Quick introduction library implementation

Erik Ahlberg
eahlb@kth.se

July 19, 2016

Contents

1 Introduction

This library implements a Monte Carlo method for finding the confidence interval for fitted parameters using `PyXSPEC`. The libarary was developed during 2016 as part of my master thesis[1]

The code consist of two parts. The first, `ErrorFinder`, is used to find the confidence interval for fitted parameters. The second, `ValidateModel`, manages the Monte Carlo simulations. `ValidateModel` is used by `ErrorFinder` and all the results of the thesis was found using `ValidateModel`.

In addition to standard `Python` packages, `Numpy`[2] and `PyXSPEC`[3] must be installed.

In figure 5 the files included in this library is shown. This tree can be placed at any location, but its internal structure must remain intact. Two examples showing how to use both parts of the library is included. These should be run with `/final` as the current working directory.

2 Using `ErrorFinder.py`

The `ErrorFinder` uses a Monte Carlo method to find the confidence interval of fitted parameters. Included here is one dummy example (`run_error.py`) with the necessary files.

In order to find the confidence interval two files must be supplied to the program, one for settings and one with the fitted parameter values.

The confidence interval is calculated in `_point.calculate_stats(...)` and can be modified if desired.

The settings file (in this example `input.txt`) specify the path to all data files, the energy channels used for each detector, the model used for fitting the data and the path to the file containing the fitted parameter values. The structure of this file is shown below.

[1] To be published at `http://kth.diva-portal.org`
[2] `www.numpy.org`
[3] `https://heasarc.gsfc.nasa.gov/xanadu/xspec/python/html/`

```
PHA_FOLDER/
[(nai0.pha,bgo0.pha), (nai1.pha,bgo1.pha)]
BAK_FOLDER/
[(nai0.bak,bgo0.bak), (nai1.bak,bgo1.bak)]
RSP_FOLDER/
[(nai0.rsp,bgo0.rsp), (nai1.rsp,bgo1.rsp)]
(**-8.0 1000.0-**, **-200.0 40000.0-**)
atable{ TableModel.fits }
results.txt
```

Figure 1: Format of the input file (`input.txt`)

All paths in the settings file is relative paths from the base folder. The base folder can be set by the optional argument `ErrorFinder(...,base_folder)`, or more permanently by modifying the global `BASE_FOLDER` variable in `ErrorFinder.py`.

The example above is of an observation with two time bins (the number elements in the arrays) using two detectors (the number of elements in the tuples). Here the data file for the NaI detector for the first time bin is located at `BASE_FOLDER/PHA_FOLDER/nai0.pha`, with the corresponding background file `BASE_FOLDER/BAK_FOLDER/nai0.bak`, ignoring energies outside the $8 - 1000\,\mathrm{keV}$ range. The data is fitted using a table model located at `BASE_FOLDER/TableModel.fits` using the parameters found in `BASE_FOLDER/results.txt`

The use of this base folder allows us to separate the data files from the code used, making it possible to use this with essentially any file structure that some scientist somewhere thought was smart for some very specific use.

Note that `PHA_FOLDER/, BAK_FOLDER/, RSP_FOLDER/` does not need to be separate folders.

The file with the fitted parameter values (in this example `results.txt`) lists the fitted parameter values and confidence intervals for each time bin. `ErrorFinder` parses this file, runs the Monte Carlo method and outputs a new file with identical format, but updated confidence intervals. This output is written by `ErrorFinder.__write_result(PATH)`, which provide a template for how to produce this format.

```
ind  pgstat  dof  tau    plus  minus  ...  z     norm
0    273.6   241  22.79  3.63  2.83   ...  0.54  128.54
1    281.2   241  24.23  3.01  2.98   ...  0.54  128.54
```

Figure 2: Format of the initial result file (`results.txt`) for the first parameter using two time bins

If one want to use this for other models some modifications is required. `ErrorFinder` and `ValidateModel` works for any `XSPEC` model (note: this has not been fully tested for native `XSPEC` models). First, and most obvious, the results file must match the parameters of the new model. Second, the code must be able to use this new format. This is achieved by modifying the `_point` class in `ErrorFinder`. An object of this class hold all the data for one time bin in the result file. This class must be modified in essentially every method to match the new format. Or, if one is more ambitious, it can be modified to handle an arbitrary model.

3 Using `ValidateModel.py`

This method is discussed extensively in the thesis. This text is intended to complement those discussions with some specifications of the implementation. Included in this library is an example of how to run these tests.

The figures and tables in the thesis were produced using functions in the `DreamStats.py`. This file is a collection of functions I used, and modified for very specific uses, over some time. Use these functions at your own risk. I will not be responsible for any erroneous results, corruption of data, loss of data and/or fires that may occur as a result of using these functions.

For connivance I will here summarize which functions were used to create which figures

`PlotDist()`: Figures 6.6, 7.6-7, 8.1, 8.5-6

`CompareInterpolation()`: Figure 7.2, 7.5

`GetSnrEvolution()`: Figures 6.1, 6.3, 8.4,

`GetOffValuePlot()`: Figures 6.4-5, 7.3-4, 8.2-3

`PlotBox()`: Appendix A

`GetTable()`: Tables 6.2-4 8.1-2

If you wish to implement your own data analysis, the result of a completed simulation is found by using `ValidateModel.LoadData(ID)`.

3.1 File structure

It is crucial that the internal file structure shown in figure 5(a) remains intact.

3.2 Creating settings files (`.ds`)

For each set of parameters that is to be tested a settings file (`.ds`) must be created. When testing a set of parameters, a series of files is created. These files specify the detector and data files used, type of simulation (type of `Sim` object) and the parameters to be tested.

In the example below two detectors are used (the first line). The type of simulation is `GridSim` (implemented in `Sim.py`). The parameters to be tested is given as an array. For each detector the paths to the data files and energy range is given. The paths are relative from the `data` folder. In this case the path to the first file would be `./data/nai0.pha`.

```
2
Grid
[1.0, 0.01, 1e-06, 2.47, 100.0, 1, 1]
##
nai0.pha
nai0.rsp
nai0.bak
**-8.0 1000.0-**
#
bgo0.pha
bgo0.rsp
bgo0.bak
**-200.0 40000.0-**
#
```

Figure 3: Format of the setting files (`.ds`)

The functions I used to generate these files are given in `settings.py`.

3.3 Running simulations

The easiest way of running a series of simulations is by using `ValidateModel.RunSimSeries(...)`. This function has several optional arguments you should familiarize yourself with before running the simulations. The most important are

`dataID`: The name of the `.ds` files that are to be used, excluding the indices

`saveID`: The name of the folder where the result will be stored, excluding the indices

`nrOfSims`: The number of iterations of the Monte Carlo method

`startindex`: The first index of the series to be run. Extremely convenient if the program were to crash halfway through the series

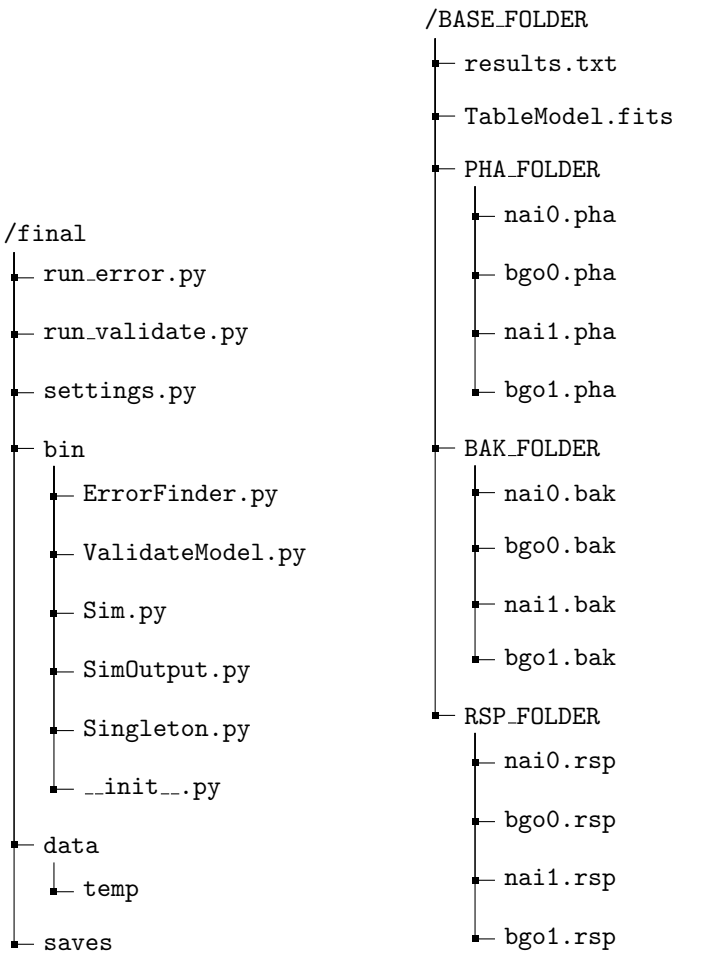
The basic principle of the method is; load a `.ds` file, run the simulations and save the result in a `.do` file (see below).

3.4 Analysis of the result

The result of a simulation will be saved at `saves/saveID/output.do`. This result can found agian with `ValidateModel.LoadData("saveID")`. `LoadData` will return a list of `SimOutput` objects, one for each iteration of the Monte Carlo method. Examples of how this data can be used is given in `DreamStats.py`.

```
[1.0, 0.05, 1e-06, 10, 100, 1, 1]
[1.0, 0.06835097197047152, 1e-06, 1.9029098156603088, 199.91590336282607, 1.0, 1.0]
["Tau", "epsilon_e", "-", "LGRB", "Gamma", "z", "norm"]
340.16631240070825
656112.4628183318
483
#
[1.0, 0.05, 1e-06, 10, 100, 1, 1]
[1.0, 0.05018417087804372, 1e-06, 10.245413967521957, 110.93661823038852, 1.0, 1.0]
["Tau", "epsilon_e", "-", "LGRB", "Gamma", "z", "norm"]
396.295722769121
1134563.0184744145
483
#
[1.0, 0.05, 1e-06, 10, 100, 1, 1]
.
.
.
```

Figure 4: Format of (`output.do`). Each iteration of the Monte Carlo method is separated by #



(a) Code library (b) The base folder

Figure 5: File structures