wobble Documentation

Release 0.0.1

Megan Bedell

Jan 19, 2021

Contents

wobble

wobble (pronounced *(/wäb.lā/)*) is an open-source python implementation of a data-driven method for deriving stellar spectra, telluric spectra, and extremely precise radial velocities simultaneously without reliance on spectral models.

Read the paper on arXiv.

CHAPTER 1

Quickstart

1.1 Installation

wobble is currently under development and not yet available through pip. However, you can install the current developer version from GitHub:

```
git clone https://github.com/megbedell/wobble.git
cd wobble
python setup.py develop
```

If you are running macOS 10.14 (Mojave) and find that setup fails at building the wobble C++ extensions, try this.

You may also need to install some requirements, notably TensorFlow (which is best installed via pip rather than conda):

```
pip install -r requirements.txt
```

If *wobble* builds with warnings and subsequently fails with a "Symbol not found" error on import, try rebuilding the C++ extensions using a different system compiler; see this issue.

1.2 Input Data

Running *wobble* on a generic data set will require some initial processing.

If you just want some example data for testing purposes, try grabbing one of these pre-processed data sets: 51 Peg, Barnard's Star.

These data sets can be easily loaded as:

```
import wobble
data = wobble.Data('filename.hdf5')
```

Assembling your own data set is possible via the built-in functions for HARPS, HARPS-N, HIRES, or ESPRESSO spectra, or by interactively passing pre-processed data. See the *API* for full documentation of data loading options.

1.3 Running *wobble*

The following is an overview of the basic steps involved in running an analysis with *wobble*. If you want a simple example script, check this demonstration Jupyter notebook, designed to be used with the 51 Peg example data.

1. Load the data.
2. Create a wobble.Results object in which to store the calculation outputs.
3. Create a wobble.Model object.

Note: Data and Results objects span all echelle orders for the input spectra (or at least as many echelle orders as you specified in the *orders* keyword when loading Data), but Model is specific to a single order.

---

4. Populate the model with one or more spectral components. Model.add_star() and Model.add_telluric() are convenience functions for this purpose.
5. Optimize the model.

Warning: The optimization scheme in *wobble* depends on L1 and L2 regularization. The default values used by the model should work fine for most HARPS data, but if you are using a custom data set from a different instrument or if you see unexpected behavior in the spectral fits then the regularization amplitudes should be tuned before *wobble* can be run reliably. See this Jupyter notebook for an example of how to do this.

6. Repeat steps 3-5 as needed for additional echelle orders.
7. (Optional) Do post-processing within the results, including combining multiple orders to get a net RV time series; applying barycentric corrections; and applying instrumental drifts as available.

Note: Currently *wobble* does not support barycentric correction calculations; those should be supplied as a part of the input data. Also, the methods of combining orders and applying barycentric shifts may be sub-optimal.

---

8. Save the results and/or write out RVs for further use.

Here's an example of what that code might look like. Again, check out the demo notebook for a slightly more detailed walkthrough, or *the API* for advanced usage.

```
results = wobble.Results(data=data)
for r in range(len(data.orders)): # loop through orders
        model = wobble.Model(data, results, r)
        model.add_star('star')
        model.add_telluric('tellurics')
        wobble.optimize_order(model)

results.combine_orders('star') # post-processing: combine all orders
results.apply_bervs('star') # post-processing: shift to barycentric frame
results.apply_drifts('star') # post-processing: remove instrumental drift

results.write_rvs('star', 'star_rvs.csv') # save just RVs
results.write('results.hdf5') # save everything
```

1.4 Accessing *wobble* Outputs

All of the outputs from *wobble* are stored in the wobble.Results object. You can download example results files corresponding to the above data files here: 51 Peg, Barnard's Star.

A saved wobble.Results object can be loaded up from disk:

```
results = wobble.Results(filename='results.hdf5')
print(results.component_names)
```

The names of the components are needed to access the associated attributes of each component. For example, let's say that two components are called 'star' and 'tellurics,' as in the example above. We can plot the mean templates for the two components in order *r* as follows:

```
import matplotlib.pyplot as plt
plt.plot(np.exp(results.star_template_xs[r]), np.exp(results.star_template_ys[r]),
                 label='star')
plt.plot(np.exp(results.tellurics_template_xs[r]), np.exp(results.tellurics_template_
↪ys[r]),
                label='tellurics')
plt.xlabel('Wavelength (Ang)')
plt.ylabel('Normalized Flux')
plt.legend()
plt.show()
```

And the RV time series can be plotted as follows:

```
plt.errorbar(results.dates, results.star_time_rvs,
                        results.star_time_sigmas, 'k.')
plt.xlabel('RV (m/s)')
plt.ylabel('JD')
plt.show()
```

Other useful quantities stored in the Results object include *results.ys_predicted*, which is an order R by epoch N by pixel M array of *y'* model predictions in the data space, and *results.[component name]_ys_predicted*, which is a same-sized array storing the contribution of a given component to the model prediction.

See the demo Jupyter notebook or the notebook used to generate figures for the paper for further examples.

CHAPTER 2

wobble API

2.1 Data

2.2 Model

2.3 Results

2.4 Regularization & Other Utilities

CHAPTER 3

Helper Scripts

A few python scripts are included in the *scripts* subdirectory. These scripts are not part of the core *wobble* functionality but may be helpful as guides for doing a few common operations related to *wobble*.

3.1 Tuning Regularization Parameters

Regularization is an important part of *wobble*'s functionality. To get the best possible results for any given data set, the regularization strengths should be tuned to suit these data. We do this with a cross-validation scheme.

The tune_regularization.py script shows an example of generating new regularization parameter files and tuning them for a given data set via cross-validation.

More details are given under the API.

3.2 Running *wobble*

Once the data have been processed and the regularization amplitudes have been set, running *wobble* should be a straightforward procedure. An overview can be found in the Quickstart section. As further examples, the scripts used to run the three analyses performed in the paper can be found under script_51peg.py, script_barnards.py, and script_HD189733.py.

CHAPTER 4

Bug Reports & Questions

wobble is an open source project under the MIT license. The source code is available on GitHub. In case of any questions or problems, please contact us via the Git Issues.