
gollum

Release 0.2.2

gully


Jan 03, 2023

GETTING STARTED

1	Installing the development version	3
2	Downloading model grids	5
3	<code>gollum</code> Quickstart	7
4	<code>gollum</code> tutorials	9
5	API	21

The goal of this repo is to provide a Python Application Programming Interface (API) to several different synthetic spectral models. *gollum* will be built on the astropy affiliated package [specutils](#), and will be inspired by the API design of [lightkurve](#). This project is loosely related to the parallel [muler](#) framework that is built on [specutils](#) and focuses on data. This project is all about models. The code itself and will have some overlap with functionality in [Starfish](#), and this project could one day become a microservice to Starfish, rather than duplicate code.

gollum makes it easy to chain routine operations in sequence:



```
from gollum.phoenix import PHOENIXSpectrum
model = PHOENIXSpectrum(teff=4100,logg=4).rotationally_broaden(28.8).rv_shift(12.4)\
        .instrumental_broaden(55000).resample(data)
```


INSTALLING THE DEVELOPMENT VERSION

Note: Conda installation is not yet available.

Pip installation is available

```
pip install gollum
```

Currently we recommend installing the developer version from source to help us with beta testing. You can install gollum into its own isolated conda python environment using the instructions below. Line 3 will create a conda environment titled *gollum_dev* that contains all the dependencies required to install *gollum*, and even all the affiliated code to make the documentation and run unit tests from scratch. Very useful! You can activate this conda environment using the code on line 4 *conda activate gollum_dev*. You could alternatively activate some other existing conda environment on your computer, but it is not guaranteed to have all of the extra packages. Our sibling package *muler* offers an identical conda environment to this one, so if you made one for muler already, you can do: *conda activate muler_dev*

```
git clone https://github.com/BrownDwarf/gollum.git
cd gollum
conda env create -f environment.yml
conda activate gollum_dev # or conda activate muler_dev if you have that already
python setup.py develop
```

Eventually you can run the tests in the *tests/* directory to double-check that everything installed correctly. Currently we are evaluating the best way to specify paths to voluminous libraries to make testing robust and quick across machines. Stay tuned!

```
py.test -vs test_precomputed.py # should work for everyone
py.test -vs test_phoenix.py # requires local PHOENIX models
py.test -vs test_sonora.py # requires local Sonora models
```

1.1 Requirements

The project appears to work on modern Mac OS, Linux, and Windows operating systems, and has been tested for Python 3.7 and above. It may work on Python 3.6, and will not work on Python 2.

DOWNLOADING MODEL GRIDS

2.1 Sonora

Currently we only support the 2018 Sonora Model grids. In a web browser, navigate to the [Sonora 2018 Zenodo website](#). Click on the Download button next to `spectra.tar` 1.2 GB. That will download the `spectra.tar` file to your `~/Downloads/` directory or equivalent default location. Untar that directory. The pattern should look something like below, depending on what your operating system defaults are, so copy-ing and past-ing this code is not likely to work— try each step at a time.

```
tar -xzf spectra.tar
mv spectra ~/libraries/raw/Sonora
```

The end result should look something like this:

```
$ ls ~/libraries/raw/Sonora/
[...]
```

<code>sp_t2100g3160nc_m0.0.gz</code>	<code>sp_t450g316nc_m0.0.gz</code>	<code>sp_t950g31nc_m0.0.gz</code>
<code>sp_t2100g316nc_m0.0.gz</code>	<code>sp_t450g31nc_m0.0.gz</code>	<code>sp_t950g562nc_m0.0.gz</code>
<code>sp_t2100g31nc_m0.0.gz</code>	<code>sp_t450g562nc_m0.0.gz</code>	<code>sp_t950g56nc_m0.0.gz</code>

```
[...]
```

2.2 PHOENIX

Note: Downloading all the PHOENIX models can take hours or days! Start the downloading early.

The PHOENIX models total over 100 GB, and generally download at a relatively slow bandwidth. The files are arranged into sub-directories for metallicity and alpha-element abundance.

You can get all of the PHOENIX models in one-fell-swoop from the command line if you have *wget*

```
cd ~/Downloads
wget -r -l 0 ftp://phoenix.astro.physik.uni-goettingen.de/HiResFITS/
```

If you don't have *wget* on your computer, please help the *gollum* grow by filing a GitHub Issue with how you resolved the problem, or what problems you are encountering.

As noted, this process will take a while as each individual file is painstakingly downloaded from a single German computer. The commandline script, as written, will preserve the directory structure— that's good! The *gollum* code demands that the directory structure is preserved. Once it's all downloaded it should look like this

```
# From inside this directory: ~/Downloads/phoenix.astro.physik.uni-goettingen.de/
$ tree
.
├── HiResFITS
│   ├── PHOENIX-ACES-AGSS-COND-2011
│   │   ├── Z-0.0
│   │   │   ├── lte02300-0.00-0.0.PHOENIX-ACES-AGSS-COND-2011-HiRes.fits
│   │   │   ├── lte02300-0.50-0.0.PHOENIX-ACES-AGSS-COND-2011-HiRes.fits
│   │   │   ├── lte02300-1.00-0.0.PHOENIX-ACES-AGSS-COND-2011-HiRes.fits
│   │   │   ├── lte02300-1.50-0.0.PHOENIX-ACES-AGSS-COND-2011-HiRes.fits
│   │   │   └── [ ... ]
│   │   ├── lte04000-2.50-0.0.PHOENIX-ACES-AGSS-COND-2011-HiRes.fits
│   │   ├── lte04000-3.00-0.0.PHOENIX-ACES-AGSS-COND-2011-HiRes.fits
│   │   ├── lte04000-3.50-0.0.PHOENIX-ACES-AGSS-COND-2011-HiRes.fits
│   │   └── [ ... ]
│   ├── Z-0.0
│   ├── Z+0.5
│   ├── Z-0.5
│   ├── Z+1.0
│   ├── Z-1.0
│   ├── Z-1.5
│   ├── Z-2.0
│   ├── Z-3.0
│   └── Z-4.0
└── WAVE_PHOENIX-ACES-AGSS-COND-2011.fits
```

While you *could* leave these directories in say `~/Downloads/phoenix.astro.physik.uni-goettingen.de/`, I recommend making a more permanent and recognizable home for these models. In particular *gollum* attempts to search a single default path for models: `~/libraries/raw/`, where the tilde `~/` denotes your home directory

```
mv ~/Downloads/phoenix.astro.physik.uni-goettingen.de/HiResFITS/PHOENIX-ACES-AGSS-COND-
  ↳ 2011 ~/libraries/raw/PHOENIX/
ls ~/libraries/raw/PHOENIX/
Z-0.0  Z+0.5  Z-0.5  Z+1.0  Z-1.0  Z-1.5  Z-2.0  Z-3.0  Z-4.0
```

Finally, you must copy the wavelength file into this directory as well. Notice that this placement breaks the native directory structure, so you must complete this step in order for *gollum* to work.

```
mv ~/Downloads/phoenix.astro.physik.uni-goettingen.de/HiResFITSWAVE_PHOENIX-ACES-AGSS-
  ↳ COND-2011.fits ~/libraries/raw/PHOENIX/
ls ~/libraries/raw/PHOENIX/
WAVE_PHOENIX-ACES-AGSS-COND-2011.fits  Z-0.0  Z+0.5  Z-0.5  Z+1.0  Z-1.0  Z-1.5  Z-2.0  ↳
  ↳ Z-3.0  Z-4.0
```

GOLLUM QUICKSTART

```
[1]: %config InlineBackend.figure_format='retina'
```

```
[2]: from gollum.phoenix import PHOENIXSpectrum
```

Simply provide the T_{eff} and $\log g$ values you desire:

```
[3]: spec = PHOENIXSpectrum(teff=5000, logg=4)
```

Normalize the spectrum by the median:

```
[4]: normalized_spectrum = spec.normalize()
```

The spectrum has wavelength, with units:

```
[5]: normalized_spectrum.wavelength
```

```
[5]: [8038.01, 8038.02, 8038.03, ..., 12848.94, 12848.96, 12848.98] Å
```

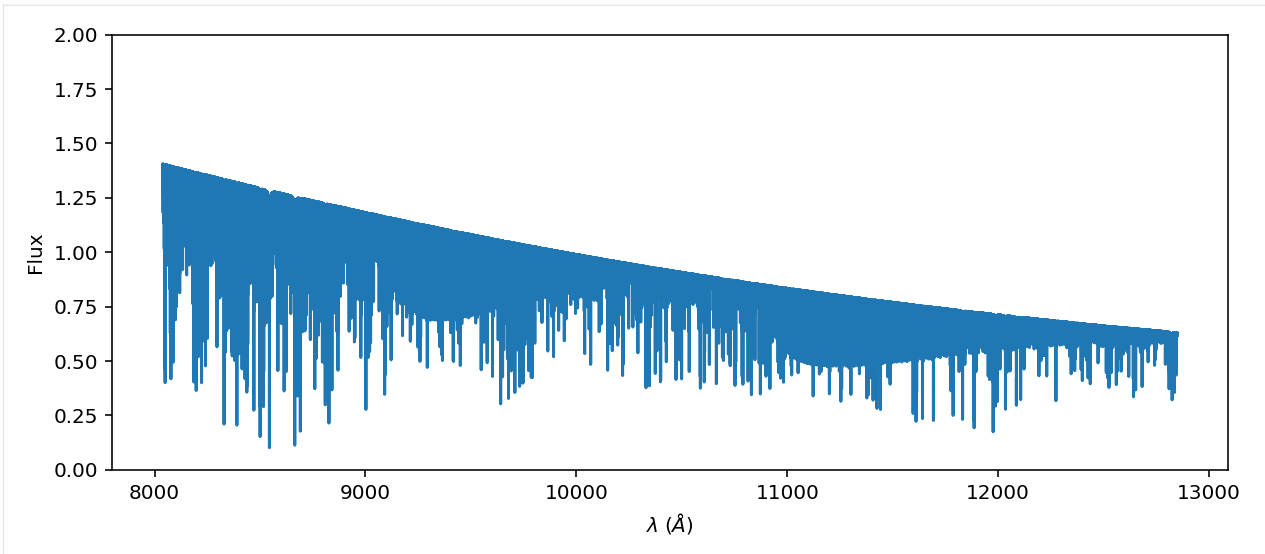
The flux vector is the same length as wavelength:

```
[6]: normalized_spectrum.flux.dtype, normalized_spectrum.shape
```

```
[6]: (dtype('float64'), (338649,))
```

```
[7]: ax = normalized_spectrum.plot()  
ax.set_ylim(0, 2)
```

```
[7]: (0.0, 2.0)
```

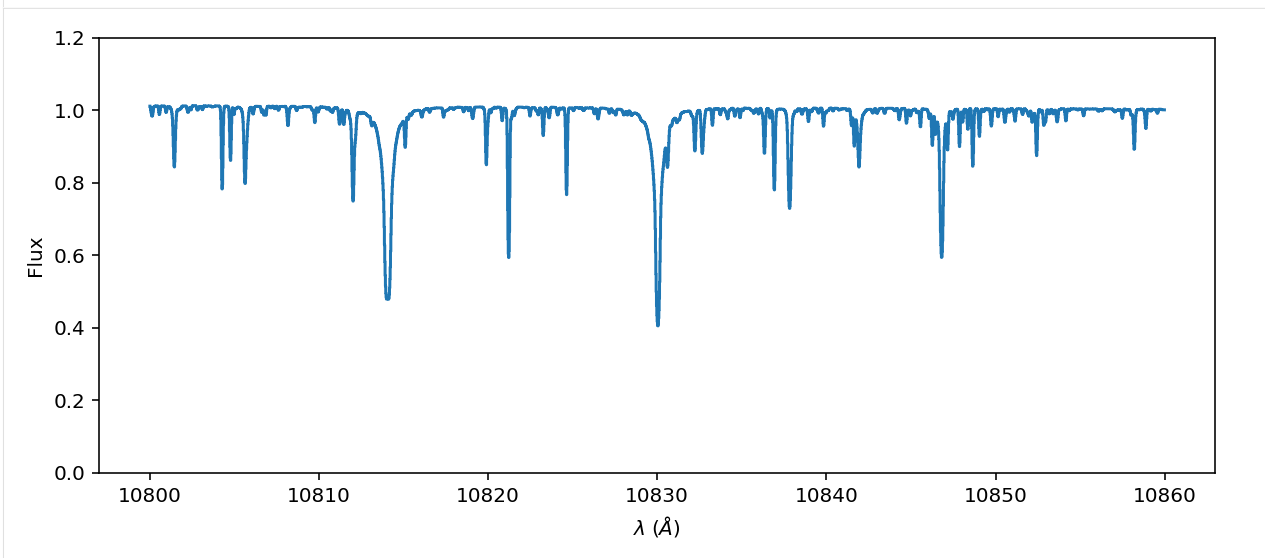


Right now we truncate the spectrum to the near-IR by default, you can change that with keyword arguments from the beginning:

```
[8]: spec = PHOENIXSpectrum(teff=5000,logg=4,wl_lo=10_800,wl_hi=10_860)
```

```
[9]: ax = spec.normalize().plot()
    ax.set_ylim(0)
```

```
[9]: (0.0, 1.2)
```



Neat! **gollum** is still under active development. Help us develop the tool by engaging with us on our [GitHub Issues page](#). You can suggest a feature, or help us brainstorm how to build this project more. Thanks!

GOLLUM TUTORIALS

4.1 Working with the PHOENIX models

The PHOENIX models are under active development. Check back soon!

```
[1]: from gollum.phoenix import PHOENIXSpectrum
import numpy as np
%config InlineBackend.figure_format='retina'
```

```
[2]: spec = PHOENIXSpectrum(teff=2800, logg=5.0)
```

```
[3]: spec.wavelength # "Angstroms"
```

```
[3]: [8038.01, 8038.02, 8038.03, ..., 12848.94, 12848.96, 12848.98] Å
```

```
[4]: spec.flux.unit # "ergs / s / cm^2 / cm"
```

```
[4]:  $\frac{\text{erg}}{\text{s cm}^3}$ 
```

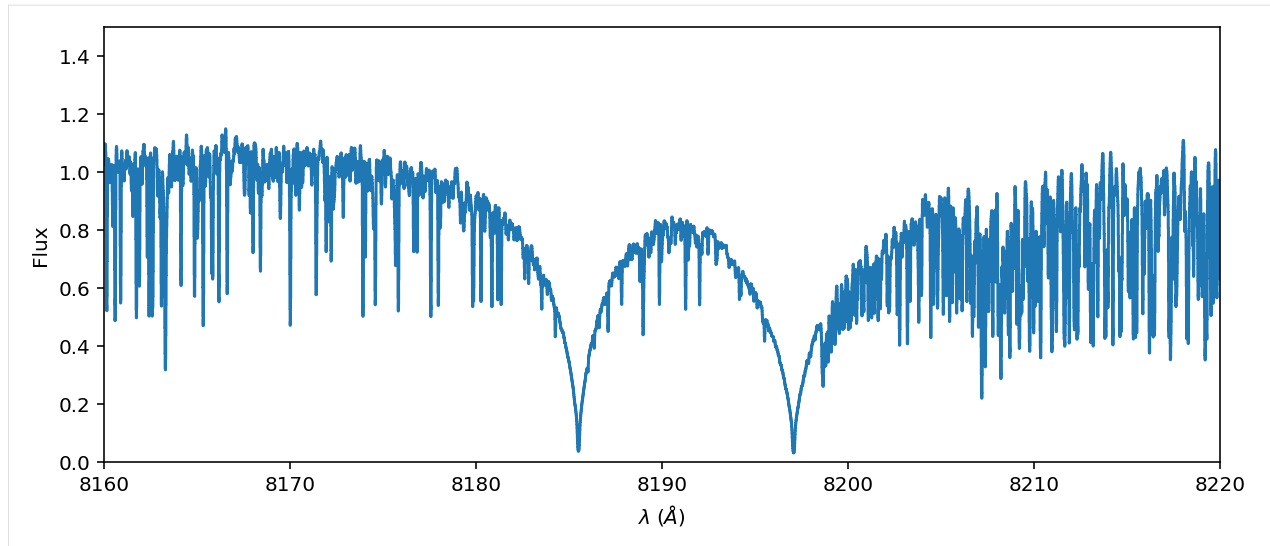
We can normalize the spectrum, which divides the spectrum by the median value, rendering the flux units *dimensionless*

```
[5]: spectrum = spec.normalize()
np.median(spectrum.flux)
```

```
[5]: 1
```

```
[6]: ax = spectrum.plot()
ax.set_xlim(8160, 8220)
ax.set_ylim(0, 1.5)
```

```
[6]: (0.0, 1.5)
```



Neat! There are many great uses for the PHOENIX models. We will be adding support for other model grids in the future.

4.2 Finding the best fit rotational broadening and radial velocity

Often in astronomy we have a data spectrum and we want to answer the question:

What is the best fit $v \sin i$ and RV for this data?

In this demo we will show some simple ways to use gollum to find the model with the best fit rotational broadening and radial velocity, assuming a fixed template.

```
[1]: from gollum.phoenix import PHOENIXSpectrum, PHOENIXGrid
import numpy as np
import astropy.units as u
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
%config InlineBackend.figure_format='retina'
```

4.2.1 For this demo we will need some real world example data

Let's use data of an A0V star from HPF and our sibling package muler.

You can get free example data from the [muler example data GitHub repository](#).

```
[2]: from muler.hpf import HPFSpectrumList, HPFSpectrum
```

```
[3]: local_file = '../..../muler_example_data/HPF/01_A0V_standards/Goldilocks_
      ↪ 20210517T054403_v1.0_0060.spectra.fits'
raw_data = HPFSpectrumList.read(local_file)
```

```
[4]: def data_clean(data):
    """Clean the HPF data with standard post-processing techniques"""
    data = data.sky_subtract(method='vector')
    data = data.deblaze()
    data = data.trim_edges((4, 2042))
    data = data.normalize()
    data = data.stitch()
    return data
```

```
[5]: full_data = data_clean(raw_data)
```

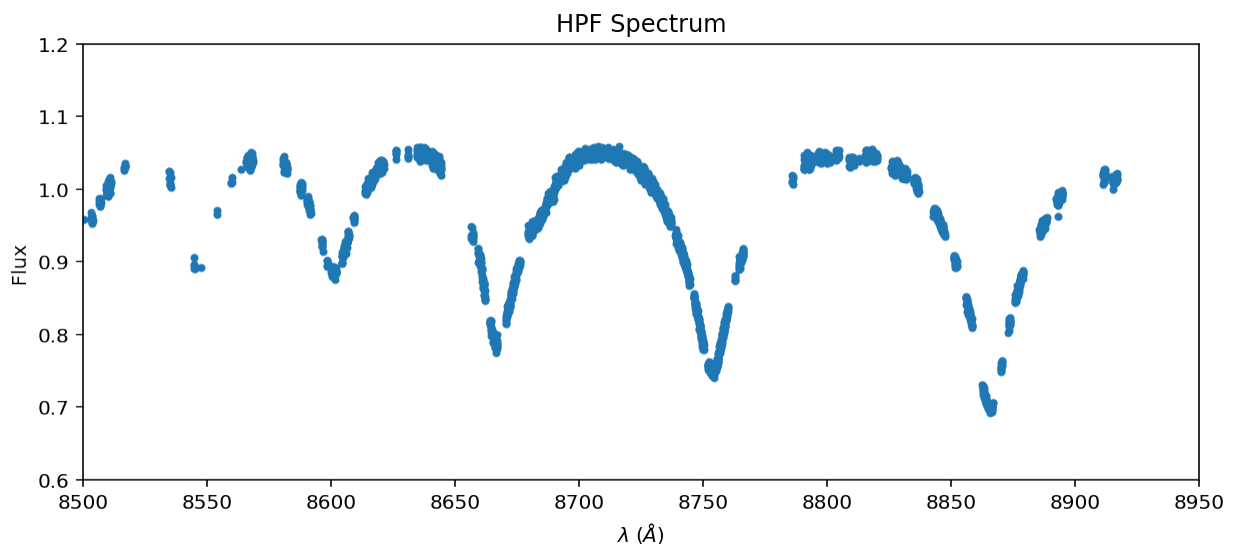
As a final step, we will mask the telluric absorption lines. This step can sometimes benefit from hand-tuning.

```
[6]: data = full_data.mask_tellurics(threshold=0.999, dilation=13)
```

We will restrict our fits to the region with the highest density of H lines.

```
[7]: data = data[8500*u.Angstrom:8950*u.Angstrom]
data = data.normalize()
```

```
[8]: ax = data.plot(marker='.', linestyle='None');
ax.set_xlim(8500, 8950);
```



OK, that's our data spectrum against which we will compare models. You can see large voids in the spectrum due to our telluric masking— that's fine, the data need not be contiguous or evenly sampled to estimate a best fit model. We will resample the model to the data.

We can choose 3 dimensions in our grid: T_{eff} , $\log g$, $\left[\frac{\text{Fe}}{\text{H}}\right]$

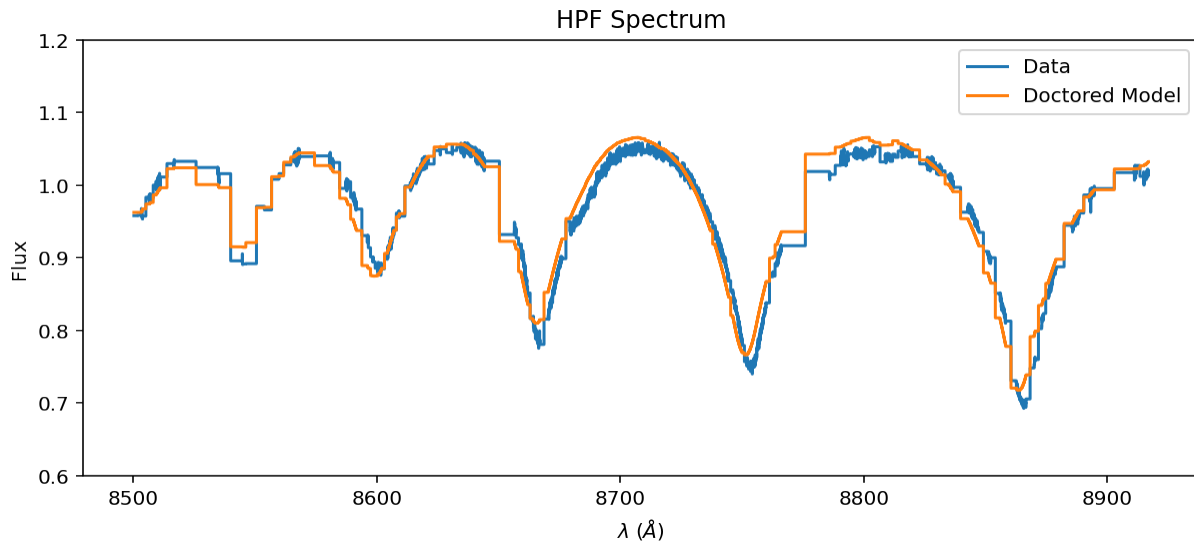
```
[9]: template = PHOENIXSpectrum(teff=9_600, logg=4.5, metallicity=0)
```

We then want to rotationally broaden and RV shift the spectrum. We'll try some guesses first:

```
[10]: guess_vsini = 150
guess_RV = -50
```

```
[11]: spec = template.rotationally_broaden(guess_vsini)\
      .rv_shift(guess_RV)\
      .instrumental_broaden(resolving_power=55_000)\
      .resample(data)\
      .normalize()
```

```
[12]: ax = data.plot(label='Data')
spec.plot(ax=ax, label='Doctored Model')
ax.legend();
```



You can see that our guess RV is off. Let's do a grid search for $v \sin i$ and RV.

```
[13]: n_vsini = 20
n_RVs = 20
vsinis = np.linspace(1, 150.0, n_vsini)
RVs = np.linspace(-100.0, 100.0, n_RVs)
```

```
[14]: loss_values = np.zeros((n_vsini, n_RVs))
```

We will compute the sum-of-the-squares-of-the-residuals ("chi-squared") for each value of $v \sin i$ and RV.

```
[15]: for i, vsini in tqdm(enumerate(vsinis), total=n_vsini):
    for j, RV in enumerate(RVs):
        spec = template.rotationally_broaden(vsini)\
              .rv_shift(RV)\
              .instrumental_broaden(resolving_power=55_000)\
              .resample(data)\
              .normalize()
        residual = data.subtract(spec, handle_meta='ff')
        chi2_loss = np.sum(0.5*residual.flux.value**2/residual.uncertainty.array**2)

        loss_values[i,j] = chi2_loss
```

```
0%|          | 0/20 [00:00<?, ?it/s]
```

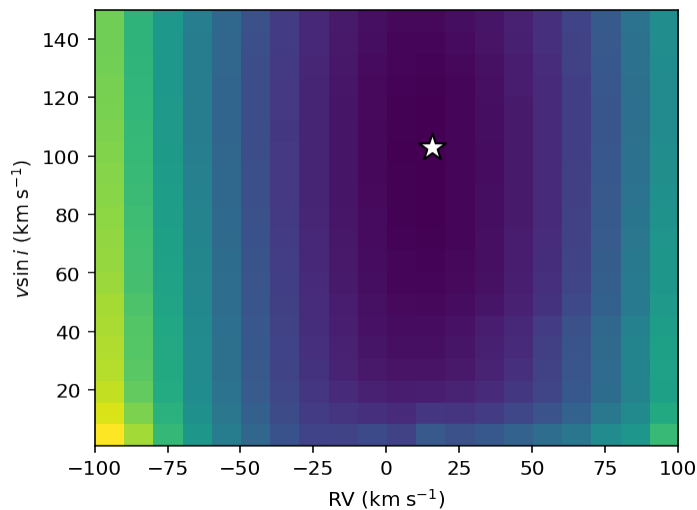
The best fit values minimizes the chi-squared.


```
[16]: best_i, best_j = np.unravel_index(np.argmin(loss_values), loss_values.shape)
```

```
[17]: best_vsini, best_RV = vsinis[best_i], RVs[best_j]
      best_RV, best_vsini
```

```
[17]: (15.789473684210535, 102.94736842105263)
```

```
[18]: extent=[RVs.min(), RVs.max(), vsinis.min(), vsinis.max()]
      plt.imshow(loss_values, extent=extent, aspect=1, origin='lower')
      plt.scatter(best_RV, best_vsini, marker='*', c='w', ec='k', s=200)
      plt.xlabel('RV ( $\mathrm{km}\mathrm{s}^{-1}$ )')
      plt.ylabel('$v\sin i$ ( $\mathrm{km}\mathrm{s}^{-1}$ )');
```

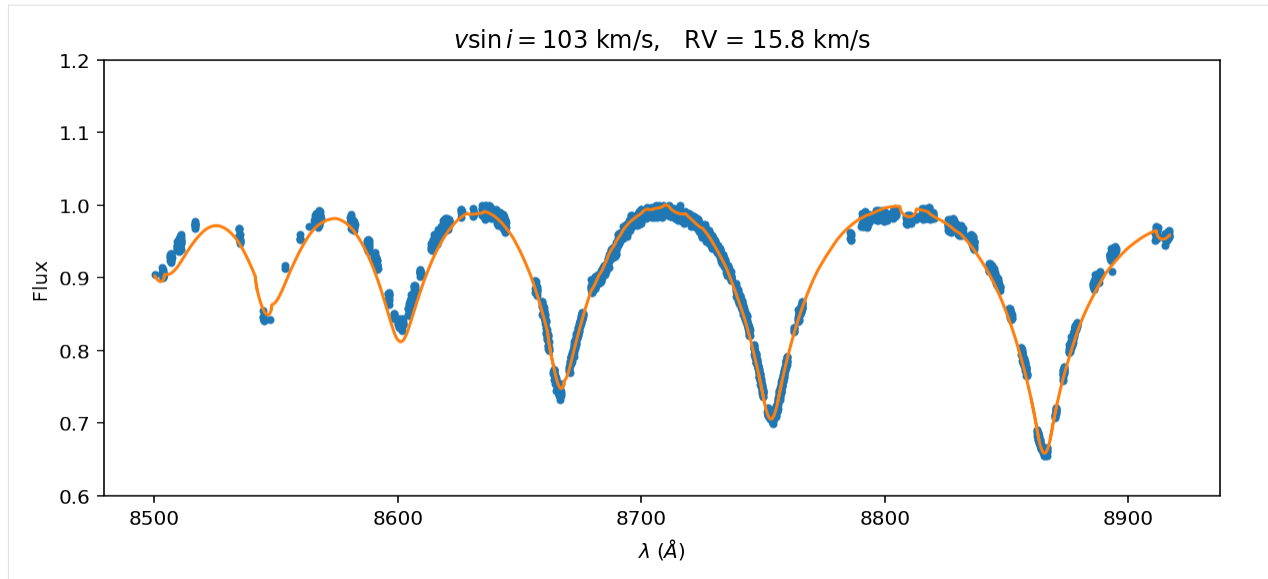


Awesome, we have found the RV and $v \sin i$ with the closest match to the data.

```
[19]: best_spec_full = template.rotationally_broaden(best_vsini)\
      .rv_shift(best_RV)\
      .instrumental_broaden(resolving_power=55_000)
      best_spec = best_spec_full[data.wavelength.min():data.wavelength.max()].normalize()
```

How does the best fit look by-eye?

```
[20]: ax=(data/data.flux.max()).plot(marker='.', linestyle='None')
      (best_spec/best_spec.flux.max()).plot(ax=ax)
      title1 = '$v\sin i$ = $'+ '{:0.0f}'.format(best_vsini)+' km/s, '
      title2 = 'RV = '+ '{:0.1f}'.format(best_RV)+' km/s'
      title = title1 + title2
      ax.set_title(title);
```



Excellent! That's much better than our initial guess.
It's still not *perfect* because the model template has imperfections.

4.3 Divide by a black body

Often in real-world astronomical applications, we want to see how a stellar spectrum varies in wavelength without the large and omnipresent crest of the black body curve.

In this tutorial we show how to remove the black body curve from a precomputed model spectrum.

```
[1]: from gollum.phoenix import PHOENIXSpectrum
    from gollum.precomputed_spectrum import PrecomputedSpectrum
    from astropy.modeling.physical_models import BlackBody
    import astropy.units as u
    import numpy as np
    %config InlineBackend.figure_format='retina'
```

```
[2]: T_eff = 5_700
```

```
[3]: original_spectrum = PHOENIXSpectrum(teff=T_eff, logg=4.5, metallicity=0,
    wl_lo=0, wl_hi=1e10) # Get the whole spectrum
```

The PHOENIX spectra have units of $\frac{\text{erg}}{\text{s cm}^2 \text{ cm}}$

```
[4]: original_spectrum.flux.unit
```

```
[4]:  $\frac{\text{erg}}{\text{s cm}^3}$ 
```

```
[5]: original_spectrum.flux.min()
```

```
[5]:  $5.5922428 \times 10^{-15} \frac{\text{erg}}{\text{s cm}^3}$ 
```

4.3.1 Make a black body spectrum with the same temperature

```
[6]: blackbody_model = BlackBody(temperature=T_eff*u.Kelvin)
     blackbody_flux_per_sr = blackbody_model(original_spectrum.wavelength)
```

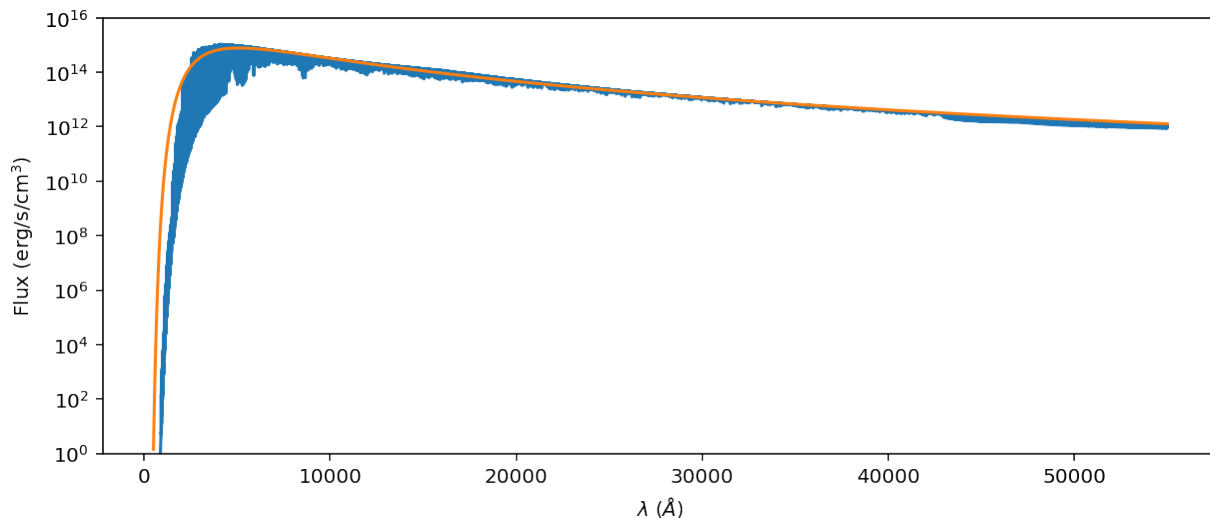
The Black Body spectra have units of $\frac{\text{erg}}{\text{s cm}^2 \text{ Hz sr}}$. To convert between the two conventions we have to do two things. First we have to pretend we are standing on the surface of the star, and multiply by π steradians. Second, we have to convert flux density in per frequency to per wavelength units. We can do that with astropy's equivalencies kwarg.

```
[7]: blackbody_flux_per_Hz = blackbody_flux_per_sr * np.pi * u.steradian
```

```
[8]: blackbody_flux = blackbody_flux_per_Hz.to(original_spectrum.flux.unit,
        equivalencies=u.spectral_density(original_spectrum.wavelength))
```

```
[9]: blackbody = PrecomputedSpectrum(flux=blackbody_flux,
        spectral_axis=original_spectrum.wavelength)
```

```
[10]: ax = original_spectrum.plot()
      blackbody.plot(ax=ax)
      ax.set_ylim(1e0, 1e16)
      ax.set_yscale('log')
      ax.set_ylabel('Flux (erg/s/cm$^3$)');
```



The plot spans 16 orders of magnitude— a huge dynamic range! Notice that the spectra have similar—but not identical—broadband spectral shapes. They should have the identical area under the curve, by the definition of effective temperature. Let's see if they do!

```
[11]: from scipy.integrate import trapezoid
```

```
[12]: original_flux = trapezoid(original_spectrum.flux, x=original_spectrum.wavelength.to(u.
    →cm))
      black_body_flux = trapezoid(blackbody.flux, x=original_spectrum.wavelength.to(u.cm))
      original_flux/black_body_flux
```

```
[12]: 0.99048532
```

The two spectral models have the same flux to within 1%, which is close enough to identical given that more spectrum resides outside the extent of the plot, and numerical artifacts limitations in the spectral modeling procedure. Let's compute the ratio spectrum to see how flat the spectrum looks. We'll first plot it over the same dynamic range as before to emphasize how much more compressed it is.

```
[13]: ratio_spectrum = original_spectrum.divide(blackbody)
```

The resulting spectrum is a ratio of fluxes with the same units, so it is dimensionless.

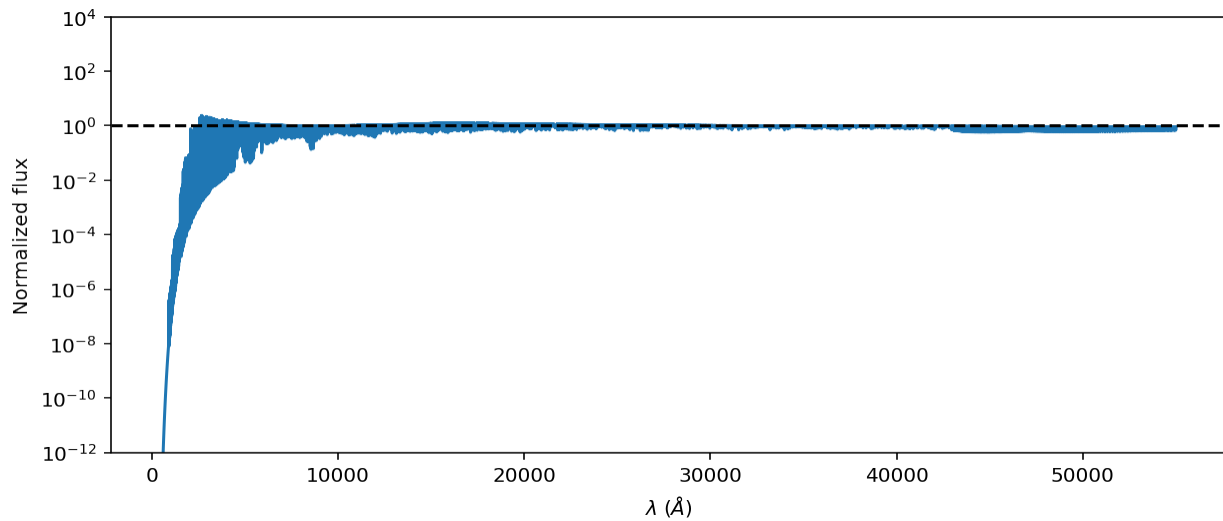
```
[14]: ratio_spectrum.flux.unit == u.dimensionless_unscaled
```

```
[14]: True
```

```
[15]: ax = ratio_spectrum.plot();

ax.set_ylim(1e-12, 1e4)
ax.set_yscale('log')
ax.set_ylabel('Normalized flux');

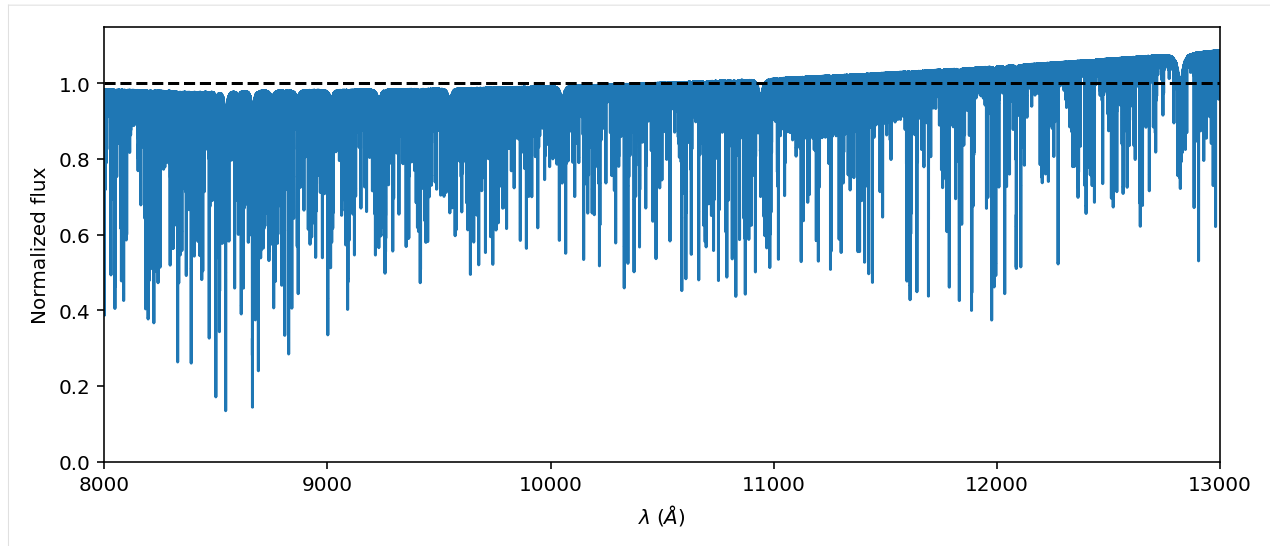
ax.axhline(1.0, linestyle='dashed', color='k');
```



At this dramatic zoom level, the flux looks pretty flat (except for the extreme ultraviolet portion of the spectrum). Let's zoom in on a region of interest from 8000 – 13000 .

```
[16]: ax = ratio_spectrum.plot(ylo=0, yhi=1.15);
ax.set_ylabel('Normalized flux');

ax.axhline(1.0, linestyle='dashed', color='k');
ax.set_xlim(8_000, 13_000);
```



OK, looks good! We have successfully used the blackbody curve to coarsely flatten the spectrum!

4.4 Interactively fit Brown Dwarf Spectra with the gollum dashboard

In this tutorial we will see how the spectra of brown dwarfs vary as a function of their intrinsic properties. We will fit observed spectra of a particular brown dwarf with the gollum dashboard, a dashboard which fits models based on properties including [effective temperature](#), [surface gravity](#), [metallicity](#), [rotational broadening](#), and [radial velocity](#). The fitting for this tutorial will be based on the Sonora-Bobcat 2021 models, which takes into account effective temperature, surface gravity, and metallicity as intrinsic values.

```
[ ]: from gollum.sonora import SonoraGrid
     from specutils import Spectrum1D
     import pandas as pd
     import astropy.units as u
```

```
[ ]: from IPython.display import HTML
     from IPython.display import Image
```

First, we will read in an example spectrum of this [ultracool dwarf](#):

2MASS J05591914-1404488

We got its data from the Keck Telescope’s [NIRSPEC spectrograph](#). A specific section of this data is displayed below.

```
[ ]: df = pd.read_csv('../data/2mass0559_59.dat',
                     delim_whitespace=True,
                     comment='#',
                     names=['wave', 'flux'])
```

```
[ ]: df.head()
```

The unit for wavelength here is [microns](#) and the unit for flux is “counts”.

```
[ ]: bdss_spectrum = Spectrum1D(spectral_axis=df.wave.values*u.micron,
                               flux=df.flux.values*u.ct)
```

```
[ ]: wl_lo, wl_hi = (bdss_spectrum.wavelength.value.min(),
                    bdss_spectrum.wavelength.value.max())
```

Next, we can read in the Sonora-Bobcat grid and show an interactive dashboard.

```
[ ]: grid = SonoraGrid(wl_lo=wl_lo, wl_hi=wl_hi)
```

Awesome! Now you can hand-in a data spectrum to overlay it onto the grid and begin fitting using the interactive sliders.

```
[ ]: grid.show_dashboard(data=bdss_spectrum, show_telluric=False)
```

The dashboard looks great!

4.5 Demo of the PHOENIX Interactive Dashboard for Stellar Spectra

In this tutorial we will see how the spectra of stars vary as a function of their intrinsic properties.

```
[ ]: from gollum.phoenix import PHOENIXGrid
     from specutils import Spectrum1D
     import pandas as pd
     import astropy.units as u
     import numpy as np
     from IPython.core.display import display, HTML
     display(HTML("<style>.container { width:100% !important; }</style>"))
```

4.5.1 Fetch example IGRINS data

```
[ ]: from muler.igrins import IGRINSSpectrum
```

```
[ ]: path = 'https://github.com/OttoStruve/muler_example_data/raw/main/IGRINS/01_IGRINS_test_
     ↪data/'
     filename='SDCH_20201202_0059.spec_a0v.fits'
     full_path = path + filename
```

```
[ ]: spec = IGRINSSpectrum(file=full_path, order=12).normalize().remove_nans().trim_edges()
     spec.plot(color=None, ylo=0.7, yhi=1.1);
```

4.5.2 Telling gollum where to find your local files

The online installation guide shows [how to download and store the PHOENIX models](#). In order to run the code below, you will need to update the `my_path` variable to the correct path on your local machine.

In this demo, the PHOENIX/ folder is houses all of the voluminous models:

```
[ ]: grid = PHOENIXGrid(teff_range=(2500, 7000), logg_range=(2, 5), metallicity_range=(0, 0.
     ↪5), wl_lo= 16200, wl_hi= 16400)
```

```
[ ]: grid.show_dashboard(data=spec)
```


API

The API is under active development. Feedback is welcomed.