Spectral Fitting of Brown Dwarf Binaries Using C++

Katrina Wright

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

Dr. Denise C. Stephens, Advisor

Department of Physics and Astronomy

Brigham Young University

April 2015

ABSTRACT

Spectral Fitting of Brown Dwarf Binaries Using C++

Katrina Wright

Department of Physics and Astronomy

Bachelor of Science

Brown dwarf spectra may be the result of binary systems. Because of this, two models must be combined and the models must be rebinned to the spectra to perform a best fit. Previous work has been in a less useful language and less general. This thesis describes the process of making code in C++ which can be used to fit any spectrum to the models.

Keywords: brown dwarf binary, computing

Table of Contents

Chapter 1

Introduction

1.1 Intro

Brown Dwarfs are celestial bodies which are too big to be considered gas giants, but too small to be considered stars. They are present throughout the universe, and are often in binary systems. My research focuses on fitting models to spectra obtained from brown dwarf systems to find what parameters the brown dwarfs best fit.

In the rest of Chapter 1, I describe brown dwarfs and the models I use, as well as some of the previous work that has been done on this topic. In Chapter 2, I discuss the goals and problems to which my research applies. In Chapter 3, I describe the code I wrote and discuss its applications and the future for this research.

1.2 Brown Dwarfs

Brown Dwarfs are of a size and mass in between stars and planets. They are larger than gas giants like Jupiter, and give off enough heat that they emit light. However, they are not quite larger enough for the fusion which powers stars and causes them to continue to give off heat. Over time, brown dwarfs, unlike stars, cool.

There are three classes of brown dwarfs: L, T, and Y. Class L brown dwarfs are the hottest and youngest, and just below class M stars in temperature. Class T are the next coolest. Class Y brown dwarfs are the coolest and oldest brown dwarfs and are difficult to detect and classify. Over time, a brown dwarf can cool so that is changes its class.

1.3 Models

Raw atmosphere models were used which have four parameters: effective temperature (T_{eff}), gravity (log g), cloud parameter (f_{sed}), and vertical mixing (K_{zz}) (Saumon & Marley 2008). The effective temperatures range from 500K to 2400K. The gravities range from 4 to 5.5 in cgs units. The cloud parameter shows how dense the clouds are, and ranges from 1 (very cloudy) to 5 (less cloudy), with another possible value of nc, which indicates no clouds. The vertical mixing has to do with how much NH_3 and CH_4 converts to N_2 and CO in the atmosphere. These models all have a very high resolution.

1.4 Previous Work

Previously, Taran Esplin, a student at BYU, studied 2M 0559 and performed a similar analysis to it as what I am doing (Esplin 2010). He wrote code which attempted a fit for the spectrum of 2M 0559 to the models provided. However, his only applied to that system. With my work, I wanted to improve on what was previously done to expand the use to more than just one spectrum.

Chapter 2

Problem

2.1 The Spectra

The brown dwarfs we are looking at are unresolved binaries. We get a spectrum, but it is not just for one object, but for two. In order to learn about them, we need to take two models for brown dwarfs and combine them and fit to the spectrum.

The models and the spectra are not originally in the same units: the model frequencies are in W/m^2/um, but the units on the spectra that are obtained are in erg/cm^2/s/Hz. Because of this, a conversion factor is needed. This requires two steps. First, it is necessary to convert from ergs/cm^2/s to W/m^2, a factor of 1000. Next, it is necessary to convert units from frequency to wavelength, which requires multiplying by c/w^2, where w is the wavelength of the spectrum at that flux. In order to do this conversion overall in one step, each flux must be multiplied by a factor of 2.99792458e11/(w^2).

This conversion is accomplished in the Spectrum class of my code.

2.2 The Models

Unfortunately, the models which we have to fit the spectra to are at a very high resolution. They are very noisy and hard to fit anything to. There is data for many more wavelengths that is useful. In order to use the models, they have to be at the same resolution as the spectra. Because of this, the models must be rebinned so they have fewer data points, and so that those data points match the spectrum they are fit to. I wrote code to take the models and resolve them to the data points in order to fix this problem.

The decrease in resolution of the models is accomplished in the Converter class of my code.

2.3 Previous Code

Previously, Taran Esplin wrote similar code in MatLab to fit a spectrum for a single system. However, that was only for one binary brown dwarf and was not applicable to many different spectra. My code improves on this because it can be used to fit any spectrum to the models.

In addition, Esplin's code was in MatLab, which is useless for Dr. Stephens since MatLab is rarely used in astronomy. My code is written in C++ instead so that it can be used and understood my more people who have interest in it.

Chapter 3

Solution

3.1 The Code

3.1.1 Process

I wrote code in C++ to do the rebinning of all the models. Now the code can be used to fit spectra. The code is in Appendix A.

In order to write my code, I referred back to the previous work done by Taran as well as some similar code written in Fortran in order to understand the splint and spline methods. I made two classes: Spectrum (Appendix A.1) and Converter (Appendix A2). Spectrum is the internal mechanism that keeps track of a spectrum or model. Converter performs the regrid so the spectrum can be fit to the model.

3.1.2 Spectrum

The spectrum class reads in data from a file about a spectrum or a model the spectrum is to be fitted to. It stores the wavelength and the flux of the spectrum so that it can be read. This class assumes that the data is in a format with each wavelength and flux pair separated on a line, and it skips lines starting with '#' as comments. It also has the function to convert the units of a spectrum as described in 2.1. The conversion must be applied to the spectrum within the function where it is used to work correctly.

The spectrum class also has the functionality to print a spectrum to a new file, so one a model is regridded, the new model spectrum can be saved.

3.1.3 Converter

The Converter class uses locate, splint, and spline from numerical recipes to regrid a model to a spectrum.

The locate function is used to find where in the model data the points for the spectrum start and end. These endpoints are used to find parameters for the spline and splint functions. Using these endpoints, the derivatives of the model at the endpoints are calculated. These derivatives are put in with the spectrum information in the spline function to interpolate in the spectrum.

Next the splint function is used to get the flux values at the interpolated values. This function takes the spectrum data, the results from spline, and each individual wavelength in the model. This produces the new fluxes for the model.

Altogether, the spline and splint functions regrid the model. At the end, it is at the same resolution and over the same range as the spectrum.

3.2 Further Work

This code will be able to be used much in the future for finding and analyzing binary brown dwarf systems. It can be used to fit the spectrum of any system to the models. For example, it would be useful in a continuation of Elora Salway's analysis of 2M 0559, a binary brown dwarf candidate, in order to fit its spectrum to the models (Salway 2015).

The function to convolve models still needs to be implemented properly before it is completely ready to fit binary brown dwarf spectra to the models. However, this code is likely to be used for years to come to analyze data from many brown dwarfs since it has the capability to be applicable to all of them.

Appendix A

C++ Code

A.1 Spectrum Class

Defines a spectrum or model with a set of wavelengths and corresponding fluxes and provides functions to convert the fluxes to the correct units, reverse the order of the information, and return information about the spectrum.

A.1.1 Spectrum Header

```
#include <iostream>
#include <string>
#include <vector>

#ifndef _SPECTRUM_H
#define _SPECTRUM_H

class Spectrum
{
private:
	std::vector<float> m_wavelengthList;
	std::vector<float> m_fluxList;
	std::string m_fluxType;

public:
	Spectrum(){}
	Spectrum(std::string filename);
```

```
        Spectrum(std::vector<float> wavelengthList, std::vector<float>
                fluxList, std::string fluxType);
        ~Spectrum(){}
        void printToFile(std::string filename);
        void convertToWperMsquaredperMicron();
        std::vector<float> getWavelengthList();
        std::vector<float> getFluxList();
        std::string getFluxType();
        void reverse();
};

#endif
```

A.1.2 Spectrum Code

```
#include "Spectrum.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <algorithm>

Spectrum::Spectrum(std::string filename)
{
std::ifstream spec_file(filename.c_str());
std::string line;
if (spec_file.is_open())
{
        while (getline(spec_file,line))
        {
                if (line.substr(0,1)!="#")
                {
                        float wavelength,flux;
                        std::istringstream is(line);
                        is >> wavelength >> flux;
                        m_wavelengthList.push_back(wavelength);
                        m_fluxList.push_back(flux);
                }
        }
        spec_file.close();
}
m_fluxType = "erg/cm^2/s/Hz";
}

Spectrum::Spectrum(std::vector<float> wavelengthList,
        std::vector<float> fluxList, std::string fluxType)
{
m_wavelengthList = wavelengthList;
```

```
m_fluxList = fluxList;
m_fluxType = fluxType;
}

void Spectrum::reverse()
{
std::reverse(m_wavelengthList.begin(), m_wavelengthList.end());
std::reverse(m_fluxList.begin(), m_fluxList.end());
}

void Spectrum::printToFile(std::string filename)
{
std::ofstream spec_file;
spec_file.open(filename.c_str());
std::string line;
for (int i=0; i<m_fluxList.size(); i++){
       std::ostringstream wtostr;
       std::ostringstream ftostr;
       float wavelength = m_wavelengthList.at(i);
       float flux = m_fluxList.at(i);
       wtostr << wavelength;
       ftostr << flux;
       line = wtostr.str() + "    " + ftostr.str() + "\n";
       spec_file << line;
}
spec_file.close();
}

void Spectrum::convertToWperMsquaredperMicron()
{
float Fnu;
float w;
if ("erg/cm^2/s/Hz" == m_fluxType)
{
       for(int i=0; i < m_fluxList.size(); i++)
       {
              Fnu = m_fluxList.at(i);
              w = m_wavelengthList.at(i);
              m_fluxList.at(i) = Fnu*2.99792458e11/(w*w);
       }
}
m_fluxType = "W/m^2/um";
}

std::vector<float> Spectrum::getWavelengthList()
{
return m_wavelengthList;
}
```

```
std::vector<float> Spectrum::getFluxList()
{
return m_fluxList;
}

std::string Spectrum::getFluxType()
{
return m_fluxType;
}
```

A.2 Converter Class

Performs regrid function on a spectrum

A.2.1 Converter Header

```
#include "Spectrum.h"
#include <vector>

#ifndef _CONVERTER_H
#define _CONVERTER_H

class Converter
{
public:
	Converter(){}
	~Converter(){}
	Spectrum regrid(Spectrum spec, Spectrum model);
};

#endif
```

A.2.2 Converter Code

```
#include "Converter.h"
extern "C" {
#include "nr.h"
}

void locate(float xx[], unsigned long n, float x, unsigned long *j);
```

```
Spectrum Converter::regrid(Spectrum spec, Spectrum model)
{
//     float* modlam = &(model.getWavelengthList()[0]);
//     float* speclam = &(spec.getWavelengthList()[0]);
//     float* modflux = &(model.getFluxList()[0]);
//     float* specflux = &(spec.getFluxList()[0]);

       // Not sure why the above wasn't working but this gets the
correct
       // pointer to call locate with - Tom
       std::vector<float> t1 = model.getWavelengthList();
       float* modlam = &t1[0];
       std::vector<float> t2 = spec.getWavelengthList();
       float* speclam = &t2[0];
       std::vector<float> t3 = model.getFluxList();
       float* modflux = &t3[0];
       std::vector<float> t4 = spec.getFluxList();
       float* specflux = &t4[0];

       unsigned long nmod = model.getWavelengthList().size();
       unsigned long nspec = spec.getWavelengthList().size();

       unsigned long nbeg = -1;
       unsigned long nend = -1;
       locate(speclam, nmod, modlam[0], &nbeg);
       locate(modlam, nmod, speclam[nspec-1], &nend);
       float d = (modflux[nbeg+1]-modflux[nbeg])/(modlam[nbeg+1]-
              modlam[nbeg]);
       float e = (modflux[nend+1]-modflux[nend])/(modlam[nend+1]-
              modlam[nend]);

       float ysec[nspec]; // needed to make this big enough to return
              everything
       spline(speclam, specflux, nspec, d, e, ysec);
       float y = 0;
       std::vector<float> finalFlux;
       for (unsigned int i = 0; i < model.getWavelengthList().size();
              i++){
              splint(speclam, specflux, ysec, nspec, modlam[i], &y);
              finalFlux.push_back(y);
       }

       Spectrum newSpec = Spectrum(model.getWavelengthList(),
              finalFlux, "erg/cm^2/s/Hz");

       return newSpec;
}
```

A.3 Main Function

Code that takes a spectrum and a model and returns a new model regridded to the spectrum.

```
#include <fstream>
#include "Spectrum.h"
#include "Converter.h"

int main(int argc, char* argv[]) {
	if (argc < 4){
		std::cout << "Please include input and output filenames
			and model to regrid to.\n";
		return 1;
	}
	std::string infile(argv[1]), outfile(argv[2]),
		modelfile(argv[3]);
	Spectrum spec = Spectrum(infile);
	spec.reverse();
	spec.convertToWperMsquaredperMicron();

	Converter conv;
	Spectrum model = Spectrum(modelfile);
	Spectrum regrid = conv.regrid(spec, model);

	regrid.printToFile(outfile);

	return 0;
}
```

Bibliography

Esplin, T. L. 2010, Statistical Spectral Fitting of the Brown Dwarf Binary System 2M0559, Bachelors Thesis (Brigham Young University)

Salway, E. 2015, In Preparation, Bachelors Thesis (Brigham Young University)

Saumon, D., & Marley, M. M. 2008, The evolution of L and T dwarfs in color-magnitude diagrams, ApJ, 689, 1327–1344

Index