TelFit Manual

Kevin Gullikson

March 31, 2014

1 Installation

This section will describe the installation procedure for the TelFit code. It is tailored for linux users, although it should work fine on Mac OSX. This program has not been tested on Windows machines, and likely requires modification to work.

1.1 Dependencies

There are several dependencies for TelFit, most of which are python packages that can be installed using the python package manager such as 'pip.' The python packages needed are listed below.

1. numpy v1.6 or greater
2. cython
3. matplotlib (used for debugging)
4. scipy
5. astropy (used for unit conversion and some constants)
6. lockfile (allows several instances of TelFit to be run at once)
7. pysynphot v0.7 or greater (Used to rebin the telluric model)
8. fortranformat (Used to make the input file for LBLRTM with the correct formatting)

The setup.py script (see Section 1.2) requires both numpy and cython, so they must be installed before this program. The setup.py script will attempt to find and install the rest of the dependencies, but I strongly recommend installing at least numpy, scipy, and matplotlib before installing TelFit if you do not have them already. If you are new to Python, the Anaconda distribution (`http://continuum.io/downloads`) has all of these packaged nicely.

TelFit also requires the LBLRTM version 12.2 telluric modeling code (in fact, this code is just a wrapper to LBLRTM). It is available from `http://rtweb.`

aer.com/lblrtm.html The package available contains LBLRTM, which does the telluric modeling, a HITRAN molecular line list, and LNFL, which converts the ascii line list into a binary version suitable for LBLRTM. If you downloaded the TelFit source code from http://www.as.utexas.edu/~kgulliks/projects.html, then the LBLRTM source code is included. However, if you installed from my git repository (https://github.com/kgullikson88/Telluric-Fitter), then you will have to put the following tarfiles into the top-level directory of TelFit (alongside setup.py).

- aerlbl_v12.2.tar.gz
- aer_v_3.2.tar.gz
- aerlnfl_v2.6.tar.gz

The setup.py script will handle the installation of LBLRTM (see Section 1.2). There have been occasional issues using the GNU fortran compiler that are fixed with the Intel compiler. While the intel compiler is not required, I recommend installing it if possible. The compiler is available for free for non-commercial use from http://software.intel.com/en-us/non-commercial-software-development.

1.2 Setup

The primary setup that needs to be done for this code is making and preparing run directories for LBLRTM. This should all be handled by the setup.py script To make everything with default settings you should run the setup script by typing the command:

```
python setup.py install [--user]
```

By default, setup.py will make four running directories called 'rundir1', 'rundir2,' 'rundir3', and 'rundir4.' The number of running directories can be set by editing the value near the top of the setup.py script. These will be placed in the current working directory. It also runs LNFL, which generates a binary linelist for use by LBLRTM. By default, the setup script will create a linelist suitable for wavelengths from 300 - 5000 nm. LBLRTM will run faster with a smaller linelist, and so you may wish to adjust these values near the top of the setup script to better suit your needs.

The running directories will contain symbolic links to the lblrtm code and the binary linelist generated by LNFL. It will also contain several input files that configure LBLRTM. The TelFit code will adjust these input files as needed, so you should not need to mess with them.

Finally, the setup.py script will ask to append a line to your /.bashrc. If you install TelFit into your home directory (using the –user option of setup.py), you should let the script do the setup for you. However, if you run setup.py as super-user, you may run into permission issues, and so you should reply 'no' to the prompt. If you say no in the prompt (or anything without a 'n' in it), you should copy-paste the command it prints to the screen right before the prompt before attempting to use TelFit.

2 Usage

In this section, I describe the various functions available to you. Example scripts are also provided in the examples directory.

2.1 MakeModel

This is the class used to directly interface with LBLRTM. You should not need to use this directly for fitting, but you may wish to use it if you just want a telluric model for certain parameters. It can be useful, for instance, to easily identify how the transmission spectrum varies as you change the abundances of various molecules. You can change the parameters of the model by editing the bottom of the MakeModel.py file. The available functions are provided below.

- `__init__(debug=False, observatory="McDonald", NumRunDirs=4, TelluricModelingDirRoot=os.environ["TELLURICMODELING"], nmolecules=12)`
  - debug: False by default. If true, it outputs a bunch of information as it goes
  - NumRunDirs: The number of running directories. This code determines which directory to use so that it does't interfere with other active modeling processes.
  - TelluricModelingRoot: This is the directory which contains all of the running directories. The environment variable should be set by the configure script (see section 1.2).
  - nmolecules: The number of molecules to use in the telluric model. Warning: the current version of this program will crash with nmolecules > 12 because the default MIPAS mid-latitude atmosphere does not have the 13th molecule. The molecule numbering is the same as for the LBLRTM code, and is reproduced below:
    1. H_2O
    2. CO_2
    3. O_3
    4. N_2O
    5. CO
    6. CH_4
    7. O_2
    8. NO
    9. SO_2
    10. NO_2
    11. NH_3
    12. HNO_3

13. OH
14. HF
15. HCl
16. HBr
17. HI
18. ClO
19. OCS
20. H_2CO
21. HOCl
22. N_2
23. HCN
24. CH_3Cl
25. H_2O_2
26. C_2H_2
27. C_2H_6
28. PH_3
29. COF_2
30. SF_6
31. H_2S
32. HCOOH
33. HO_2
34. O
35. $ClONO_2$
36. NO+
37. HOBr
38. C_2H_4
39. CH_3O

- `EditProfile(self, profilename, profile_height, profile_value)`

  Use this function to give a new atmospheric profile. This will generally give more accurate results than just using the generic mid-latitude profile. Instructions for getting a profile from the Global Data Assimilation System (GDAS) meteorological archive are shown in Appendix A.

  – profilename: The name of the profile you want to edit. Available names are:
    * Pressure
    * Temperature
    * Molecule (choose one from the list above, without subscripts. Will cause an error if you choose a molecule with a molecule number greater than nmolecules)

- profile_height: The height of the profile data points (in km)
- profile_value: The value of the profile at each point (units vary)

- `MakeModel(self, pressure=795.0, temperature=283.0, lowfreq=4000, highfreq=4600, angle=45.0, humidity=50.0, co2=368.5, o3=3.9e-2, n2o=0.32, co=0.14, ch4=1.8, o2=2.1e5, no=1.1e-19, so2=1e-4, no2=1e-4, nh3=1e-4, hno3=5.6e-4, lat=30.6, alt=2.1, wavegrid=None, resolution=None, save=False, libfile=None)`

  This function will make a telluric transmission spectrum model with the given parameters.

  - pressure: the pressure at the observatory altitude, in hPa
  - temperature: the temperature at the observatory altitude, in K
  - lowfreq: the lower wavenumber of the model to be created (in cm^{-1}
  - highfreq: the upper wavenumber of the model to be created (in cm^{-1}
  - angle: the zenith angle of the telescope at the time of observation
  - humidity: The relative humidity at the telescope altitude (in percent)
  - co2 through hno3: The abundances of the indicated molecules at the observatory altitude (in ppmv)
  - lat: The latitude of the observatory (in degrees)
  - alt: The altitude of the observatory (in km)
  - wavegrid: Interpolate the model onto the wavelength (not *wavenumber*) grid.
  - resolution: Convolve with a Gaussian profile to reduce the resolution of the model (given as $R \equiv \lambda/(\Delta\lambda)$)
  - save: if True, it will save the output model as a two-column ascii file. The location of the model is printed to the screen.
  - libfile: a convenience option useful for creating a library of telluric spectra. Give the name of the file with this keyword variable, and the code will append the location of the model to the libfile. This keyword is ignored if save = False

2.2 TelluricFitter

This is the main code, used to fit a telluric model to some data. Before calling Fit(), you *must* at least tell it which variables to fit. I also recommend using the InputData, AdjustValue, SetObservatory and SetBounds methods before fitting. The following methods are the ones useful for actual fitting; there are other methods in the code, but they are not meant for direct use and so I do not document them.

- `__init__(debug=False, debug_level=2)`

  This is just the class initialization function.

  - debug: If True, it prints out a bunch of information, and may generate some plots using matplotlib
  - debug_level: Ignored if debug=False. Otherwise, determines how much information is printed to the terminal output. Higher numbers (up to 4) increase the verbosity.

- `FitVariable(vardict)`

  Add one or more variables to the list being fit. The input should be a dictionary where the key is the parameter name and the value is the initial guess value for that parameter.

- `AdjustVariable(vardict)`

  Adjust the value of a parameter, without telling the code to fit that variable. Useful for things like the telescope zenith angle, which will be known much better than the column density of water, for example. The input should be a dictionary where the key is the parameter name and the value is the value for that parameter. As of TelFit v0.2, setting a variable with AdjustVariable will *unset* the fitting flag.

- `SetBounds(bounddict)`

  Set bounds on a variable that you wish to fit. It will let you set bounds on unfit parameters, but will have no effect. The input should be a dictionary with the name of the parameter as the key, and a list with the upper and lower bounds as the value.

- `SetObservatory(observatory)`

  Set the observatory your observations are from. This just defines the latitude and altitude, which MakeModel needs (See section 2.1) The input can either be a dictionary with keys of "latitude" and "altitude" (and the corresponding values), or one of the following preset strings:

  - CTIO
  - La Silla
  - Paranal
  - Mauna Kea
  - McDonald

- `ImportData(data)`

Give the fitter the data to be fit. The input should be a DataStructures.xypoint instance (see section 2.3)

- `EditAtmosphereProfile(profilename, profile_height, profile_value)`

  This is identical to the the version in MakeModel (see Section 2.1).

- `IgnoreRegions(region)`

  Tells the fitter to ignore certain regions of the spectrum in the chi-squared calculation. Useful for stellar or strong interstellar lines. The input should be one of the following:

  1. A list of size two, where the elements are the lower and upper wavelengths of the region to ignore (in nm)
  2. A list of lists, where each sublist is as above. This just allows the user to call the function once instead of several times for each region.

- ```
  Fit(resolution_fit_mode="SVD", fit_primary=False,
           adjust_wave="data", continuum_fit_order=7,
           wavelength_fit_order=3)
  ```

  Here is the all-important fit method. Calling this will begin the fitting process, which can take some time. It returns a DataStructures.xypoint instance (see Section 2.3) for the best fit model. The parameters are enumerated below.

  – data: The data you want to fit, given as a DataStructures.xypoint object (see Section 2.3). Prior to TelFit v0.2, this had to be given separately in the ImportData method.

  – resolution_fit_mode: The method to use for fitting the detector resolution. Choices are 'SVD' or 'gauss'. SVD performs a singular value decomposition to estimate the instrumental broadening profile. The gauss mode finds the best-fit gaussian instrumental profile. SVD is liable to fit noise for weak lines and/or low S/N data, and is problematic for extremely strong lines as well. It can however be better than the gauss method for intermediate strength lines, and is a little bit faster as well.

  – fit_primary: If True, it will generate a savitzky-golay smoothing spline to the data after dividing by the telluric model in each iteration of the fitting loop. This will only work for very rotationally broadened spectra, and has had limited success in testing so far. Use with caution!

  – return_resolution: controls whether the best-fit resolution is returned to the user. One case I have used this for is to fit echelle data of late-type stars by getting all the best-fit parameters from redder orders with more telluric lines and fewer stellar lines, and then applying those atmospheric parameters to the rest of the orders.
  ```

- adjust_wave: Can be either 'data' or 'model'. If 'data', then the wavelength fit will edit the wavelength solution of the data to fit the telluric model. If 'model', it will adjust the wavelengths in the telluric model. Using the 'data' option can therefore help to wavelength calibrate the data. Note however, that doing so will usually introduce an unphysical (but constant) velocity shift to the data, because the conversion from vacuum to air wavelengths in the model is done in a very approximate manner. The wavelength shift will be of order ~ 1 km s^{-1}.
- continuum_fit_order: The polynomial order to use in the continuum fit. The fit is done in each iteration of the telluric fitting loop, and so gets better as the fit converges.
- wavelength_fit_order: The polynomial order to use in the wavelength adjustment. Note that this is not fitting pixels -¿ wavelength. It fits wavelength -¿ better wavelength (i.e. lam(pixel) = f(lam_0(pixel)))

2.3 DataStructures

This just provides the xypoint class, which has the following attributes, all of which are numpy.ndarray objects:

- x: The wavelength array
- y: The flux
- cont: The continuum level of the flux
- err: The error in each pixel. If not given, this is just taken as the square root of the flux.

The xypoint class also has the following convenience methods:

- copy: Returns a deep copy of the xypoint object.
- size: Returns the size of the xypoint. This is taken from the 'x' attribute, so may be wrong if they are not all the same size (but something has gone wrong if they aren't)
- output(outfilename): Outputs the xypoint object to the given file as a 4-column ascii with the following columns:
  1. x (wavelength) array
  2. y (flux) array
  3. continuum array
  4. error array

The xypoint object supports slicing as well, although the output depends on how it is accessed:

- xypoint[index]: Returns a tuple of length 4, containing the x,y,cont,and err values at the given index.

- xypoint[index1:index2]: Returns an xypoint object which is a copy of the original between the given indices.
- xypoint[list]: Returns an xypoint object identical to the original, but only at the indices given in the list

A Using the Global Data Assimilation System meteorological archive

In most cases, a good atmosphere profile goes a long ways to improving the telluric fit. This section explains the usage of the Global Data Assimilation System meteorological archive, and how to input the results into TelFit. These instructions were made on March 31, 2014. If the archive website changes after publication, you are on your own! The steps for getting an appropriate atmosphere profile are listed below:

1. Enter the latitude and longitude of the observatory
2. In the Sounding row, choose the entry titled "GDAS (1 deg, 3 hourly, Global)." For observatories in the United States, the NAM sounding data, labeled "NAM (12km, 3 hourly, U.S.)," will be slightly more accurate. Hit the "Go" button in sounding row.
3. Choose the appropriate UT date.
4. Choose the appropriate UT time in the first row.
5. Choose "Text only" in the "Output Options" row.
6. Choose "Text listing" in the "Graphics" row.
7. Complete the captcha and hit the "Get Sounding" button
8. Copy the second block of information into a text file. It will look something like this:

```
PRESS HGT(MSL) TEMP DEW PT  WND DIR  WND SPD
HPA       M      C     C       DEG     M/S
E = Estimated Surface Height

1015.     0.   27.1   22.8    92.3     4.2
1000.   130.   25.8   22.0    93.3     4.5
 975.   354.   23.6   21.6    94.6     4.6
 950.   581.   21.6   20.3    98.5     4.7
 925.   813.   20.3   17.6   108.5     4.2
 900.  1049.   18.9   16.0    96.2     3.7
 875.  1291.   17.6   14.4    95.9     3.3
 850.  1539.   16.2   12.8   102.7     2.8
```

```
825.  1793.   14.8   10.7   110.5     2.4
800.  2053.   13.6    8.2   114.3     1.9
775.  2320.   12.8    4.5    98.0     1.3
750.  2595.   12.2   -0.9    39.3     1.5
725.  2878.   11.4   -5.1    13.8     2.9
700.  3170.    9.7   -1.9     5.5     4.4
650.  3781.    5.8   -4.2   358.6     5.8
600.  4431.    2.5  -12.6   353.8     5.7
550.  5130.   -0.6  -23.6   338.2     4.6
500.  5884.   -5.4  -24.9   322.3     5.0
450.  6701.  -11.1  -22.5   318.2     8.5
400.  7594.  -17.5  -33.7   314.5    11.2
350.  8579.  -24.3  -45.5   312.3    12.5
300.  9682.  -33.4  -48.3   318.2    12.9
250. 10934.  -43.8  -55.2   318.3    12.7
200. 12393.  -55.3  -60.3   303.2    12.7
150. 14165.  -68.9  -73.2   305.7    10.4
100. 16532.  -75.7  -83.4   318.7     8.4
 50. 20622.  -65.0  -87.3   144.1     2.9
```

The text file you created has the pressure, temperature, and dew point at several atmosphere heights. You should read in the appropriate columns in your Python code (NumPy's loadtxt command works well for this). You must do some quick unit conversion and sorting before inputting the atmosphere profile to TelFit. Some sample code to read in the profile, sort it, convert the dew point to a humidity, convert the height to km, and convert the temperature to Kelvin is shown below. Most of the code is self explanatory, except for the conversion from dew point temperature to mixing ratio in ppmv. The equation and constants were taken from a compilation of formulas for humidity, available at `http://www.vaisala.com/Vaisala%20Documents/Application%20notes/Humidity_Conversion_Formulas_B210973EN-F.pdf`

Listing 1: Code to adjust units from GDAS archive

```
#Read in GDAS atmosphere profile information
Pres,height,Temp,dew = numpy.loadtxt(atmosphere_fname,
                                     usecols=(0,1,2,3),
                                     unpack=True)
sorter = numpy.argsort(height)
height = height[sorter]
Pres = Pres[sorter]
Temp = Temp[sorter]
dew = dew[sorter]

#Convert dew point temperature to ppmv
```

```
Pw = 6.116441 * 10**(7.591386*Temp/(Temp + 240.7263))
h2o = Pw / (Pres-Pw) * 1e6

height /= 1000.0
Temp += 273.15
```