Data analysis with `spectr`

Alan Heays | February 2, 2022

Contents

1 Introduction

This document and accomanying example scripts were test against `spectr` vesion `v1.3.0`.

2 Installation

2.1 Source code

GPL source code is publically available on github at https://github.com/aheays/spectr. The repository can be cloned locally with `git clone --depth=1 https://github.com/aheays/spectr.git` (this will require git to be installed `sudo apt install git` on ubuntu). A subsequent `git pull` while inside the `spectr` source directory will download any updates. Otherwise, zip-archives of particular version can be downloaded e.g., https://github.com/aheays/spectr/archive/refs/tags/v1.3.0.zip.

2.2 Python dependencies

This module has been tested and works with python 3.9 and 3.10, and will not work with version <3.9. Many non-standard python libraries are used and are available from linux distributions or via pip, with the following package names:

```
bidict cycler hitran-api brukeropusreader dill h5py matplotlib
numpy openpyxl periodictable scipy sympy xmltodict pyqt5 py3nj
```

2.3 Building the fortran extensions

Fortran code used to generate python extensions with `f2py` and needed to speed up some calculations. Some submodules will still import and run fine without these. To compile the extensions run `make` within the spectr source directory. The options in the `Makefile` are for the gfortran compiler and use the `lapack` library. The following distribution packages might be sufficient to compile the code under linux:

- Debian / Ubuntu: `gfortran python3.9-dev liblapack-dev`
- Arch: `gcc-gfortran lapack`

If the Makefile is not working, then the following should be enough to get things running:

```
f2py3 -c --quiet -llapack --f90flags="-Wall -ffree-line-length-none" --opt="-O3" fortran_tools.f90
```

2.4 Matplotlib dependencies

Under ubuntu, installing the packages `pyqt5` and `qt5-default` seem to be enough to get matplotlib to draw windowed figures.

2.5 Installing in a linux virtual environment

This is a recipe for installing spectr and its dependencies in a virtual environment, and not messing up the system python.

- Install python3.9 (or 3.10) somehow. Under ubuntu with `sudo apt install python3.9`
- Create a virtual environment using python3.9: `python3.9 -m venv venv3.9`
- Start the virtual environment: `source venv3.9/bin/activate`
- Install the python dependencies with pip: `pip install bidict cycler hitran-api brukeropusreader dill h5py matplotlib numpy openpyxl periodictable scipy sympy xmltodict ; pip install py3nj` *For some reason py3nj must be intalled after everything else (2022-01-18).*
- The matplotlib Qt dependency can also be installed with pip, `pip install pyqt5`, but Qt must be installed on the system itself, e.g, with `sudo apt install qt5-default`
- Clone spectr code with `git clone --depth=1 https://github.com/aheays/spectr.git`
- Add `spectr` to the python path so it can be imported from anywhere

  ```
  cd venv3.9/lib/python3.9/site-packages/
  ln -s ../../../../spectr .
  cd -
  ```

- Add `qplot` to the path so it can be run from within the virtual environment

  ```
  cd venv3.9/bin
  ln -s ../../spectr/qplot .
  cd -
  ```

- I wrote a script that attempts to install the virtual environment: `bash install_in_venv.sh`

2.6 Testing the installation

Test by importing spectr and trying to plot something

```
source venv3.9/bin/activate
echo "from spectr.env import *" | python
qplot examples/absorption/data/2021_11_30_bcgr.0
```

3 Usage

3.1 Importing `spectr`

A command to import all submodules and many common functions directly into the working namespace is `from spectr.env import *`. Otherwise only the needed submodules can be imported, e.g., `import spectr.spectrum`

3.2 Optimising things

The `optimiser.Optimiser` class is used to conveniently construct model objects with parameters that can be fit to experimental data. The real-number input arguments of most methods of objects base-classed on `Optimiser` can be marked for optimisation by replacing their values with a optimiser.Parameter object. This has the abbreviated definition:

```
P(value=float,
  vary=True|False,
  step=float,
  uncertainty=float,
  bounds=(float,float))
```

Only the first argument is required. For example, `x=P(2,True,1e-5,bounds=(0,100))` defines a parameter `x` that will be varied from an initial value of 2 but constrained to the range 0 to 100. When computing the finite-difference approximation to the linear dependence of model error on `x` a step size of 1×10^{-5} will be used. The fitting uncertainty `unc` will be set automatically after optimisation. Multiple `Optimiser` objects can be combined in a hierarchy, so that multiple spectra can be fit at once to optimise a common parameter, for example a temperature-dependence coefficient fit to spectra at multiple temperatures.

3.3 Encoding diatomic quantum numbers

3.4 `qplot`

This is a command line programming for making line plots, e.g., `qplot datafile`, or `qplot -h` for a list of options.

4 Example scripts

4.1 To run the scripts

First start a virtual environment if needed with `source path/to/venv3.9/bin/activate` then to run a script:

```
cd examples/subdirectory
python script.py
```

Output from the scripts are in `examples/subdirectory/output`.

4.2 Modelling absorption spectra

Example scripts showing how to quantify species in infrared absorption spectra.

4.2.1 `examples/absorption/fit_one_line.py`

This function automatically fits the column density and pressure broadening of HITRAN speces in IR absorption spectra. This will work sometimes but fail when the spectrum is more complex. The frequency range is given by the preset `characteristic_infrared_bands` values in spectr/data/species_data.py. To fit other species (or multiple) `'HCN'` with e.g., `'CO'` or `'H2O'`.

4.2.2 `examples/absorption/fit_manually_1.py`

This is a lower-level script that fits a small part of one spectrum. The model results are output to `examples/output/fit_manually_1/model_of_experiment`.

4.2.3 `examples/absorption/fit_manually_2.py`

A script that fits an entire spectrum including all identified species.

4.2.4 `examples/absorption/fit_manually_3.py`

A script that fits the CO fundamental band including more instrumental effects.

4.2.5 `examples/absorption/fit_absorption_1.py`

This script uses a FitAbsorption object to more conveniently fit multiple species in a spectrum.

4.2.6 `examples/absorption/fit_absorption_2.py`

A FitAbsorption script that fits multiple species in multiple spectra.

4.2.7 `examples/absorption/fit_absorption_3.py`

A FitAbsorption script to that fits a cross section file (downloaded from HITRAN) to an experimental spectrum. It also uses a measured background rather than a fitted spline curve.

4.3 Modelling emission spectra

4.3.1 `examples/emission/compute_emission_band.py`

Computes a list of diatomic level energies from molecular constants of electronic-vibrational bands, and then computes a linelist of electric-dipole transitions.

4.3.2 `examples/emission/fit_emission_band_1.py`

Uses a linelist computed from molecular constants to fit the temperature of N_2 emission in a laboratory spectrum.

4.3.3 `examples/emission/fit_emission_band_2.py`

Fit the temperature and molecular constants of multiple N_2 emission bands in a laboratory spectrum.

4.3.4 `examples/emission/fit_emission_band_3.py`

Fit an effective J-dependent rotational temperature to some emission bands.

4.4 Analysing ARGO output

4.4.1 `examples/argo/analyse_argo_1.py`

Load a model and print whats in it.

4.4.2 `examples/argo/analyse_argo_2.py`

Plot the dominant production and destruction reactions affecting particular species.

4.4.3 `examples/argo/analyse_argo_3.py`

Compare two models.

5 Submodules

5.1 `env.py`

Conveniently import all submodules.

5.2 dataset.py

Storage, manipulation, and plotting of tabular data. Allows for the recursive calculation of derived quantities

5.3 tools.py

Functions for performing common mathematical and scripting tasks.

5.4 plotting.py

Functions for plotting built on matplotlib.

5.5 convert.py

Unit conversion, species name conversion, and various conversion formulae.

5.6 optimise.py

General class for conveniently and hierarchically building numerical models with optimisable parameters.

5.7 atmosphere.py

Classes for analysing atmospheric photochemistry.

5.8 lines.py

Dataset subclasses for storing atomic and molecular line data.

5.9 levels.py

Dataset subclasses for storing atomic and molecular level data.

5.10 bruker.py

Interact with output files of Bruker OPUS spectroscopic acquisition and analysis software.

5.11 database.py

Interface to internal spectroscopic and chemistry database.

5.12 electronic_states.py

Calculation of diatomic level energies from potential-energy curves.

5.13 exceptions.py

Exception used to internally communicate failure conditions.

5.14 hitran.py

Access HITRAN spectroscopic data with hapy.

5.15 lineshapes.py

Simulate individual and groups of spectra lines of various shapes.

5.16 quantum_numbers.py

Functions for manipulating atomic and molecular quantum numbers.

5.17 spectrum.py

Classes for manipulating and modelling of experimental spectroscopic datea.

5.18 `thermochemistry.py`

Functions for computing thermochemical equilibrium with ggchem.

5.19 `viblevel.py`

Classes for simulating diatomic levels and lines defined by effective Hamiltonians.

5.20 `fortran_tools.f90`

Various fortran functions and subroutines.