure of depletion and sample the warm/hot interstellar medium. We selected the 1560 Å multiplet of C I due to the large exposure times needed to cover the 1656 Å multiplet, and the lack of FUV flux in these objects preventing the use of multiplets such as 1277 or 1328 Å.

For HD 184875 we used the fluxes from the TD1 satellite, which are $1.7 \times 10^{-11}$ at 1565 Å, $2.4 \times 11$ at 1965 Å, and $1.8 \times 11$ at 2365 Å. 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 1489 Å setting using the 0.2x0.09 slit and 900 seconds at 2113 Å using the 31x0.05NDA.

For HD 192640 low resolution, large aperture IUE SWP spectra show a flux at 1656 Å of $6 \times 10^{-12}$ erg/cm$^2$/s/Å, and a flux of no greater than $3 \times 10^{-11}$ erg/cm$^2$/s/Å anywhere in the 1763 Å observation band. For the longer wavelength setting near 2113 Å IUE low resolution, large aperture LWP observations show a flux of $3 \times 10^{-11}$ erg/cm$^2$/s/Å across the wavelength region of interest. These fluxes result in an exposure time of 5400 seconds at 1598 Å using the 0.2x0.09 slit and at the 2113 Å 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.

  Publications from previous HST programs:


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

<table>
<thead>
<tr>
<th>Target</th>
<th>File</th>
<th>Date</th>
<th>ExpTime</th>
<th>S/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD 332329</td>
<td>20040704 0017</td>
<td>2004-07-05 23:54:13</td>
<td>1.800</td>
<td>48.2</td>
</tr>
<tr>
<td>HD 192640</td>
<td>19990604 0032</td>
<td>1999-06-05 01:50:37</td>
<td>1.384</td>
<td>323.9</td>
</tr>
<tr>
<td>HD 185732</td>
<td>19990707 0016</td>
<td>1999-07-07 23:14:31</td>
<td>3.001</td>
<td>101.3</td>
</tr>
<tr>
<td>HD 184471</td>
<td>19940924 0009</td>
<td>1994-09-24 19:12:07</td>
<td>0.702</td>
<td>23.0</td>
</tr>
<tr>
<td>BD+344101</td>
<td>19940918 0013</td>
<td>1994-09-18 20:28:02</td>
<td>0.901</td>
<td>19.3</td>
</tr>
<tr>
<td>HD 184875</td>
<td>19960728 0014</td>
<td>1996-07-28 22:15:11</td>
<td>1.410</td>
<td>322.6</td>
</tr>
<tr>
<td>HD 184066</td>
<td>20050826 0013</td>
<td>2005-08-26 21:05:55</td>
<td>0.500</td>
<td>298.6</td>
</tr>
<tr>
<td>HD 182989</td>
<td>19960625 0040</td>
<td>1996-06-26 00:50:34</td>
<td>1.685</td>
<td>116.0</td>
</tr>
<tr>
<td>HD 190397</td>
<td>20060624 0017</td>
<td>2006-06-25 00:11:05</td>
<td>1.800</td>
<td>57.5</td>
</tr>
<tr>
<td>HD 197345</td>
<td>20040823 0040</td>
<td>2004-08-23 22:52:39</td>
<td>0.060</td>
<td>336.8</td>
</tr>
<tr>
<td>HD 172167A</td>
<td>19960502 0040</td>
<td>1996-05-03 03:03:07</td>
<td>0.062</td>
<td>483.8</td>
</tr>
<tr>
<td>HD 195692A</td>
<td>20010807 0022</td>
<td>2001-08-07 22:23:57</td>
<td>0.600</td>
<td>146.2</td>
</tr>
<tr>
<td>HD 188854A</td>
<td>20000721 0024</td>
<td>2000-07-21 22:46:41</td>
<td>1.801</td>
<td>145.2</td>
</tr>
<tr>
<td>HD 200723</td>
<td>19990813 0031</td>
<td>1999-08-14 00:54:29</td>
<td>2.812</td>
<td>297.6</td>
</tr>
<tr>
<td>HD 198343</td>
<td>20001005 0017</td>
<td>2000-10-05 20:54:15</td>
<td>3.001</td>
<td>162.1</td>
</tr>
<tr>
<td>HD 173977</td>
<td>20020920 0007</td>
<td>2002-09-20 18:49:18</td>
<td>7.201</td>
<td>193.2</td>
</tr>
<tr>
<td>HD 186155</td>
<td>19990810 0023</td>
<td>1999-08-10 21:44:50</td>
<td>1.306</td>
<td>298.3</td>
</tr>
<tr>
<td>BD+293448</td>
<td>19940924 0008</td>
<td>1994-09-24 18:56:43</td>
<td>0.701</td>
<td>16.6</td>
</tr>
<tr>
<td>HD 194093</td>
<td>19960823 0020</td>
<td>1996-08-23 22:12:43</td>
<td>0.120</td>
<td>513.5</td>
</tr>
<tr>
<td>HD 198226A</td>
<td>19990817 0020</td>
<td>1999-08-18 01:11:59</td>
<td>1.570</td>
<td>92.5</td>
</tr>
<tr>
<td>HD 190537A</td>
<td>20000720 0021</td>
<td>2000-07-21 00:14:39</td>
<td>0.901</td>
<td>147.0</td>
</tr>
<tr>
<td>HD 341233</td>
<td>20000805 0006</td>
<td>2000-08-05 20:15:56</td>
<td>3.601</td>
<td>51.1</td>
</tr>
<tr>
<td>HD 190508</td>
<td>20050805 0015</td>
<td>2005-08-05 23:34:58</td>
<td>1.200</td>
<td>32.7</td>
</tr>
<tr>
<td>HD 200019</td>
<td>20000804 0013</td>
<td>2000-08-04 22:01:55</td>
<td>1.201</td>
<td>113.8</td>
</tr>
<tr>
<td>HD 199939</td>
<td>20020602 0045</td>
<td>2002-06-23 00:53:25</td>
<td>0.481</td>
<td>78.6</td>
</tr>
</tbody>
</table>

α Numbers correspond to annotations in Figure 5.2 and Tables 5.1 - 5.2.

β Elodie file names given by “elodie:yyymmdd/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

<table>
<thead>
<tr>
<th>Target</th>
<th>File</th>
<th>Date</th>
<th>ExpTime</th>
<th>S/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 HD 335070</td>
<td>20010825 0012</td>
<td>2001-08-25 21:57:14</td>
<td>4.501</td>
<td>54.4</td>
</tr>
<tr>
<td>32 HD 193891</td>
<td>20000706 0007</td>
<td>2000-07-06 20:53:03</td>
<td>1.201</td>
<td>105.9</td>
</tr>
<tr>
<td>33 HD 170527</td>
<td>20031103 0009</td>
<td>2003-11-03 18:07:57</td>
<td>2.700</td>
<td>166.4</td>
</tr>
<tr>
<td>34 HD 198149</td>
<td>19950911 0017</td>
<td>1995-09-11 21:23:23</td>
<td>0.300</td>
<td>405.2</td>
</tr>
<tr>
<td>35 BD+483149</td>
<td>20010731 0030</td>
<td>2001-08-01 00:27:20</td>
<td>1.501</td>
<td>84.0</td>
</tr>
<tr>
<td>36 HD 332698</td>
<td>20031004 0011</td>
<td>2003-10-04 00:05:47</td>
<td>5.563</td>
<td>12.8</td>
</tr>
<tr>
<td>37 HD 187372</td>
<td>20000809 0018</td>
<td>2000-08-09 22:12:40</td>
<td>0.601</td>
<td>182.9</td>
</tr>
<tr>
<td>38 HD 186619</td>
<td>20030703 0023</td>
<td>2003-07-03 23:23:47</td>
<td>0.300</td>
<td>76.5</td>
</tr>
<tr>
<td>39 HD 186532</td>
<td>20030703 0021</td>
<td>2003-07-03 23:08:41</td>
<td>0.360</td>
<td>61.1</td>
</tr>
<tr>
<td>40 HD 177808</td>
<td>20030703 0013</td>
<td>2003-07-03 21:39:20</td>
<td>0.360</td>
<td>79.5</td>
</tr>
<tr>
<td>41 HD 172381</td>
<td>20010802 0035</td>
<td>2001-08-03 00:01:44</td>
<td>0.421</td>
<td>52.2</td>
</tr>
<tr>
<td>42 HD 190964</td>
<td>20030703 0029</td>
<td>2003-07-04 00:17:46</td>
<td>0.300</td>
<td>54.3</td>
</tr>
<tr>
<td>43 HD 197939</td>
<td>20030703 0031</td>
<td>2003-07-04 00:45:06</td>
<td>0.300</td>
<td>47.2</td>
</tr>
</tbody>
</table>

\(^{a}\) Numbers correspond to annotations in Figure 5.2 and Tables 5.1 - 5.2.

\(^{b}\) Elodie file names given by “elodie:yyyymmdd/nmnn”.

\(^{c}\) Metadata taken directly from FITS headers.

\(^{d}\) At the beginning of the exposure. Format is yyyy−mm−dd hh:mm:ss

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

α Numbers correspond to annotations in Figure 5.3 and Tables 5.3 - 5.4.
β Elodie file names given by “elodie:yyyymmdd/chhh”.
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

<table>
<thead>
<tr>
<th>Target</th>
<th>File</th>
<th>Date</th>
<th>ExpTime</th>
<th>S/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 HD 191649</td>
<td>19970817 0032</td>
<td>1997-08-17 23:44:52</td>
<td>0.901</td>
<td>120.3</td>
</tr>
<tr>
<td>32 HD 185414</td>
<td>19970818 0015</td>
<td>1997-08-18 22:21:58</td>
<td>0.901</td>
<td>178.1</td>
</tr>
<tr>
<td>33 HD 187237</td>
<td>19970909 0021</td>
<td>1997-09-09 20:53:22</td>
<td>0.721</td>
<td>126.5</td>
</tr>
<tr>
<td>34 HD 186427</td>
<td>19980808 0010</td>
<td>1998-08-08 21:03:54</td>
<td>0.601</td>
<td>156.2</td>
</tr>
<tr>
<td>35 HD 188326A</td>
<td>19980808 0018</td>
<td>1998-08-08 23:24:01</td>
<td>0.901</td>
<td>119.3</td>
</tr>
<tr>
<td>36 HD 187462</td>
<td>19980810 0015</td>
<td>1998-08-10 22:30:06</td>
<td>0.701</td>
<td>119.9</td>
</tr>
<tr>
<td>37 HD 197310</td>
<td>19980810 0020</td>
<td>1998-08-11 00:12:43</td>
<td>0.900</td>
<td>111.5</td>
</tr>
<tr>
<td>38 HD 195988</td>
<td>19980901 0018</td>
<td>1998-09-01 22:25:17</td>
<td>0.901</td>
<td>109.3</td>
</tr>
<tr>
<td>39 HD 187123A</td>
<td>19990916 0036</td>
<td>1999-09-16 20:51:41</td>
<td>0.901</td>
<td>88.4</td>
</tr>
<tr>
<td>40 HD 198483</td>
<td>19990926 0017</td>
<td>1999-09-26 20:21:45</td>
<td>0.300</td>
<td>46.4</td>
</tr>
<tr>
<td>41 HD 198809</td>
<td>20000706 0015</td>
<td>2000-07-07 00:31:12</td>
<td>0.301</td>
<td>218.1</td>
</tr>
<tr>
<td>42 HD 184499</td>
<td>20030710 0032</td>
<td>2003-07-11 23:56:30</td>
<td>0.900</td>
<td>136.1</td>
</tr>
<tr>
<td>43 HD 188650</td>
<td>20031101 0010</td>
<td>2003-11-01 18:12:46</td>
<td>1.500</td>
<td>261.1</td>
</tr>
<tr>
<td>44 HD 176377</td>
<td>20040706 0007</td>
<td>2004-07-06 23:25:16</td>
<td>1.805</td>
<td>186.2</td>
</tr>
<tr>
<td>45 HD 197666</td>
<td>20040726 0029</td>
<td>2004-07-27 00:41:51</td>
<td>1.200</td>
<td>48.3</td>
</tr>
<tr>
<td>47 HD 187748</td>
<td>20040731 0014</td>
<td>2004-07-31 22:42:52</td>
<td>0.600</td>
<td>101.6</td>
</tr>
<tr>
<td>48 HD 187876</td>
<td>20040731 0015</td>
<td>2004-07-31 22:58:00</td>
<td>1.200</td>
<td>87.5</td>
</tr>
<tr>
<td>49 HD 190771</td>
<td>20040901 0012</td>
<td>2004-09-01 20:01:19</td>
<td>1.800</td>
<td>273.5</td>
</tr>
<tr>
<td>50 HD 189749</td>
<td>20040927 0010</td>
<td>2004-09-27 20:53:50</td>
<td>1.200</td>
<td>63.7</td>
</tr>
<tr>
<td>51 HD 170357</td>
<td>20041122 0017</td>
<td>2004-11-22 17:10:24</td>
<td>1.201</td>
<td>47.7</td>
</tr>
<tr>
<td>52 HD 182758</td>
<td>20041123 0012</td>
<td>2004-11-23 17:14:01</td>
<td>1.200</td>
<td>57.5</td>
</tr>
<tr>
<td>53 HD 181047</td>
<td>20050502 0021</td>
<td>2005-05-29 02:09:04</td>
<td>1.100</td>
<td>54.4</td>
</tr>
<tr>
<td>54 HD 196361</td>
<td>20050803 0013</td>
<td>2005-08-03 23:25:06</td>
<td>1.200</td>
<td>28.2</td>
</tr>
<tr>
<td>55 HD 172393</td>
<td>20050805 0010</td>
<td>2005-08-05 21:45:23</td>
<td>1.200</td>
<td>48.8</td>
</tr>
<tr>
<td>56 HD 175441</td>
<td>20050913 0007</td>
<td>2005-09-13 19:23:26</td>
<td>1.500</td>
<td>64.6</td>
</tr>
<tr>
<td>57 HD 185269</td>
<td>20060608 0028</td>
<td>2006-06-09 01:25:25</td>
<td>2.400</td>
<td>139.3</td>
</tr>
<tr>
<td>58 HD 176670</td>
<td>19960902 0009</td>
<td>1996-09-02 20:35:23</td>
<td>0.601</td>
<td>227.7</td>
</tr>
<tr>
<td>59 HD 188947</td>
<td>19970828 0014</td>
<td>1997-08-28 22:52:29</td>
<td>0.091</td>
<td>126.5</td>
</tr>
<tr>
<td>60 HD 199870</td>
<td>20021028 0017</td>
<td>2002-10-28 21:03:10</td>
<td>0.902</td>
<td>220.8</td>
</tr>
</tbody>
</table>

α Numbers correspond to annotations in Figure 5.3 and Tables 5.3 - 5.4.
β Elodie file names given by “elodie:yyyymmdd/nrn”.
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

<table>
<thead>
<tr>
<th>Target</th>
<th>File $^\beta$</th>
<th>Date $^\gamma$</th>
<th>ExpTime $^\gamma$</th>
<th>S/N $^\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BD+293427</td>
<td>19940918 0009</td>
<td>1994-09-18 19:16:24</td>
<td>0.801</td>
<td>18.7</td>
</tr>
<tr>
<td>HD 340577</td>
<td>19940918 0015</td>
<td>1994-09-18 20:52:00</td>
<td>0.901</td>
<td>27.6</td>
</tr>
<tr>
<td>HD 200405</td>
<td>19940918 0023</td>
<td>1994-09-18 23:35:40</td>
<td>0.703</td>
<td>26.6</td>
</tr>
<tr>
<td>HD 191742</td>
<td>19940919 0009</td>
<td>1994-09-19 20:01:26</td>
<td>1.203</td>
<td>26.0</td>
</tr>
<tr>
<td>HD 192678</td>
<td>19940924 0012</td>
<td>1994-09-24 20:01:23</td>
<td>0.603</td>
<td>51.3</td>
</tr>
<tr>
<td>HD 195725</td>
<td>19990809 0021</td>
<td>1999-08-09 22:40:21</td>
<td>1.004</td>
<td>316.4</td>
</tr>
<tr>
<td>HD 177196</td>
<td>19990813 0019</td>
<td>1999-08-13 20:54:11</td>
<td>1.600</td>
<td>303.4</td>
</tr>
<tr>
<td>HD 183534</td>
<td>20060531 0018</td>
<td>2006-06-01 01:10:57</td>
<td>4.501</td>
<td>154.1</td>
</tr>
<tr>
<td>BD+353616A</td>
<td>19940918 0010</td>
<td>1994-09-18 19:36:40</td>
<td>0.602</td>
<td>17.4</td>
</tr>
<tr>
<td>HD 341037</td>
<td>19940924 0015</td>
<td>1994-09-24 20:55:23</td>
<td>0.801</td>
<td>27.7</td>
</tr>
<tr>
<td>HD 197572</td>
<td>19960822 0011</td>
<td>1996-08-22 21:00:20</td>
<td>0.000</td>
<td>324.6</td>
</tr>
<tr>
<td>HD 198726</td>
<td>19960823 0027</td>
<td>1996-08-24 00:01:34</td>
<td>2.001</td>
<td>257.3</td>
</tr>
<tr>
<td>BD+423935</td>
<td>19970614 0020</td>
<td>1997-06-15 00:59:00</td>
<td>0.601</td>
<td>19.7</td>
</tr>
<tr>
<td>HD 238932A</td>
<td>19970617 0017</td>
<td>1997-06-17 23:51:12</td>
<td>0.301</td>
<td>18.9</td>
</tr>
<tr>
<td>BD+423607</td>
<td>19970822 0008</td>
<td>1997-08-22 20:57:07</td>
<td>3.602</td>
<td>59.7</td>
</tr>
<tr>
<td>HD 194951</td>
<td>19990706 0018</td>
<td>1999-07-07 02:29:49</td>
<td>0.900</td>
<td>81.7</td>
</tr>
<tr>
<td>HD 331319</td>
<td>19990709 0016</td>
<td>1999-07-10 01:06:21</td>
<td>5.401</td>
<td>98.1</td>
</tr>
<tr>
<td>HD 195069</td>
<td>20030625 0017</td>
<td>2003-06-26 00:15:07</td>
<td>7.202</td>
<td>560.0</td>
</tr>
<tr>
<td>HD 193370</td>
<td>20030725 0014</td>
<td>2003-07-26 00:30:27</td>
<td>0.600</td>
<td>241.1</td>
</tr>
<tr>
<td>HD 200805</td>
<td>20030729 0010</td>
<td>2003-07-30 00:13:00</td>
<td>4.501</td>
<td>152.7</td>
</tr>
<tr>
<td>HD 178593</td>
<td>20040801 0011</td>
<td>2004-08-01 22:23:03</td>
<td>1.506</td>
<td>103.7</td>
</tr>
<tr>
<td>HD 188307</td>
<td>20040802 0016</td>
<td>2004-08-03 23:55:22</td>
<td>1.200</td>
<td>28.6</td>
</tr>
<tr>
<td>HD 187253</td>
<td>20050722 0034</td>
<td>2005-07-23 00:41:41</td>
<td>1.200</td>
<td>31.8</td>
</tr>
<tr>
<td>HD 185239</td>
<td>20050913 0008</td>
<td>2005-09-13 19:55:05</td>
<td>1.500</td>
<td>75.3</td>
</tr>
<tr>
<td>HD 175225</td>
<td>19960503 0033</td>
<td>1996-05-04 02:09:59</td>
<td>0.601</td>
<td>216.8</td>
</tr>
<tr>
<td>HD 173701</td>
<td>19961023 0025</td>
<td>1996-10-23 18:00:10</td>
<td>0.901</td>
<td>133.0</td>
</tr>
<tr>
<td>HD 177153</td>
<td>19970820 0011</td>
<td>1997-08-20 20:55:14</td>
<td>0.901</td>
<td>132.8</td>
</tr>
<tr>
<td>HD 185351</td>
<td>19970823 0006</td>
<td>1997-08-23 20:29:39</td>
<td>0.630</td>
<td>267.0</td>
</tr>
<tr>
<td>BD+413931</td>
<td>19970823 0012</td>
<td>1997-08-23 22:30:41</td>
<td>3.601</td>
<td>68.2</td>
</tr>
<tr>
<td>HD 175306</td>
<td>19970827 0016</td>
<td>1997-08-27 20:02:16</td>
<td>0.601</td>
<td>24.4</td>
</tr>
</tbody>
</table>

$^\beta$ Elodie file names given by “elodie:yyyyymdd/nmmn”.

$^\gamma$ Metadata taken directly from FITS headers.

$^\delta$ At the beginning of the exposure. Format is yyyy−mm−dd hh:mm:ss

$^\epsilon$ As reported by the archive.

94
TABLE B.6
Observational Metadata for Non-OB Inconclusive Targets – B

<table>
<thead>
<tr>
<th>Target</th>
<th>File $^\beta$</th>
<th>Date $^\gamma$</th>
<th>ExpTime $^\gamma$</th>
<th>S/N $^\gamma$</th>
<th>$10^4 s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD 182488</td>
<td>199809010009</td>
<td>1998-09-01 20:02:18</td>
<td>0.901</td>
<td>163.6</td>
<td></td>
</tr>
<tr>
<td>HD 175900</td>
<td>199908140014</td>
<td>1999-08-14 22:37:57</td>
<td>0.904</td>
<td>91.8</td>
<td></td>
</tr>
<tr>
<td>HD 193795</td>
<td>199909260015</td>
<td>1999-09-26 20:05:14</td>
<td>0.300</td>
<td>33.7</td>
<td></td>
</tr>
<tr>
<td>HD 200391</td>
<td>200007070007</td>
<td>2000-07-07 21:35:17</td>
<td>1.201</td>
<td>95.2</td>
<td></td>
</tr>
<tr>
<td>HD 181655</td>
<td>200007310015</td>
<td>2000-07-31 23:06:50</td>
<td>0.751</td>
<td>183.9</td>
<td></td>
</tr>
<tr>
<td>BD+364301</td>
<td>200008040016</td>
<td>2000-08-04 23:21:21</td>
<td>3.601</td>
<td>74.8</td>
<td></td>
</tr>
<tr>
<td>HD 179484A</td>
<td>200008130020</td>
<td>2000-08-13 22:27:28</td>
<td>1.801</td>
<td>73.8</td>
<td></td>
</tr>
<tr>
<td>HD 190360A</td>
<td>200008240023</td>
<td>2000-08-24 23:03:22</td>
<td>0.721</td>
<td>206.8</td>
<td></td>
</tr>
<tr>
<td>HD 178911A</td>
<td>200104020032</td>
<td>2001-04-02 02:59:15</td>
<td>1.201</td>
<td>107.3</td>
<td></td>
</tr>
<tr>
<td>HD 199598</td>
<td>200107210021</td>
<td>2001-07-21 00:46:46</td>
<td>0.901</td>
<td>151.9</td>
<td></td>
</tr>
<tr>
<td>HD 173605</td>
<td>200107280005</td>
<td>2001-07-28 20:09:56</td>
<td>1.501</td>
<td>127.5</td>
<td></td>
</tr>
<tr>
<td>HD 190228</td>
<td>200108110015</td>
<td>2001-08-11 22:12:07</td>
<td>0.901</td>
<td>125.9</td>
<td></td>
</tr>
<tr>
<td>HD 188326</td>
<td>200108120017</td>
<td>2001-08-12 22:25:24</td>
<td>1.501</td>
<td>112.7</td>
<td></td>
</tr>
<tr>
<td>HD 187123</td>
<td>200108140018</td>
<td>2001-08-14 22:52:25</td>
<td>1.001</td>
<td>96.4</td>
<td></td>
</tr>
<tr>
<td>HD 182736</td>
<td>200110100022</td>
<td>2001-10-10 21:42:19</td>
<td>0.901</td>
<td>116.6</td>
<td></td>
</tr>
<tr>
<td>HD 171242</td>
<td>200110110012</td>
<td>2001-10-11 19:32:37</td>
<td>0.901</td>
<td>75.5</td>
<td></td>
</tr>
<tr>
<td>HD 185657</td>
<td>200210280015</td>
<td>2002-10-28 20:24:06</td>
<td>1.343</td>
<td>200.9</td>
<td></td>
</tr>
<tr>
<td>HD 171007</td>
<td>200305210022</td>
<td>2003-05-21 00:49:22</td>
<td>1.801</td>
<td>42.3</td>
<td></td>
</tr>
<tr>
<td>HD 193216A</td>
<td>200307150013</td>
<td>2003-07-15 23:21:17</td>
<td>0.900</td>
<td>50.1</td>
<td></td>
</tr>
<tr>
<td>HD 178911B</td>
<td>200307170020</td>
<td>2003-07-17 00:47:04</td>
<td>0.900</td>
<td>117.0</td>
<td></td>
</tr>
<tr>
<td>HD 200102</td>
<td>200307250019</td>
<td>2003-07-25 02:02:02</td>
<td>3.001</td>
<td>258.1</td>
<td></td>
</tr>
<tr>
<td>HD 190403</td>
<td>200307270011</td>
<td>2003-07-27 23:26:14</td>
<td>3.601</td>
<td>156.8</td>
<td></td>
</tr>
<tr>
<td>HD 174104</td>
<td>200307290007</td>
<td>2003-07-29 20:08:57</td>
<td>5.001</td>
<td>122.4</td>
<td></td>
</tr>
<tr>
<td>HD 180161</td>
<td>200407100010</td>
<td>2004-07-11 00:43:52</td>
<td>2.401</td>
<td>129.8</td>
<td></td>
</tr>
<tr>
<td>HD 180683</td>
<td>200407260024</td>
<td>2004-07-26 22:57:15</td>
<td>1.200</td>
<td>50.4</td>
<td></td>
</tr>
<tr>
<td>HD 197037</td>
<td>200407260027</td>
<td>2004-07-26 23:53:04</td>
<td>1.200</td>
<td>75.0</td>
<td></td>
</tr>
<tr>
<td>HD 180502</td>
<td>200407280012</td>
<td>2004-07-28 22:20:08</td>
<td>1.200</td>
<td>80.1</td>
<td></td>
</tr>
<tr>
<td>HD 197488</td>
<td>200407280016</td>
<td>2004-07-29 23:58:53</td>
<td>1.200</td>
<td>79.0</td>
<td></td>
</tr>
<tr>
<td>HD 175425</td>
<td>200409260012</td>
<td>2004-09-26 19:17:56</td>
<td>1.200</td>
<td>57.1</td>
<td></td>
</tr>
</tbody>
</table>

$^\beta$ Elodie file names given by “elodie:yyyyymmdd/nnnn”.

$^\gamma$ Metadata taken directly from FITS headers.

$^\delta$ At the beginning of the exposure. Format is yyyy−mm−dd hh:mm:ss

$^\epsilon$ As reported by the archive.
### TABLE B.7
Observational Metadata for Non-OB Inconclusive Targets – C

<table>
<thead>
<tr>
<th>Target</th>
<th>File $^\beta$</th>
<th>Date $^\gamma$</th>
<th>ExpTime $^\gamma$</th>
<th>S/N $^\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD 193215</td>
<td>20041001</td>
<td>2004-10-01 19:11:51</td>
<td>1.200</td>
<td>57.1</td>
</tr>
<tr>
<td>HD 172557</td>
<td>20041002</td>
<td>2004-10-02 18:44:12</td>
<td>1.200</td>
<td>60.3</td>
</tr>
<tr>
<td>HD 184601</td>
<td>20041002</td>
<td>2004-10-02 19:30:53</td>
<td>1.200</td>
<td>69.1</td>
</tr>
<tr>
<td>HD 199100</td>
<td>20041002</td>
<td>2004-10-02 20:16:51</td>
<td>1.200</td>
<td>91.8</td>
</tr>
<tr>
<td>HD 195987</td>
<td>20041009</td>
<td>2004-10-09 17:53:28</td>
<td>1.802</td>
<td>126.3</td>
</tr>
<tr>
<td>HD 187548</td>
<td>20050529</td>
<td>2005-05-30 00:28:37</td>
<td>1.500</td>
<td>43.9</td>
</tr>
<tr>
<td>HD 177780</td>
<td>20050530</td>
<td>2005-05-31 01:51:44</td>
<td>1.500</td>
<td>22.8</td>
</tr>
<tr>
<td>HD 190605</td>
<td>20050616</td>
<td>2005-06-17 02:09:52</td>
<td>1.200</td>
<td>82.6</td>
</tr>
<tr>
<td>HD 178911</td>
<td>20050711</td>
<td>2005-07-12 00:25:49</td>
<td>1.017</td>
<td>110.0</td>
</tr>
<tr>
<td>HD 178450</td>
<td>20050714</td>
<td>2005-07-14 22:56:23</td>
<td>2.108</td>
<td>88.6</td>
</tr>
<tr>
<td>HD 173024</td>
<td>20050723</td>
<td>2005-07-23 21:41:57</td>
<td>1.200</td>
<td>35.2</td>
</tr>
<tr>
<td>HD 173289</td>
<td>20050720</td>
<td>2005-07-26 00:05:31</td>
<td>1.200</td>
<td>38.3</td>
</tr>
<tr>
<td>HD 197140</td>
<td>20050805</td>
<td>2005-08-06 00:02:23</td>
<td>1.200</td>
<td>44.3</td>
</tr>
<tr>
<td>HD 197207</td>
<td>20050912</td>
<td>2005-09-12 21:36:10</td>
<td>1.500</td>
<td>58.4</td>
</tr>
<tr>
<td>HD 189087</td>
<td>19961023</td>
<td>1996-10-23 18:19:44</td>
<td>0.801</td>
<td>103.7</td>
</tr>
<tr>
<td>HD 234677</td>
<td>19970521</td>
<td>1997-05-22 00:54:31</td>
<td>3.601</td>
<td>134.7</td>
</tr>
<tr>
<td>HD 197989</td>
<td>19970823</td>
<td>1997-08-23 21:16:55</td>
<td>0.021</td>
<td>204.7</td>
</tr>
<tr>
<td>HD 183255</td>
<td>19970924</td>
<td>1997-09-24 19:45:41</td>
<td>5.401</td>
<td>215.5</td>
</tr>
<tr>
<td>HD 191026</td>
<td>19971108</td>
<td>1997-11-08 18:31:03</td>
<td>1.200</td>
<td>226.7</td>
</tr>
<tr>
<td>HD 338867</td>
<td>19980318</td>
<td>1998-03-19 03:24:27</td>
<td>0.602</td>
<td>43.7</td>
</tr>
<tr>
<td>HD 199956</td>
<td>20000708</td>
<td>2000-07-08 22:08:06</td>
<td>2.401</td>
<td>140.0</td>
</tr>
<tr>
<td>HD 199546</td>
<td>20010730</td>
<td>2001-07-30 20:39:07</td>
<td>1.201</td>
<td>91.2</td>
</tr>
<tr>
<td>HD 189806</td>
<td>20010731</td>
<td>2001-07-31 20:28:22</td>
<td>1.201</td>
<td>76.8</td>
</tr>
<tr>
<td>HD 181209</td>
<td>20010731</td>
<td>2001-07-31 20:55:43</td>
<td>1.202</td>
<td>68.2</td>
</tr>
<tr>
<td>HD 188056</td>
<td>20010809</td>
<td>2001-08-09 22:28:08</td>
<td>0.603</td>
<td>95.3</td>
</tr>
<tr>
<td>BD+333930</td>
<td>20010825</td>
<td>2001-08-25 21:11:28</td>
<td>2.401</td>
<td>124.5</td>
</tr>
<tr>
<td>BD+334140</td>
<td>20010826</td>
<td>2001-08-26 21:53:42</td>
<td>3.601</td>
<td>58.8</td>
</tr>
<tr>
<td>HD 192787</td>
<td>20021022</td>
<td>2002-10-22 18:07:01</td>
<td>1.641</td>
<td>261.8</td>
</tr>
<tr>
<td>HD 196134</td>
<td>20021027</td>
<td>2002-10-27 20:49:15</td>
<td>2.401</td>
<td>209.3</td>
</tr>
</tbody>
</table>

$^\beta$ Elodie file names given by “elodie:yyyyMMdd/mmmn”.

$^\gamma$ Metadata taken directly from FITS headers.

$^\delta$ At the beginning of the exposure. Format is yyyy – mm – dd hh : mm : ss

$^\epsilon$ As reported by the archive.
TABLE B.8
Observational Metadata for Non-OB Inconclusive Targets – D

<table>
<thead>
<tr>
<th>Target</th>
<th>File $^\beta$</th>
<th>Date $^\gamma\delta$</th>
<th>ExpTime $^\gamma$</th>
<th>S/N $^\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD 176408</td>
<td>20021028 0013</td>
<td>2002-10-28 19:57:56</td>
<td>0.900 274.0</td>
<td></td>
</tr>
<tr>
<td>HD 198550</td>
<td>20030711 0031</td>
<td>2003-07-12 00:47:33</td>
<td>0.900 59.4</td>
<td></td>
</tr>
<tr>
<td>HD 192910</td>
<td>20030725 0013</td>
<td>2003-07-26 00:14:39</td>
<td>0.360 152.3</td>
<td></td>
</tr>
<tr>
<td>HD 193469</td>
<td>20030726 0017</td>
<td>2003-07-27 02:15:11</td>
<td>2.101 188.3</td>
<td></td>
</tr>
<tr>
<td>HD 198794</td>
<td>20030727 0012</td>
<td>2003-07-28 00:32:39</td>
<td>3.600 114.6</td>
<td></td>
</tr>
<tr>
<td>HD 190470</td>
<td>20040712 0012</td>
<td>2004-07-13 01:33:42</td>
<td>3.300 70.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$^\beta$ Elodie file names given by “elodie:yyyyymmdd/nmm”.

$^\gamma$ Metadata taken directly from FITS headers.

$^\delta$ At the beginning of the exposure. Format is yyyy-mm-dd hh:mm:ss

$^\epsilon$ As reported by the archive.
<table>
<thead>
<tr>
<th>Target</th>
<th>File</th>
<th>Date</th>
<th>ExpTime</th>
<th>S/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>GJ 0815</td>
<td>20010902 0018</td>
<td>2001-09-02 21:36:39</td>
<td>1.201</td>
<td>29.7</td>
</tr>
<tr>
<td>HD 189063</td>
<td>20020618 0027</td>
<td>2002-06-19 01:53:06</td>
<td>0.482</td>
<td>79.8</td>
</tr>
<tr>
<td>HD 178003</td>
<td>20030703 0014</td>
<td>2003-07-03 21:49:54</td>
<td>0.360</td>
<td>49.3</td>
</tr>
<tr>
<td>HD 180450</td>
<td>20030703 0017</td>
<td>2003-07-03 22:23:43</td>
<td>0.360</td>
<td>45.2</td>
</tr>
<tr>
<td>HD 179869</td>
<td>20030703 0018</td>
<td>2003-07-03 22:34:21</td>
<td>0.401</td>
<td>52.3</td>
</tr>
<tr>
<td>HD 187849</td>
<td>20030703 0024</td>
<td>2003-07-03 23:32:47</td>
<td>0.300</td>
<td>98.5</td>
</tr>
<tr>
<td>HD 190544</td>
<td>20030703 0026</td>
<td>2003-07-03 23:52:15</td>
<td>0.300</td>
<td>88.8</td>
</tr>
<tr>
<td>HD 173740</td>
<td>20040822 0011</td>
<td>2004-08-22 21:34:21</td>
<td>7.201</td>
<td>104.9</td>
</tr>
<tr>
<td>HD 331161</td>
<td>19940925 0014</td>
<td>1994-09-25 20:14:54</td>
<td>3.601</td>
<td>17.6</td>
</tr>
<tr>
<td>HD 190163</td>
<td>20000809 0020</td>
<td>2000-08-09 22:34:45</td>
<td>0.900</td>
<td>67.4</td>
</tr>
<tr>
<td>HD 182917</td>
<td>20030703 0019</td>
<td>2003-07-03 22:44:36</td>
<td>0.500</td>
<td>50.6</td>
</tr>
</tbody>
</table>

β Elodie file names given by “elodie:yyyyymdd/nmm”.
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>
<thead>
<tr>
<th>Target</th>
<th>File $^\beta$</th>
<th>Date $^\gamma$</th>
<th>ExpTime $^\gamma$</th>
<th>S/N $^\gamma$</th>
<th>Detla $^\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD 188209</td>
<td>19970917 0012</td>
<td>1997-09-17 21:36:34</td>
<td>1.803</td>
<td>417.5</td>
<td>1997-09-17 21:36:34</td>
</tr>
<tr>
<td>HD 194335</td>
<td>19970921 0027</td>
<td>1997-09-21 23:04:10</td>
<td>1.201</td>
<td>243.6</td>
<td>1997-09-21 23:04:10</td>
</tr>
<tr>
<td>HD 200120</td>
<td>20030808 0014</td>
<td>2003-08-08 22:09:33</td>
<td>1.200</td>
<td>376.4</td>
<td>2003-08-08 22:09:33</td>
</tr>
<tr>
<td>HD 182255</td>
<td>20031106 0018</td>
<td>2003-11-06 17:37:00</td>
<td>3.601</td>
<td>512.2</td>
<td>2003-11-06 17:37:00</td>
</tr>
<tr>
<td>HD 174638</td>
<td>20010906 0025</td>
<td>2001-09-06 21:22:35</td>
<td>0.241</td>
<td>249.7</td>
<td>2001-09-06 21:22:35</td>
</tr>
<tr>
<td>HD 176437</td>
<td>19960425 0031</td>
<td>1996-04-26 02:27:27</td>
<td>0.301</td>
<td>64.5</td>
<td>1996-04-26 02:27:27</td>
</tr>
<tr>
<td>HD 198183</td>
<td>20040823 0041</td>
<td>2004-08-23 23:05:39</td>
<td>0.600</td>
<td>85.9</td>
<td>2004-08-23 23:05:39</td>
</tr>
<tr>
<td>HD 190864</td>
<td>20010812 0018</td>
<td>2001-08-12 22:56:35</td>
<td>2.402</td>
<td>144.2</td>
<td>2001-08-12 22:56:35</td>
</tr>
<tr>
<td>HD 193514</td>
<td>20040828 0008</td>
<td>2004-08-28 00:04:36</td>
<td>2.401</td>
<td>129.4</td>
<td>2004-08-28 00:04:36</td>
</tr>
</tbody>
</table>

$^\beta$ Elodie file names given by “elodie:yyyyymmdd/nmnm”.

$^\gamma$ Metadata taken directly from FITS headers.

$^\delta$ At the beginning of the exposure. Format is yyyy-mm-dd hh:mm:ss

$^\epsilon$ As reported by the archive.
| Target | File $^\beta$ | Date $^\delta$ | ExpTime $^\gamma$ | S/N $^\epsilon$
|-------|--------------|---------------|-----------------|-------
| 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

$^\beta$ Elodie file names given by “elodie:yyyyMMdd/mnnn”.

$^\gamma$ Metadata taken directly from FITS headers.

$^\delta$ At the beginning of the exposure. Format is yyyy−mm−dd hh:mm:ss

$^\epsilon$ 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__

```python
# 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, \n    Observatory, Montage, Plot, Spectrum, Measure, Profile

# elevate 'Spectrum' to the package level
from .Slipy.Spectrum import Spectrum
```
C.2 Algorithms, Functions

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

# No FunctionsError implemented yet

def Gaussian(x, *params):
    A, mu, sigma = params
    return A * np.exp(-0.5 * (x - mu)**2 / sigma**2)

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

def InvertedGaussian(x, *params):
    return 1 - Gaussian(x, *params)

def Lorentzian(x, *params):
    x0, gamma = params
    return 1 / ((2 * (x - x0) / gamma)**2 + 1)

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

def NormalizedVoigt(x, *params):
    x0, sigma, gamma = params
    return w((x-x0)**2 + 1)*gamma) / (sigma * np.sqrt(np.pi)) .real / (sigma * np.sqrt(2 * np.pi))

def Voigt(x, *params):
    A, x0, sigma, gamma = params
    A -> amplitude of the profile
    x0 -> center of the profile
    sigma -> the Gaussian width
    gamma -> the Lorentzian width
```

Reference:
return params[0] * NormalizedVoigt(x, *params[1:]) / NormalizedVoigt(0, 0, *params[2:])

def InvertedLorentzian(x, *params):
    """An Inverted Lorentzian (i.e., \( A \cdot \text{Lorentzian}(x) \)).""
    A, x0, gamma = params
    return A / Lorentzian(x, x0, gamma)
C.3 Algorithms: KernelFit

```python
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3) # slipy/Algorithms/KernelFit.py

Non-parametric Kernel Regression.

```
if hasattr(x, 'unit') and not hasattr(self.x, 'unit'):
    raise KernelFitError('The provided array has units but the ' 'original domain does not!')
if hasattr(self.x, 'unit') and not hasattr(x, 'unit'):
    raise KernelFitError('The provided array does not have units ' 'but the original domain did!')
# copy 'x' such that the return type is the same as the input type
y = x.copy()

# fix units though
if hasattr(x, 'unit'):
    y = y.value * self.y.unit
for a, point in enumerate(x):
    weights = Gaussian(point, 1, self.x, self.bandwidth)
    # weights should be dimensionless
    if hasattr(x, 'unit'):
        weights = weights.decompose()
    args = np.where(np.isfinite(self.y))
    y[a] = np.sum(weights[args] * self.y[args]) / np.sum(weights[args])
return y
import numpy as np
from astropy import units as u
from slipy.error import slipyerror
from framework.options import optionserror
from archives.atomicdata import atomicerror
from archives.atomicdata import AtomicData, MortonTable

class atomicerror(slipyerror):
    pass

class ionmanager:
    def __init__(self):
        self.data = AtomicData.MortonTable
        # march through the table and apply units, compute oscillator strengths
        for a, entry in enumerate(self.data):
            Ion = entry[2]
            # conversion from 'cm-' to 'Angstrom' (Mohr & Taylor 2008)
            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 key is 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')
except OptionsError as err:
    print(f'--- OptionsError: {err}
raise AtomicError("Failed keyword assignment from __call__ to IonManager!")

if isinstance(key, tuple):
    if len(key) != 2:
        raise AtomicError("tuples expected to be length 2 on __call__ to 'IonManager!")
    table = self.Between(*key)
else:
    table = self.__getitem__(key)

# map entries to index value
lookup_options = { 'air': 0, 'vacuum': 1, 'ion': 2, 'elox': 3, 'logwf': 4, 'fvalue': 5 }

if lookup not in lookup_options:
    raise AtomicError("{!r} is not an available search option!".format(lookup))

if wavelength not in [ 'air', 'vacuum' ]:
    raise AtomicError("Only 'air' and 'vacuum' wavelengths are understood!")

if isinstance(key, str):
    # alter the index dictionary for column changes
    lookup_options = { 'air': 0, 'vacuum': 1, 'elox': 2, 'logwf': 3, 'fvalue': 4 }

    if lookup == 'ion':
        # 'ion' column won't be present in the returned 'table' !!!
        raise AtomicError("You provided the name of an ion but requested the names 'of the ions as the return value!")

    if not isinstance(key, tuple) and not isinstance(key, str):
        return tuple([ table[ lookup_options[wavelength] ], table[ lookup_options[lookup] ] ])

    return [ # return pairs of wavelength and 'lookup' for each 'line' found
        tuple([ line[ lookup_options[wavelength] ], line[ lookup_options[lookup] ] ])
        for line in table
    ]

def __getitem__(self, index):
    # access methods.
    if the 'index' is a string value giving the name of a particular ion, this method
    returns the entries in the data for that ion.
    if the 'index' is a numerical value, this method returns the table entry closest
    in wavelength (vacuum).
    if the 'index' is a slice object (e.g., '500-550') it returns the segment of the
    table data on that range. A 'step' is not acceptable (e.g., ':0.5').
    if isinstance(index, slice):
        start, stop, step = index.start, index.stop, index.step
        if step: raise AtomicError("You cannot slice the table with a 'step' size!")
        if not start:
            # the first vacuum wavelength in the table
            start = self.data[0][1]
        if not stop:
            # the first vacuum wavelength in the table
            stop = self.data[-1][1]
        return self.Between(start, stop)
    elif isinstance(index, str):
        if index not in self.ions:
            raise AtomicError("'{!r}' is not a recognized or available ion in the 'data'!").format(index))
        return self.ions[index]
    else:
        hasattr(index, 'unit'):
            index = index.to(u.Angstrom).value
        proximity = (np.array([entry[1].value for entry in self.data]) - index)**2
        return self.data[ proximity.argmin() ]

def below(self, wavelength):
    # Return all table entries below the given wavelength. If units are not given,
Angstroms are implied.

if not hasattr(wavelength, 'unit'):
    wavelength *= u.Angstrom

if wavelength > self.data[-1][1] or wavelength < self.data[0][1]:
    raise AtomicError("Cannot access wavelengths outside the data available!")

return [ entry for entry in self.data if entry[1] < wavelength ]

def Above(self, wavelength):
    """
    Return all table entries below the given wavelength. If units are not given, 
    Angstroms are implied.
    """

    if not hasattr(wavelength, 'unit'):
        wavelength *= u.Angstrom

    if wavelength > self.data[-1][1] or wavelength < self.data[0][1]:
        raise AtomicError("Cannot access wavelengths outside the data available!")

    return [ entry for entry in self.data if entry[1] > wavelength ]

def Between(self, wavelengthA, wavelengthB):
    """
    Return all table entries between the given wavelengths. If units are not given, 
    Angstroms are implied.
    """

    if not hasattr(wavelengthA, 'unit'):
        wavelengthA *= u.Angstrom
    if not hasattr(wavelengthB, 'unit'):
        wavelengthB *= u.Angstrom

    if wavelengthA > self.data[-1][1] or wavelengthA < self.data[0][1]:
        raise AtomicError("Cannot access wavelengths outside the data available!")
    if wavelengthB > self.data[-1][1] or wavelengthB < self.data[0][1]:
        raise AtomicError("Cannot access wavelengths outside the data available!")

    if wavelengthA > wavelengthB:
        # that doesn't make sense, switch the order
        wavelengthA, wavelengthB = wavelengthB, wavelengthA

    return [ entry for entry in self.data if entry[1] > wavelengthA ]

# create a static instance of the IonManager class.
IonSearch = IonManager()

class Ion():
    """
    An object for declaring atomic ions.
    An 'Ion' should have a name, wavelength (air or vacuum), oscillator strength,
    and a transition probability (Einstein coefficient).
    """

    Much of this data is available from the .Data.Archives.AtomicData module searchable 
    with the .Data.Archives.IonManager class. In the future, we hope to implement an 
    internal database of transition probabilities for each ion in the AtomicData module 
    from the Morton 2003 publication. Currently, the user must provide this data.
    The NIST Atomic Spectra Database Lines Data is a good source for atomic data values: 
    http://physics.nist.gov/PhysRefData/ASD/lines_form.html

    def __init__(self, name=None, wavelength=None, fvalue=None, A=None, **kwargs):
        """
        Create a new 'Ion'. If no arguments are given the state remains uninitialized. 
        The 'name' (e.g., 'Ca III') is used to connect with the data in the 
        .Data.Archives.AtomicData module via the IonManager class. The 'wavelength' need 
        not necessarily be the exact value of the line; the line closest to that given 
        for the 'name'd ion is used. This is by default the wavelength in vacuum, but can 
        be the wavelength in air if the keyword argument 'medium='air'" is given. 
        The created member attribute 'wavelength' and 'fvalue' are automatically assigned 
        based on this procedure, but can be explicitly assigned if the "fvalue" is 
        provided directly. The transition probability (Einstein coefficient) 'A' simply 
        attached as 'A'. If not units are given for 'A', 's-1' is assigned. 
        """

        try:
            options = Options( **kwargs, {
                'medium' : 'vacuum' # alternative is 'air'
            })

            self.medium = options('medium')

            if self.medium not in ['vacuum', 'air']:
                raise AtomicError("From Ion.__init__(), the only allowed values for 'medium' 
                                   are 'vacuum' and 'air'!")

        except OptionsError as err:
print('  ' + OptionsError(', err)
raise AtomicError('From Ion.__init__(), that was not an acceptable keyword assignment '
'for an 'Ion'!)"

if not name or not wavelength or fvalue:
  # simply assign whatever is provided.
  self.name = name
  self.wavelength = wavelength
  self.fvalue = fvalue
  self.A = A
else:
  if not hasattr(wavelength, 'unit'):
    wavelength = u.Angstrom
  found_ions = IonSearch(name, wavelength=self.medium)
  wavelengths = np.array([(wavelength - entry[0]).value for entry in found_ions])**2
  closest_ion = wavelengths.argmin()
  # assign the closest wavelength to that available
  self.wavelength = found_ions[closest_ion][0].to(wavelength.unit)
  self.fvalue = found_ions[closest_ion][1] * u.dimensionless_unscaled
  self.name = name
  if A and not hasattr(A, 'unit'):
    A /= u.s
  self.A = A

def __str__(self):
  """Show the available information for this Ion.
  ""
  return ('Ion: {}\n  ' + 'Name: {}\n  ' + 'Wavelength: {}\n  ' + 'fvalue: {}\n  ' + 'A: {}').format(self.name, self.wavelength, self.fvalue, self.A))

def __repr__(self):
  """The same as the 'str' representation.
  ""
  return str(self)
import os, shutil, numpy as np
from urllib.request import urlopen

class ElodieError(SlipyError):
    pass

class Archive:
    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,
                {'infilename': default_infile,  # path to input file
                 'catalogs': ['H0', 'H1', 'H2', 'G1', 'G2'],  # catalogues to keep
                 })

            # parameter assignments
            infile = options['infilename']
            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.strip() 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[2:2])

        # 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():
                if options['dataset'] not in
                    idx = np.array([np.array([options[1],-1], dtype=np.int_).argmax()]
                files[target] = options[idx][0]

        # members
        self.data = targets
        self.files = files
        self.names = [
            join('-', join(x.split('/')[1]), split('/')[-1]) + '.fits'
            for x in files.values()]

        def Script(filename, pipeline=''):  # Construct url script for Elodie archive given 'filename' and optionally 'pipeline' instructions (e.g., 'zwr/fca[1,nor]').
```
import os
from elodie import (G

Y return G.join([Ghttp://atlas.obs-hp.fr/elodie/Ecgi?zc=1&ac=", filename, pipeline, &amp;mime:application/x-fits])

G

Y def download( *files, **kwargs ):

Y try:

Y # function parameter defaults

Y options = Options( kwargs, (          
Y     'verbose': True, # display messages, progress
Y     'resample': (-1, -1, -1), # handled by Elodie
Y     'normalize': True; # continue normalization
Y     'outpath': '.', # directory for downloaded files
Y     'names': [], # alternative output names for 'files'
Y )

Y # function parameter assignments

Y verbose = options('verbose')
Y resample = options('resample')
Y normalize = options('normalize')
Y outpath = options('outpath')
Y names = options('names')

Y # check for 'resampled' assignment

Y if 'resample' not in kwargs:

Y     resample = None

Y elif len(resample) == 3:

Y     raise ElodieError('Download() expects 'resample' to be of length 3.')

Y # set default names

Y if not names:

Y     names = [G.join([fname.split('.')[1]]) for fname in files]
Y     names = [G.join([fname.split('.')[2]]) for fname in names]
Y     names = [fname + '.fits' for fname in names]

Y elif len(names) != len(files):

Y     raise ElodieError('Download() expects 'names' option to be of length equal to that of the number of 'files' arguments.')

Y except OptionsError as err:

Y     print( --> OptionsError: ', err)

Y     raise ElodieError('Failed keyword assignment from Download().')

Y if not resample and not normalize:

Y     pipeline = ''

Y else:

Y     pipeline = R2+wrs'

Y if normalize:

Y     pipeline += [Gfca[nor]']

Y if resample:

Y     resample = [str(x) for x in resample]
Y     pipeline += [Gwrs[1,'] + ',join(resample) + ']'

Y if verbose:

Y     display = Monitor(ETC=True)

Y files = len(files)

Y print(\n
Y for a, spectra in enumerate(files):

Y     # show progress

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

Y     # download file

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

Y         shutil.copyfileobj(response, outfile)

Y     if verbose:

Y         display.complete()

Y     display.elapsed()
```

112
C.6 Framework Argument

```python
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)
# slipy/framework/argument.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(kwargs[arg]) not in options[arg]:
                raise ArgumentError('Option {} expects {}, given {}.
                        format(arg, type(options[arg]), type(kwargs[arg])))
        # accept reassignment
        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 at {}, 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 {} to {} for argument 
                        format(self.t, self.name))
                    self.value = True if value == 'True' else False
                elif type(value) is int:
                    raise ArgumentError('Invalid conversion from {} to {} for argument 
                    format(self.t, self.name))
                else:
                    self.value = True if value else False
                elif type(value) is not bool:
                    raise ArgumentError('Invalid conversion from {} to 
                    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))
```

113
C.7 Framework: Command

```python
# 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 '0func' pattern
            })
        if options('exe'):
            function = clargs[0].split(':\')
            if len(function) != 2 or function[0] == function[1]:
                raise CommandError('Incorrect formatting in function call.\n                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\nOptionsError', err)
    raise CommandError('from Parse')
except ValueError as key:
    raise CommandError('Incorrect formatting of keyword arguments.'
```
C.8 .. Framework . Display

```python
# 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 . 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 numerical 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(err)
            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 += '(%0.2f) % %.format( self.percent + 100 )
        if self.ETC:
            display += self.__estimatedcompletiontime()
        sys.stdout.write('
\$\%3H % ()
'.format(display))
        sys.stdout.flush()

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

115
if time() - self.last > self.freq:
    # refresh rate surpassed.
    # update time of last call and percent complete
    self.last = time()
    self.percent = float(i) / float(imax)
    # display progress bar
    self.__build()

def complete(self):
    """
    Call to finalize the progress bar.
    """
    if self.inline:
        sys.stdout.write('\r\033[K\r')
        sys.stdout.flush()
    else:
        self.percent = 1
        self.numbers = False
        self.etc = False
        self.__build()
        sys.stdout.write(' complete\n')
        sys.stdout.flush()

def elapsed(self):
    """
    Display total time elapsed since instantiation.
    """
    total = time() - self.start
    abrv = ['d', 'h', 'm', 's']
    unit = [60*60, 60, 1, 1]
    count = ['d':0, 'h':0, 'm':0, 's':0]
    for item in abrv:
        while total > unit[item]:
            total -= unit[item]
            count[item] += 1
    else: count[item] = math.floor(total)
    total = [item.format(w, u)
             for u, v in zip(abrv, [count[v] for v in abrv])
             if count[u]]
    total = ' Time Elapsed: ' + ':'.join(total) + ' 
' + total
    sys.stdout.write(total)
    sys.stdout.flush()
C.9 Framework Measurement

```python
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)

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):
    srepr = '
'.join([label + ' = ' + str(attr) for attr, label in zip(['Measurement',
        'self.value = self.unit', 'self.error', 'self.name', 'self.notes'],
        [value, unit, 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 '
'.join([n for n in zip(attr, name) if attr]).format(*show)
```
C.10 Framework Options

```python
# 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 SlipError
from .Argument import Argument as Arg, ArgumentError

class OptionsError(SlipError):
    # 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()}
    except AttributeError as err:
        raise OptionsError('Options object expects dictionary types.')(err)
    except KeyError as key:
        raise OptionsError('{} was not a recognized option.'.format(key))(key)
    except ArgumentError as err:
        print('{} --> ArgumentError: {}, msg '.format(key, key))(err)
        raise OptionsError('Failed assignment.')(err)
    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))(key)
    def items(self):
        # Access options.items() values.
        return {k: v.value for k, v in self.options.items()}.items()
```
C.11  SLiPy. Correlate

```python
# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPL-3)
# slipy/slpym/Correlate.py

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

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

def rms(array):
    """Return the root mean square of an array."""
    if type(array) is not np.ndarray or len(array.shape) != 1:
        raise CorrelateError("RMS() expects 1D numpy arrays")
    return np.sqrt((array**2).sum()) / len(array)

def 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:
        options = Options(kwars, {'lag': 25 # pixels to shift for correlation})
        if type(spectrumA) 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")
        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("-
```
import os, sys, fnmatch
from astropy.io import fits as pyfits
from .. import SlipyError
from .Framework import Parse, CommandError
from .Framework.Display import DisplayError
from .Spectrum import Spectrum, SpectrumError
from .Simbad import Position, Distance, Sptype, IDList, SimbadError

class FitsError(SlipyError):
    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, f) for f 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, f) for dirpath, dirnames, filenames in os.walk(toplevel) for f 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
        'crpix': crpix,  # reference pixel header keyword
        'crval': crval,  # value at reference pixel
        'cdelt': cdelt,  # resolution (delta lambda)
    }
    try:
        # convert 'files' to list
        files = list(files)
    except:
        # available key word arguments
        options = Options(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
              'crpix': crpix,  # reference pixel header keyword
              'crval': crval,  # value at reference pixel
              'cdelt': cdelt,  # resolution (delta lambda)
        })
        # search for files matching 'pattern'
        if toplevel:
            # search for files matching 'pattern'
find = MFind if recursive else Find
files = find( toplevel, pattern )

if verbose:
    # import iteratively, displaying progress
    display = Monitor()
    nfiles = len(files)
data = []
print(' Importing data from {} fits files ...'.format(nfiles))
for a, filename in enumerate(files):
    display.progress(a, nfiles)
data.append( Spectrum(filename, wavecal=wavecal, crpix1=crpix1, crval1=crval1, cdelt1=cdelt1) )
display.complete()
return data

except OptionsError as err:
    print(' --> OptionsError: ', err)
    raise FitsError('Data retrieval failure.')
except SpectrumError as err:
    print(' --> SpectrumError: ', err)
    raise FitsError('Failed to construct spectrum.')

def Header( filename, keyword = None, **kwargs ):
    """
    Retrieve 'keyword' from FITS header in 'filename'. If not provided a
    keyword, the entire header object is returned.
    """
    try:
        options = Options( kwargs,
            ( 'is_main' : False )
        )
        is_main = options('is_main')
        with pyfits.open(filename) as hdulist:
            header = hdulist[0].header
            if keyword: header = header[keyword]
        if is_main:
            print( header )
            return
        else:
            return header

    except OptionsError as err:
        print(' --> OptionsError: ', err)
        raise FitsError('Failed keyword assignment in Header.')
    except KeyError as key:
        raise FitsError('Header element '{} was not accessible '.
                        'From {}'.format(keyword, filename))

def Search( files, **kwargs ):
    """
    Extract object names from Fits 'files' and use Simbad.py
    to resolve the 'attribute' (a required keyword argument)
    from the SIMBAD astronomical database.
    """
    kwgars = {
            'verbose' : True , # display messages, progress
            'toplevel' : '' , # search under 'toplevel' directory
            'pattern' : '*.fits' , # for files under 'toplevel'
            'recursive' : False , # search recursively under 'toplevel'
            'attribute' : '' , # attribute to search for (no default)
            'is_main' : False , # reserved for calls from Main()
        }
    try:
        # convert 'files' to list
        files = list(files)
    # available keyword arguments
    options = Options( kwgars,
        { 'verbose' : True , # display messages, progress
          'toplevel' : '' , # search under 'toplevel' directory
          'pattern' : '*.fits' , # for files under 'toplevel'
          'recursive' : False , # search recursively under 'toplevel'
          'attribute' : '' , # attribute to search for (no default)
          'is_main' : False # reserved for calls from Main()
        })
    # assign parameters
```python
verbos = options('verbos')
toplevel = options('toplevel')
pattern = options('pattern')
recursive = options('recursive')
attribute = options('attribute')
is_main = options('is_main')

# Available search functions from Simbad.py
SimbadSearch = {
    'Position': Position,  # ra, dec (degrees)
    'Distance': Distance,  # in parsecs
    'Sptype': Sptype,  # spectral type
    'IDlist': IDlist  # alternate IDs
}

if not attribute:
    raise FitsError("An 'attribute' must be specified for Search().")

if attribute not in SimbadSearch:
    raise FitsError("'{}' is not an available search criteria."
        .format(attribute))

if toplevel:
    # Search for files in 'toplevel' directory
    files = find if recursive else find
    nfiles = len(files)
display = Monitor()

if verbose:
    # Read object names iteratively
    print("Reading object names for {0} Fits files ...".format(nfiles))
    obj_ids = []
    for a, name in enumerate(files):
        display.progress(a, nfiles)
        obj_ids.append( Header(name, 'object') )
    display.complete()
else:
    obj_ids = [ Header(name, 'object') for name in files ]

if verbose:
    # Query for 'attribute' iteratively
    print("Searching for '{}'s with SIMBAD ...".format(attribute))
    results = []
    for a, obj in enumerate(obj_ids):
        display.progress(a, nfiles)
        results.append( SimbadSearch[attribute](obj) )
    display.complete()
else:
    results = [ SimbadSearch[attribute](obj) for obj in obj_ids ]

if is_main:
    formatted = {
        'Position':(1.2f) (1.2f),
        'Distance':(0.2f),
        'Sptype': '{}'
    }
    for item in results:
        if type(item) is list:
            print( formatted[attribute].format(*item))
        else:
            print( formatted[attribute].format(item))

else:
    return results

except OSError as err:
    print("Failed assignment for Search().")
raise

except SimbadError as err:
    print("Simbad failed.")
raise

def PositionSort( center, radius, *files, **kwargs ):
    ""
    Return a list of files from 'files' that lie in a 'radius' (in degrees)
    from 'center', based on the 'ra' (right ascension) and 'dec' (declination).
    ""
    kwargs = {
        'ra' : 'pos1',  # header element for right ascension
        'dec' : 'pos2',  # header element for declination
        'obj' : 'object',  # header element for object id
        'raconvert' : True,  # convert decimal hours to decimal degrees
        'deconvert' : True,  # convert decimal degrees to decimal degrees
        'verbose' : True,  # display messages, progress
        'toplevel' : '}',  # 'toplevel' directory to look for files in
        'recursive' : False,  # search 'recursively' below 'toplevel'
        'pattern' : '*.fits',  # glob 'pattern' for file search
        'useSimbad' : False  # use Simbad instead of header elements
    }
```

122
try:
    # function parameter defaults
    options = Options( kwargs, )
    (  
        'ra' : 'pos',  # header element for right ascension
        'dec' : 'pos',  # header element for declination
        'obj' : 'object',  # header element for object id
        'raconvert' : True,  # convert decimal hours to decimal degrees
        'verbose' : True,  # display messages, progress
        'toplevel' : 'toplevel',  # 'toplevel' directory for file search
        'recursive' : False,  # search 'recursive'ly below 'toplevel'
        'pattern' : '*.fits',  # glob 'pattern' for file search
        'useSimbad' : False  # use Simbad instead of header elements
    )

    # function parameter assignments
    ra = options('ra')
    dec = options('dec')
    obj = options('obj')
    raconvert = options('raconvert')
    verbose = options('verbose')
    toplevel = options('toplevel')
    recursive = options('recursive')
    pattern = options('pattern')
    useSimbad = options('useSimbad')

except OptionsError as err:
    print( OptionsError('Failed keyword assignment in PositionSort().') )
    raise

raise FitsError( Failed keyword assignment in PositionSort(). )

# check arguments
if not hasattrCENTERiterable__):  
    raise FitsError(PositionSort() expects 'center' argument to be 'iterable' and have two elements. )
if len(center) != 2:  
    raise FitsError(PositionSort() expects 'center' argument to have 'exactly two elements. ')
if not isinstance(radius, Number):
    raise FitsError(PositionSort() expects 'radius' argument to 'be a 'Number. ')
for a, f in enumerate(files):
    if not isinstance(f, str):
        raise FitsError(PositionSort() expects 'str' like arguments 'for all 'files' (from argument {}).')

    # convert 'files' to list type
    files = list(files)
    # look under 'toplevel' if requested
    if toplevel:
        files += find(toplevel, pattern)

    if verbose:
        # create display object
        display = Monitor()
        nfiles = len(files)
        # initialize blank lists
        pos1, pos2 = [], []
    if not useSimbad:
        if verbose:
            print('Retrieving position information from {} files ... '.format(nfiles))
            # check file headers for requested information
            for a, fitsfile in enumerate(files):
                try:
                    alpha = Header(fitsfile, ra)
                    delta = Header(fitsfile, dec)
                    if raconvert: alpha *= 180 / 12
                    pos1.append(alpha)
                    pos2.append(delta)
                except FitsError as err:
                    # attempt to get info from SIMBAD instead
                    print('\nFailed to retrieve position from file {}, contacted SIMBAD ...'.format(a))
                    pos = Position(Header(fitsfile, obj))
                    print(\'\n\n\')
                    pos1.append(pos[0])
                    pos2.append(pos[1])

                if verbose: display.progress(a + 1, nfiles)
            else:
                # use the Simbad module to search for positions
                for a, fitsfile in enumerate(files):
                    pos = Position(Header(fitsfile, obj))
                    pos1.append(pos[0])
                    pos2.append(pos[1])
                if verbose: display.progress(a, nfiles)

        else:
            # erase progress bar

except Exception as err:
    print(\nFailed to retrieve position from file {}, contacted SIMBAD ...'.format(a))
    pos = Position(Header(fitsfile, obj))
    print(\'\n\n\')
    pos1.append(pos[0])
    pos2.append(pos[1])
    if verbose: display.progress(a, nfiles)
```python
if verbose:
    display_complete()
    print('\033[K Compiling list of files ... ')

# keep files for targets within range
keepers = [ f for p1, p2, f in zip(pos1, pos2, files)
            if abs(p1 - center[0]) < radius and abs(p2 - center[1]) < radius ]

# account for p1 = 0 & center ~ 360 like comparisons
keepers = [ f for p1, p2, f in zip(pos1, pos2, files)
            if abs(p1 - center[0]) < radius and abs(p2 - center[1]) < radius ]

# account for p1 = 360 & center ~ 0 like comparisons
keepers = [ f for p1, p2, f in zip(pos1, pos2, files)
            if abs(p1 - center[0] - 360) < radius and abs(p2 - center[1]) < radius ]

if verbose: print('\033[A\r\033[K Compiling list of files ... done')

# exclude any potential double countings
return list(set(keepers))


def Main( clargs ) :
    """
    Main function. See __doc__ for details.
    """

    if len(clargs) < 2:
        # show usage
        print( __doc__)
        return 0

    # Available functions for execution
    executable = {
        'Header' : Header, # Header function
        'Search' : Search, # Search function
    }

    try:
        # Parse command line arguments
        function, argS, kwargS = Parse( clargs[1:] )
        # show function usage
        print( executable[function].__doc__ )
        return 0
    
    # run execution
    executable[function]( **argS, is_main=True, **kwargS )
    return 0

    except CommandError as err:
        print( ' --> CommandError: ' + str(err) )
        return 1

    except KeyError as key:
        print( ' --> () was not a recognized function.' + format(key) )
        return 1

    except FitsError as err:
        # don't let uncaught self exception pass if from main.
        print( ' --> FitsError: ' + format(err) + msg )
        return 1

    except Exception as err:
        # don't let uncaught self exception pass if from main.
        print( ' --> Exception: ' + format(err) + msg )
        return 1

    if __name__ == '__main__':
        # call Main function. exit 0 or 1
        sys.exit( Main(sys.argv) )
```

124
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, sh, numpy as np
from subprocess import check_output as call, CalledProcessError
from sys import stdout
from numbers import number
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):
    # 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 be mindful of their choices. If the side lengths cannot
    # be subdivided into well-behaved (rational) segments, higher decimal places
    # will be lost in the SubFieldArchiveList() 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_site_centers = np.linspace(dec_bottom_site, dec_top_site, grid[1])

    return ra_site_centers, dec_site_centers

def 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 and 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)
```
# Building mosaic for `{}`. Generating meta-data table ...

# Generate image meta-data table
output = call(['mimgtbl', 'images', 'images.tbl']).decode('utf-8')
if 'ERROR' in output: raise MontageError('Failed `mimgtbl` from `{}`'.format(folder))
if verbose:
    stdout.write('done.n Generating FITS header template ...
    stdout.flush()

# Create mosaic FITS header template
output = call(['mcreatehdr', '-p', 'images', 'images.tbl', 'template.hdr', 'template.hdr']).decode('utf-8')
if 'ERROR' in output: raise MontageError('Failed `mcreatehdr` in `{}`'.format(folder))
if verbose:
    stdout.write('done.n Reprojecting images ...
    stdout.flush()

# Reproject images
output = call(['mprojexec', '-p', 'images', 'images.tbl', 'template.hdr', 'projected', 'fits.tbl']).decode('utf-8')
if 'ERROR' in output: raise MontageError('Failed `mprojexec` in `{}`'.format(folder))
if verbose:
    stdout.write('done.n Generating new image meta-data table for projected images ...
    stdout.flush()

# Create new meta-data table for reprojected images
output = call(['mimgtbl', 'projected', 'proj-images.tbl']).decode('utf-8')
if 'ERROR' in output: raise MontageError('Failed `mimgtbl` in `{}`'.format(folder))
if not bkmodel:
    # Simply co-add images
    if verbose:
        stdout.write('done.n Co-adding images ...
        stdout.flush()
    output = call(['madd', '-p', 'projected', 'proj-images.tbl', 'template.hdr', 'final/mosaic.fits']).decode('utf-8')
    if 'ERROR' in output: raise MontageError('Failed `madd` in `{}`'.format(folder))
else:
    # Fit overlaps for background corrections
    if verbose:
        stdout.write('done.n Fitting overlaps for background corrections ...
        stdout.flush()
    output = call(['moverlaps', 'proj-images.tbl', 'diffs.tbl']).decode('utf-8')
    if 'ERROR' in output: raise MontageError('Failed `moverlaps` in `{}`'.format(folder))
    # Perform background subtractions on overlaps
    if verbose:
        stdout.write('done.n Performing background subtractions on overlaps ...
        stdout.flush()
    output = call(['mdiffexec', '-p', 'projected', 'diffs.tbl', 'template.hdr', 'differences']).decode('utf-8')
    if 'ERROR' in output: raise MontageError('Failed `mdiffexec` in `{}`'.format(folder))
    # Computing plane-fitting coefficients
    if verbose:
        stdout.write('done.n Computing plane-fitting coefficients ...
        stdout.flush()
    output = call(['mfitexec', 'diffs.tbl', 'fits.tbl', 'differences']).decode('utf-8')
    if 'ERROR' in output: raise MontageError('Failed `mfitexec` in `{}`'.format(folder))
    # Create table of background corrections
    if verbose:
        stdout.write('done.n Creating table of background corrections ...
        stdout.flush()
    output = call(['mbgmodel', 'proj-images.tbl', 'fits.tbl', 'corrections.tbl']).decode('utf-8')
    if 'ERROR' in output: raise MontageError('Failed `mbgmodel` in `{}`'.format(folder))
'()' .format(folder))

# apply background matching to reprojected images
if verbose:
    stdout.write('done\n Applying background matching to ')
    stdout.write('reprojected images ... ')
stdout.flush()
output = call(['mbgexec', '-p', 'projected', 'proj-images.tbl',
               'corrected.tbl'], decode='utf-8')
if 'ERROR' in output: raise MontageError('Failed `mbgexec` in ')
'()'.format(folder))

# co-add images for final mosaic
if verbose:
    stdout.write('done\n Co-adding corrected images ... ')
stdout.flush()
output = call(['madd', '-p', 'corrected', 'proj-images.tbl',
               'template.hdr', 'final/mosaic.fits']), decode='utf-8')
if 'ERROR' in output: raise MontageError('Failed `madd` in ')
'()'.format(folder))

# finished mosaic
if verbose:
    stdout.write('done\n')
stdout.flush()
except CalledProcessError as err:
    print('>>> CalledProcessError', err)
raise MontageError('Failed process from Mosaic()')

class SubField:
    def __init__(self, center, sides, grid, **kwargs):
        Create 'site' grid for SubField.
        try:
            # function parameter defaults
            options = Options( kwargs,
                {'survey': 'DSS', # DSS, SDSS, 2MASS
                 'band': 'DSS2B', # filter for 'survey', see 'bands' dict
                 'pad': 0.0 # amount to add (degrees) around 'sites' }
            )
            # function parameter assignments
            survey = options('survey').upper()
            band = options('band').upper()
            pad = options('pad')

            # available bands for each survey
            bands = [
                # Two-Micron All-Sky Survey
                '2MASS': ['J', 'H', 'K'],
                # Sloan Digital Sky Survey
                'SDSS': ['U', 'G', 'R', 'I', 'Z'],
                # STScI Digitized Sky Survey
                'DSS': ['DSS1B', 'DSS1R', 'DSS2B', 'DSS2R', 'DSS2R', 'Quick-V']
            ]

            # check for appropriate survey, band
            if survey not in bands:
                raise MontageError('() was not a recognized survey for ')
                'the Montage Python module.'.format(survey))
            if band not in bands[survey]:
                raise MontageError('() was not a recognized filter band ')
                for the () survey.'.format(band, survey))

            # check arguments
            if not hasattr(center, '__iter__') or
            not hasattr(sides, '__iter__') or
            not hasattr(grid, '__iter__ '):
                raise MontageError('SubField() expects array-like arguments for ')
                'center', 'sides', and 'grid' arguments. ')
            if len(center) != 2 or len(sides) != 2 or len(grid) != 2:
                raise MontageError('SubField() expects center, sides and ')
                'grid' arguments to have length two. ')

            # SolveGrid()
            ra_site_centers, dec_site_centers = SolveGrid(sides, grid)
            # relative to SubField center
            ra_site_centers += center[0]
            dec_site_centers += center[1]

            # record number of 'sites' along each axis
            self.num_ra_sites = grid[0]
            self.num_dec_sites = grid[1]

            # build arguments for subprocess call
```python
self.archive_command_list = [
    ['marchivelist', survey, band, '[{:2f} \dots{:2f}].format(ra_site,
    dec_site), str(pad + sides[0]/grid[0]),
    pad + sides[1]/grid[1]), 'remote OBJ']
for ra_site in ra_site_centers
for dec_site in dec_site_centers
]

# make current directory 'home'
self.location = os.path.abspath('')

# new tree structure
self.folders = [os.path.join(self.location, 'Site-{:04d}.format(a+)')
for a in range(len(self.archive_command_list))]

# initialize folder structure with 'images' directory
for folder in self.folders:
    abspath = os.path.join(folder, 'images')
    if not os.path.exists(abspath):
        os.makedirs(abspath)
except OptionsError as err:
    print('--- OptionsError:', err)
    raise MontageError('Failed keyword assignment in SubField()')

def ArchiveList(self, **kwargs):
    # Run the 'marchivelist' command on the 'site' grid.
    try:
        # function parameter defaults
        options = Options(kwargs,
        {'verbose': True # display messages, progress
        })
        # function parameter assignments
        verbose = options('verbose')
        for a, command in enumerate(self.archive_command_list):
            # navigate to 'images' directory
            os.chdir(os.path.join(self.folders[a], 'images'))
            if verbose:
                stdout.write('Running 'marchivelist' on {} ...
                .format(os.path.basename(self.folders[a])))
                stdout.flush()
            # run 'marchivelist'
            output = call(command).decode('utf-8')
            if 'ERROR' in output or 'count=0' in output:
                raise MontageError('Failed archive list from archive() '
                'command {} (output: {}).'.format(command, output))
            if verbose: stdout.write('done')
            if verbose:
                stdout.write('
')
                stdout.flush()
        except OptionsError as err:
            print('--- OptionsError:', err)
            raise MontageError('Failed keyword assigned from SubField.exec()')
        except CalledProcessError as err:
            print('--- CalledProcessError:', err)
            raise MontageError('marchivelist' returned exit status 1.)

def ArchiveExec(self, **kwargs):
    # Run 'marchivexec' on each 'site' in the SubField.
    try:
        # function parameter defaults
        options = Options(kwargs,
        {'verbose': True # display messages, progress
        })
        # function parameter assignments
        verbose = options('verbose')
        for a, folder in enumerate(self.folders):
            # navigate to site folder
            os.chdir(os.path.join(folder, 'images'))
            if verbose:
                stdout.write('Running 'marchivexec' on {} ...
                .format(os.path.basename(folder)))
                stdout.flush()
        # run 'marchivexec'
```
```python
def build(self, resolution, **kwargs):
    '''Run the build process for the 'sites' in this SubField. See the
    Montage Mosaic() function documentation.
    '''
    try:
        # function parameter options
        options = Options(kwargs,
            {'verbose':True,  # display message, progress
             'bkmodel':True  # run background modelling procedure
            })
        # function parameter assignments
        verbose = options['verbose']
        bkmodel = options['bkmodel']
    except OptionsError as err:
        print('  --> OptionsError:', err)
        raise MontageError('Failed keyword assignment in ArchiveExec().')
    except CalledProcessError as err:
        print('  --> CalledProcessError:', err)
        raise MontageError('ArchiveExec() returned exit status 1.()')

    if verbose:
        stdout.write('done\n')
        stdout.flush()
    # setup folder structure
    for subdir in ['corrected', 'projected', 'differences', 'final']:
        for folder in self.folders:
            abspath = os.path.join(folder, subdir)
            if not os.path.exists(abspath):
                os.makedirs(abspath)
            if verbose:
                stdout.write('Setting up folder structure ...\n')
                stdout.flush()
            # run Mosaic() on all 'site's
            Mosaic(resolution, *self.folders, verbose=verbose, bkmodel=bkmodel)
    if verbose:
        stdout.write('done\n')
        stdout.flush()
    # run Mosaic() on all 'site's
    Mosaic(resolution, *self.folders, verbose=verbose, bkmodel=bkmodel)

    if verbose:
        stdout.write('done\n')
        stdout.flush()
    # run Mosaic() on all 'site's
    Mosaic(resolution, *self.folders, verbose=verbose, bkmodel=bkmodel)
```

129
if verbose:
    stdout.write('Copying image from {} ...
    os.path.basename(folder))
    stdout.flush()

# Copy over image
sh.copy( os.path.join(folder, 'final/mosaic.fits'), os.path.join(
    master_images, 'mosaic({}).fits'.format(a + 1)) )

if verbose: stdout.write('done')

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

def Merge(self, resolution, **kwargs):
    ...
    Merge(resolution, **kwargs ):
    Merge all 'site' mosaics into a single master SubField mosaic. The only
keyword options are 'verbose' (default: True) and 'bkmodel' (default: True).
    See Montage.Mosaic().
    ...

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

    except OptionsError as err:
        print('Failed keyword assignment in SubField.Merge().')
        raise MontageError('Failed keyword assignment in SubField.Merge().')

    # check for master directory
    master_dir = os.path.join(self.location, 'master')
    if not os.path.exists(master_dir):
        raise MontageError('No master directory detected for Subfield')
        raise MontageError('No master directory detected for Subfield')

    # change directories
    os.chdir(master_dir)

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

    # run Mosaic of 'site' mosaics
    Mosaic(resolution, '{}', verbose=verbose, bkmodel=bkmodel)

class Field:
    ...
    Mosaic manager for 'Montage'.
    ...
    def __init__(self, center, sides, grid, subgrid, **kwargs):
        ...
        Initialize a Field centered on 'center' (array-like of length two with
right ascension and declination in decimal degrees) and side lengths
'sides' (array-like of length two in units of decimal degrees in
right ascension and declination respectively). 'grid' (array-like of
length two) specifies the grid layout (e.g., (4,4) says 4x4) to
subdivide the field. The 'subgrid' is the same but pertaining to the
further subdivision of each SubField into a grid layout of 'sites'.
    See SubField class.

    kwargs = {
        'verbose': True, # display message, progress
        'survey': 'DSS', # JMASS, SDSS, DSS
        'band': 'DSS2B' # filter 'band' pertaining to 'survey'
    }

    try:
        # function parameter defaults
        options = Options( kwargs, {
            'verbose': True, # display message, progress
            'survey': 'DSS', # JMASS, SDSS, DSS
            'band': 'DSS2B' # filter 'band' pertaining to 'survey'
        })

        # function parameter assignments
        verbose = options('verbose')
        survey = options('survey')
        band = options('band')
except OptionsError as err:
    raise MontageError(f"Failed keyword assignment in Field().")

# available bands for each survey
bands = {
    # Two-Micron All-Sky Survey
    '2MASS': ['J', 'H', 'K'],
    # Sloan Digital Sky Survey
    'SDSS': ['g', 'r', 'i', 'z', 'y'],
    # STScI Digitized Sky Survey
    'DSS': ['DSS1B', 'DSS1R', 'DSS2B', 'DSS2R', 'DSS2IR', 'Quick-9'],
}

# check for appropriate survey, band
if survey not in bands:
    raise MontageError(f"{} was not a recognized survey for {}.")

if band not in bands[survey]:
    raise MontageError(f"{} was not a recognized filter band for {}.")

# check arguments
if not hasattr(center, '__iter__') or not hasattr(subgrid, '__iter__'):  # Field() expects array-like arguments.
    raise MontageError(f"Field() expects arguments to be length two."

if verbose:
    stdout.write(f"Setting up a x field around {(grid[0], grid[1], center[0], center[1])}.
    stdout.flush()

    # solveGrid()
    self.ra_centers, self.dec_centers = solveGrid(sides, grid)

    # relative to field center
    self.ra_centers = center[0]
    self.dec_centers = center[1]

    # side lengths for SubField's
    self.sub_sides = sides / grid

    # set current directory to 'Field directory'.
    self.location = os.path.abspath('.

    # run SubField directories
    self.folders = [os.path.join(self.location, 'SubField-{:03d}'.format(a+1))
                    for a in range(len(self.ra_centers) * len(self.dec_centers))
    ]

    # create SubField directories
    for folder in self.folders:
        if not os.path.exists(folder):
            os.makedirs(folder)

        # zip together all ra, dec pairs
        self.sub_centers = self.sub_centers + [self.ra_centers for ra in self.ra_centers for self.dec_centers ]

    # initialize empty list of SubField's
    self.subfields = []

    # initialize all SubField's
    for a, folder, sub_center in zip(range(len(self.folders)), self.folders, self.sub_centers):
        if verbose:
            stdout.write(f"Initialzing () ...
            os.path.basename(folder))

        if verbose:
            stdout.write(f"done"

            if verbose:
                stdout.write(f"\n")

        def Archivelist(self, **kwargs):
            ... run 'ArchiveList()' on all SubFields. The only keyword option is
'verbose' (default: True).

try:
  # function parameter defaults
  options = Options(kwars,
                   {'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.ArchiveList().')
if verbose:
  stdout.write('\n Running ArchiveList() on all SubFields ... ')
  stdout.write('\n  
')
  stdout.flush()
# Run ArchiveList() on all SubField's:
for a, subfield in enumerate(self.subfields):
  if verbose:
    stdout.write('\n Running ArchiveList() on {} ... ')
    subfield: ArchiveList(verbos False)
    stdout.write('done')
if verbose:
  stdout.write('\n')
  stdout.flush()
def ArchiveExec(self, **kwars):
  # Run 'ArchiveExec()' on all SubFields. The only keyword option is
  # 'verbose' (default: True).
  try:
    # function parameter defaults
    options = Options(kwars,
                     {'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 ArchiveExec() on all SubFields ... ')
  stdout.write('\n  
')
  stdout.flush()
# Run ArchiveExec() on all SubField's:
for a, subfield in enumerate(self.subfields):
  if verbose:
    stdout.write('\n Running ArchiveExec() on {} ... ')
    subfield: ArchiveExec(verbos False)
    stdout.write('done')
if verbose:
  stdout.write('done')
def Build(self, resolution, **kwars):
  Build(resolution, **kwars):
  Run the build process for all SubFields in this Field. See the
documentation for Montage.Mosaic() and SubField.Build().
  kwars = {
    'verbose': True, # display messages, progress
    'bkmodel': True # run background modelling procedures.
  }
try:
    # function parameter defaults
    options = Options(kwars,
        # 'verbose': True, # display messages, progress
        # 'bkmodel': True, # run background modelling procedure
    )
except OptionsError as err:
    print( '-> OptionsError', err)
raise MontageError("Failed keyword assignment in 'Field::ArchiveExec()'")

if verbose:
    stdout.write("\nRunning Build() on all SubField's ...
")
    stdout.write("\n\n")
    stdout.write("\n")
    stdout.flush()

# Run ArchiveExec() on all SubField's
for a subfield in enumerate(self.subfields):
    if verbose:
        stdout.write("\n")
        stdout.write("\n")
        stdout.write("\n")
        stdout.flush()
    def Collect(self, **kwars):
        # Run Collect() on all SubFields of this Field.
        try:
            # function parameter defaults
            options = Options(kwars,
                # 'verbose': True, # display messages, progress
            )
        except OptionsError as err:
            print( '-> OptionsError', err)
            raise MontageError("Failed keyword assignment in 'Field::ArchiveExec()'")
        if verbose:
            stdout.write("\nRunning Collect() on all SubField's ...
")
            stdout.write("\n\n")
            stdout.write("\n")
            stdout.flush()
        # Run ArchiveExec() on all SubField's
        for a subfield in enumerate(self.subfields):
            if verbose:
                stdout.write("\n")
                stdout.write("\n")
                stdout.write("\n")
                stdout.flush()
    def Merge(self, resolution, **kwars):
        try:
            # function parameter defaults
            options = Options(kwars,
                # 'verbose': True, # display messages, progress
            )
        except OptionsError as err:
            print( '-> OptionsError', err)
            raise MontageError("Failed keyword assignment in 'Field::ArchiveExec()'")
        if verbose:
            stdout.write("\n")
            stdout.write("\n")
            stdout.write("\n")
            stdout.flush()
'verbose': True, # display messages, progress
)

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

except OptionsError as err:
    print(\n        '--> OptionsError: ', err)
raise MontageError("Failed keyword assignment in '{0}".".format('Field.ArchiveExec()).")

if verbose:
    stdout.write("\n Running Merge() on all SubField's ... ")
    stdout.write("\n" + ' ' + 70 + '\n")
    stdout.write("\n" + ' ' + 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 '{0}' / '{1}' ... ")
        .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(\n                '--> 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)

    # collection of master mosaics
    for a, folder in enumerate(self.folders):
        if verbose:
            stdout.write("\n Copying image from '{0}' ... '{1}'\n")
            .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 '{0}'")
            .format(os.path.basename(folder))

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

    if verbose: stdout.write("done")
if verbose:
    stdout.write('
')
    stdout.flush()

# change directories to 'master'
os.chdir(master_dir)

# run Mosaic() on all SubField 'master's
Mosaic(resolution, '.', verbose=verbose, bkmodel=bkmodel)
Classes for defining observatory parameters (similar to the IRAF task).

```python
from astropy import units as u

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 __str__(self):
        return str(self)

    def __repr__(self):
        return str(self)

    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 = 43.93834 * u.degree # North
            self.altitude = 650 * u.meter
            self.timezone = 1 * u.hourangle
            self.resolution = 42000 * u.dimensionless_unscaled

    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 = 31.963333333 * 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 = 31.963333333 * 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 = -30.1652778 * 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 = -28.743331111 * u.degree # North
            self.altitude = 2347 * u.meter
            self.timezone = 4 * u.hourangle
```

C.14 .. SLiPy . Observatory

# Copyright (c) Geoffrey Lentner 2015. All Rights Reserved.
# See LICENSE (GPLv3)

# SLiPy/Observatory.py

# 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 __str__(self):
        return str(self)

    def __repr__(self):
        return str(self)

    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 = 43.93834 * u.degree # North
            self.altitude = 650 * u.meter
            self.timezone = 1 * u.hourangle
            self.resolution = 42000 * u.dimensionless_unscaled

    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 = 31.963333333 * 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 = 31.963333333 * 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 = -30.1652778 * 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 = -28.743331111 * u.degree # North
            self.altitude = 2347 * u.meter
            self.timezone = 4 * u.hourangle
class PARANAL(Observatory):
    # European Southern Observatory: Paranal
    def __init__(self):
        self.name = "European Southern Observatory: Paranal"
        self.longitude = 70.403333333 * u.degree # West
        self.latitude = -23.375 * u.degree # North
        self.altitude = 2635 * u.meter
        self.timezone = 4 * u.hourangle

class LICK(Observatory):
    # Lick Observatory
    def __init__(self):
        self.name = "Lick Observatory"
        self.longitude = 121.636666667 * u.degree # West
        self.latitude = 37.343333333 * u.degree # North
        self.altitude = 1290 * u.meter
        self.timezone = 8 * u.hourangle

# Observatory entry from a conversation with Craig Foltz 8/20/97.
# House was changed and the "mmt" entry was removed.
class MMT(Observatory):
    # MMT Observatory
    def __init__(self):
        self.name = "MMT Observatory"
        self.longitude = 110.885 * u.degree # West
        self.latitude = 31.688333333 * u.degree # North
        self.altitude = 2000 * u.meter
        self.timezone = 7 * u.hourangle

class CFHT(Observatory):
    # Canada-France-Hawaii Telescope
    def __init__(self):
        self.name = "Canada-France-Hawaii Telescope"
        self.longitude = 155.471666667 * u.degree # West
        self.latitude = 19.8256666667 * u.degree # North
        self.altitude = 4215 * u.meter
        self.timezone = 10 * u.hourangle

class LAPALMA(Observatory):
    # Roque de los Muchachos, La Palma
    def __init__(self):
        self.name = "Roque de los Muchachos, La Palma"
        self.longitude = 17.88 * u.degree # West
        self.latitude = 28.758333333 * u.degree # North
        self.altitude = 2327 * u.meter
        self.timezone = 8 * u.hourangle

class MSO(Observatory):
    # Mt. Stromlo Observatory
    def __init__(self):
        self.name = "Mt. Stromlo Observatory"
        self.longitude = 210.976666667 * u.degree # West
        self.latitude = -34.67935 * u.degree # North
        self.altitude = 1149 * u.meter
        self.timezone = -10 * u.hourangle

class SSO(Observatory):
    # Siding Spring Observatory
    def __init__(self):
        self.name = "Siding Spring Observatory"
        self.longitude = 210.938805556 * u.degree # West
        self.latitude = -30.7266388889 * u.degree # North
        self.altitude = 1149 * u.meter
        self.timezone = -10 * u.hourangle

class AAO(Observatory):
    # Anglo-Australian Observatory
    def __init__(self):
        self.name = "Anglo-Australian Observatory"
        self.longitude = 210.933913889 * u.degree # West
        self.latitude = -30.7229611111 * u.degree # North
        self.altitude = 1164 * u.meter
        self.timezone = -10 * u.hourangle

class MCDONALD(Observatory):
    #
McDonald Observatory

```python
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

```python
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 MTDOLOW(Observatory):

Catalina Observatory: 61 inch telescope

```python
def __init__(self):
    self.name = "Catalina Observatory: 61 inch telescope"
    self.longitude = 110.7316666667 * 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

```python
def __init__(self):
    self.name = "Dominion Astrophysical Observatory"
    self.longitude = 123.4166666667 * u.degree # West
    self.latitude = 48.5216666667 * u.degree # North
    self.altitude = 229 * u.meter
    self.timezone = 7 * u.hourangle
```

# Submitted by Patrick Vielle 5/4/93

class SP(Observatory):

Observatorio Astronomico Nacional, San Pedro Martir

```python
def __init__(self):
    self.name = "Observatorio Astronomico Nacional, San Pedro Martir"
    self.longitude = 115.486944444 * u.degree # West
    self.latitude = 31.8291666667 * u.degree # North
    self.altitude = 2100 * u.meter
    self.timezone = 7 * u.hourangle
```

# Submitted by Patrick Vielle 5/4/93

class TONA(Observatory):

Observatorio Astronomico Nacional, Tonantzintla

```python
def __init__(self):
    self.name = "Observatorio Astronomico Nacional, Tonantzintla"
    self.longitude = 98.3138888889 * u.degree # West
    self.latitude = 19.0327777778 * u.degree # North
    self.timezone = 8 * u.hourangle
```

# Submitted by Ron Hamilton 8/18/93
class PALOMAR(Observatory):

The Hale Telescope

```python
def __init__(self):
    self.name = "The Hale Telescope"
    self.longitude = 116.863 * u.degree # West
    self.latitude = 33.356 * u.degree # North
    self.altitude = 1786 * u.meter
    self.timezone = 8 * u.hourangle
```

# Submitted by Pat Seitzer 10/31/93

class DMM(Observatory):

Michigan-Dartmouth-MIT Observatory

```python
def __init__(self):
    self.name = "Michigan-Dartmouth-MIT Observatory"
    self.longitude = 111.6166666667 * u.degree # West
    self.latitude = 31.355 * u.degree # North
    self.altitude = 1938.5 * u.meter
    self.timezone = 7 * u.hourangle
```

# Submitted by Ignacio Ferrin 9/1/94
```python
class NOV(Observatory):
    """
    National Observatory of Venezuela
    """
    def __init__(self):
        self.name = 'National Observatory of Venezuela'
        self.longitude = 70.866666666667 * u.degree
        self.latitude = 8.79 * u.degree
        self.altitude = 3610 * u.meter
        self.timezone = 4 * u.hourangle

# Submitted by Alan Welty 10/28/94
class RMO(Observatory):
    """
    Black Moshannon Observatory
    """
    def __init__(self):
        self.name = 'Black Moshannon Observatory'
        self.longitude = 78.085 * u.degree
        self.latitude = 40.821666666667 * u.degree
        self.altitude = 738 * u.meter
        self.timezone = 5 * u.hourangle

# Submitted by Biwei Jiang 1/1/2895
class BAO(Observatory):
    """
    Beijing XingLong Observatory
    """
    def __init__(self):
        self.name = 'Beijing XingLong Observatory'
        self.longitude = 124.245 * u.degree
        self.latitude = 40.9333333333 * u.degree
        self.altitude = 958 * 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 = 115.4783333333 * u.degree
        self.latitude = 19.8283333333 * u.degree
        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 = 148.418866666 * u.degree
        self.latitude = 45.8858888889 * u.degree
        self.altitude = 1413.69 * u.meter
        self.timezone = -1 * u.hourangle

# Submitted by Michael Ledlow 8/8/96
class AMO(Observatory):
    """
    Apache Point Observatory
    """
    def __init__(self):
        self.name = 'Apache Point Observatory'
        self.longitude = 105.82 * u.degree
        self.latitude = 32.78 * u.degree
        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
        self.latitude = 35.0966666667 * u.degree
        self.altitude = 2198 * u.meter
        self.timezone = 7 * u.hourangle

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

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

# Submitted by S. Giridhar 6/28/63
class IA(Observatory):
    
    def __init__(self):
        self.name = 'Indian Astronomical Observatory, Hanle'
        self.longitude = 281.83583 * u.degrees # West
        self.latitude = 32.7794 * u.degrees # North
        self.altitude = 4500 * u.meter
        self.timezone = -5.5 * u.hourangle

    # Submitted by Doug Mink 1/6/97
class FLWO(Observatory):
    
    def __init__(self):
        self.name = 'Whipple Observatory'
        self.longitude = 110.8775 * u.degrees # West
        self.latitude = 31.6899444444 * u.degrees # North
        self.altitude = 2320 * u.meter
        self.timezone = 7 * u.hourangle

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

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

    # Submitted by John Menzies 12/31/99
class SAO(Observatory):
    
    def __init__(self):
        self.name = 'South African Astronomical Observatory'
        self.longitude = 339.1893055556 * u.degrees # West
        self.latitude = -31.6285555556 * u.degrees # North
        self.altitude = 1798 * u.meter
        self.timezone = -2 * u.hourangle

    # Submitted by Jorge Federico Gonzalez 12/10/98
class CASLEO(Observatory):
    
    def __init__(self):
        self.name = 'Complejo Astronomico El Leoncito, San Juan'
        self.longitude = 69.3 * u.degrees # West
        self.latitude = -30.2088333333 * u.degrees # North
        self.altitude = 2552 * u.meter
        self.timezone = 3 * u.hourangle

    # Submitted by Jorge Federico Gonzalez 12/10/98
class BOSQUE(Observatory):
    
    def __init__(self):
class ROZHEN(Observatory):
    "National Astronomical Observatory Rozhen - Bulgaria"
    def __init__(self):
        self.name = "National Astronomical Observatory Rozhen - Bulgaria"
        self.longitude = 55.261111
        self.latitude = 41.37833333
        self.altitude = 2168
        self.timezone = -1

# Submitted by Ilian Iliev 1/19/99

class IRTF(Observatory):
    "NASA Infrared Telescope Facility"
    def __init__(self):
        self.name = "NASA Infrared Telescope Facility"
        self.longitude = 155.47199
        self.latitude = 19.26218
        self.altitude = 4168
        self.timezone = 10

# Submitted by Bill Vacca 7/14/99

class BSU(Observatory):
    "Bowling Green State Univ Observatory"
    def __init__(self):
        self.name = "Bowling Green State Univ Observatory"
        self.longitude = 83.65966667
        self.latitude = 41.37833333
        self.altitude = 2168
        self.timezone = 5

# Submitted by Andy Layden 7/16/99

class DSAZ(Observatory):
    "Deutsch-Spanisches Observatorium Calar Alto - Spain"
    def __init__(self):
        self.name = "Deutsch-Spanisches Observatorium Calar Alto - Spain"
        self.longitude = 2.54625
        self.latitude = 37.22681111
        self.altitude = 2168
        self.timezone = -1

# 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
        self.latitude = 37.22681111
        self.altitude = 2168
        self.timezone = -1

# Submitted by Oliver-Mark Cordes 8/5/99

class HMO(Observatory):
    "Observatorium Hoher List (Universitaet Bonn) - Germany"
    def __init__(self):
        self.name = "Observatorium Hoher List (Universitaet Bonn) - Germany"
        self.longitude = 6.88
        self.latitude = 50.16276
        self.altitude = 541
        self.timezone = -1

# Submitted by Steven Majewski 8/27/99

class LMD(Observatory):
    "Leander McCormick Observatory"
    def __init__(self):
        self.name = "Leander McCormick Observatory"
        self.longitude = 78.52333333
        self.latitude = 38.73333333
        self.altitude = 264
        self.timezone = 5

# Submitted by Steven Majewski 8/27/99

class FMO(Observatory):
```python
def __init__(self):
    self.name = "Fan Mountain Observatory"
    self.longitude = 78.6933333333 * u.deg  # West
    self.latitude = 37.8733333333 * u.deg  # North
    self.altitude = 566 * u.m
    self.timezone = 0 * u.hourangle

# Submitted by Kie & McLeod 10/13/1999
class WHITIN(Observatory):
    #
    Whitin Observatory, Wellesley College
    #

    def __init__(self):
        self.name = "Whitin Observatory, Wellesley College"
        self.longitude = 71.385833 * u.deg  # West
        self.latitude = 42.295 * u.deg  # North
        self.altitude = 362 * u.m
        self.timezone = 0 * 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 = 3.3847222222 * u.deg  # West
            self.latitude = 37.6641666667 * u.deg  # North
            self.altitude = 2996 * u.m
            self.timezone = -1 * u.hourangle

        # Gemini Observatory - Submitted by Inger Jorgensen
        #
        class GEMINI(NORTH(Observatory):
            #
            Gemini North Observatory
            #

            def __init__(self):
                self.name = "Gemini North Observatory"
                self.longitude = 19.8238815 * u.deg  # West
                self.latitude = 42.134 * u.m
                self.timezone = 10 * u.hourangle

        # Gemini-North
        # These values are from the aerial survey made Sept 25, 1996
        # http://www.ifla.hawaii.edu/eko/coordinates.html
        class GEMINI(SOUTH(Observatory):
            #
            Gemini South Observatory
            #

            def __init__(self):
                self.name = "Gemini South Observatory"
                self.longitude = -19.75925 * u.deg  # West
                self.latitude = -29.7716666667 * u.deg  # North
                self.altitude = 2737. * u.m
                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 = 78.7233333333 * u.deg  # West
                self.latitude = -25.7716666667 * u.deg  # North
                self.altitude = 2722. * u.m
                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.deg  # West
                self.latitude = -29.7433333333 * u.deg  # North
                self.altitude = 2347 * u.m
                self.timezone = 4 * u.hourangle

```
class PARANAL(Observatory):
    European Southern Observatory: Paranal
    def __init__(self):
        self.name = "European Southern Observatory: Paranal."
        self.longitude = 70.40333333 * u.degree # West
        self.latitude = -23.375 * u.degree # North
        self.altitude = 2635 * u.meter
        self.timezone = 4 * u.hourangle

# The following additional entries were supplied by Peter Weilbacher
# weilbach@uni-mw.gwdg.de on 28 Jan 2002 13:12:59 -0700.
class ESOMT(Observatory):
    European Southern Observatory, NTT, La Silla
    def __init__(self):
        self.name = "European Southern Observatory, NTT, La Silla."
        self.longitude = 70.7317422222 * u.degree # West
        self.latitude = -28.7448777778 * u.degree # North
        self.altitude = 2375 * u.meter
        self.timezone = 4 * u.hourangle

class ESO36M(Observatory):
    European Southern Observatory, 3.6m Telescope, La Silla
    def __init__(self):
        self.name = "European Southern Observatory, 3.6m Telescope, La Silla."
        self.longitude = 70.7296127778 * u.degree # West
        self.latitude = -28.7428294444 * u.degree # North
        self.altitude = 2400 * u.meter
        self.timezone = 4 * u.hourangle

class ESOVLT(Observatory):
    European Southern Observatory, VLT, Paranal
    def __init__(self):
        self.name = "European Southern Observatory, VLT, Paranal."
        self.longitude = 70.4922 * u.degree # West
        self.latitude = -24.6253 * u.degree # North
        self.altitude = 2648 * u.meter
        self.timezone = 4 * u.hourangle

# Submitted by Giovanni Catanzaro, 7/17/03.
class SLN(Observatory):
    SLN - Catania Astrophysical Observatory
    def __init__(self):
        self.name = "SLN - Catania Astrophysical Observatory."
        self.longitude = 345.026666667 * u.degree # West
        self.latitude = 37.6916666667 * u.degree # North
        self.altitude = 1725 * u.meter
        self.timezone = -1 * u.hourangle

# Submitted by Ahmet Devlen, 4/21/04.
class EUO(Observatory):
    Ege University Observatory
    def __init__(self):
        self.name = "Ege University Observatory"
        self.longitude = -36.625 * u.degree # West
        self.latitude = 38.339333333 * u.degree # North
        self.altitude = 795. * u.meter
        self.timezone = 2 * u.hourangle

# Submitted by Zeki Aslan 8/15/85 who said the "tno" entry was wrong.
class TNO(Observatory):
    Turkkiye National Observatory
    def __init__(self):
        self.name = "Turkiye National Observatory"
        self.longitude = -29.6666666667 * u.degree # West
        self.latitude = 36.8244444444 * u.degree # North
        self.altitude = 2555. * u.meter
        self.timezone = 2 * u.hourangle

class TUG(Observatory):
    TUBITAK National Observatory, Turkey
    def __init__(self):
        self.name = "TUBITAK National Observatory, Turkey."
        self.longitude = -29.6666666667 * u.degree # West
```python
self.latitude = 36.825 * u.degree  # North
self.altitude = 2547. * u-meter
self.timezone = -2 * u.hourangle

# Submitted 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

# Submitted by Jeevan C. Bandey 7/28/05
# Changed to M longitude MUF 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 = 80.54361 * u.degree  # West
    self.latitude = 29.36 * u.degree  # North
    self.altitude = 1950. * u-meter
    self.timezone = -5.5 * u.hourangle

# Submitted by Eduardo Fernandez Lazo 10/28/05
class OARP(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 = -33.9524488889 * 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 = 72.105277778 * 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 = 32.594444444 * u.degree  # West
    self.latitude = -28.961111111 * 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 = 248.873333333 * u.degree  # West
    self.latitude = 23.4803333333 * u.degree  # North
    self.altitude = 2862. * u-meter
    self.timezone = -8 * u.hourangle

# Submitted by Mairan Teodoro 1/27/10
class SDAR(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
```
```python
class 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 MET(Observatory):
    def __init__(self):
        self.name = "McDonald Observatory - Hobby-Eberly Telescope"
        self.longitude = 104.01472222 * 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 JCMO(Observatory):
    def __init__(self):
        self.name = "Jack C. Davis Observatory, Western Nevada College"
        self.longitude = 119.79066667 * 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):
    def __init__(self):
        self.name = "Langkawi National Observatory"
        self.longitude = 100.2188888889 * u.degree # West
        self.latitude = 6.38594444444 * u.degree # North
        self.altitude = 117 * u.meter
        self.timezone = -8 * u.hourangle
```

145
C.15 SLiPy Plot

```python
# 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 ..Framework.Options import Options, OptionsError
from .Fits import 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.xargs = []
            self.yargs = []
            self.targs = []
            self.xkwargs = {}
            self.ykwargs = {}
            self.tkwargs = {}
            self.xkwargs = {}
            self.ykwargs = {}
            self.tkwargs = {}
            self.txkwargs = {}
            self.tzkwargs = {}
            self.xlimits = {spectrum[wave]  # set x limits to the data
                             self.xlimits = [spectrum[wave].min(),
                                             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()
```

146
def xlim(self, xmin, xmax):
    ""
    Handle to pyplot.xlim
    ""
    self.xlimits = [xmin, xmax]
    self.ax.set_xlim(xmin, xmax)

def ylim(self, ymin, ymax):
    ""
    Handle to pyplot.ylim
    ""
    self.ylimits = [ymin, ymax]
    self.ax.set_ylim(ymin, ymax)

def xlabel(self, *args, **kwargs):
    """x axis label."
    self.xargs = args
    self.xkwargs = kwargs
    self.ax.set_xlabel(*args, **kwargs)

def ylabel(self, *args, **kwargs):
    """y axis label."
    self.yargs = args
    self.ykwargs = kwargs
    self.ax.set_ylabel(*args, **kwargs)

def title(self, *args, **kwargs):
    """title for plot."
    self.targs = args
    self.tkargs = kwargs
    self.ax.set_title(*args, **kwargs)

def legend(self, *args, **kwargs):
    """legend for plot."
    self.largs = args
    self.lkwargs = kwargs
    plt.legend(*args, **kwargs)

def text(self, *args, **kwargs):
    """display text over plot."
    self.txargs = args
    self.txkwargs = kwargs
    self.ax.text(*args, **kwargs)

def txtclear(self):
    """Clear all 'text' from figure."
    self.txargs = []
    self.txkwargs = []
    self.draw()

def markers(self, *args):
    """Reassign the values for the 'marker' s in the figure. The number
    of arguments must equal the number of spectra in the figure. This
    starts out as one, but will increase for ever SPlot.overlay()."
    if len(args) != len(self.data):
        raise PlotError('() arguments were given but there are {} spectra plotted in this figure!'.format(len(args), len(self.data)))
    for a, mark in enumerate(args):
        if type(mark) is not str:
            raise PlotError('Arguments given to SPlot.markers() must be () but argument {} was {}'.format('""", a[1], type(mark)))
    self.marker = list(args)

def _build(self, picker = False):
    """Make the plot."
    if picker:
        self.restore()
        self.ax.plot(self.data[0], self.data[1], self.marker[0], label = self.label[0], picker = True)
    else:
        for x, y, m, l in zip(self.data[0], self.data[1], self.marker, self.label):
            self.ax.plot(x, y, m, label=l)
if self.xargs or self.xkwargs:
    self.xlabel(*self.xargs, **self.xkwargs)
if self.yargs or self.ykwargs:
    self.ylabel(*self.yargs, **self.ykwargs)
if self.targs or self.tkwargs:
    self.title(*self.targs, **self.tkwargs)
if self.largs or self.lkwargs:
    self.legend(*self.largs, **self.lkwargs)
if self.txargs or self.txkwargs:
    for args, kwargs in zip(self.txargs, self.txkwargs):
        self.ax.text(*args, **kwargs)
if self.xlims:
    self.xlim(*self.xlims)
if self.ylims:
    self.ylim(*self.ylims)
if self.gridv:
    self.grid(self.gridv)
if self.use_tex:
    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')
def refresh(self):
    ""
    pyplot.draw()
    pyplot.draw()

def draw(self, picker=False):
    ""
    Re-build the plot
    ""
    self.ax.clear()
    self.___build(picker=picker)
    pyplot.draw()

def close(self):
    ""
    Close the plot.
    ""
    pyplot.close('Spectral-Plot (SLiPy)')

def grid(self, value):
    ""
    Show grid on plot.
    ""
    self.gridv = value
    pyplot.grid(value)

def save(self, filename):
    ""
    Save plot to 'filename'. Must have extension for formatting.
    ""
    if type(filename) is not str:
        raise PlotError('filename should be of type str.')
    if len(filename.split('.')) < 2:
        raise PlotError('filename needs an extension.')
    pyplot.savefig(filename, format=filename.split('.')[-1])

def xoffset(self, value):
    ""
    Toggle the offset for the x axis
    ""
    pyplot.gca().get_xaxis().get_major_formatter().set_useOffset(value)

def yoffset(self, value):
    ""
    Toggle the offset for the y axis
    ""
    pyplot.gca().get_yaxis().get_major_formatter().set_useOffset(value)

def tight_layout(self):
    ""
    pyplot.tight_layout()
    ""
    pyplot.tight_layout()

def overlay(self, splots):
    ""
    Overlay (add) spectra to this plot from other 'splots'.
    ""
    for a, plot in enumerate(splots):
# check data type
if type(plot) is not SPlot:
    raise PlotError('SPlot overlay expects ' 'type SPlot! (from argument {}).'.format(a))

# add data
self.data += plot.data
self.wave += plot.wave
self.marker += plot.marker
self.label += plot.label

def restore(self):
    ...  # Restore self.data and self.wave from possible 'overlay's.
    ...
    self.data = [ self.data[8] ]
    self.wave = [ self.wave[8] ]
    self.marker = [ self.marker[8] ]
    self.label = [ self.label[8] ]

def desired(plot):
    ...  # Helper function for Iterate. Prompts user to keep 'plot';
    ...  # returns True or False.
    ...  # draw the plot
    plot.draw()
    ...  # prompt the user for input
    prompt = input('{}[K keep -> '] {y/[n]/x}? ' .format(plot.name)).strip()
    # ensure valid response while True:
    if prompt not in ['y', 'n', 'x', 'x']:
        # invalid input. prompt again
        print('{}[K keep not a recognized response. .' .format(prompt))
        prompt = input('{}[K keep -> '] {y/[n]/x}? ' .format(plot.name)).strip()
    else:
        # clear the error message
        print('{}[K User exited early, saving results. ')
        break
    if prompt in ['n', 'n']:
        return False
    elif prompt in ['y']:
        return True
    else:
        # the user input 'x'
        raise KeyboardInterrupt('{}[K User exited early, saving results. ')

def Iterate( *plots, **kwargs):
    ...  # Iterate thru 'plots' to inspect data. The user marks 'plots' of
    ...  # interest. The function returns a list of 'names' marked.
    try:
        options = Options( kwars,
            {  # keep': 'name' # alternatively, 'plot'  
                'keep': 'name'  
            })
        keep = options('keep')
        if keep not in ['name', 'plot']:
            raise PlotError('Iterate expects either 'name' or 'plot' for '
                'keyword argument 'keep'.')
    # check input arguments
    for plot in plots:
        if not hasattr(plot, 'draw'):
            raise PlotError('Iterate expects objects to '  
                'have a 'draw' method.')
        if not hasattr(plot, 'name'):
            raise PlotError('Iterate expects objects to '  
                'have a 'name' method.')
    # clear some space
    print('
')

    keepers = []
    for a, plot in enumerate(plots):
        print('{}[K Showing plot {} of {} ... ' .format(a, len(plots)))
        if desired(plot):
            if keep == 'name':
                keepers.append(plot.name)
elif keep == 'plot'
    keepers.append( plot )

return keepers

except OptionsError as err:
    print('OptionsError', err)
    raise PlotError('Failed to initialize iterate.')

except KeyboardInterrupt as x:
    print(x)

return keepers
C.16  .. SLiPy . Profile

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

** Profile fitting tasks for spectra. **

```
Given 'splot' of type 'SPlot', the user selects four points on the spectrum and a parameterized function is fit (an inverted Lorentzian by default). Optionally, 'splot' can be of type spectrum and a basic SPlot will be created for you. If the user gives an alternative 'function', 'params' (parameters) must be provided. 'params' is to be the first guess, 'p0' given to scipy...curve_fit; the user can provide them explicitly, or in the form of functions with the template 'function(xarray, yarray)'

where 'xarray' and 'yarray' are the 'wave' and 'data' arrays extracted between the two points selected by the user.
The gap is jumped using ID interpolation (scipy...interpld).

```python
options = Options( kwars, {
  'kind' : 'cubic', # given to scipy...interpld for continuum
  'bandwith' : 0.1*u.m, # user should provide this!
  # 'rms' : False, # measure the RMS of the line, continuum ')
  })

kind = options('kind')
bandwidth = options('bandwith')

# rms = options('rms')

except OptionsError as err:
  print('Please select four points identifying the spectral line.')
  raise ProfileError('Unrecognized option given to Extract()')

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.data[0].copy()
yl = splot.data.data[0].copy()
x = xl + wave[2] + wave[1]
y = x + wave[2] + wave[1]
line = Spectrum(y1, x1)

# extract continuum arrays
xc = splot.wave.data[0].copy()
yc = splot.data.data[0].copy()

# inside outer-most selections
yc = yc[xc < wave[3] > wave[0]]
x = xc[xc < wave[3] > wave[0]]

# keep wavelengthes whole domain for later
xx = xc.copy()
yy = yc.copy()

# but not the line
yc = yc[~np.logical_or(xc < wave[1], xc > wave[2])]""""""""""""""""""""""""""

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

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

# continuum inside the line
cont_inside = interp(xc)

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

# 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) - 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 beforehand.

'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, '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)
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 spectrum 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 beforehand.

'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). ... 
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 
else: uncertainty = np.array([np.abs((upper - W).value), np.abs((lower - W).value)]) * tau.wave.unit 
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):
    if not hasattr(ion, 'Atomic.Ion') or not isinstance(ion, Atomic.Ion): 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 = 1.1332 / 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([np.abs((N - upper).value), np.abs((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 (1995)
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, 0.151], 
])
class FittingGUI:
    """
    Graphical Interface (keeps references alive) for fitting analytical
    profiles to spectral data.
    """
    def __init__(self, splot, obs=None, ion=None, function='Voigt', **kwargs):
        Build widget elements based on a spectrum plotted in 'splot' (type SPlot).
        'splot' may also be a Spectrum object (for which a SPlot will be created).
        Make initial guesses at parameters, build widgets, render the figure.
        try:
            options = Options(kwags, {
                'kind': 'cubic', # given to interp1d for continue
                'bandwidth': 0.1*u.nm, # user should provide this!
            })
            kind = options['kind']
            bandwidth = options['bandwidth']
        except OptionsError as err:
            print('Unhandled option given to FittingGUI.__init__()!')
            raise ProfileError('Unrecognized option given to FittingGUI.__init__()!')
        if function not in ['Voigt', 'Lorentzian', 'Gaussian']:
            raise ProfileError('The only currently implemented functions,
            'for fitting profiles are 'Voigt', 'Lorentzian' and 'Gaussian'!')
        if not isinstance(obs, Observatory):
            raise ProfileError('From FittingGUI.__init__(), if the absorption line
            'parameters 'obs' or 'ion' are provided, both must be given!')
        if self.any_line_parameters and not self.has_line_parameters:
            raise ProfileError('From FittingGUI.__init__(). If the absorption line
            'parameters 'obs' or 'ion' are provided, both must be given!')
        if function == 'Voigt':
            raise ProfileError('From FittingGUI.__init__(), in order to compute the
            'broadening from the instrument profile of the 'obs'ervatory and the
            'column density for the 'ion', it is expected that we try to fit a 'Voigt'
            'profile 'function'!')
    if not hasattr(obs, 'resolution'):
        raise ProfileError('From FittingGUI.__init__(), if an observatory 'obs' is
        'provided, it needs to have a member 'resolution'!')
    if not hasattr(ion, 'wavelength') or not hasattr(ion.wavelength, 'unit'):
        raise ProfileError('From FittingGUI.__init__(), the provided 'ion' either
        'did not have a 'wavelength' attribute or it has one without units!')
    if not hasattr(ion, 'fvalue'):
        raise ProfileError('From FittingGUI.__init__(), the provided 'ion' does not
        'have an 'fvalue' (oscillator strength) attribute!')
    if hasattr(ion.fvalue, 'unit') and ion.fvalue_unit != u.dimensionless_unscaled:
        raise ProfileError('From FittingGUI.__init__(), the oscillator strength,
        'fvalue' for an ion must be a dimensionless Quantity!')
    if not hasattr(ion, 'A') or not ion.A:
        raise ProfileError('From FittingGUI.__init__(), the provided 'ion' does not
        'have an 'A' (transition probability) attribute!')
    if hasattr(ion.A, 'unit') and ion.A.unit != u.Unit('s^{-1}):
        raise ProfileError('From FittingGUI.__init__(), the transition probability,
        'A', from the 'ion' must be in units of 's^{-1}!' if units are present!)
# the instrument broadening is from R = lambda / delta_lambda
# FWHM = 2 * sqrt(2 * log 2) sigma for the Gaussian instrument profile
self.R = obs_resolution

self.sigma_instrument = (ion.wavelength / self.R) / (2 * np.sqrt(2 * np.log(2)))

self.sigma_instrument_squared = self.sigma_instrument.value ** 2

# save parameters for later
self.wavelength = ion.wavelength

self.fvalue = ion.fvalue

self.A = ion.A

# the FWHM of the intrinsic line profile (Lorentzian) is proportional to the
# transition probability (Einstein coeff.) 'A'...
# convert from km s^-1 to wavelength units

self.gamma = (ion.wavelength * (ion.wavelength * ion.A / (2 * np.pi)), to(u.km / u.s) / c.si).value

# the leading constant in the computation of 'W' (per Angstrom per cm^-2)
# self.leading_constant = (m.e.si + c.si) / (np.sqrt(np.pi) + m.si**2 + ion.fvalue)
# self.leading_constant = 1 / (ion.fvalue * ion.wavelength.to(u.A)**2 + np.pi *
# m.emu + m.e.si + c.to(u.km/u.s)**2)).value

# setting 'function' to 'ModifiedVoigt' only makes a change in how the 'Voigt'
# profile is evaluated by using 'self.gamma' instead of self.Params[...]['Gamma']

function = 'ModifiedVoigt'

# --------------- -------------------------------

print('\nWe need to extract the lines from the continuum before we begin the fitting process.

# extract the line, continuum, rms from the spectrum

self.line, self.continuum = Extract(splot, bandwidth=brightness, kind=kind)

self.rms = self.continuum.rms

print('\nNow select the peaks of each line to be fit.\n
print('Initial guesses will be made for each line marked.\n
input('Press <Return> after making your selections ...\n
# grab all the selected points

global selected

def selected(points = np.array([
    [ entry.value for entry in selected['wave'] ],
    [ entry.value for entry in selected['data'] ]
])

# point pairs in ascending order by wavelength

points = points[points[:,0].argsort()]

# domain size of the line and the number of components

self.domainsize = self.line.wave.value[-1] - self.line.wave.value[0]

self.numlines = np.shape(points)[1] - 4

if self.numlines < 1:
raise ProfileError('FittingGUI() expects at least one line to be selected for fitting!')

# final spectral line profile is convolution of the LSP and the
# line profile function. Gaussian on Gaussian, Gaussian on Lorentzian,
# etc... Set the functional form given requested profile function

self.evaluate = self.SetFunction(function)

self.Parameters = 

self.Parameters['L'] + str(line + 1) = self.Parameterize(function, loc)

for line, loc in enumerate(points[::, 2::2, 1])

# grab the actual Figure object and it's axis to keep references

self.splot = splot

self.fig = splot.fig

self.ax = splot.ax

# refresh image, but keep any changes in axis limits

splot.xlim = +splot.ax.get_xlim()

splot.ylim = +splot.ax.get_ylim()

splot.draw()

# bring up plot to make room for sliders

plt.subplots_adjust(bottom = 0.30)

# resample continuum onto line domain

self.continuum = self.continuum.copy()

self.continuum.resample(self.line)
# common domain for plotting, strip units, wavelengths from continuum
self.x = self.continuum.wave.value
self.continuum = self.continuum.data.value

# copy of continuum allows for adjustments
self.y = self.continuum.copy()

# save the mid-level of the continuum to avoid over-calculations
self.continuum_level = self.continuum.mean()

# add plots of each line, keep dictionary of handles
self.Component = {
    line : plt.plot(self.x, self.y - self.Evaluate(self.x, **parameters), 'k-'),
    for line, parameters in self.Params.items():
        }

# add plot of the continuum
self.Continuum = plt.plot(self.x, self.y, 'r-', lw=2)

# add plot of superposition of each component
self.Combination = plt.plot(self.x, self.y - self.Supersition(), 'g-')

# fix the limits on plots
xmin, xmax = splot.ax.get_xlim()
ymin, ymax = splot.ax.get_ylim()
plt.axis([xmin, xmax, ymin, ymax])

# axes for parameter sliders
self.Axis = {
    key : axis = ax , xpos = , ypos = dy , xsize , ysize
    line : plt.axes([0.10, 0.05, + (k+1)* + 0.03, 0.65, 0.02], axisbg = 'white')
    for k, line in enumerate(self.Params.keys()):
        }

# add an axis to make adjustments to the continuum
self.Axis["Continuum"] = plt.axes([0.10, 0.05, 0.65, 0.02], axisbg='white')

# create the sliders for the parameters
self.Slider = {
    }

# Slider 'key' and widget
param = widgets.Slider(
    self.Axis[param], # which axis to put slider on
    param, # name of parameter (and the slider)
    self.Min(param), # set the minimum of the slider
    self.Max(param), # set the maximum of the slider
    valinit = self.Params[param][0] # initial value
    )

# create a slider for each parameter
for param in self.Params.keys():
    self.Slider[param] = widgets.Slider(self.Axis[param], param, 
        valinit = self.Params[param][0], # initial value
        )

# create the slider for the continuum (10% adjustments)
self.Slider["Continuum"] = widgets.Slider(self.Axis["Continuum"], "Continuum",
    0.98 * self.continuum_level, 1.10 * self.continuum_level,
    valinit = self.continuum_level)

# connect sliders to update function
for slider in self.Slider.values():
    slider.on_changed(self.update)

# create axis for radio buttons
self.RadioAxis = plt.axes([0.05, 0.1, 0.1, 0.02], 
    axisbg = 'white', frameon = False)

# create the radio button widget
self.Radio = widgets.RadioButtons(self.RadioAxis, 
    tuple(["L" + str(i+1) for i in range(self.numLines)]), active = 0)

# connect the radio button to it's update function
self.Radio.on_clicked(self.ToggleComponent)

# set current component as the first line
self.current_component = 'L'

# make currently selected line bold
self.Component[s["L"]].set_linewidth(2)

# variable allows for the radio buttons to not screw-up the sliders/graphics
# when switching between lines
self.still = False

# add the initial text for 'N' and 'b' if applicable.
# text is along RPE below the y-axis
if self.has_line_parameters:
def Parameterize(self, function, loc):
    """
    Choose the initial parameters of 'function' given the peak 'loc'.
    """
    if function == 'Voigt':
        return (Gamma): 0.10 * self.domainsize / self.numlines,
        'Sigma': 0.20 * self.domainsize / self.numlines, 'Peak': loc[0],
        'Depth': self.continue(loc[0]).value - loc[1])
    elif function == 'Lorentzian':
        return (FWHM): 0.5 * self.domainsize / self.numlines, 'Peak': loc[0],
        'Depth': self.continue(loc[0]).value - loc[1])
    elif function == 'Gaussian':
        return (FWHM): 0.5 * self.domainsize / self.numlines, 'Peak': loc[0],
        'Depth': self.continue(loc[0]).value - loc[1])
    elif function == 'ModifiedVoigt':
        return (Sigma) = (self.Maximize(Sigma) - self.Minimize(Sigma)) / 2,
        'Peak': loc[0], 'Depth': self.continue(loc[0]).value - loc[1])
    else:
        raise ProfileError('From FittingGUI.Parameterize(), the only '
        'currently implemented functions are the 'Gaussian', 'Lorentzian', and '
        'Voigt' profiles!')

def SetFunction(self, function):
    """
    Return how to 'evaluate' the profile given the 'function'.
    """
    options = {'Voigt': self.__Voigt, 'Lorentzian': self.__Lorentzian,
               'Gaussian': self.__Gaussian, 'ModifiedVoigt': self.__ModifiedVoigt}
    if function not in options:
        raise ProfileError('From FittingGUI.SetFunction(), the only '
        'currently implemented functions are the 'Voigt', 'Lorentzian', and '
        'the 'Gaussian' profiles!!')
    return options[function]

def __Gaussian(self, x, **params):
    """
    A Gaussian profile. See ..Algorithms.Functions.Gaussian
    """
    return Gaussian(x, params['Depth'], params['Peak'], params['FWHM'] / 2.354820045038942)  

def __Lorentzian(self, x, **params):
    """
    A Lorentzian profile. See ..Algorithms.Functions.Lorentzian
    """
    return params['Depth'] * Lorentzian(x, params['Peak'], params['FWHM'])

def __Voigt(self, x, **params):
    """
    The Voigt profile. See ..Algorithms.Functions.Voigt
    """
    return Voigt(x, params['Depth'], params['Peak'], params['Sigma'], params['Gamma'])

def __ModifiedVoigt(self, x, **params):
    """
    This is the Voigt profile, but 'gamma' was already set.
    """
    return Voigt(x, params['Depth'], params['Peak'], params['Sigma'], self.gamma)

def SuperPosition(self):
    """
    Superposition of each line component (blended line)
    """
    combined = np.zeros(np.shape(self.x))
    for parameters in self.Params.values():
        combined += self.Evaluate(self.x, **parameters)
    return combined

def Minimum(self, param):
    """
    Set the lower bound on the 'param'eter for its slider.
    """
    if param == 'Peak':
        return self.x[0]
    elif param == 'FWHM':
        return 1e-6
elif param == 'Depth':
    return 1e-6
elif param == 'Sigma':
    if self.Evaluate != self._ModifiedVoigt:
        return 1e-6
    else:
        return self.sigma_instrument.value
elif param == 'Gamma':
    return 1e-6
else:
    raise ProfileError('From FittingGUI: Minimum() \( \{\} \) is not currently implemented as a parameter to set the minimum.' 'for!'.format(param))

def Maximum(self, param):
    '''
    Set the upper bound on the 'param'eter for its slider.
    '''
    if param == 'Peak':
        return self.x[-1]
    elif param == 'FWHM':
        return 0.5 * self.domainsize
    elif param == 'Depth':
        return 1.5 * self.continuum.max()
    elif param == 'Sigma':
        return 0.9 * self.domainsize / self.numlines
    elif param == 'Gamma':
        return 0.9 * self.domainsize / self.numlines
    else:
        raise ProfileError('From FittingGUI: Maximum() \( \{\} \) is not currently implemented as a parameter to set the maximum.' 'for!'.format(param))

def Update(self, val):
    '''
    Cycle thru Sliders and update Parameter dictionary. Re-draw graphs.
    '''
    line = self.current_component
    for parameter, slider in self.slider.items():
        if parameter == 'Continuum':
            self.y = self.continuum + (slider.val - self.continuum_level)
        else:
            self.Params[line][parameter] = slider.val
    # update the appropriate graph data, based on new parameters
    for line, parameters in self.Params.items():
        self.Component[line].set_ydata(self.y - self.Evaluate(self.x, **parameters))
    self.update_graph
    self.update_cont
    self.update_comb
    self.update_preview()

def ToggleComponent(self, label):
    '''
    Toggle function for the radio buttons. Switch between line components
    'L1', 'L2', etc. Update the sliders to reflect changing parameters.
    '''
    self.current_component = label
    if self.has_line_parameters:
        self.Update_Preview()

        for line in self.Component.keys():
            if line == label:
                self.Component[line].set_linewidth(2)
            else:
```python
self.Component[line].set_linewidth(1)

# update the sliders to reflect the current component
for parameter, slider in self.Slider.items():
    if parameter != 'Continuum':
        slider.set_val(self.Params[label][parameter])

# update the displayed 'b', 'N' and 'z' if present
if self_has_line_parameters:
    self.Update_Preview()

# give control back to sliders
self.stall = False

# push updates to graph
self.fig.canvas.draw_idle()

def solve_b(self):
    ...
    Given the current line component, solve for the broadening parameter, 'b',
    given the instrument profile and the observed broadening.
    ...
    # b = sqrt(2 * sigma_v)
    sigma_obs = self.Params[self.current_component]['Sigma']
    sigma_v = np.sqrt(sigma_obs**2 - self.sigma_instrument_squared) / self.wavelength
    return b, np.sqrt(c * sigma_v / self.wavelength)

    # return c * np.sqrt(1 + (c * sigma_v / self.wavelength))
    # To return the expected wavelength...
    # return const * c

def solve_tau_P(self):
    ...
    Solve for the apparent optical depth at line center.
    ...
    line_data = self.Component[self.current_component].get_ydata()
    line_wave = self.Component[self.current_component].get_xdata()
    line_center = line_data.argmin()
    return np.log(line.y[line_center] / line_data[line_center])

def solve_W(self):
    ...
    Solve for the equivalent width of the currently selected line.
    ...
    line_data = self.Component[self.current_component].get_ydata()
    line_wave = self.Component[self.current_component].get_xdata()
    return np.exp(-tau) * self.wavelength

    # equivalent width (dimensionless)
    # W = self.solve_W() / self.wavelength

    # M = self.solve_M() / self.wavelength

    # m_e c^2 / pi c^2 \approx 1.13 \text{ cm}^{-1} ??? How is that?
    const = (1.1312 / u.cm) / (self.fvalue * self.wavelength.to(u.cm))
    return const * W

def solve_v(self):
    ...
    Solve for the velocity of the line given the expected wavelength.
    The result is returned in km s^{-1}.
    ...
    line_center = self.Params[self.current_component]['Peak'] * self.wavelength
    return c.to(u.km / u.second) * (line_center - self.wavelength) / self.wavelength

def Update_Preview(self):
    ...
    Re-compute the 'b', 'N', and 'v' values, update the text in the plot.
    ...
    if self.b, self.N, self.v = self.solve_b(), self.solve_N(), self.solve_v():
        self.preview.set_text('b: (%10.4f) km s^{-1} b: (%10.4f) cm^{-2}
            tau_b: (%10.4f)', format(self.solve_v().value, self.solve_b().value, self.solve_W().value, self.solve_N().value, self.solve_tau_P())

def GetSpectra(self):
    ...
    Return the current graphs as spectrum objects.
    ...
    return [self.Component[line].get_ydata() * self.line.data_unit, self.x * self.line.wave/unit] for line in self.component.keys()

def GetContinuum(self):
    ...
    Return the current graph of the continuum.
```

continuum = Spectrum(self.y * self.line.data.unit, self.x * self.line.wave.unit)
continuum.rs = self.rs
return continuum

def kill(self):
    """
    Close the plot to destroy widgets and restore the 'splot' to its original state.
    """
    plt.close(self.fig)
    self.splot.fig = plt.figure("Spectral Plot (SlipY)")
    self.splot.ax = self.splot.fig.add_subplot(111)
    self.splot.draw()

    def Multifit(splot, error=None, obs=None, ion=None, function='Voigt', measure=True,
                 boost=None, **kwargs):
        """
        The Multifit routine takes a 'splot' figure (type SPlot) and allows the
user to interactively fit line profiles. 'splot' may optionally be of type
Spectrum, in which case a SPlot figure will be created for you. This function
creates a fittingGUI object (not documented here) which uses the Profile.Extract()
routine first ("kwars" are passed to this function). As in the Extract routine, the user
will select four points that mark the boundaries of the line (blended or otherwise)
and some surrounding continuum to sample. The KernelFit(
...Algorithms KernelFit) routine
is applied with the user specified 'bandwidth' to smooth the noise out of the continuum
and interpolate, of type 'kind', across the line gap. After this, the user is asked to
further select points marking the peaks of the line(s). The number of selection points
indicates the number of lines to be fit. If you wish to deblend two or more lines, select
all suspect locations. These are not only used to determine the number of lines to fit
but to make initial guesses at the parameters for each line and determine reasonable scales
for the sliders.
        """
        After these selections, a slider for each parameter appears along with radio buttons for
each line. The user can interactively adjusts the parameter(s) for a given line by
selecting it (the radio buttons) and dragging a slider. The parameters available to
adjust depend on the 'function' argument. There are currently three available line shapes:
'Gaussian', 'Lorentzian', and 'Voigt'. See...Algorithms.functions for information on
these.
        """
        Each line is represented by a black, dashed curve in the plot. The currently selected line
is bold. The combined (i.e., blended) line is plotted as a solid green curve.
        """
        If an Observatory with a 'resolution' is provided via the 'obs' argument then the
thermal broadening parameter 'b' can be computed (and displayed). This is only applicable
with either a 'Gaussian' or 'Voigt' profile. Further, an 'ion' needs to be provided in
this scenario (type...Data.Atomic.Ion) with an oscillator strength (fvalue), transition
probability (f) and wavelength as members. With these data, we can interactively show
the broadening parameter, the velocity, and the apparent column density during the fitting
process. Note: the 'gamma' slider will disappear in this context because it has already
been determined via the transition probability.
        """
        Each slider is labeled on the left side with the name of the parameter it controls. At
the right of each slider is the current value of that parameter. The radio buttons are
labeled 'L1', 'L2', etc. for each line.
        """
        This routine returns both a list of Spectrum 'lines' as well as the 'continuum' that was
fit using fittingGUI. This function acts as both a single line fitting tool as well
as a deblending tool. By default, the final parameterizations are attached to each spectrum
as a dictionary, '.parameters'.'
        """
        If 'measure' is passed as True (the default), the equivalent width is computed and attached
to each spectrum ('L1', 'L2', etc...) and can be accessed with 'W'. If 'b' and/or 'N' are
available for each line these will be attached as well, accessible as '.b' and '.N'
respectively. This functionality can be suppressed by setting 'measure' to False. 'boost'
and 'error' are passed to EquivalentWidth() and ColumnDensity().
        """
# running the user interface
    gui = fittingGUI(splot, obs=obs, ion=ion, function=function, **kwargs)
    input("(In press <return> when you are finished fitting lines ...)

# attach the parameter dictionaries to each spectrum
    lines = gui.GetSpectra()
    for a, parameterization in enumerate(sorted(gui.Params.keys())):
        lines[a].parameters = gui.Params[parameterization]

# attach the continuum to the line list
    continuum = gui.GetContinuum()
    if not measure:
        return lines, continuum

    if gui.has_line_parameters:
        for line, key in zip(lines, sorted(gui.Params.keys())):
            # set the line to 'L1', 'L2', etc...
            gui.current_component = key
        # attach the measured quantities
        notes = "Measurement made using Profile.Multifit() from SlipY"
        line.W = EquivalentWidth(line, continuum, error=error, boost=boost)
        line.N = ColumnDensity(line, continuum, ion, error=error, boost=boost)

162
line.b = Measurement(gui.solve_b(), name='Doppler Broadening Parameter', notes=notes)
line.v = Measurement(gui.solve_v(), name='Velocity', notes=notes)
gui.kill()
return lines, continuum
C.17  .. SLiPy : Simbad

```python
# Copyright (C) Geoffrey Lentner 2015. All Rights Reserved.
# This module allows the user to query the SIMBAD astronomical database from
# inside Python or shell commands/scripts.

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 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.,
'alpha lea', 'HD 123475', 'del cyg', etc ... ) if the name is two parts
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.

```
'dtype': float, # output datatype
def __init__(self, identifier, criteria, default=\float, **kwargs):
    Initiate query to SIMBAD database.
    
    # check argument types
    if type(identifier) is not str or type(criteria) is not str:
        raise SimbadError('Simbad.Query function expects str.
    ’types for arguments.’)
    try:
        # keyword argument options for Query
        self.options = Options( **kwargs,
            {'parse' : True # parse SIMBAD return file
             'full' : False # return full line of info
             'dtype' : default, # convert return data
             'is_main': False # called from Main()
        )
        # assignments
        self.parse = self.options('parse')
        self.full = self.options('full')
        self.dtype = self.options('dtype')
        self.is_main = self.options('is_main')

        # query SIMBAD database
        with urlopen(Script(identifier, criteria )) as response:
            self.data = str( response.read().decode('utf-8') ).strip()
    except OptionsError as err:
        print(err.error)
        raise SimbadError('Simbad.Query was not constructed'
    ‘for {}”.format(identifier))
    except URLError as error:
        raise SimbadError('failed to contact SIMBAD database for'
    ‘{}”.format(identifier) )

    if 'not found' in self.data or 'error' in self.data:
        raise SimbadError('{}' could not be resolved by SIMBAD.'
    .format(identifier))
    if self.parse:
        # pre-parse operation common to all criteria
        self.data = self.data.split('data')[-1]

def __call__(self):
    Retrieve data from Query
    return self.data
def Position( identifier, **kwargs ):
    Handle to the Query class with criteria="SC08(d,C)"
    query = Query( identifier, 'SC08(d,C)', **kwargs )

    if query.full:
        query.data = query.data.split(\n)[\-\-]
    elif query.parse:
        # extract relevant data
        query.data = query.data.split(\-\-).split(\‘
    if len( query.data ) == 1:
        # dec had " not "
        query.data = query.data[\-\-].split(\‘
    query.data[\-\-] = query.data[\-\-] + query.data[\-\-]
    # return formatted data type
    query.data = [ query.dtype(pos) * u.degree for pos in query.data ]

    if query.is_main:
        if query.full or not query.parse:
            print( query() )
        else:
            print( (\0.2f) \(1.2f)’.format(*query())
    else: return query.data
def Distance( identifier, **kwargs ):
    Handle to the Query class with criteria="SPLX"
    query = Query( identifier, 'SPLX', **kwargs )

    if query.full:
        query.data = query.data.split(\n)[\-\-]
    elif query.parse:
        # extract relevant data
        query.data = query.data.split(\-\-).split(\‘
    if len( query.data ) == 1:
        # dec had " not "
        query.data = query.data[\-\-].split(\‘
    query.data[\-\-] = query.data[\-\-] + query.data[\-\-]
    # return formatted data type
    query.data = [ query.dtype(pos) * u.degree for pos in query.data ]

    if query.is_main:
        if query.full or not query.parse:
            print( query() )
        else:
            print( (\0.2f) \(1.2f)’.format(*query())
    else: return query.data
if query.full:
    data = query.data.split(\n)[\-1]

elif query.parse:
    # extract relevant data
    data = query.data.split()
    if data[0] == '\n':
        # nothing found!
        raise SimbadError('No distance found for {}' .format(identifier))
    try:
        # convert milli-arcseconds to parsecs
        result = u.pc / ( query.dtype(data[0]) / 1000.0 )
    except ValueError as err:
        raise SimbadError('Use a numeric type for Simbad.Distance!')
    if data[2][0] == '[' and data[2][-1] == ']':
        # TODO: understand SIMBAD error format
        # an uncertainty was given by SIMBAD
        # uncertainty = u.pc * 0.001 * query.dtype(data[0]) * query.dtype(data[2][1:-1]) / ( [0.001 * query.dtype(data[0])] )**2
        # uncertainty = result * query.dtype(data[2][1:-1])
        uncertainty = None
    else:
        uncertainty = None
else:
    data = query.data
if query.is_main:
    if query.full or not query.parse:
        print( data )
    else:
        print('([@:.2f])' .format( data ) )
elif not query.parse:
    return data
else:
    return result

# Measurement(result, error=uncertainty, name='Distance',
# notes='Retrieved from SIMBAD database for "{}"' .format(identifier))

def sptype(identifier, **kwargs):
    """
    Handle to the query class with criteria='SP'.
    """
    query = Query(identifier, 'SP', **kwargs)
    if query.full:
        # return last full line of query
        query.data = query.data.split(\n)[\-1]
    elif query.parse:
        # extract relevant data
        query.data = query.data.split()[1]
    if query.is_main:
        print( query() )
    else:
        return query()

def idlist(identifier, **kwargs):
    """
    Handle to the query class with criteria='IDLIST'.
    With 'parse' = True, return a list of alternate IDs for
    the 'identifier' provided.
    """
    query = Query(identifier, 'IDLIST', **kwargs)
    if query.parse:
        # extract relevant data
        query.data = query.data.split(' '')[\-1].strip().split(\n)
    if query.is_main:
        for line in query.data:
            print(line)
    else:
        return query()

def main( clargs ):
    """
    Main Function. See _doc__ for details.
    """
    if len(clargs) < 2:
        # show usage
        print( __doc__ )
        return 
    # Available functions for execution
try:
    # Parse command line arguments
    function, args, kwargs = Parse( clargs[1:] )
    if not args and not kwargs or args[0] == 'help':
        # show function usage
        print( executable[function].__doc__ )
        return 0
    # run execution
    for identifier in args:
        executable[function] ( identifier, is_main=True, **kwargs )
    return 0
except CommandError as err:
    print( ' --> CommandError: ', err.msg )
    return 1
except KeyError as key:
    print( ' --> {} was not a recognized function.'.format(key) )
    return 1
except SimbadError as err:
    # don't let uncaught self exception pass if from main.
    print( ' --> SimbadError: ', err.msg )
    return 1
except Exception as err:
    print( ' --> Unrecognized error with query for {}' .format(args[0]) )
    print( ' --> Exception: {}' .format(err) )
    return 1
if __name__ == '__main__':
    # Main return 0 or 1
    exit( Main( argv ) )
C.18  SLiPy . Telluric

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

Telluric - Corrections for atmospheric absorption 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 is
        needed to be provided for Telluric.Correct().')

    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 at least one
        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
```

168
```python
best = None
for cal in calibration:
    # best horizontal pixel shift
    shift = Xcorr( spectrum, cal, lag=lag)
    # build matrices with identical rows (len=npix-shift)
    if shift < 0:
        calmatrix = np.tile(cal.data[:shift], (trials, 1))
        objmatrix = np.tile(spectrum.data[-shift:], (trials, 1))
    elif shift > 0:
        calmatrix = np.tile(cal.data[shift:], (trials, 1))
        objmatrix = np.tile(spectrum.data[-shift:], (trials, 1))
    else:
        calmatrix = np.tile(cal.data, (trials, 1))
        objmatrix = np.tile(spectrum.data, (trials, 1))
    # amplitude matrix has identical columns
    ampmatrix = np.tile(amp, (size,1)).T
    # remove units for dimensionless operations
    calmatrix = calmatrix.value
    objmatrix = objmatrix.value
    # flip arrays for amplification
    diff = objmatrix - (1 - (1 - calmatrix) * ampmatrix)
    # compute the RMS for each trial
    rmsvector = np.sqrt(np.sum(diff**2, axis=1) / size)
    if not best:
        # if first pass, assign to 'best'
        best = (rmsvector.min(), rmsvector.argmax(), shift, cal)
    elif rmsvector.min() < best[0]:
        # this fit is better, save it to 'best'
        best = (rmsvector.min(), rmsvector.argmax(), shift, cal)
    # results of calibration fitting
    index = best[1] # amplitude
    shift = best[2] # Xcorr
    cal = best[3] # which calibration spectrum
    # we can't update an attribute...
update = spectrum.data.value
# divide spectrum
if shift < 0:
    update[+shift] /= 1 - (1 - cal.data[shift].value) * amp[index]
elif shift > 0:
    update[-shift] /= 1 - (1 - cal.data[shift].value) * amp[index]
else:
    update /= 1 - (1 - cal.data.value) * amp[index]
# re-incorporate units
spectrum.data = update * u.Unit(spectrum.yunits)
```

from astropy.io import fits as pyfits
from astropy.constants import c
from astropy import units as u
from .. import SlipyError
from .. import helcorr
from .. import observatory
from .. import spectrum
from .. import options
from .. import monitor

class VelocityError(SlipyError):
    """Exception specific to the velocity module""
    pass

def headerinfo(handler):
    """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
    universal time of observation and the right ascension and declination
    of the target.

    try:
        with pyfits.open(handler) 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 = '\n'.format(year, month, day, UT, ra, dec)
    except IOError as error:
        raise VelocityError('Failure in {} from headerinfo'.format(handler))
    return info

def irafinput(files, **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).
    
    kwars = {
        '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'
    }
    try:
        # dictionary of options
        options = Options(kwars,
                          {'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 = ffind 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' observatory 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 observatory 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, 'unit'):
                raise VelocityError('From Heliocorrect(), in spectrum (), "ra" doesn't have units!'.format(a))
            if not hasattr(spectrum, 'unit'):
                raise VelocityError('From Heliocorrect(), in spectrum (), "dec" doesn't have units!'.format(a))
            if not hasattr(spectrum, '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 'astropy/...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.degree).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)
            if verbose:
                display.progress(a, len(spectra))
        # finalize progress bar (erase)
        if verbose:
            display.complete()
except OptionsError as err:
    print( 'Failed to perform HeliCorrect() task.' )
except DisplayError as err:
    print( 'Exception from Display.Monitor in HeliCorrect()' )

def BaryCorrect( obs, *spectra, **kwargs ):
    """
    Perform barycentric velocity corrects on 'spectra' based on
    'observatory' 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( 'HeliCorrect() 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( 'HeliCorrect expects the observatory to have all three: latitude, longitude, and altitude attributes.' )
        # check 'spectra' arguments
        for aL spectrum in enumerate(spectra):
            if type(spectrum) is not Spectrum:
                raise VelocityError( 'HeliCorrect() 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 HeliCorrect()' )
            if verbose:
                display = Monitor()
                print( 'Running HeliCorrect on {} spectra ...
'.format( len(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 then.
        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.second),value)
        ){km s^-1} = u.Unit('~')
        # apply correction to wave vector.
        del(lambda) / Lambda = del(V) / c
        spectrum.wave = spectrum.wave * hcorr / c
        # show progress if desired
        if verbose: display.progress(a, len(spectra))
        # finalize progress bar (erase)
        if verbose: display.complete()
    except OptionsError as err:
        print( 'Failed to perform HeliCorrect() task.' )
    except DisplayError as err:
        print( 'Exception from Display.Monitor in HeliCorrect()' )
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)