Yafit - Yet Another Fitting Tool

Version 0.1-

Starlink User Note
Mark Taylor
6 Feb 2007

$Id: yafit-user.xml,v 1.8 2007/02/12 18:01:14 mbt Exp $

Abstract

Yafit is a tool for fitting curves. Although it could be used in other contexts, it is mainly intended for fitting observed photometric data to calculated model spectra. In general one uses it by preparing a file containing one or more sets of observations, preparing a file containing a number of model spectra, and then running the fitting program which identifies the model that most closely fits each set of observations. The results can be viewed graphically and/or output in a number of formats.

Contents

1 Introduction

Yafit is a package intended for identifying which spectral models from a given set are best fits to one or more observational data sets. Operation is as follows:

1. Prepare a file containing the observations to fit
2. Prepare a file containing calculated model spectra to fit the observations against
3. Perform the fit

In general this will be done by executing three separate commands. Section 2 explains how to invoke commands from the Yafit package, and Section 3.1, Section 3.2 and Section 3.3 describe the three steps above. Note it is the user's responsibility to ensure that the quantities to be fitted against each other (the observation and model data) are compatible, that is, represent the same X and Y quantities and have compatible units.

To use Yafit, you should read Section 2 to find out how to invoke the commands, and then Section 3 for an explanation of how to use the tool to fit your data. The other sections can be consulted as required.

2 Invoking Yafit commands

Yafit is a java program, usually distributed as the file `yafit.jar`. It requires a Java Runtime Environment (JRE) version 1.4 or later to run.

The package comprises a number of different *tasks*; tasks are called things like `fit` and `plotobs`. They are invoked by giving the task name followed by a set of arguments, usually in the form `name=value`. Each command and its arguments are described in detail in Appendix A

There are two ways of invoking the program:

Directly with java:

```
java -jar yafit.jar <task-name> <task-args>
```

With script (unix only):

```
yafit <task-name> <task-args>
```

The `yafit` script is a convenience for running on unix-like operating systems: if you don't have the script, you can extract it from the jar file like this:

```
unzip yafit.jar yafit
chmod +x yafit
```

To run the script file, you must have the `java` command on your path, and the `yafit` script should be in the same directory as `yafit.jar`.

Invoking the program with the `-help` flag gives a summary of the available tasks:

```
yafit -help
```

```
Usage:
   yafit [-help] [-version] [-verbose] [-debug] [-bench]
         <task-name> <task-args>

   yafit <task-name> -help

   Known tasks:
      fit
      funcs
      copymodel
      tableobs
      plotmodel
      plotobs
```

Usage information (a list of parameters) for each individual task can be obtained by supplying the `-help` flag after the name of the task itself, e.g.

```
yafit fit -help
```

Full help for each command is given in the reference section, Appendix A.

3 How to fit spectra using Yafit

3.1 Preparing Observation Data

Before fitting, you must prepare your observation data in a way that the program can understand. This involves generating a `.yobs` file which contains spectral information about one or more sources. Typically this will consist of photometric observations at a number of pass bands for each of several sources of interest. Currently the `tableobs` task is provided to convert observations in a table for which each column represents observations in a given pass band to a `.yobs` file.

Suppose that you have photometric observations in a table as follows:

```
#       RA       DEC    PHOTOZ        J    J_ERR        H    H_ERR        K    K_ERR
   119.608   21.410      2.52    11.94    0.022    12.16    0.024    12.57    0.018
    53.094  -27.839      3.05    12.33    0.022    12.34    0.026    12.67    0.020
   213.870  -46.515      1.65    17.32    0.106    17.57    0.192    17.49    0.176
   204.143  -29.089      0.50    15.93    0.041    16.29    0.071    16.48    0.073
```

This represents observations of four objects in each of three photometric bands (J, H and K). The fluxes are here represented as magnitude values, and the flux at each of the bands has an associated error value. The fitting program must identify each of these bands with a region of the model spectrum and determine how close the observation is to the model flux in that region to see how well the observation fits the model. The flux error in each case provides a measure of how large deviations are to be tolerated. An additional column (PHOTOZ) gives the known redshift for each object and the sky position is given by the RA and DEC columns.

The `tableobs` task can be used to generate an observation file from this table, by specifying which columns give flux values and flux errors for each band pass. You can also provide expressions which perform numeric value transformations such as unit conversions if required; recall that the X (wavelength) and Y (flux) values in your observation file must be compatible with those you supply in your model file before the fitting is performed.

To define which columns correspond to which band passes in the table, you must prepare a *key file*, which lists the flux value and error columns corresponding to each band pass given in the table. For the table above, this file would look something like the following:

```
#  yColName     yerrColName        x    xWidth
   J            J_ERR          12400     5000
   H            H_ERR          16600     4000
   K            K_ERR          21600     6000
```

The format of this file is a four-column ASCII table - the column names (and any other lines starting with a '#' character) are ignored. Each line of it must contain four entries in order:

1. Name of column giving Y (flux) values
2. Name of column giving Y (flux) errors
3. X (wavelength) value of band pass centre for these fluxes
4. X (wavelength) width of band pass filter for these fluxes

This key file therefore identifies the meanings of six of the columns in the input table above.

When you run `tableobs` you can also specify information about how the X and Y values should be written out. You can label their names and units and give an expression for how to calculate them from the values in the input table. Note the names and units you give are only annotations and will not cause any conversions to take place apart from those you specify by providing the expressions. Nevertheless it is a good idea to provide these in order to document your data files. You may well wish to convert the X and Y values to more suitable quantities, since they must be in a form which can be meaningfully fitted against the model data to be provided. For instance in the above case, you will probably want to convert the Y values from magnitudes to fluxes. A flexible expression

language, described in Section 6, is available to perform these conversions; you will mostly use a few functions such as `abToJansky`.

Finally, you can specify redshift value if the input table contains one. This may be necessary to determine how model spectra are to be shifted along the X axis prior to attempting a fit.

Here is an example of a `tableobs` command to prepare a `.yobs` file from the input table above.

```
yafit tableobs in=demo.txt ifmt=ascii
               key=jhk.key
               redshift=PHOTOZ
               xname=Wavelength xunit=Angstrom x=x
               yname=Flux yunit=Jansky y='abToJansky(y)'
               out=demo.yobs
```

In detail, this works as follows:

`in=demo.txt ifmt=ascii`
The input table is named and its format is given as an ASCII table. The format types here are those permitted by STILTS including VOTable, FITS, CSV and others. See SUN/256.

`key=jhk.key`
Gives the location of the 4-column 'key file' whose format is described above. This must be in ASCII format.

`redshift=PHOTOZ`
Defines the column in the input table which gives redshift. This is not required if redshift is not known or not represented in the table. This parameter can only be sensibly used if the X axis is a wavelength measure. The value of the parameter can be an algebraic expression using column names from the table.

`xname=Wavelength xunit=Angstrom x=x`
Labels the output X values to be Wavelength quantities with units of Angstrom. The `x=x` is optional, but it means that the output X values will be equal to those supplied in the .key file. You could change the units here by writing, for instance, "`xunit=micron x=x*1e-4`". The X widths are automatically adjusted to the same scale as the X values (using numerical differentiation).

`yname=Flux yunit=Jansky y='abToJansky(y)'`
Labels the output Y values to be flux quantities with units of Jansky. Additionally, says that the output Y values are to be calculated from the values in the keyed columns of the input table using the pre-defined "`abToJansky()`" function, which converts from AB magnitudes to Jansky. The Y errors are automatically adjusted to the same scale as the Y values (using numerical differentiation).

`out=demo.yobs`
Writes the result to a .yobs file as named. This is in fact a VOTable, and can for instance be viewed in TOPCAT, but it has some special characteristics, and it may be best to think of it as an opaque file format used by Yafit. Note that the output file will contain all the original table information from the input table, and this will be propagated to any output tables, so that for instance information in the RA and DEC columns is not lost.

A full reference for the `tableobs` task is given in Appendix A.6.

Having prepared your .yobs file, you can view it using the `plotobs` command if you wish before using it for actual fitting.

3.2 Preparing Model Data

Before fitting, you must prepare your set of model spectra in a way that the program can understand. Yafit understands a number of model formats, including outputs from certain spectral

simulation codes. You can either provide your models in one of these formats directly or convert them to the special `.ymod` format prior to use. The `copymodel` task can convert from the various foreign model formats to `.ymod`, optionally applying custom transformations to the data. Even if your model data is in one of the formats that Yafit understands, you may wish to use `copymodel` so as to apply these additional conversions, for instance annotating or modifying the units on the X (wavelength) or Y (flux) axes to match those presented in the observation data.

Here is an example invocation of `copymodel`:

```
yafit copymodel ifmt=galaxev
                in=bc2003_salp.a1 in=bc2003_salp.a2
                xname=Wavelength xunit=Angstrom
                yname=Luminosity yunit=erg/s
                y=3.826e33*y
                out=gal.ymod
```

In detail, this works as follows:

`ifmt=galaxev`
: Gives the format in which the input files will be supplied. `galaxev` is one of the formats that Yafit understands (see Section 4).

`in=bc2003_salp.a1 in=bc2003_salp.a2`
: Names the two input files in galaxev format. Any number of input model files can be given - they will be concatenated together before output.

`xname=Wavelength xunit=Angstrom`
: Labels the axis name and unit for the independent (X) variable.

`yname=Luminosity unit=erg/s`
: Labels the axis name and unit for the dependent (Y) variable.

`y=3.826e33*y`
: Multiplies the input Y (flux) values by a given constant. This sort of thing may be required to convert the values from those used in the input model data to match those used by the observations during the fit.

`out=gal.ymod`
: Gives the name of the output file to which the converted model data will be written. The output file is actaully a specially prepared VOTable or FITS table, and can e.g. be viewed in TOPCAT if required.

A full reference for the `copymodel` task is given in Appendix A.1.

Having prepared your .ymod file, you can view it using the `plotmodel` command if you wish before using it for actual fitting.

3.3 Performing the Fit

Having prepared your input observation and model data, you run the `fit` task to compare the two. This compares each input observation with each input model and presents the results in one or more ways. The most straightforward use of the results is to assess which of the input models matches the observation best, and then output this information. However, some output modes allow you to see the goodness of fit for several of the models in addition to the best fit one.

As with the other tasks, the `fit` task has a number of parameters to configure its operation. These fall into the following categories:

Input Specification
: The `model` and `modelfmt` parameters give the location of the input model data file and the `obs` parameter gives the location of the input observation data. These should be prepared as

described in Section 3.1 and Section 3.2.

Fitting Configuration
The `smoother`, `scale` and `fitcalc` parameters control the details of how the goodness of fit between a given observation and a given model is to be calculated.

Output Specification
The `gui`, `summary`, `bestfits` and `allfits` parameters control what the program does with the results having calculated them. Some of the possibilities are display them in a graphical window and output a table which contains rows associating the best fit with each input observation.

A full reference for the `fit` task is given in Appendix A.2.

NOTE: The most important thing to remember when performing a fit is that the axes of the presented observation and model files must be compatible. If the wavelength axis for the observations is in Angstrom and for the models is in microns your results won't make much sense. Having the same units for the two flux axes may not be necessary - since the models will normally be scaled by a constant to provide the best fit, an arbitrary multiplicative constant between the two units may be unimportant unless you need to make use of the fitted scaling constant. However even in this case you must ensure that the axes are compatible (e.g. both are fluxes, rather than one being in magnitudes).

An example invocation is as follows:

```
yafit fit obs=demo.yobs model=demo.ymod
          gui=true bestfits=out.vot
```

This fits the observations in file `demo.yobs` against the model data in `demo.ymod`. Default settings are used for the fitting. By way of output, the results will be presented graphically (`gui=true`) and the output written as a VOTable `out.vot` in which each row gives the details of an observation and the model which fitted it best, as well as the goodness of fit statistic and scaling constant used.

4 Model Formats

Yafit can understand model data in a number of formats as written by various spectral simulation codes. This list will probably need to be expanded in the future. You can contact the author if a format that you need is not here, but note that some well-defined description of a format must be available in order for a handler to be written.

YModel
YModel is the name given to the 'native' format used by Yafit for model data. It is in fact a FITS or VOTable table in which each row represents one model spectrum. The first column contains a vector of X values, and the second a vector (of the same size) of Y values, and these two therefore together define a function Y(X) which represents the spectrum. There may be any number of additional columns present, giving other information about each model. It is possible to write YModel files directly and to manipulate them using TOPCAT or STILTS, but in most cases YModel files will be written using the `copymodel` task.

Galaxev
Spectra extracted into files by **Galaxev**'s `galaxevpl` (a.k.a. `gpl`) tool can be used. These look something like this:

```
# Output file name = bc2003_hr_m42_salp_ssp_exp200.a1
# Input file name  = bc2003_hr_m42_salp_ssp_exp200.ised
# Column           2          3          4
# Record         106        107        108
# Age (yr)      4.500E+07  4.750E+07  5.000E+07
# Lambda(A)       Flux       Flux       Flux
  9.100000E+01  6.685E-19  6.602E-19  6.520E-19
  9.400000E+01  7.586E-19  7.491E-19  7.398E-19
    ...
```

Starburst99
Output files written by the **Starburst99** program can be used. These look something like this:

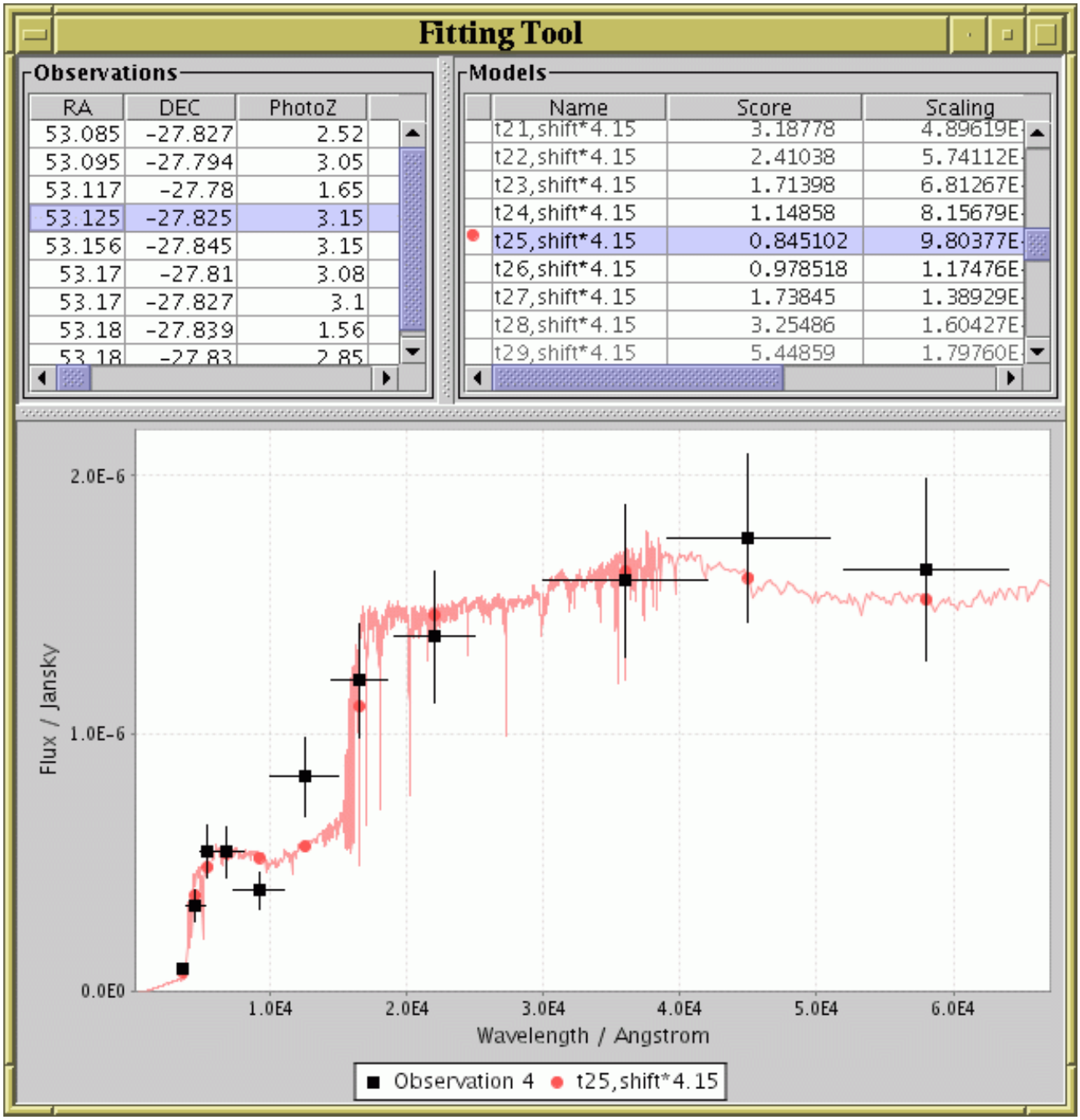
```
MODEL DESIGNATION: eyles01100FM2
MODEL GENERATED ON 20060527  AT 035003.6

             COMPUTED SYNTHETIC SPECTRUM

TIME [YR]    WAVELENGTH [A]   LOG TOTAL  LOG STELLAR  LOG NEBULAR  [ERG/SEC/A]
.10000E+05          91.00      25.60479    25.60479   -15.00000
.10000E+05          94.00      26.34040    26.34040   -15.00000
   ...
```

SvoTar
The Spanish Virtual Observatory (SVO) provides a service from the web page http://svo.laeff.esa.es/theory/db2vo/html/ which returns grids of model spectra given certain criteria entered in an HTML form. The result it returns is a tar archive containing a number of VOTable files each of which holds a single calculated spectrum. These tar files (compressed or uncompressed) can be understood directly by Yafit as a list of model spectra.

SidewaysVOT
This is an ad-hoc format which can be used to store model data in a standard tabular format as a VOTable. The first column of the table is used to store X (wavelength) values, and each subsequent column stores the Y (flux) values for a different model spectrum. All the Y values in the Nth row correspond to the X value in that row and so on. An M-column sideways table contains M-1 model spectra. The only additional information about each spectrum is a name or label, which is supplied by the column name of the column which it represents.

5 Graphical Display

Some of the tasks in Yafit provide a graphical display on the screen; `plotobs` plots observational data, `plotmodel` plots model data, and the `gui` mode of `fit` shows both. These displays use a similar format and visual controls, as shown in the figure.



Window displayed by the `gui` mode of the `fit` task.

The window consists of two parts: at the top are one or more tables giving details of the model or observation data being displayed, and at the bottom is the graphical display itself. In this case the top left part shows the table of observations (one per row) and the top right part shows the table of models (one per row). In each case, the highlighted row or rows indicate which observation/model is being drawn, and you can select observations/models for display by clicking on these tables to highlight rows. Clicking on a row while holding down the CTRL button will usually add/subtract the relevant row from the display list, while doing it without CTRL will select the relevant row as the only one for display.

The continuous coloured line or lines represent model data, and the points with error bars represent observational data points. The vertical bars on each point represent the error associated with the flux at that point, and the horizontal bars represent the width of the band pass. In this case the coloured dots give the position of the Y samples taken from the model data for comparison with the observed points - this is determined from the actual model curve according to some smoothing function as determined by the parameters of the fitting task (typically, by averaging over the band pass width).

Some control over the form of the plot is available. First, you can resize the window to resize the plot. Second, you can zoom in by dragging the mouse down and to the right, or zoom back to "normal" by dragging up and left. Thirdly, by clicking with the right mouse button on the plot you are presented with a number of options to zoom in and out on one or both axes and to print the plot.

The graphical windows displayed by `plotobs` and `plotmodel` work in a similar way.

6 Expression Language

Some of the parameters to Yafit tasks allow you to enter an algebraic expression in terms of named variables or column names. For instance the `x` and `y` parameters in the `tableobs` and `copymodel` have values which are expresions. This section explains the expression language that is used.

Form most purposes, the language is pretty intutive. The normal algebraic operators (`+`, `-`, `*` and `/`) are available, numbers are written in the usual way, and variable names such as those representing table columns must start with a letter and continue with a sequence of letters, numbers and underscores.

As well as these, there are a number of pre-defined functions which can be used. As well as standard ones such as `exp()`, `cos()`, `max()`, `sqrt()` and so on, there are some useful astronomically specific ones, including

- `luminosityDistance(z,H0,omegaM,omegaLambda)`
- `mToMpc(distMetres)`
- `abToJansky(magAB)`
- `janskyToAb(fluxJansky)`

and others. These are not described here, but you can see full documentation of all the available functions by running the `funcs` task.

The expression language is powerful and extensible, however, most users of Yafit will not need to use many of its features, so it is not discussed in detail here. For a full description see the manuals for STILTS or TOPCAT, which use the same technology.

A Command Reference

This section gives reference descriptions of each of the tasks available in the Yafit package, with detailed descriptions of their parameters. A more tutorial introduction to their use is provided in some cases in the main part of this document.

A.1 copymodel

`copymodel` is used to prepare files containing model spectra, ready for fitting by the `fit` task. It takes model data in one of the forms described in Section 4 as input, optionally manipulates it in accordance with the parameters, and writes the output to a `.ymod` file suitable for input into other fitting tasks. The main manipulations which can be performed are changing the X and Y values (using the `x` and `y` parameters) and labelling the axes and their units.

See Section 3.2 for some more discussion and examples.

Usage:

```
Usage: copymodel [-help] [-debug]
                 in=<model-file>
                 [ifmt=ymodel|galaxev|starburst99|svotar|sideways-vot]
                 [out=<out-file>]
                 [ofmt=ymodel-fits|ymodel-votable|ymodel-votable-binary]
                 [x=<expr>]
                 [y=<expr>]
                 [xname=<value>]
                 [xunit=<value>]
                 [yname=<value>]
                 [yunit=<value>]
```

Parameters:

`ifmt = ymodel|galaxev|starburst99|svotar|sideways-vot`
Defines the format of the file given by the `in` parameter. See Section 4 for a list of known formats.

[Default: `ymodel`]

`in = <model-file>`
Input model data files. The format is given by the `ifmt` parameter. Multiple model files (in the same format) may be specified by giving this parameter multiple times.

`ofmt = ymodel-fits|ymodel-votable|ymodel-votable-binary`
Gives the format in which the output will be written. The basic format is `ymodel`, but it is possible to choose which variant (FITS, VOTable etc) of that format is used by specifying this parameter. It doesn't matter much which is used though there may be performance implications.

[Default: `ymodel-votable`]

`out = <out-file>`
Gives the name of the output file to which the model data will be written in `ymodel` format.

[Default: `-`]

`x = <expr>`
Formula for the X (wavelength) values in the output model file in terms of the X values in the input model file. The default value of "`x`" simply makes the one equal to the other, however it is possible to use algebraic expressions here in terms of `x` and `y` (the Y values from the input model file). Normal arithmetic operators as well as some special functions

may be used - see Section 6.

[Default: `x`]

`xname = <value>`
Gives a name for the output X axis, such as "Wavelength". This does not affect processing, but may be used to annotate the displayed or output results, so it is a good idea to supply a value if known to reduce confusion.

`xunit = <value>`
Gives a unit for the output X axis, such as "Angstrom". This does not affect processing, but may be used to annotate the displayed or output results, so it is a good idea to supply a value if known to reduce confusion.

`y = <expr>`
Formula for the Y (flux) values in the output model file in terms of the Y values in the input model file. The default value of "`y`" simply makes the one equal to the other, however it is possible to use algebraic expressions here in terms of `y` and `x` (the X values from the input model file). Normal arithmetic operators as well as some special functions such as `abToJansky()` may be used - see Section 6.

[Default: `y`]

`yname = <value>`
Gives a name for the output Y axis, such as "Flux". This does not affect processing, but may be used to annotate the displayed or output results, so it is a good idea to supply a value if known to reduce confusion.

`yunit = <value>`
Gives a unit for the output Y axis, such as "Jansky". This does not affect processing, but may be used to annotate the displayed or output results, so it is a good idea to supply a value if known to reduce confusion.

A.2 fit

`fit` is the task which performs the actual fitting of observed data to model spectra. The main inputs are a set of observations supplied as a `.yobs` file, and a set of model data supplied as a `.ymod` file or in one of the other formats described in Section 4. Some of the other parameters describe how the fitting will be done, and some specify what form the result will be output in.

See Section 3.3 for some more discussion and examples.

Usage:

```
Usage: fit [-help] [-debug]
           model=<model-file>
           [modelfmt=ymodel|galaxev|starburst99|svotar|sideways-vot]
           obs=<obs-file>
           [smoother=square|point]
           [scale=true|false]
           [fitcalc=chi2|poisson|unscaled]
           [gui=true|false]
           [summary=<out-file>]
           [bestfits=<out-table>]
           [bestfitsfmt=<out-format>]
```

Parameters:

`bestfits = <out-table>`
Destination filename for a table containing the fitting results. This will be a table with one

row for each observation giving the observation details and the details of the model which fits it best, along with the goodness-of-fit score and scaling factor if appropriate. The format of the table is given by the bestfitsfmt parameter.

bestfitsfmt = <out-format>
Table format for `bestfits` table. Must be one of the permitted STILTS formats such as `votable`, `fits`, `ascii`, `text`, `csv`, ... See SUN/256 for a full list.

[Default: `(auto)`]

fitcalc = chi2|poisson|unscaled
Determines how the goodness of fit score will be calculated.

- `chi2`: chi squared using value errors
- `poisson`: chi squared using Poisson errors
- `unscaled`: absolute root mean squared distance

[Default: `chi2`]

gui = true|false
If true a window will be displayed plotting the observations and best fit model curves.

[Default: `true`]

model = <model-file>
File containing the models used for fitting. This will typically be a `ymodel` file but may be in some other format. The format is given by the `modelfmt` parameter. Multiple model files (in the same format) may be specified by giving this parameter multiple times.

modelfmt = ymodel|galaxev|starburst99|svotar|sideways-vot
Defines the format of the file given by the `model` parameter. See Section 4 for a list of known formats.

[Default: `ymodel`]

obs = <obs-file>
File containing the observations to be fitted against. This must be in `yobs` format.

scale = true|false
Whether to scale model curves by a multiplicative constant to achieve the best fit. If true, each model curve will be scaled by whatever factor gives the best fit, otherwise its absolute amplitude will be used.

[Default: `true`]

smoother = square|point
Describes how model curves will be sampled over the width of each observation point for the observations listed in the `obs` parameter. The options are currently

- `square`: the model curve will be uniformly averaged over the width of the observation
- `point`: a linear interpolation of the model curve will be taken at the central point of the observation

[Default: `square`]

summary = <out-file>
Gives the destination file for summary output. This is just a list of the best fit model for each observation. By default, it is written to standard output ("-"), but it can be sent to a file by giving its name here.

[Default: `-`]

A.3 funcs

`funcs` is a documentation utility which allows you to browse the functions available in the expression language used when specifying some expressions to Yafit, described in Section 6.

Invoking the command causes a window to pop up on the display with two parts. The left hand panel contains a tree-like representation of the functions available - the top level shows the classes (categories) into which the functions are divided, and if you open these up (by double clicking on them) each contains a list of functions and constants in that class. If you click on any of these classes or their constituent functions or constants, a full descritption of what they are and how to use them will appear in the right hand panel.

Usage:

```
Usage: funcs [-help] [-debug]
```

Parameters:

A.4 plotmodel

`plotmodel` provides a graphical display of the model data in a `.ymod` file or one of the other formats described in Section 4. The upper part of the window shows a table with one row for each of the input models, and the lower part shows a plot with all of the models drawn as different colour curves. You can highlight one or more of these models (plot them in black) by clicking on the relevant row(s) in the table at the top. Use CTRL-click to select multiple rows. Other elements of the GUI are explained in Section 5.

Usage:

```
Usage: plotmodel [-help] [-debug]
                 in=<model-file>
                 [ifmt=ymodel|galaxev|starburst99|svotar|sideways-vot]
```

Parameters:

`ifmt = ymodel|galaxev|starburst99|svotar|sideways-vot`
Defines the format of the file given by the `in` parameter. See Section 4 for a list of known formats.

[Default: `ymodel`]

`in = <model-file>`
File containing the model data to plot. The format is given by the `ifmt` parameter. Multiple model files (in the same format) may be specified by giving this parameter multiple times.

A.5 plotobs

`plotobs` provides a graphical display of the observation data in a `.yobs` file. The upper part of the window shows a table with one row for each of the input observations, and the lower part shows a plot with the points represented by the observations each plotted in a different colour. You can highlight one or more of these observations (overplot them in black) by clicking the relevant row(s)

in the table at the top. Use CTRL-click to select multiple rows. Other elements of the GUI are explained in Section 5.

Usage:

```
Usage: plotobs [-help] [-debug]
               in=<obs-file>
```

Parameters:

`in = <obs-file>`
File containing the observations to be plotted. This must be in `yobs` format.

A.6 tableobs

`tableobs` is used to prepare `.yobs` files from tables that contain observation data. A `yobs` file is the natural format in which Yafit stores information about observed spectra, but observations are not usually originally supplied in this form, so `tableobs` should be used to prepare them in this way. To use this task, you must have a table which contains data from one or more observations and a *key file* which describes what each column of the table means, in particular which columns represent flux values and flux errors at which wavelengths.

The key file is an ASCII table format containing optional comment lines (blank or starting with the '#' character) followed by a series of lines each describing the observations in one band pass. Each of these lines has four space-separated entries:

1. **yColName:** name of the table column containing Y value
2. **yerrColName:** name of the table column containing Y errors (or "`null`")
3. **x:** central X value at which measurements are taken
4. **xWidth:** width of X range over which measurements are taken

The X and Y here are typically wavelength and flux values, so that `x` is typically a band pass central wavelength and `xWidth` is the band pass full width. However, these X and Y values may be interpreted differently if desired, for instance Y may be in magnitudes rather than flux units.

Note that the Y error column may contain the special value "`null`" if no the input table contains no error information for one or more Y values. In this case however beware that error-less Y values will not be used by the `chi2` goodness-of-fit measure in the `fit` task, since chi-squared fitting makes no sense without error estimates. To introduce given non-null error values you should pre-process the table (e.g. using STILTS) to provide a column containing suitable numeric values.

The input table which the key file describes is a table in any of the formats recognised by the SUN/256 package, including FITS, VOTable, CSV and ASCII. In general it is necessary to say what format it is in with the `ifmt` parameter.

See Section 3.1 for some more discussion and examples.

Usage:

```
Usage: tableobs [-help] [-debug]
                in=<table>
                [ifmt=<in-format>]
                key=<key-file>
                [out=<obs-file>]
                [redshift=<expr>]
                [x=<expr>]
                [y=<expr>]
```

```
[xname=<value>]
[xunit=<value>]
[yname=<value>]
[yunit=<value>]
```

Parameters:

ifmt = <in-format>
Gives the format of the table located by the in parameter (votable, fits, csv etc).

[Default: (auto)]

in = <table>
Location of the table which contains the input observation data. The columns of this table must contain the Y (flux) values and errors at different band passes. Which column means what is specified by the key parameter. The table located by this parameter may be in any of the formats supported by STIL, as defined by the ifmt parameter.

key = <key-file>
Location of file containing the four-column *key* file that describes which columns of the input table give which information. See the command description for more detail.

out = <obs-file>
Name of the file to which the output yobs file will be written.

[Default: -]

redshift = <expr>
May give an expression for redshift of the input observations in terms of quantities in the input table. If there is one, this will normally be expressed just as the name of the column in the input table which gives redshift. However, it can be an algebraic expression combining column names from the input table - see Section 6. A redshift should only be given if the X values are wavelength (not, for instance, if they are frequencies) since otherwise the fitting program will not treat them properly.

x = <expr>
Formula for the X (wavelength) values in the output observation file in terms of the X values in the key file. The default value of "x" simply makes the one equal to the other, however it is possible to use algebraic expressions here in terms of x, for instance in order to modify the scaling (change units). Normal arithmetic operators as well as some special functions may be used - see Section 6.

[Default: x]

xname = <value>
Gives a name for the output X axis, such as "Wavelength". This does not affect processing, but may be used to annotate the displayed or output results, so it is a good idea to supply a value if known to reduce confusion.

[Default: X]

xunit = <value>
Gives a unit for the output X axis, such as "Angstrom". This does not affect processing, but may be used to annotate the displayed or output results, so it is a good idea to supply a value if known to reduce confusion.

y = <expr>
Formula for the Y (flux) values in the output observation file in terms of the Y values in the input table. the default value of "y" simpley makes the one equal to the other, however it is possible to use algebraic expressions here in terms of y, x (the X values from the input key file), and any of the values in the input table referred to by their column name. Normal arithmetic operators as well as some special functions such as abToJansky() may be used - see Section 6

[Default: y]

yname = <value>

Gives a name for the output Y axis, such as "Flux". This does not affect processing, but may be used to annotate the displayed or output results, so it is a good idea to supply a value if known to reduce confusion.

[Default: Y]

yunit = <value>

Gives a unit for the output Y axis, such as "Jansky". This does not affect processing, but may be used to annotate the displayed or output results, so it is a good idea to supply a value if known to reduce confusion.