Megan's Guide to Brown Dwarf Spectroscopic Model Fitting with **MASTIFF**

Megan Tannock

September 11, 2022

Please report any errors, mistakes, or code bugs to me at mtannock@uwo.ca

Contents

1 Introduction

1.1 What Are We Doing, and Why Are We Doing It?

This guide is intended to teach you how to use my spectroscopic model fitting tools, **Mastiff**, which I developed for the model fitting in Tannock et al. (2021) and Tannock et al. (2022). If you use these codes, please cite these two works!

Model photospheres are computed using complex physics and chemistry, and utilise detailed experimental and theoretical line lists. Such models can be used to generate model spectra, and these models are becoming increasingly accurate and well-matched to observed spectra. Models can be used to determine the physical parameters of brown dwarfs based on their spectra, and by shifting and broadening the models we can determine the radial velocities (RVs) and projected rotation velocities ($v \sin i$ s) of brown dwarfs.

Some of the publicly available models include the suite of models based on the PHOENIX code, such as BT-Settl and AMES-Cond (Allard et al., 2001, 2012). Another family of models are based on the Ackerman & Marley (2001) cloud model, and include the Saumon & Marley 2008 (herein SM08), Morley et al. 2012 (herein Morley), Sonora Bobcat (Marley et al. 2021; herein Sonora Bobcat), and Sonora Bobcat Alt A (Tannock et al. 2022, Hood et al. in prep.; herein Bobcat Alt A) models. Please see Section 1.3.3 of my PhD thesis[1] for a complete description of these models and their history.

These models are available on grids of effective temperature ($T_{\rm eff}$), surface gravity ($\log g$), and for some models, metallicity and sedimentation efficiency ($f_{\rm sed}$). Sedimentation efficiency is a parameter that describes the clouds in the model. I additionally add a grid of RV and $v \sin i$ to the models.

My approach to the model fitting is one of brute-force, where I compare every possible combination of the above parameters to an observed spectrum, and calculate the chi square statistic (χ^2). Every value is saved to an output file, allowing the user to identify the best fitting model, and to compute whatever statistics they desire.

The basic steps to fitting any model with any parameters to an observed spectrum are as follows:

1. Read in the model and data
2. Broaden the model to the desired $v \sin i$ by convolving the spectrum with a kernel
3. Shift the model to the desired RV
4. Resample the wavelength scale of the model to that of the data
5. Normalize the data and model to peak at unity
6. Compute the chi square statistic (χ^2)

There are many nuances and details involved in this process, and in this guide I will take you through the entire process, as it is laid out in my model fitting tools.

1.2 What is MASTIFF?

All good software needs a good name. I have decided to call the suite of IDL codes that perform the fitting **Mastiff** in honour of my dog, Dino (see Figure 1). This is a brute-force approach, and mastiffs are very strong. And if you really want it to be a contrived astronomy acronym, I guess it can stand for 'Megan's Astronomy Spectra Toolkit Is For Fitting.'

2 The Codes of MASTIFF

My suite of model fitting tools, herein **Mastiff**, are written in the Interactive Data Language (IDL). There is a **main code**, `fitmodelgrid_final.pro`, which performs the steps of the fitting in order by calling a series of functions. This main code has many customizable parameters and features, which are all set in a configuration file.

[1] https://ir.lib.uwo.ca/etd/8354/

Figure 1: Dino, a Very Good Boy (VGB).

The **Mastiff** codes and functions are named as follows, in alphabetical order:

- `broadenforvsini_final.pro`
- `chisquarestats_final.pro`
- `chopregion_final.pro`
- `contiuumdiv_polynomial_final.pro`
- `fitmodelgrid_final.pro`
- `getmodelnames_final.pro`
- `getmodelparameters_final.pro`
- `getregions_final.pro`
- `maketelluricmask_final.pro`
- `readconfigfile_final.pro`
- `readdata_final.pro`
- `readmodels_final.pro`
- `resamplemodel_final.pro`
- `savebestmodel_final.pro`
- `scalemodelanddata_final.pro`

Each of these codes starts with detailed documentation on how to call the code or function, and describes all of the parameters, but I will take you through these codes in the order they appear in `fitmodelgrid_final.pro`.

Mastiff is available on Github[2] in the repository `data1/Mastiff_ModelFittingTools`. This repository contains all of the custom IDL codes, a README file, a copy of this PDF, and two sets of sample data for you to try. The first sample data set in the directory `ExampleFitting1/` is the data set we will use in this Guide.

The model fitting also makes use of the IDL Astronomy User's Library[3].

[2] https://github.com/megantannock/Mastiff_ModelFittingTools
[3] https://idlastro.gsfc.nasa.gov/

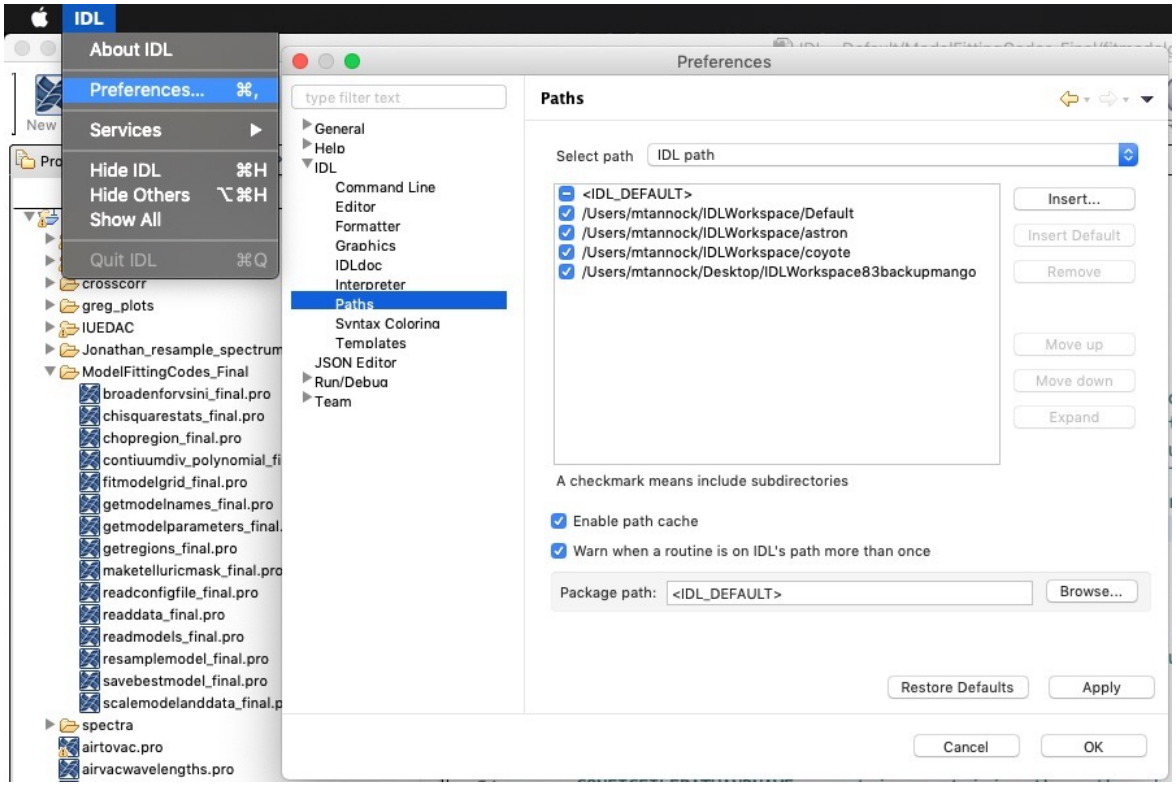

Figure 2: How to edit your IDL paths using `idlde`. Click the 'Insert...' button to add new directories.

2.1 Setting Up IDL

Mastiff was written to run on IDL 8.7.1 and have been confirmed to work on IDL 8.8.2. There is no guarantee these codes will run on other versions.

To get **Mastiff** up and running, you will need to copy the `IDLCodes/` directory and files to your `IDLWorkspace`. If you do not have a folder called `IDLWorkspace/` (or similar) in your home directory, try typing `idlde` in a terminal to open the IDL Developer's Environment. A workspace should be set up for you when you launch `idlde` for the first time. You should also download the IDL Astronomy User's Library[4], and place it in your IDLWorkspace. Ensure that these codes are in the IDL path. The easiest way to do this is through `idlde`, in the Preferences Tab (See Figure 2). When code and function names appear blue in `idlde`, it is recognizing your paths correctly.

Codes and functions in IDL should compile automatically when they are called for the first time in a session, but if there is an issue with your paths, or if you make a change to your code while IDL is active, you can compile a code with the following, from inside of `idlde` or from the command line when IDL is open:

```
.compile -v '/PathToYourCode/YourCode.pro'
```

3 Getting Started

3.1 Your Data

This is not a tutorial on reducing spectroscopic data. I am assuming you have a reduced spectrum that has been calibrated to vacuum wavelengths, with telluric features removed, and corrected for the barycentric velocity at the time of observation. The data do not need to be flux calibrated or normalized (the code will normalize everything), but must be in F_λ units. The data can be in any format you like (fits, txt, csv, etc).

For this guide, we will use a snippet of the T6 dwarf spectrum published in Tannock et al. (2022). This spectrum is small part of a Gemini South/IGRINS spectrum, with the wavelength range 1.77396 to 1.79888 μm

[4] https://idlastro.gsfc.nasa.gov/

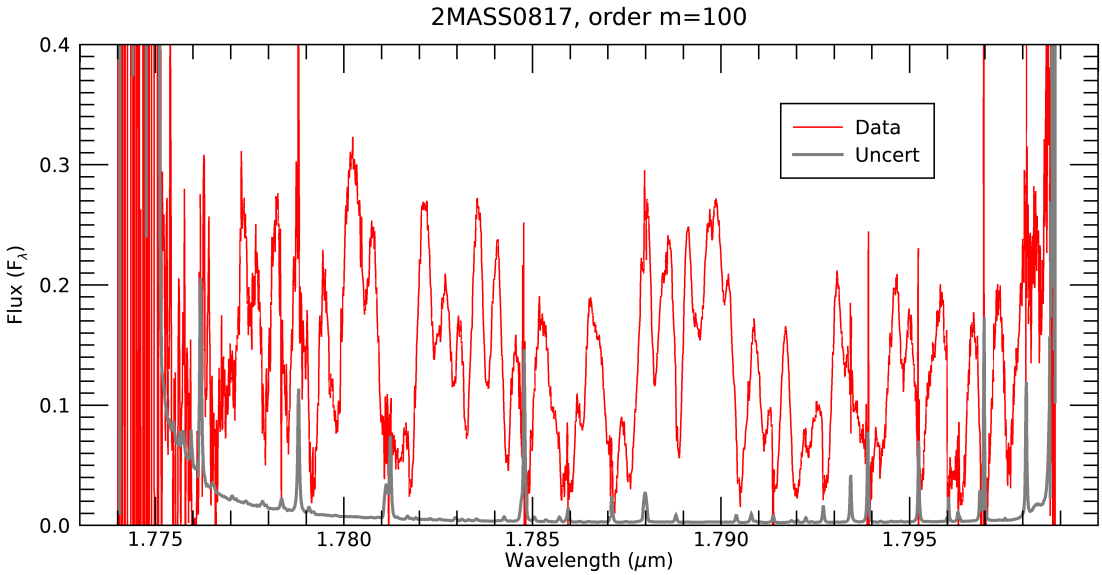

Figure 3: The sample data for this guide. This is order m=100 of a Gemini South/IGRINS spectrum of the T6 dwarf 2MASS0817 from Tannock et al. (2022).

(order m = 100), and is shown in Figure 3. IGRINS has a resolution of $R = \lambda/\Delta\lambda \sim 45,000$, and this is a high signal-to-noise (SNR) spectrum. But, as you can see, there are a few potential issues with this spectrum. The edges are very noisy, and there appear to be a few positive and negative spikes throughout the spectrum. As we will see soon, these are the remnants of an imperfect telluric correction, but **Mastiff** can account for this. This spectrum is available as a text file in the `SampleData1/` directory inside of the `Mastiff_ModelFittingTools` repository.[5]

To summarize, to use **Mastiff**, your **data** must:

- Be wavelength calibrated to vacuum wavelengths
- Be telluric corrected
- Be corrected for the barycentric velocity
- Have wavelength in microns (μm)
- Have Flux in F_λ units
- Be in any file format

3.2 The Models

As mentioned above, the spectroscopic models are available on grids of effective temperature ($T_{\rm eff}$), gravity (g or $\log g$), and for some models, metallicity and sedimentation efficiency ($f_{\rm sed}$). $T_{\rm eff}$ is always given in Kelvin, and $f_{\rm sed}$ is unitless. Depending on the model family, gravity is given as g in m/s^2, or as $\log g$ where g is in cgs units (cm/s^2). So a g=1000 m/s^2 model may also be written as $\log g$=5.0 after converting g to cm/s^2.

The models typically have flux in F_ν (erg/cm^2/s/Hz), so this must be converted to F_λ (erg/cm^2/s/μm) before comparing them to the data:

$$F_\lambda = \frac{F_\nu c}{\lambda^2} \tag{1}$$

where c is the speed of light. Some models have other conversions that are required, and you should consult their documentation. The wavelength units must also be converted to microns. The units do not need to be converted before running **Mastiff**. The units can be converted when the model is read in, so you should keep your models in the same format as they are available in.

[5] https://github.com/megantannock/Mastiff_ModelFittingTools

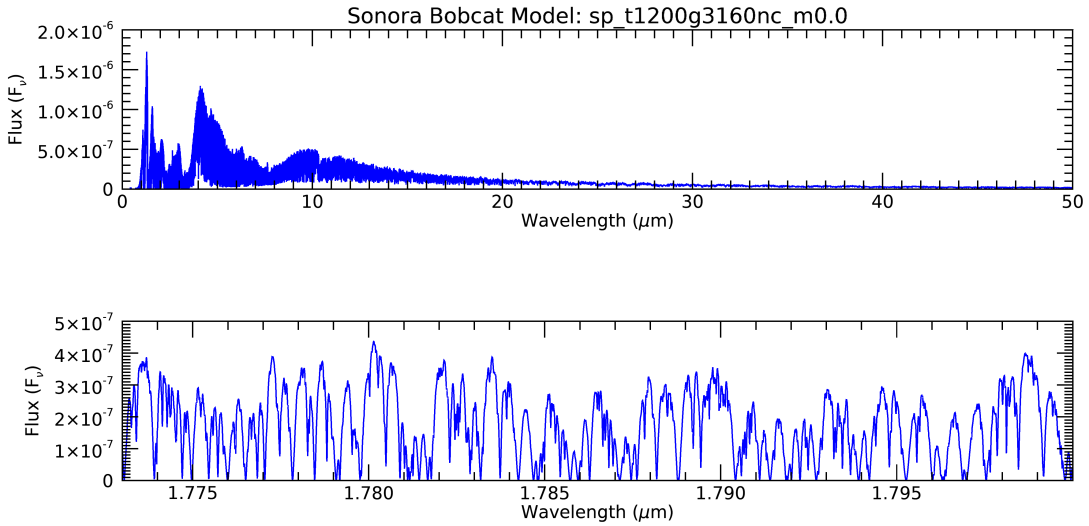

Figure 4: A Sonora Bobcat model (Marley et al., 2021), with $T_{\rm eff}$=1200 K, and g=3160 m/s^2 ($\log g$=5.5). The resolution of this model is $R \sim 200,000$. *Top panel:* the full wavelength coverage of the model. *Bottom panel:* Cropped to match the wavelengths of Figure 3.

The models have no rotation ($v \sin i = 0$ km/s) and no shift for radial velocities (RV=0km/s), and have resolutions $R > 100,000$.

You can use whichever models you like, but the code has already has the capability to handle the following models: BT-Settl, SM08, Morley, Sonora Bobcat, and Sonora Bobcat Alt A.

In this guide we will fit Sonora Bobcat models to our data. An example of a Sonora Bobcat model is shown in Figure 4. This model has its original F_ν flux units and wavelength in microns. The flux will need to be converted. This model also has $R \sim 200,000$ but we will match it to the resolution of our data, after we apply a rotation velocity broadening and radial velocity shift.

3.3 The Configuration File

The entire fitting process of **Mastiff** is dictated by the 27-line **configuration file**. The lines of the configuration file must have the following format:

```
Keyword_Name = value
```

with a space-equals sign-space in between the keyword name and the keyword value. The keyword name and keyword value must not contain spaces, and strings should not have quotations (‘’’’) around them.

Here is the list of the keywords in the configuration file with a description of each. An example of a configuration file (the one I used for this guide) is shown in Figure 5. Whenever a keyword from the configuration file is mentioned in this guide, I will highlight it in red. In Section 4 I will go through each step of the model fitting in order, and add additional details and figures relating to these keyword.

The order of these keyword is not necessarily the easiest one (with all related keywords grouped together), but I hope these descriptions will keep it clear. These descriptions are also available in the documentation of both `FITMODELGRID_FINAL.PRO` and `READCONFIGFILE_FINAL.PRO`.

The configuration file keywords:

- `objectname` = The name of your observed object (or whatever you want to identify it by), must be a string with no spaces. See Section 4.6 for more details.
- `objectpath` = The path to your observed spectrum data. See Section 4.2 for more details.

- `objectfilename` = The name of your observed spectrum's flux file. Your input data must be previously corrected for the barycentric velocity. See Section 4.2 for more details.
- `objecterrorname` = The name of your observed spectrum's error file (may be the same file as above). Your input data must be previously corrected for the barycentric velocity. See Section 4.2 for more details.
- `objectinstrument` = The name of the instrument you used, so you can read the data in correctly. Options are: FIRE, GNIRS, or IGRINS. If your data differs from these formats, add a new `CASE` statement to read the data in. Make sure the flux is in F_λ units and the wavelength is in microns. You can add these conversions in `readdata_final.pro`, you do not have to convert the units before running this code. See Section 4.2 for more details.
- `modeltype` = The name of the model family you want to fit to. Options are: `MORLEY`, `SAUMON`, `SONORA-BOBCAT`, and `BOBCAT-ALTA`. If you have a different model, add a new `CASE` statement to read the model in. Make sure the flux is in F_λ units and the wavelength is in microns. You can add these conversions in `readmodels_final.pro`, you do not have to convert the units before running this code. See Sections 4.7 and 4.7 for more details.
- `modelpath` = The path to your models. See Section 4.7 for more details.
- `modelnamesfile` = The path and name of a text file which contains the name of each model you wish to fit to. See Section 4.7 for more details.
- `wavelengthregionname` = A name for the region you want to fit to, for example: Jband. This is to name the output file. Must be a string with no spaces. See Section 4.6 for more details.
- `wavelengthregion` = The wavelength coverage you want to fit to. Must be in the format 1.23,1.56 with the comma included, no spaces. Units are microns. See Sections 4.1 and 4.4, Figure 8 for more details.
- `normalizationregion` = The wavelength coverage you want to normalize to. Must be in the format 1.25,1.50 with the comma included, no spaces. Units are microns. See Section 4.4 for more details.
- `dividecontinuum` = You can divide out the continuum if you wish. Set this to the degree of the polynomial you want to fit. If you don't want to divide the continuum, use 0. See Section 4.4 for more details.
- `outfilepath` = The path where you would like the output file written to. See Section 4.6 for more details.
- `rundate` = The date, used for the output file's name. Must be a string with no spaces. See Section 4.6 for more details.
- `vsinistartval` = The start value for the range of $v \sin i$ value you want to test. Units are km/s. See Section 4.1 for more details.
- `vsiniendval` = The end value for the range of $v \sin i$ value you want to test. Units are km/s. See Section 4.1 for more details.
- `vsinistep` = The step size for the range of $v \sin i$ value you want to test. Units are km/s. See Section 4.1 for more details.
- `rvshiftstartval` = The start value for the range of RV value you want to test. Units are km/s. See Section 4.1 for more details.
- `rvshiftendval` = The end value for the range of RV value you want to test. Units are km/s. See Section 4.1 for more details.
- `rvstep` = The step size for the range of RV value you want to test. Units are km/s. See Section 4.1 for more details.
- `limbdarkeningcoefficient` = Numeric scalar giving the limb-darkening coefficient. Use 0.6 as the default value. This is used in `lsf_rotate.pro`, see documentation [6], and Section 4.8 for more details.

[6] https://idlastro.gsfc.nasa.gov/ftp/pro/astro/lsf_rotate.pro

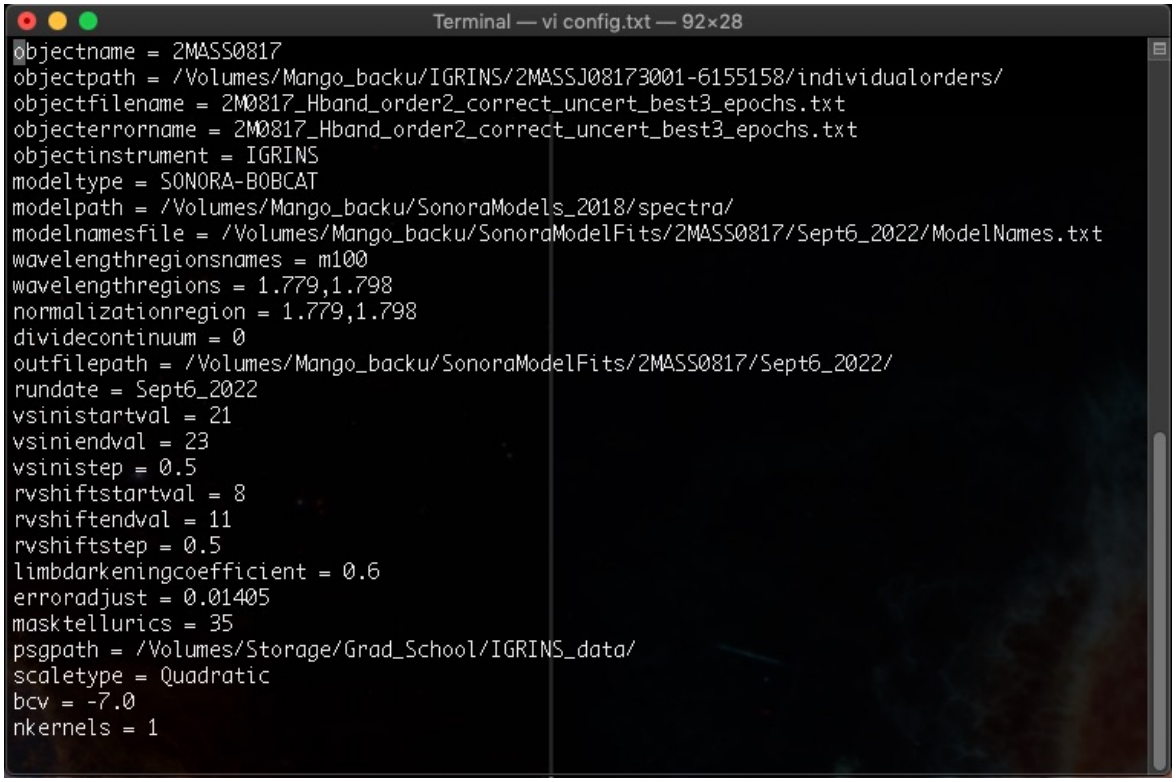

Figure 5: The configuration file used for the model fitting in this guide.

- `erroradjust` = You can add an extra constant to be added in quadrature to your observed spectrum's uncertainty. This can be used to account for uncertainty in the models (which do not come with uncertainties). You can use trial an error to find the value which gives a reduced chi square of 1. See Section 4.5 for more details.

- `psgpath` = The path to the Planetary Spectrum Generator (PSG; Villanueva et al. 2018)[7] transmittance spectrum, if you're masking the tellurics. If you're not masking, enter your oufilepath. See Section 4.3 for more details.

- `masktellurics` = You can apply a mask to the wavelengths where strong telluric features are present. Enter the value (in %) of a cut off of the model transmittance to mask. For example: Mask all lines which have less than 30% transmittance - use value 30. The mask is based on the atmospheric transmittance from the PSG. If you do not want to use a mask, enter 0. See Section 4.3 and Figures 6 and 7 for more details.

- `scaletype` = This code can do some fancy scaling and applying offsets. Options are: `None`, `Scale`, `ScaleAndOffset`, `Linear`, `LinearAndOffset`, `Quadratic`, `QuadraticAndOffset`. See `SCALEMODELANDDATA_FINAL.PRO` documentation and Section 4.10 for more details.

- `bcv` = The barycentric velocity correction that was applied to your data in km/s. See Section 4.3 for more details.

- `nkernels` = If your model's resolution changes a lot across the wavelengths of interest, you may end up overbroadening or underbroadening the ends of the spectrum by using the same kernel across the whole spectrum. You can split the spectrum in to N sections and broadening each section with its own kernel, then stitching the resulting spectrum back together. If you do not want to do this, set numkernels=1. See Section 4.8 for more details.

[7] https://psg.gsfc.nasa.gov/

3.4 Starting the Fitting

Once your configuration file is complete, you can start **Mastiff** by entering the following in to a terminal running IDL:

```
FITMODELGRID_FINAL, '/path/configfilename'
```

where `path` is the path to where your configuration file is located, and `configfilename` is the name of the text file containing your configuration keywords. So your call might look something like this:

```
FITMODELGRID_FINAL, '/Users/mtannock/WorkStuff/test_config.txt'
```

There is also an **optional** flag you can set to produce a figure (.png, .pdf, and .ps) of the best fitting model, and to save the data for that figure:

```
FITMODELGRID_FINAL, '/Users/mtannock/WorkStuff/test_config.txt', /MAKEFIGURES
```

This flag calls `savebestmodel_final.pro` at the end of the fitting. `savebestmodel_final.pro` can also be called after the fact using the same configuration file:

```
SAVEBESTMODEL_FINAL, '/Users/mtannock/WorkStuff/test_config.txt'
```

4 Model Fitting, Step by Step

In this section I will describe each step of **Mastiff**, as they appear in the main code, `FITMODELGRID_FINAL.PRO`, and provide some helpful figures of the intermediate steps.

4.1 Initial Set Up

`FITMODELGRID_FINAL.PRO` starts by printing some useful information, such as how to cite the software and the time the code starts running. Depending on the model grid you are testing, a run can take a long time, so it is helpful to know what time the code started and ended.

The **configuration file** is read in by the `readconfigfile_final.pro` function, which splits up the lines of the configuration file (using the spaces, so it is important to format your configuration file correctly!) and prints the configuration file out, so you can check that you have entered information correctly. An array of strings called 'keywords' containing the values of the keywords is returned.

`FITMODELGRID_FINAL.PRO` then renames the elements of the 'keywords' array so that the code is easier to read.

The grids of $v \sin i$ and RV are generated as arrays based on the `vsinistartval`, `vsiniendval`, `vsinistep`, `rvshiftstartval`, `rvshiftendval`, and `rvshiftstep` keywords from the configuration file.

The `getregions_final.pro` function takes the `wavelengthregionname`, `wavelengthregion` (or `normalizationregion`), and `dividecontinuum` keywords and splits the strings at the comma. The function changes the numeric variables to numbers instead of strings.

4.2 Reading in the Data

Now we're ready to read in our observed data. The `readdata_final.pro` function does this for us, using the `objectinstrument`, `objectpath`, `objectfilename`, and `objecterrorname` keywords.

Within `readdata_final.pro`, a `CASE` statement based on `objectinstrument` determines exactly how we're going to read in the data. `objectpath` and `objectfilename` or `objecterrorname` are concatonated to find the file, and it is read in. If the units are not in F_λ units and the wavelengths are not in microns, this conversion can be added to this part of the code. If you have data that differs from the existing formats, simply add another `CASE` statement to `readdata_final.pro` that suits your data.

Lastly, the slit width (in pixels) of the spectrograph is set. This is needed for the telluric line masking later in the code.

Now we have data to work with! We will do a couple more steps to prepare it before moving on to the model.

4.3 Telluric Masking

Telluric masking is an **optional** step for preparing the data. This is only necessary if your telluric correction is poor, leaving residuals that may otherwise drive the chi square.

Whether or not to use the telluric masking is set by the masktellurics keyword. If it is set to 0, the masking is skipped, and if it is set to any other number (from 1 to 100), the masking will proceed.

The masking uses an Earth's Transmittance spectrum, produced by the Planetary Spectrum Generator (PSG; Villanueva et al. 2018)[8], and a list of OH emission lines (Rousselot et al., 2000). These files, psg_alldefault_JHKband_data.txt and ohlines.dat, respectively, are provided in the Github Repository[9] along with the IDL codes, in the directory PSG/. They can be placed in any location, and that location is specified by the psgpath keyword. The two file names are hardcoded in the telluric correction section of FITMODELGRID_FINAL.pro, and only has coverage for the near-infrared (J, H, and K bands), so if you wish to use a different mask, you must edit this part of the code.

These two files are read in, and then the function maketelluricmask_final.pro generates the mask for this data set, based on the bcv and masktellurics keywords set in the configuration code, along with the slitwidth set by readdata_final.pro.

In correcting for the barycentric velocity, the data (and telluric features) are shifted, so the bcv is required to shift the Earth's transmittance and OH line list to match the data. The masktellurics keyword doubles as a flag and a threshold for the masking. The slitwidth is used to determine the approximate widths of the OH emission lines in the data set.

The transmittance spectrum was generated with resolution $R \sim 50,000$. It is matched to the resolution of the data using the resamplemodel_final.pro function, which performs a simple interpolation using IDL's interpol function.

The transmittance spectrum is shown in Figure 6. The depths of these lines indicate how much light is transmitted through Earth's atmosphere; at 1.0, 100% of the light goes through, at 0.5, 50% of the light goes through, and at 0.0, 0% of the light goes through. The masktellurics keyword sets a threshold - at any wavelengths the transmittance spectrum goes below the masktellurics value, the corresponding data points will be set to NaN in the spectrum. maketelluricmask_final.pro determines these regions using IDL's where function.

The list of OH emission lines contains discrete wavelengths for the positions of the lines. maketelluricmask_final.pro finds the data points in our spectrum corresponding to those positions and sets the flux to NaN. The data points within the slitwidth (in pixels) number of data points on either side of the line, to account for instrumental broadening.

Figure 7 shows the transmittance spectrum for our region of interest (top panel) along with the masktellurics threshold we set in the configuration file, and the appearance of the data before and after the mask has been applied (bottom panel).

4.4 Trimming and Normalizing the Data

Next, we have the option to trim the edges of our spectrum, or select a smaller part of the spectrum. This is performed by the function chopregion_final.pro. The wavelengthregion values are used with IDL's where function to select a section of the data. If you do not wish to remove the edges, simply set the wavelengthregion values to your wavelength coverage.

For the data we're fitting in this guide, we use this keyword to trim off the noisy edges. This data set has a lot of overlap between the echelle orders, so we can be generous in how much we trim off. Figure 8 shows the before and after of our data set using the wavelengthregion with value 1.779,1.798.

After trimming, we normalize the data. The normalizationregion keyword values are used with IDL's where function to get the part of the spectrum corresponding to this region. The mean is taken for that region and used to normalize the data and uncertainty. This allows you to normalize based on a specific region of the data, rather than the entire spectrum. This is helpful if you're fitting a broad wavelength coverage, or a section of the spectrum that has a jump, like a molecular bandhead.

Now, **optionally**, the data may have the continuum removed with a simple polynomial (this tool does not do anything fancy like a spline fit). The dividecontinuum keyword sets the degree of the polynomial to be fit to the data. If you do not want to remove the continuum, set dividecontinuum to 0. The contiuumdiv_polynomial_final.pro function smooths the data a little and then fits a polynomial of degree

[8] https://psg.gsfc.nasa.gov/
[9] https://github.com/megantannock/Mastiff_ModelFittingTools

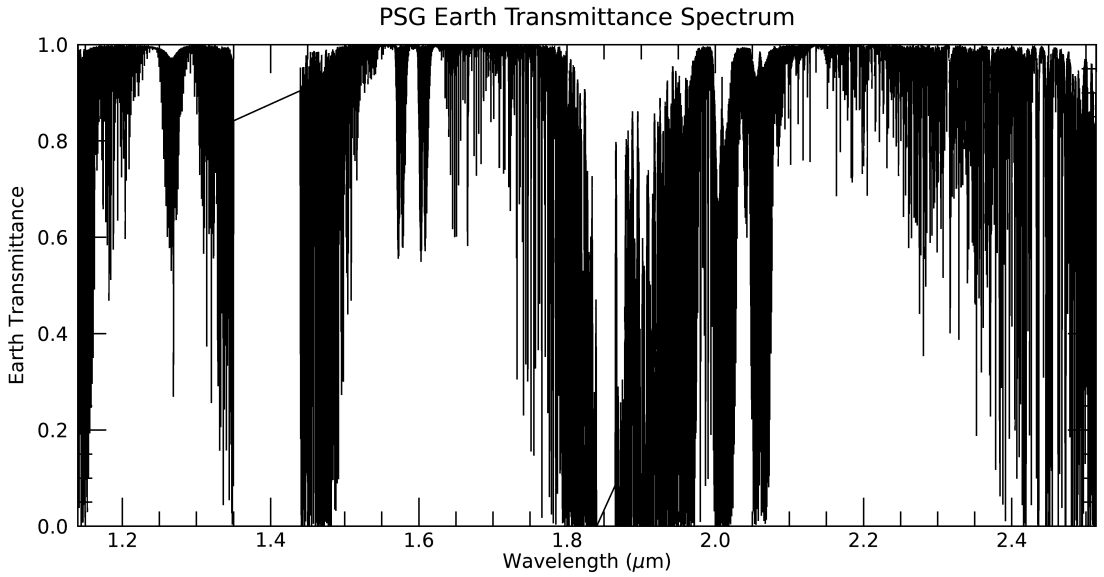

Figure 6: The Earth's transmittance spectrum generated by the PSG for the J, H, and K bands of the near-infrared. The depths of these lines indicate how much light is transmitted through Earth's atmosphere; at 1.0, 100% of the light goes through, at 0.5, 50% of the light goes through, and at 0.0, 0% of the light goes through. Note that there is a gap between the J and H bands from about 1.350 μm to 1.44 μm, and between the H and K bands from about 1.840 μm to 1.865 μm. If these regions are required, they will need to be generated with the PSG.

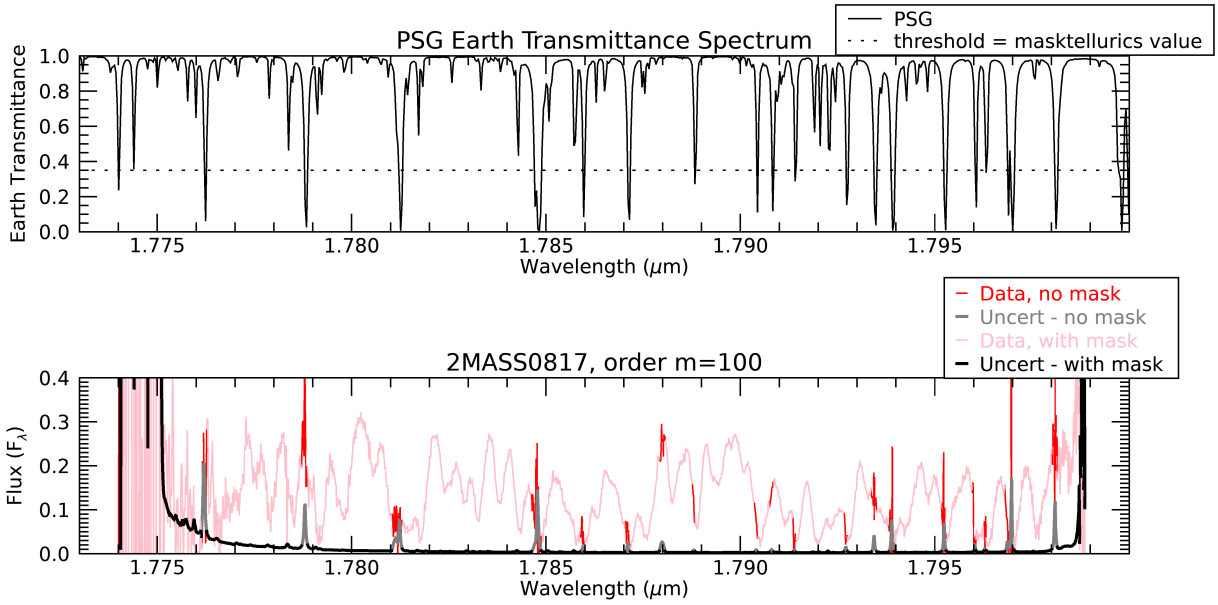

Figure 7: *Top panel:* The Earth's transmittance spectrum generated by the PSG for a narrow wavelength region. Here a threshold of 0.35 (35%) has been set with the `masktellurics` keyword, and is indicated with a dashed line. This means that anywhere the transmittance goes below this line, the values in the spectrum will be set to NaN. The positions of the OH emission lines are not shown, but they are also masked out in the bottom panel. *Bottom panel:* The data before and after the mask has been applied. The red line (no mask) is plotted under the pink line (with mask), so anywhere red peaks out is where the data points have been removed.

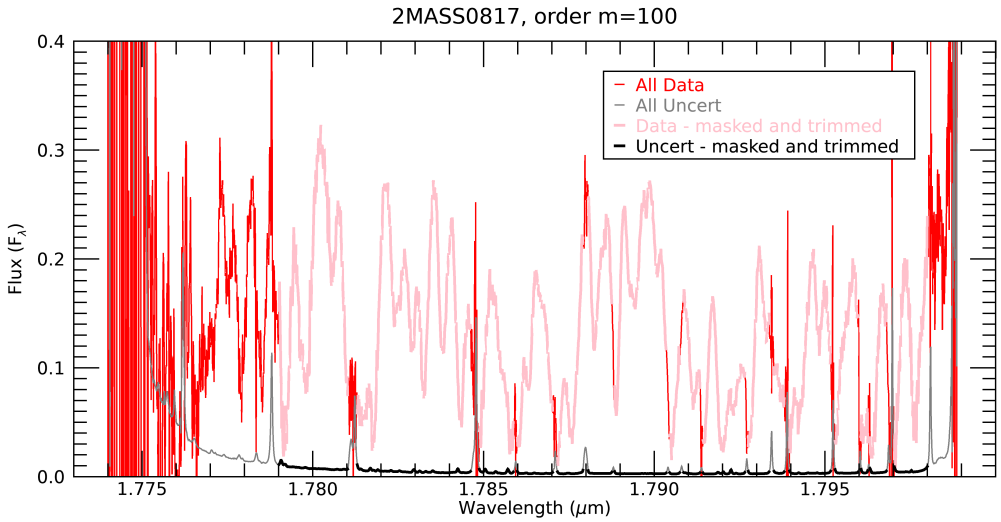

Figure 8: The before and after of our data set using the `wavelengthregion` keyword to trim to `1.779,1.798`. The telluric mask has also been applied to the pink line.

`dividecontinuum` to the data using IDL's `ROBUST_POLY_FIT.pro` function. The fit is returned to the main code and divided out of the data and uncertainty.

4.5 Accounting for the Uncertainty in the Models

The spectroscopic models do not come with uncertainties, although there are uncertainties on their flux values. To account for this missing uncertainty, the `erroradjust` keyword can be used to add some uncertainty in quadrature to the uncertainty on the data. Ideally this is a value which produces a reduced chi square (χ^2_R) of 1 for the best fitting model. This value can be determined through an iteration.

The uncertainty used in the χ^2 calculations is:

$$\sigma_{\rm final} = \sqrt{{\sigma_{\rm data}}^2 + {\sigma_{\rm erroradjust}}^2} \tag{2}$$

4.6 Setting Up the Output Files

The final step before we read in the models and start comparing models to data is to set up our output file. A file is produced in this step, and will have information appended to it as the code continues. As stated earlier, this is a brute force approach, where we compute the χ^2 for every model on the grid compared to the data, so this file can get quite large if you're running a big grid.

The configuration file header contains the complete configuration file, as a quick check of what you did in this fitting run, and then has the header for the parameters of every model on the grid and the χ^2 values.

Model Filename, $T_{\rm eff}$, log(g) (or g, but the header will say log g), $f_{\rm sed}$, χ^2, degrees of freedom, χ^2_R, $v \sin i$ (km/s), and RV (km/s), followed by some constants that are related to the `scaletype` keyword we will discuss in Section 4.10. There is a `CASE` statement for each `scaletype` option that will set up the columns for the appropriate number of constants.

The name and location of the output file are set by the `outfilepath`, `objectname`, `rundate`, and `wavelengthregionname` keywords, in a string of the form:

```
outfilepath + objectname + '_'+ rundate + '_'+wavelengthregionname + '.dat'
```

So for the keywords shown in Figure 5, the output path and file name will be:

```
/Volumes/Mango_backu/SonoraModelFits/2MASS0817/Sept6_2022/2MASS0817_Sept6_2022_m100.dat
```

Figure 9: And example of how the `modelnamesfile` file should be formatted.

4.7 Reading In the Models

At long last, we are ready to read in the models!

You will tell the code which models to test by setting up a file that will be located in the `modelnamesfile` keyword. `modelnamesfile` should include the path and name of the file. The structure of the file is a text file with one model name on each line, as shown in Figure 9. The `getmodelnames_final.pro` function reads in this file and puts the filenames in to an array of strings.

Now we begin a series of nested `FOR` loops to explore the grid of models, rotation velocities, ($v \sin i$s) and radial velocities (RVs). The outtermost loop is the models loop. Reading in a model is a time consuming step, so we want to read in each model only once. The code prints useful information as it runs, including what model is currently being processed and the time it was started.

First the `getmodelparameters_final.pro` function extracts the models parameters ($T_{\rm eff}$, $\log g$, and $f_{\rm sed}$) from the file name. A `CASE` statement based on the `modeltype` keyword splits up the file name and returns the $T_{\rm eff}$, $\log g$, and $f_{\rm sed}$ values as strings. **If a model does not have the $f_{\rm sed}$ parameter, a value of 0 will be returned for $f_{\rm sed}$.**

Additional cases may be added to the `getmodelparameters_final.pro` function if you are using a set of models that is not currently available (currently available: BT-Settl, SM08, Morley, Sonora Bobcat, and Sonora Bobcat Alt A).

The model is read in by the `readmodels_final.pro` function using the `modeltype` and `modelpath` keywords, and the model name. A `CASE` statement based on `modeltype` determines how the model should be read in and converts the units if required.

For our example in this guide, The Sonora Bobcat models have wavelength in microns, but flux in F_ν units. Figure 10 shows the before and after of this unit conversion. The F_ν shown in this figure is the same as in Figure 4.

The model is then trimmed using the `chopregion_final.pro` function based on the `wavelengthregion` keyword, but a little extra wavelength coverage (0.02 μm) is left on either end of the model, to allow for shifting of large radial velocities.

If the `dividecontinuum` keyword is set to anything other than 0, the `contiuumdiv_polynomial_final.pro` function fits the continuum and divides it out in the same way as the data (See Section 4.4). The model will be normalized using the `normalizationregion` and `scaletype` keywords later in the code (See Section 4.10).

4.8 Applying Rotational Broadening

The next `FOR` loop loops through the rotation velocities. This is more computationally intensive than the radial velocity shifting, so it comes before the RV shift. We want to do computationally intensive things as few times as possible. The velocities we loop through are defined by the `vsinistartval`, `vsiniendval`, and `vsinistep` keywords.

The model The `broadenforvsini_final.pro` function performs the broadening. The current $v \sin i$ value, along with the model wavelength and flux arrays are passed in with the `nkernels` and `limbdarkeningcoefficient` keywords.

The `lsf_rotate.pro` function from the IDL Astronomy User's Library[10] generates a kernel based on the wavelength spacing, input $v \sin i$, and `limbdarkeningcoefficient` keyword. The default value of

[10] https://idlastro.gsfc.nasa.gov/

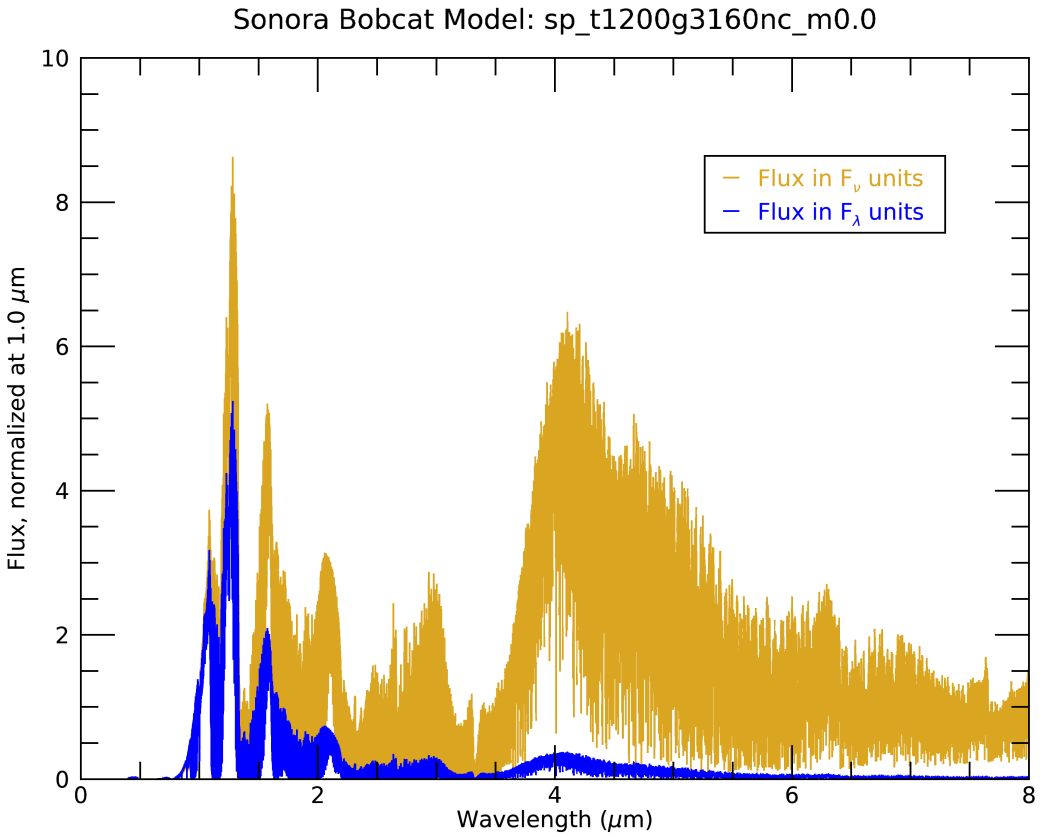

Figure 10: A Sonora Bobcat model before an after converting the flux units from F_ν to F_λ. Both models are normalized to a value of 1.0 at wavelength = 1.0 μm.

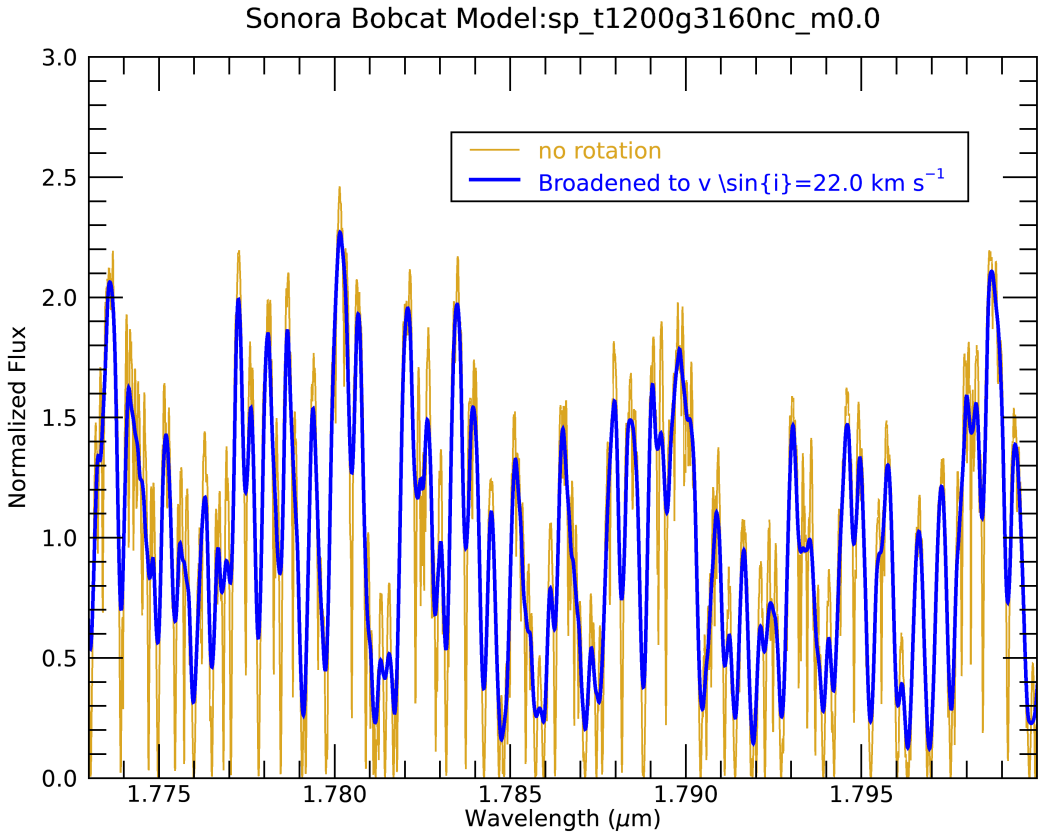

Figure 11: The Sonora Bobcat model before and after applying a rotatioional broadening of 22.0 km/s.

`limbdarkeningcoefficient` is 0.6, and you should consult the `lsf_rotate.pro` function's documentation[11] for more information on this parameter. See also: Chapter 17 of Gray (1992).

The kernel produced by `lsf_rotate.pro` is then convolved with the model flux, using IDL's `CONVOL` function, and the resulting smoothed spectrum is returned to the main code. See Figure 11 to see how the data changes by being broadened.

The kernel is based on the wavelength spacing at the centre of your wavelength coverage. If the spacing or resolution of your wavelength array changes greatly over your wavelength coverage, this can result in an over- or under-broadening at the ends of the spectrum. See Figure 12 for an illustration of this effect.

The `nkernels` keyword allows for an approximation of an evolving kernel to overcome this effect. Technically, a convolution requires a constant kernel, so something with a changing kernel would actually be a linear transform. But here we take the simple approach of dividing the spectrum in to a number of sections (the value of the `nkernels` keyword) and determining a kernel for each of those sections. Each section is broadened with its own kernel, then the sections are stitched together and returned to the main code. This works best if there is not a large jump in the data (like a molecular bandhead, for example), as each section needs to be normalized before it is stitched to the others.

If you wish to use the same kernel everywhere, use `nkernels`=1. This is recommended for all models, which are very high resolution compared to the data.

4.9 Radial Velocity Shifting

The third and innermost `FOR` loop loops through the radial velocities defined by the `rvshiftstartval`, `rvshiftendval`, and `rvshiftstep` keywords. This loop also does the final resolution matching, scaling, and writes out the results to the output file (see Section 4.10).

The wavelengths of the model are shifted for the current RV value simply by:

[11] https://idlastro.gsfc.nasa.gov/ftp/pro/astro/lsf_rotate.pro

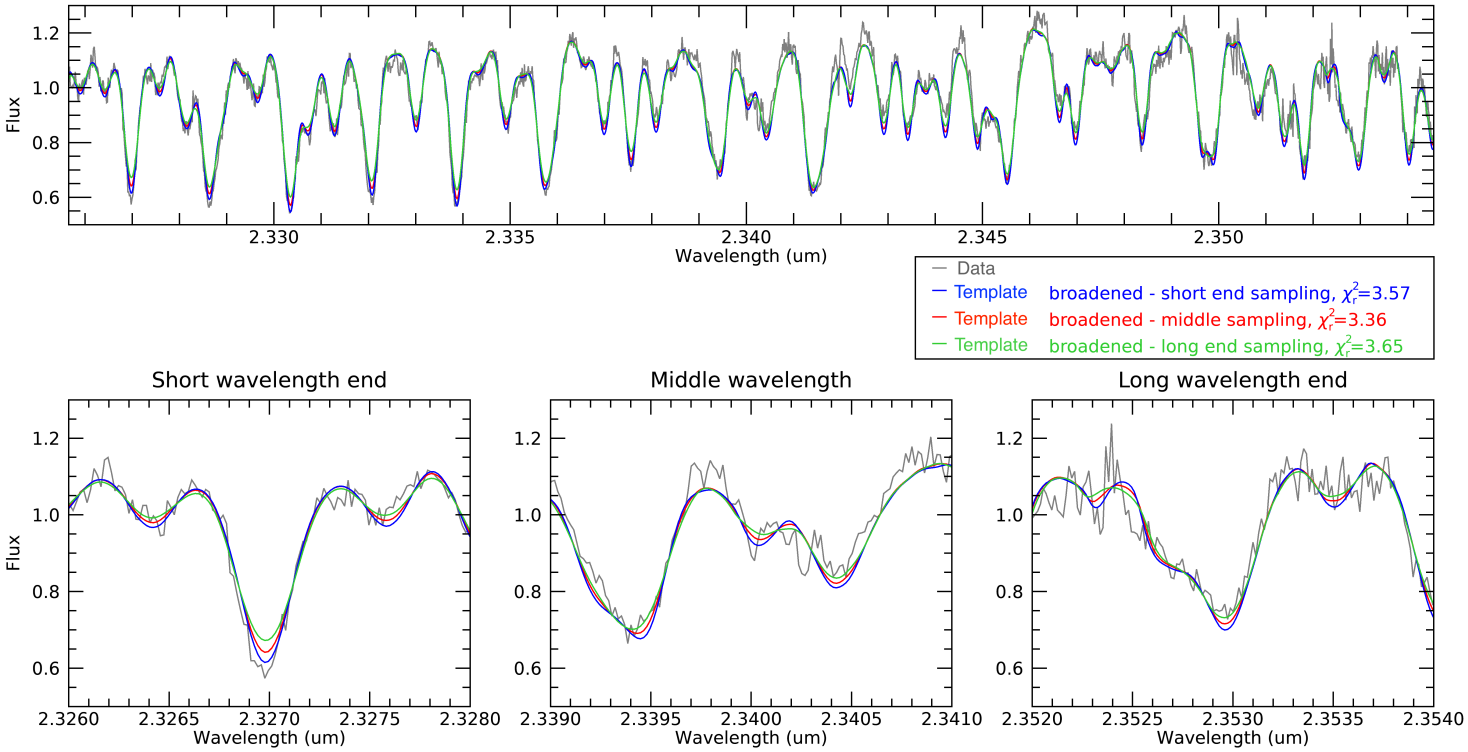

Figure 12: An example of how unevenly sampled data can affect your rotational broadening. *Top panel:* The full wavelength coverage. *Bottom panels:* Zoomed in on three different regions of the top panel.
The figure shows a comparison of a brown dwarf spectrum (grey) and a more slowly rotating template brown dwarf (coloured lines; observed with the same instrument and settings), broadened to fit the data. This data is higher resolution at the long wavelength end than it is at the short wavelength end. If you use a kernel based on the spacing at the centre, the centre will fit well, but the short wavelength end gets overbroadened and the long wavelength end gets underbroadened (see red line). The template has been divided in to three sections and a kernel was generated for each one. Across the bottom panels we can see that the blue line fits best at the short wavelength end, the red line fits best for the middle section, and the green line fits best for the long wavelength end. The blue, red, and green lines correspond to kernels based on the short wavelength section, middle seciton, and long wavelength section, respectively. *Note:* This data is not for 2M0817, the sample data used in this guide, it is for another set of brown dwarfs.

$$\lambda_{\text{shifted}} = \lambda\left(1 + \frac{RV}{c}\right) \tag{3}$$

where c is the speed of light in the same units as RV (km/s).

4.10 Final Touches: Resampling and Optimized Scaling to Minimize the χ^2

Before computing the χ^2 statistic and writing out the fitting information to the output file, the resolution/ wavelength scale of the model must be matched to the data. The `resamplemodel_final.pro` function does this resampling.

The model is then normalized based on the mean of the `normalizationregion`, and finally the `scalemodelanddata_final.pro` function is called to do some fancy optimized normalization. `scalemodelanddata_final.pro` has a few different options for how it scales and offsets the data. A CASE statement determines how this is performed based on the `scaletype` keyword.

The options for the `scaletype` keyword are: `None`, `Scale`, `ScaleAndOffset`, `Linear`, `LinearAndOffset`, `Quadratic`, and `QuadraticAndOffset`. Each of these defines a way of scaling or offsetting the data and model. For each option a "Goodness of Fit" equation, G, is written and we take the derivative of G, set it to zero, and solve the resulting system of equations for constants which minimize G. See Suárez et al. (2021) and Tannock et al. (2022) for a similar approach.

Here are the descriptions and G equations for each case, where D_i is the data, σ_i is the uncertainty, M_i is the model, and λ_i is the wavelength, each for element i of the respective arrays. m, a, b, c, and d are the constants, and the sums are over the arrays (of N elements) which contain our spectra:

- **None**: Do not apply any optimized scaling or offsets. Use the simple normalization of the Data and Model (each normalized separately).
- **Scale**: Determine a multiplicative factor for the model which minimizes G:

$$G = \sum_{i=1}^{N}\left(\frac{D_i - a\,M_i}{\sigma_i}\right)^2 \tag{4}$$

  System of equations we need to solve: $\partial G/\partial a = 0$
- **ScaleAndOffset**: Determine a multiplicative factor for the model, and flux offset for the data which minimizes G:

$$G = \sum_{i=1}^{N}\left(\frac{(D_i + d) - a\,M_i}{\sigma_i}\right)^2 \tag{5}$$

  System of equations we need to solve: $\partial G/\partial d = 0$; $\partial G/\partial a = 0$
- **Linear**: Determine a linear fit which minimizes G:

$$G = \sum_{i=1}^{N}\left(\frac{D_i - (m\,\lambda_i + b)\,M_i}{\sigma_i}\right)^2 \tag{6}$$

  System of equations we need to solve: $\partial G/\partial m = 0$; $\partial G/\partial b = 0$.
  The model is multiplied by the linear fit to simplify the math here, but in the code the data is divided by this fit at the end.
- **LinearAndOffset**: Determine a linear fit for the model and flux offset for the data which minimizes G:

$$G = \sum_{i=1}^{N}\left(\frac{(D_i + d) - (m\,\lambda_i + b)\,M_i}{\sigma_i}\right)^2 \tag{7}$$

  System of equations we need to solve: $\partial G/\partial d = 0$; $\partial G/\partial b = 0$; $\partial G/\partial m = 0$.
  The model is multiplied by the linear fit to simplify the math here, but in the code the data is divided by this fit at the end.

- **Quadratic**: Determine a quadratic fit for the model which minimizes G:

$$G = \sum_{i=1}^{N} \left(\frac{D_i - (a\,\lambda_i^2 + b\,\lambda_i + c)\,M_i}{\sigma_i} \right)^2 \tag{8}$$

  System of equations we need to solve: $\partial G/\partial a = 0$; $\partial G/\partial b = 0$; $\partial G/\partial c = 0$.
  The model is multiplied by the quadratic fit to simplify the math here, but in the code the data is divided by this fit at the end.

- **QuadraticAndOffset**: Determine a quadratic fit for the model, and flux offset for the data which minimizes G:

$$G = \sum_{i=1}^{N} \left(\frac{(D_i + d) - (a\,\lambda_i^2 + b\,\lambda_i + c)\,M_i}{\sigma_i} \right)^2 \tag{9}$$

  System of equations we need to solve: $\partial G/\partial a = 0$; $\partial G/\partial b = 0$; $\partial G/\partial c = 0$; $\partial G/\partial d = 0$.
  The model is multiplied by the quadratic fit to simplify the math here, but in the code the data is divided by this fit at the end.

The derivatives were taken by hand, but I use IDL's linear algebra tools to solve the systems of equations. The data and model are then corrected according to `scaletype` selected, and the constants which minimize G are returned to the main code to be printed in the output file.

For the sample data used in this guide, I found that the residuals turned up at the ends of the orders (See Tannock et al. 2022) due to an instrumental effect. To correct for this, a quadratic correction is used, and the `scaletype` keyword is set to Quadratic in the configuration file. Figure 13 shows the data before and after this correction.

4.11 Calculation of χ^2 and Saving the Output File

Now, finally, we are ready to calculate the χ^2 between the data and model. The `chisquarestats_final.pro` function calculates the χ^2 and reduced χ^2 (χ^2_R):

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{D_i - M_i}{\sigma_i} \right)^2 \tag{10}$$

and

$$\chi^2_R = \frac{1}{N-1} \sum_{i=1}^{N} \left(\frac{D_i - M_i}{\sigma_i} \right)^2 \tag{11}$$

where D is the data, with all of our masking, trimming, normalizing, and fitting, M is the model, σ is the uncertainty from Equation 2 , and $N-1$ is the number of degrees of freedom (d.o.f.).

Finally, the main code writes out the model name, model parameters ($T_{\rm eff}$, $\log g$, $f_{\rm sed}$), $v \sin i$, RV, χ^2, χ^2_R, d.o.f., and constants described in Section 4.10 to the output file. A `CASE` statement based on the `scaletype` keyword determines which constants will be entered in the output file.

As a reminder, the output file is named:
`outfilepath + objectname + '_'+ rundate + '_'+wavelengthregionname + '.dat'`
(so for the sample configuration file in Figure 5:
`/Volumes/Mango_backu/SonoraModelFits/2MASS0817/Sept6_2022/2MASS0817_Sept6_2022_m100.dat`
)

The `FOR` loops repeat for every combination of model, $v \sin i$, and RV, and the code concludes by printing some more useful information, like the location of the output file and the time the code started and ended running. The first few lines of the output file for the test data in this guide are shown in Figure 14.

The very last step is the **optional** flag (`MAKEFIGURES`) for generating a figure of the best fit. The `savebestmodel_final.pro` function is called with the same configuration file, but after reading the configuration file, it also reads the output file of `FITMODELGRID_FINAL.pro`, identifies the single best fitting model from the lowest χ^2_R value, and repeats the above steps exactly, but for that one model. `savebestmodel_final.pro` opens (and closes) a new window showing a three panel figure, as shown in Figure 15 for our sample data. The

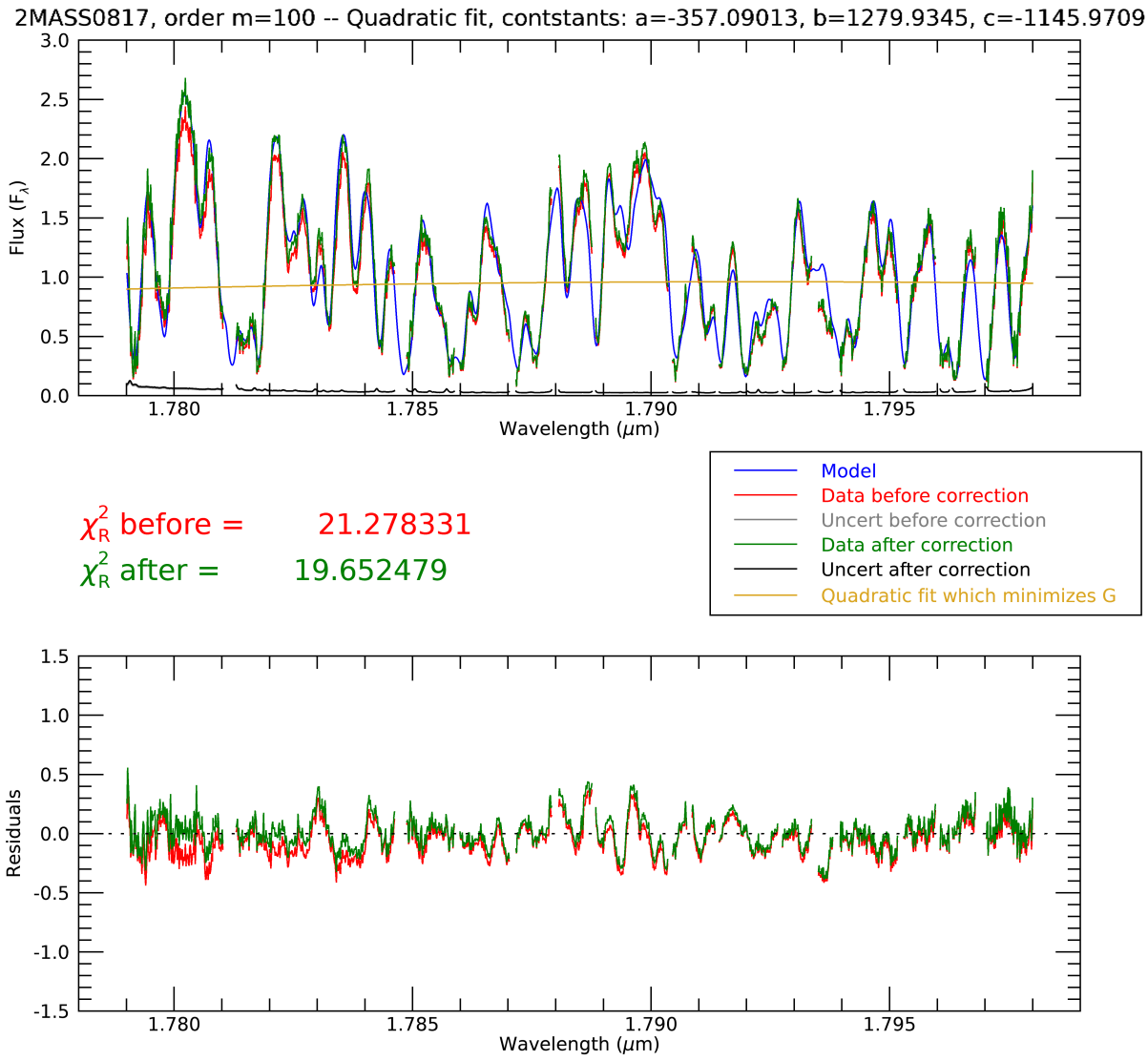

Figure 13: The data before and after the quadratic correction to minimize G. For this data set, the quadratic correction corrects an instrumental effect that otherwise results in a curve in the residuals at the ends of the order. *Top panel:* The data, uncertainty, model, and quadratic fit which minimizes G. *Bottom panel:* The residuals.

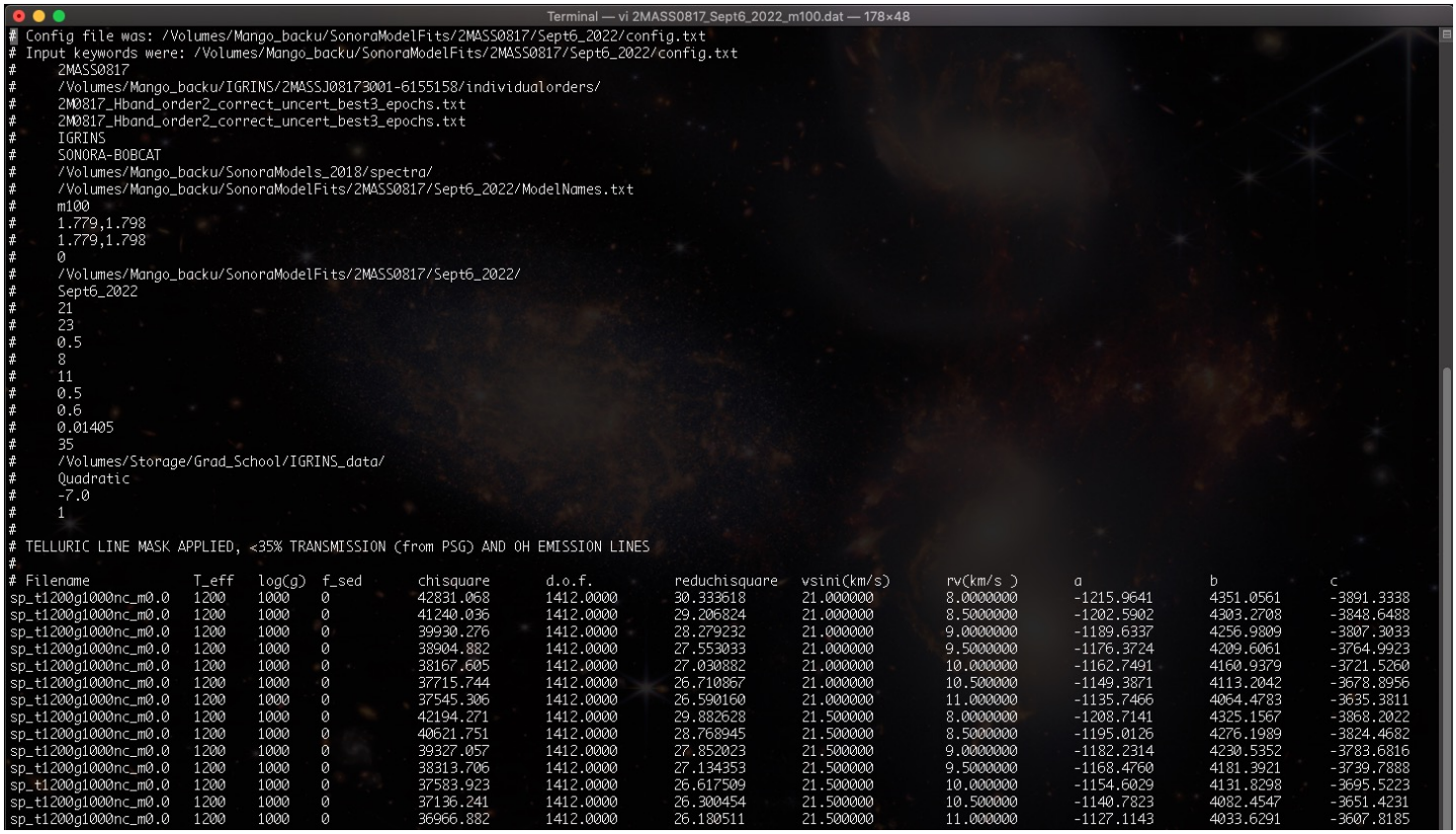

Figure 14: The first few lines of the output file for the configuration file used in this guide. The output file fits shows all of the values set in the configuration file, then all of the information for every model on the grid. Note that all of the $f_{\rm sed}$ values are 0 because the Sonora Bobcat models do not have the $f_{\rm sed}$ parameter.

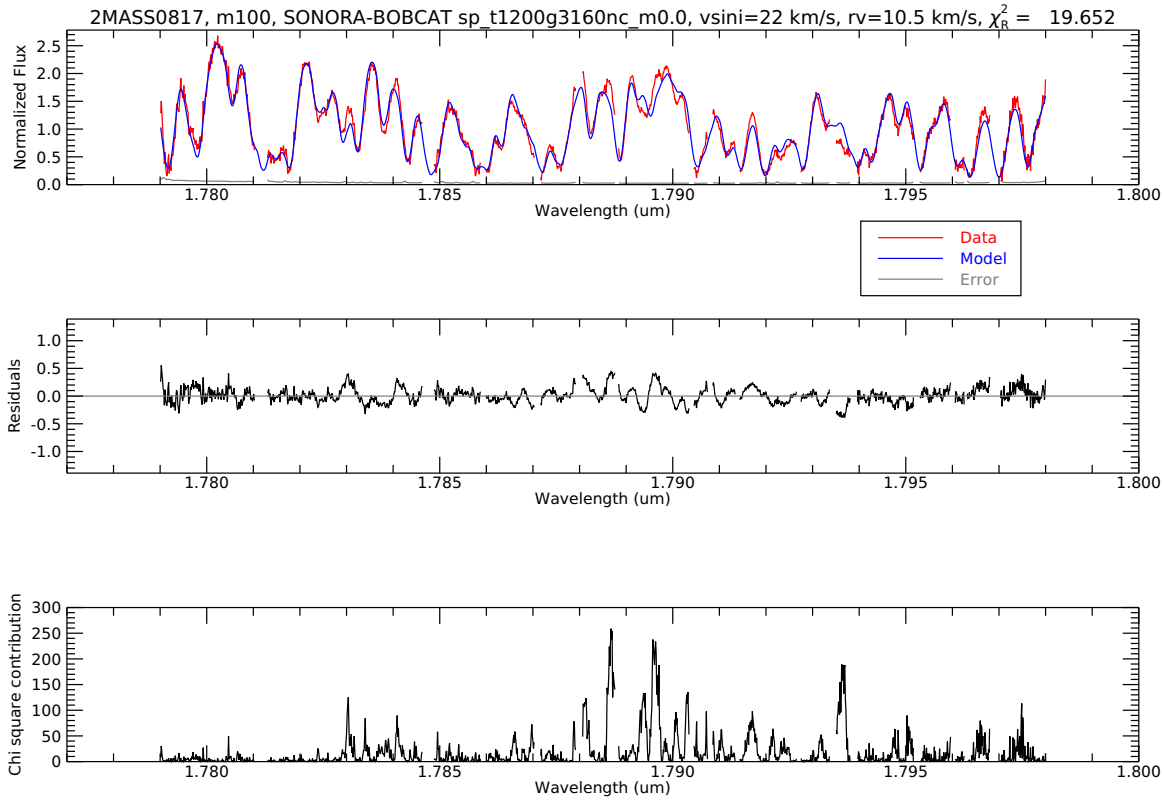

Figure 15: The final figure produced by `FITMODELGRID_FINAL.pro` and `savebestmodel_final.pro` when the `MAKEFIGURES` flag is enabled.

figure includes three panels; the top panel shows the best fitting model and data, the middle panel shows the residuals, and the bottom panel shows the χ^2 contribution at each wavelength. The bottom panel is particularly helpful for identifying regions where the model deviates from the data, or regions that may need to be masked out.

The figure is saved as a .png, .ps, and .pdf. The data in the top panel of the figure is also saved in text file. `savebestmodel_final.pro` creates a new directory called `figures` in the directory in the `outfilepath` keyword, and saves the figures and textfiles there with filenames in the following format, using. the `objectname`, `rundate`, and `wavelengthregionname` keywords:

```
objectname + '_'+ rundate + '_'+ wavelengthregionname + '_bestfit_figure.png'
```

and

```
objectname + '_'+ rundate + '_'+ wavelengthregionname + '_bestfit_figuredata.dat'
```

So for the sample configuration file in this guide (Figure 5), the file names are `2MASS0817_Sept6_2022_m100_bestfit_figure.png` (and .ps and .pdf) and `2MASS0817_Sept6_2022_m100_bestfit_figuredata.dat` .

5 What Do I Do With This Large Output File?

5.1 Figures

If you did not set the `/MAKEFIGURES` flag on `FITMODELGRID_FINAL.pro`, a figure of the data and the best fitting model can be made after the fact by calling the `savebestmodel_final.pro` code with the same configuration

file:

```
SAVEBESTMODEL_FINAL, '/Users/mtannock/WorkStuff/test_config.txt'
```

5.2 Probabilities and Likelihoods

With the χ^2 values (not χ^2_R) in the output file you can perform various statistics. For example, the probability (p) of obtaining a certain χ^2 value for a given model (with a particular $T_{\rm eff}$, g, $f_{\rm sed}$, $v \sin i$, and RV) is proportional to the likelihood, $\mathcal{L}$, where $\mathcal{L} = \prod_{i=1}^{N} \mathcal{L}_i$, and $\mathcal{L}_i$ is the likelihood for a particular data point (one element in the array of your spectrum composed of N data points):

$$\mathcal{L}_i = \exp\left(\frac{-(D_i - M_i)^2}{2\sigma_i^2}\right) \tag{12}$$

where D_i is your data point, σ_i is the uncertainty on that data point, and M_i is your model value corresponding to that data point. So the product, which is the likelihood of a given model, is:

$$\begin{aligned}
\mathcal{L} = \prod_{i=1}^{N} \mathcal{L}_i &= \exp\left(\frac{-(D_1 - M_1)^2}{2\sigma_1^2}\right) \times \exp\left(\frac{-(D_2 - M_2)^2}{2\sigma_2^2}\right) \times \exp\left(\frac{-(D_3 - M_3)^2}{2\sigma_3^2}\right) \times ... \\
&= \exp\left(\frac{-(D_1 - M_1)^2}{2\sigma_1^2} + \frac{-(D_2 - M_2)^2}{2\sigma_2^2} + \frac{-(D_3 - M_3)^2}{2\sigma_3^2} + ...\right) \\
&= \exp\left(\left(\frac{-1}{2}\right)\left(\frac{(D_1 - M_1)^2}{\sigma_1^2} + \frac{(D_2 - M_2)^2}{\sigma_2^2} + \frac{(D_3 - M_3)^2}{\sigma_3^2} + ...\right)\right) \\
&= \exp\left(\left(\frac{-1}{2}\right)\left(\sum_{i=1}^{N} \frac{(D_i - M_i)^2}{\sigma_i^2}\right)\right) \\
&= \exp\left(\frac{-1}{2}\chi^2\right) \\
&\propto p
\end{aligned} \tag{13}$$

This can be repeated for every single model, and then normalized so that the sum of all likelihoods is equal to 1: $\sum \mathcal{L}_{models} = 1$. See Chapter 6 of Wall & Jenkins (2003), the talk by Tom Loredo at the 2018 Carl Sagan Exoplanet Workshop[12], and lecture slides by Chris Blake[13] for more details.

Sometimes you are dealing with very large χ^2 values that can cause computation problems in the exponential. Here is a cute way to **calculate the normalized likelihood for a given model** when your χ^2s are large:

Consider the probability of a model 'a': $p_a = e^{-\chi^2_a/2}$ (from Eq. 14). The normalized likelihood for model a is: $\mathcal{L}_a = p_a/(\sum_{k=1}^{M} p_k)$ for a set of M models. So let's expand that a bit:

$$\begin{aligned}
\mathcal{L}_a = \frac{p_a}{\sum_{k=1}^{M} p_k} &= \frac{p_a}{p_a + p_b + p_c + ...} \\
&= \frac{e^{-\chi^2_a/2}}{e^{-\chi^2_a/2} + e^{-\chi^2_b/2} + e^{-\chi^2_c/2} + ...}
\end{aligned} \tag{14}$$

Let's invert $\mathcal{L}_a$ to combine some terms:

[12] Slides: https://nexsci.caltech.edu/workshop/2018/presentations/Sagan18_Loredo.pdf ; Talk: http://nexsci.caltech.edu/workshop/2018/presentations/SSW2018_Loredo.mp4

[13] https://astronomy.swin.edu.au/~cblake/StatsLecture3.pdf

$$
\begin{aligned}
\frac{1}{\mathcal{L}_a} &= \frac{e^{-\chi_a^2/2} + e^{-\chi_b^2/2} + e^{-\chi_c^2/2} + ...}{e^{-\chi_a^2/2}} \\
&= \frac{e^{-\chi_a^2/2}}{e^{-\chi_a^2/2}} + \frac{e^{-\chi_b^2/2}}{e^{-\chi_a^2/2}} + \frac{e^{-\chi_c^2/2}}{e^{-\chi_a^2/2}} + ... \\
&= 1 + e^{(-\chi_b^2/2)+(\chi_a^2/2)} + e^{(-\chi_c^2/2)+(\chi_a^2/2)} + ... \\
&= 1 + e^{\frac{1}{2}(\chi_a^2-\chi_b^2)} + e^{\frac{1}{2}(\chi_a^2-\chi_c^2)} + ... \\
&= \sum_{k=1}^{M} e^{\frac{1}{2}(\chi_a^2-\chi_k^2)}
\end{aligned}
\tag{15}
$$

And inverting this again to obtain the normalized likelihood for model a ($\mathcal{L}_a$):

$$
\mathcal{L}_a = \frac{1}{\sum_{k=1}^{M} e^{\frac{1}{2}(\chi_a^2-\chi_k^2)}} \tag{16}
$$

You can use the normalized likelihoods to examine the behaviour of each model parameter ($T_{\rm eff}$, g, $f_{\rm sed}$, $v \sin i$, and RV) for all other parameters by marginalizing over all other parameters. See slide 25 of Chris Blake's slides.[14] You can marginalize over each parameter and plot something like a corner plot[15] (it's more of a histogram with this type of data - a corner plot comes from an MCMC) and to obtain a probability distribution. Assuming that distribution is a Gaussian, you can obtain standard deviations, percentiles, etc.

5.3 Weighted Averages

If you have fit many regions in a spectrum (for example, IGRINS data has 46 orders that can be fit independently) you compute weighted averages for each model parameter ($T_{\rm eff}$, g, $f_{\rm sed}$, $v \sin i$, and RV). The IPAC website[16] has a nice list of equations. Usually we weight values by $1/\sigma^2$, but $\exp\left(\frac{-1}{2}\chi_R^2\right)$ (reduced χ^2 this time) is a reasonable approximation for the weight. This is the technique used in Tannock et al. (2021) and Tannock et al. (2022).

6 Final Notes

Mastiff is what you can call "Embarrassingly Parallel," [17] although I prefer the alternative term, "pleasingly parallel" because there is nothing wrong with these codes. Whether embarrassing or pleasing, this means that multiple instances of the code can be run to fit multiple spectra (multiple sections of a spectrum from one object, like different orders, or different spectra entirely). All you have to do is set up different configuration files and you're only limited by the number of cores on your machine. Of course the main code could also be re-written to do the `FOR` loops in parallel as well, to make it even faster, but that's a project for another day.

When I have many fitting instances to run, more than the available cores on my machine, I like to also write a wrapper that calls instances of `FITMODELGRID_FINAL.pro` back to back. Then I don't have to check in X minutes or hours later to start the next fitting instance. This might look something like:

```
pro runfitting
FITMODELGRID_FINAL, '/Users/mtannock/WorkStuff/config_1.txt', /MAKEFIGURES
FITMODELGRID_FINAL, '/Users/mtannock/WorkStuff/config_2.txt', /MAKEFIGURES
FITMODELGRID_FINAL, '/Users/mtannock/WorkStuff/config_3.txt', /MAKEFIGURES
end
```

and I would run `runfitting.pro` from the command line.

That's about it! Thanks for using **Mastiff**! I hope this guide was clear and useful. Please reach out by sending an email to mtannock@uwo.ca if you have any questions or if you find any errors in the guide or codes.

[14] https://astronomy.swin.edu.au/~cblake/StatsLecture3.pdf
[15] https://corner.readthedocs.io/en/latest/
[16] https://wise2.ipac.caltech.edu/docs/release/allwise/expsup/sec5_3bv.html
[17] https://en.wikipedia.org/wiki/Embarrassingly_parallel

References

Ackerman A. S., Marley M. S., 2001, , 556, 872

Allard F., Hauschildt P. H., Alexander D. R., Tamanai A., Schweitzer A., 2001, , 556, 357

Allard F., Homeier D., Freytag B., 2012, Philosophical Transactions of the Royal Society of London Series A, 370, 2765

Gray D. F., 1992, The observation and analysis of stellar photospheres.. Cambridge University Press

Marley M. S., et al., 2021, , 920, 85

Morley C. V., Fortney J. J., Marley M. S., Visscher C., Saumon D., Leggett S. K., 2012, , 756, 172

Rousselot P., Lidman C., Cuby J. G., Moreels G., Monnet G., 2000, , 354, 1134

Saumon D., Marley M. S., 2008, , 689, 1327

Suárez G., Metchev S., Leggett S. K., Saumon D., Marley M. S., 2021, , 920, 99

Tannock M. E., et al., 2021, , 161, 224

Tannock M. E., Metchev S., Hood C. E., Mace G. N., Fortney J. J., Morley C. V., Jaffe D. T., Lupu R., 2022, , 514, 3160

Villanueva G. L., Smith M. D., Protopapa S., Faggi S., Mandell A. M., 2018, , 217, 86

Wall J. V., Jenkins C. R., 2003, Practical Statistics for Astronomers. Cambridge Observing Handbooks for Research Astronomers, Cambridge University Press, doi:10.1017/CBO9780511536618