Absorption **LI**ne **S**oftware
ALIS, version 1.0

Ryan J. Cooke

June 4, 2015

Contents

1 Introduction

If you're looking for software that will help you to fit spectral features, then you've come to the write place, otherwise, you (probably) won't want to continue reading. This package was originally written as "Absorption LIne Software" (i.e. ALIS), but has now been customised so that you can fit pretty much anything you want (i.e. emission, absorption or both!). In many aspects (and in other aspects not), ALIS is similar to code VPFIT, written by Bob Carswell, John Webb, and others. If you're wanting to perform Voigt profile absorption line fitting using chi-squared minimization, I would suggest that you download, install and use both ALIS and VPFIT. They should both (in principle) give you the answer.

This 'manual' is designed to be a user (and troubleshooting) guide that will help you to either install, update, run and develop ALIS. There is also a troubleshooting section (see Section 10) which you should consult if you run into any trouble while performing the above actions.

As with all software packages, this one isn't perfect. If you run aground by trying to do something I (or others) haven't yet tried or tested, then please contact me. I would happy to try and incorporate your request in the next version, or suggest how best to get around your problem in the near future (but please consult the troubleshooting guide first, as your answer may be there!).

Throughout this guide, you will see several grey text boxes

> ... a bit like this one

These boxes indicate commands that should be issued at either a terminal, or may indicate code that is used by ALIS. At the time of first writing this documentation, I'm certain that I've forgotten to list *every* feature that is implemented. If you feel something could be documented or explained better, please let me know. Finally, although it is absolutely not necessary, if you would like to use the code for your work and want to cite it correctly, please use the following reference:

Cooke R. (2016), JOURNAL, REFERENCE.

Other than that, thanks for using ALIS, and I hope you find it helpful!

1.1 Software and Documentation License Agreement

2 Installing and Updating ALIS

ALIS is written in the PYTHON software environment, it can be used (at least in principle) either as it comes (recommended), on the python command line, or can be imported into your own PYTHON code.

2.1 Software Requirements

I tried to use as few "additional" packages as possible when writing ALIS, but you really can't beat the following PYTHON essentials (in fact, they are probably already installed on your system, and if not, they should be!). If you want to use ALIS, you will need to download and install the following packages:

- PYTHON (required, version $\geq 2.6.5$)
- NUMPY (required)
- MATPLOTLIB (required)
- ASTROPY (required, version ≥ 0.2)
- CUDA (optional — only needed if you want to use GPU multiprocessing – in development)
- PYCUDA (optional — only needed if you want to use GPU multiprocessing – in development)

2.2 Recommended Installation

I strongly advise you to install python and the other dependencies using 'Anaconda' or 'Canopy'. In what follows, I will assume that you wish to use Canopy as your python install.

Download and install Enthought Canopy, which can be downloaded from the following website: `https://www.enthought.com/products/canopy/`

This should be fairly straightforward. Once installed, open Canopy and click on the 'Package Manager'. In this new window, search for and install 'astropy'.

In your `~/.cshrc` file (or the equivalent file for the shell you are using), create an alias for the Canopy version of python. On a Mac, it will be something like:

```
alias python "/Applications/Canopy.app/appdata/canopy-1.4.0.1938.macosx-x86_64/Canopy.app/
              Contents/MacOS/python"
```

where `canopy-1.4.0.1938.macosx-x86_64` will probably be different for your computer. You will also need to set the `PYTHONPATH`, which defines the location of the installed python packages. If you open Canopy, click on the 'Editor' (this option was available on the same screen where you selected 'Package Manager'). Select 'Create new file', and insert the following text:

```
import numpy
import matplotlib
import astropy
print numpy.__file__
print matplotlib.__file__
print astropy.__file__
```

Click the green play button at the top of the window (i.e. the green triangle pointing to the right), and this will generate some text in the bottom panel. It might look something like (on a Mac):

```
/Users/rcooke/Library/Enthought/Canopy_64bit/User/lib/python2.7/site-packages/numpy/__init__.pyc
/Users/rcooke/Library/Enthought/Canopy_64bit/User/lib/python2.7/site-packages/matplotlib/__init__.pyc
/Users/rcooke/Library/Enthought/Canopy_64bit/User/lib/python2.7/site-packages/astropy/__init__.pyc
```

In this example, you will need to set your PYTHONPATH environment variable to:

```
setenv PYTHONPATH "/Applications/Canopy.app/appdata/canopy-1.4.0.1938.macosx-x86_64/
                   Canopy.app/Contents/lib/python2.7/site-packages/"
setenv PYTHONPATH ${PYTHONPATH}:"/Users/rcooke/Library/Enthought/Canopy_64bit/
                                 User/lib/python2.7/site-packages"
```

Note that the first line of the PYTHONPATH points to the Canopy application directory (and is similar to – but not the same as – the alias you set above for python). The second line appends a directory to the PYTHONPATH, which tells python where to find the numpy, matplotlib, and astropy libraries. The directory to use on the second line is determined from the python script you just executed. By inspecting the printout, you only need to specify the full directory path up to the 'site-packages' folder (i.e. ignore the `numpy/__init__.pyc` bit). If one of these packages is located in a different directory, you will need to append this directory to the PYTHONPATH as well. In the example above, all packages are located in the same directory.

Once your python path is set, you are ready to proceed with the Section 2.3. If you want to check that you've setup python correctly, open a new terminal window, type `python` at the command prompt, and try to import `numpy, matplotlib,` and `astropy`, with the command `import numpy` etc. If you receive no errors, then you have setup your python environment correctly.

2.3 Installing ALIS

I assume that you succeeded with your python setup... If not, ALIS probably won't work very well (if it all). There are just 3 simple steps to install ALIS:

Step 1. Go to the directory you keep all of your software (or the directory you want to install ALIS), for example:

```
cd /home/username/software/
```

Step 2. Check out the latest version of the code:

```
git clone https://github.com/rcooke-ast/ALIS.git
```

Step 3. Create the following alias for the software (place this in your .cshrc file):

```
alias alis 'python /home/username/software/ALIS/src/alis.py'
```

Step 4. (Optional) Add the ALIS directory to your `pythonpath` (if you want to call ALIS from within your own PYTHON script):

If you have already specified your `pythonpath` somewhere in your .cshrc, use:

```
setenv PYTHONPATH $PYTHONPATH:/home/username/software/ALIS/src
```

otherwise (this is the first `pythonpath` you've specified — are you sure?), use:

```
setenv PYTHONPATH /home/username/software/ALIS/src
```

That's it! But don't forget to source your .cshrc file for steps 3 & 4 to take effect.

3 How ALIS works

This section is designed to give you a brief overview of how the software works behind the scenes, and provides some suggestions for how you can change/modify aspects of the code (without harming ALIS!).

In general, you will need to create a `.mod` model file (e.g. `myfirstfit.mod`) which tells ALIS how to run. You will then execute the code by typing the following on a command line[1]:

```
alis myfirstfit.mod
```

From here, ALIS will read in the default settings, change these settings as you specify in the `.mod` file, load the data, load the model, fit the model to the data, and provide a bunch of output (which

[1] Yes, a GUI is on the cards... eventually.

you can turn on/off as you require). I'll now briefly describe each of these separately in the following sections.

At this point, I wish to acknowledge that I have used the MPFIT software package (Markwardt, 2009), which I have modified for the purposes of ALIS. MPFIT was originally called LMFIT when it was a FORTRAN code long, long ago. From here, it was rewritten in IDL by **Craig Markwardt** (and received the new name MPFIT), and was then converted into PYTHON by **Mark Rivers**. Finally, **Sergey Koposov** updated the PYTHON version of MPFIT to use NUMPY, rather than its predecessor (NUMERIC). The current (and all preceding versions) of MPFIT uses a Levenberg-Marquardt least-squares minimisation algorithm to derive the model parameters that best fit the data (i.e. the parameters that minimise the difference between the data and the model, weighted by the error in the data). The underlying assumption here is that the data are normally-distributed. The relevant updates that I have made to MPFIT includes CPU multiprocessing, a more informative screen output that is suited to the ALIS software, and a few (almost) inconsequential fixes.

3.1 The settings.alis file

This file contains the default settings that ALIS needs in order to run. I do not recommend that you change the parameters in this file (since you can change each and every setting in your `.mod` file). But if you insist on changing something, here is a brief description of what you can/can't do. This file contains a three-argument command. The first argument is just a name (or identifier) that ALIS uses. The second argument is the name given to a particular setting. The third argument is the value of this setting.

The first argument can take the following values: `run` (which tells ALIS how to run), `chisq` (to set the details of the chi-squared minimisation, including convergence properties), `plot` (to describe how the data/model should be prepared for plotting), `out` (the information you want to output), `sim` (to perform Monte Carlo simulations), and `generate` (to generate fake data).

Table 1 provides a summary of the permitted arguments/values, and includes a description for each of the parameters. As mentioned earlier, you can also change these parameters manually for each fit that you perform, by including the same three argument commands (separated by whitespace or tabs) in your `.mod` input model file.

Table 1: THE ALLOWED PARAMETER SETTINGS FOR ALIS

Arg. 1	Arg. 2	Default Value	Allowed Values	Description
`run`	`atomic`	atomic.xml	str	The name of the table that contains the relevant atomic transitions (must be a VOTable with suffix `.xml`)
`run`	`bintype`	km/s	km/s, A	Set the type of pixels to use – If pixels have constant velocity use "km/s", if pixels have constant Angstroms use "A"

Continued on next page...

Table 1 – continued from previous page

Arg. 1	Arg. 2	Default Value	Allowed Values	Description
`run`	`blind`	True	True, False	Run a blind analysis – In order to print out the best-fitting model, this must be set to False
`run`	`capvalue`	None	None, float > 0.0	If not None, set the model value to be the capvalue everywhere the model exceeds the capvalue
`run`	`convergence`	False	True, False	Run a convergence check
`run`	`convnostop`	False	True, False	Continue to reduce the chi-squared stopping criteria until the parameters have converged
`run`	`convcriteria`	0.2	float > 0.0	Convergence criteria in units of the parameter error
`run`	`datatype`	default	string	Specify the type of data being read in (the default is the IRAF standard). `HIRESredux` will read in data reduced by HIRES_REDUX. `UVESpopler` will read in data combined with UVES_POPLER.
`run`	`limpar`	False	True, False	If an initial parameter's value is outside the model limits, set the initial value to the limit. Otherwise, an error is raised.
`run`	`ncpus`	-1	int	Number of CPUs to use (-1 means all bar one CPU, -2 means all bar two CPUs)
`run`	`ngpus`	0	int ≥ 0	Number of processes to send to the GPU/GPUs (must be ≥ 0)

Continued on next page...

Table 1 – continued from previous page

Arg. 1	Arg. 2	Default Value	Allowed Values	Description
`run`	`nsubpix`	5	int ≥ 0	Number of sub-pixels per 1 standard deviation to interpolate all models over – higher values give higher precision
`run`	`nsubmin`	5	int ≥ 0	Minimum number of sub-pixels per 1 pixel to interpolate all models over – higher values give higher precision
`run`	`nsubmax`	21	int ≥ 0	Maximum number of sub-pixels per 1 pixel to interpolate all models over – higher values give higher precision
`run`	`renew_subpix`	True	True, False	If True, the subpixellation will be calculated after every iteration
`run`	`warn_subpix`	100	int ≥ 0	If the number of sub-pixels exceeds this amount, the user will be warned
`chisq`	`atol`	0.001	float ≥ 0.0	Termination criteria – this measures the absolute change desired in the sum of squares
`chisq`	`ftol`	1.0E-10	float ≥ 0.0	Termination criteria – this measures the relative error desired in the sum of squares
`chisq`	`gtol`	1.0E-10	float ≥ 0.0	Termination criteria – this measures the orthogonality desired between the function vector and the columns of the Jacobian
`chisq`	`xtol`	1.0E-10	float ≥ 0.0	Termination criteria – this measures the relative error desired in the approximate solution

Continued on next page...

Table 1 – continued from previous page

Arg. 1	Arg. 2	Default Value	Allowed Values	Description
`chisq`	`fstep`	20.0	float > 0.0	Factor above machine-precision to use for step size
`chisq`	`maxiter`	20000	int > 1	Maximum number of iterations before giving up
`chisq`	`miniter`	1	int > 1	Minimum number of iterations before checking convergence criteria
`plot`	`fits`	True	True, False	Plot the best-fitting model with the data?
`plot`	`residuals`	False	True, False	Plot the residuals for the best-fitting model?
`plot`	`only`	False	True, False	Don't fit the data, just plot the input data and model
`plot`	`pages`	all	int, all	Which pages to plot? Either provide comma-separated integers or the text string 'all'
`plot`	`dims`	3x3	(int)x(int)	Specify the plotting dimensions (ROWSxCOLUMNS)
`plot`	`fitregions`	False	True, False	Indicate the regions of data that are being used in the chi-squared minimation (i.e. that specified by fitrange)?
`plot`	`ticks`	True	True, False	Plot tick marks above the spectrum to indicate model components?
`plot`	`ticklabels`	False	True, False	Plot labels above the tick marks to identify model components?
`out`	`covar`	""	str	Output the covariance matrix, to a file with name given by third argument (No output if the argument is "")

Continued on next page...

Table 1 – continued from previous page

Arg. 1	Arg. 2	Default Value	Allowed Values	Description
`out`	`convtest`	""	str	Output the details of the convergence test, to a file with name given by third argument (No output if the argument is "")
`out`	`fits`	False	True, False	Output the best fitting model fits?
`out`	`onefits`	False	True, False	Output the best fitting models to a single fits file?
`out`	`model`	True	True, False	Output the best fitting model parameters?
`out`	`overwrite`	False	True, False	Overwrite existing files when writing out?
`out`	`plots`	""	str	Save the output plots to a pdf file, to a file with name given by third argument (No output if the argument is "")
`out`	`sm`	False	True, False	Generate a SuperMongo plotting script?
`out`	`reletter`	False	True, False	Set to True if you want ALIS to reletter the tied/fixed parameters starting from 'A'
`out`	`verbose`	2	0, 1, 2	Level of screen output (0 is for no screen output, 1 is low level output, 2 is output everything)
`sim`	`random`	None	int > 0	Number of random simulations to perform
`sim`	`perturb`	None	int > 0	Number of simulations to perform where the starting parameters are perturbed
`sim`	`systematics`	False	True, False	In addition to running random simulations, do you want to run systematics simulations? `sim+random` must be > 0 to run systematics simulations.

Continued on next page...

Table 1 – continued from previous page

Arg. 1	Arg. 2	Default Value	Allowed Values	Description
`sim`	`beginfrom`	""	str	An input file called `<filename>.mod.out` that contains the starting parameters and errors for the simulations (NOTE: you will need to also output the corresponding covariance matrix for these parameters, with a filename `<filename>.mod.covar`)
`sim`	`startid`	0	int ≥ 0	A starting ID label for the Monte Carlo simulations. This will be incremented by 1 for each new simulation until `sim+random` simulations have been performed.
`sim`	`systmodule`	None	str, None	If the user writes their own module to deal with systematics, specify the name of this module here. None will use the default, built-in systematics
`sim`	`newstart`	True	True, False	Generate a new set of starting parameters from the best-fitting covariance matrix?
`sim`	`dirname`	sims	str	Name of the folder to dump the output from the Monte Carlo runs
`sim`	`edgecut`	4.0	float ≥ 0.0	Number of standard deviations (based on instrumental FWHM) to reject from generated data (due to edge effects)
`generate`	`data`	False	True, False	Generate fake data (instead of fit)?
`generate`	`pixelsize`	2.5	float > 0.0	Pixel size (in units of `run+bintype`) for the generated wavelength array

Continued on next page...

Table 1 – continued from previous page

Arg. 1	Arg. 2	Default Value	Allowed Values	Description
`generate`	`peaksnr`	0.0	float ≥ 0.0	Signal-to-noise ratio (at the peak of the model) for the generated data (0.0 is used for perfect data)
`generate`	`skyfrac`	0.0	float ≥ 0.0	What is the fractional contribution of the sky (relative to the peak of the model). The condition `skyfrac` $<$ (peak of model / `peaksnr`) must hold.
`iterate`	`model`	None	str, None	Make dynamic changes to the model using a user-specified function ('None' means do not iterate). Two arguments (separated by a comma - no spaces) are allowed, but not necessary. The first argument is the name of the module, the second is any text string that you want passed to the module.

where int, float and str respectively refer to an integer, floating point, and character string value.

3.2 The atomic.xml file

This file may never concern you, but it is good to know what it does, just in case you ever need it. This file contains all of the atomic data for a given list of atomic transitions that I think will be useful to you. It is largely derived from the compilation by Bob Carswell, and is used in VPFIT (with a few updates that I've found useful). This is a file that is continuously updated to include the latest laboratory measurements and additional transitions that others find useful. If you would like to easily view, update, or add new values to this table, I would recommend TOPCAT. However, I suggest that you make your own copy of this file, make changes to it, and load *your* version of the atomic data file into ALIS. This can be done by either changing the default atomic data file listed in `settings.alis` (i.e. `run atomic atomic.xml`), or you can change this setting in your `.mod` file. That said, if you have data for new transitions that I have not included, (and you think they may also be of use to someone else), please let me know, and I will update the default version too. The column data in atomic.xml are as follows:

(1) Mass Number
(2) Atomic Mass (in amu)
(3) Solar Isotopic **Number/Mass** — **TBC** Abundance
(4) Element Name
(5) Ionization stage
(6) Vacuum Wavelength (in Å)
(7) Oscillator Strength
(8) Transition Probability (in s^{-1})
(9) The q-value of the transition (for use with fine-structure constant variation)
(10) The K-value of the transition (for use with proton-to-electron mass ratio variation)

3.3 The `.mod` model file

The `.mod` file contains all of the information that ALIS needs in order to perform a fit. In short, you need to include three sections to your `.mod` file, the first section is where you can adjust the default settings of ALIS (for example, change the number of CPUs that the code uses to find the best-fitting solution). The second section provides the details of a given data file that you want ALIS to read in. The third section contains the details of the model you wish to fit to the data. There's an optional fourth section, which tells ALIS to link a model parameter to another parameter, through some expression. Therefore, your `.mod` file should look something like the following:

```
# ALIS will ignore all text to the right of a '#' symbol.
# This is a comment line.

#- ->
ALIS will also ignore all script between these comment & arrow symbols.
<- -#

<three parameter arguments>

data read
<tell ALIS where the data are and what to do with it>
data end

model read
<tell ALIS what model you want to fit to the data>
model end

link read
<tell ALIS what links to enforce between model parameters>
link end
```

where the bold arguments in angle brackets are to be chosen from a list of commands that are described in the following three subsections.

3.3.1 Three parameter arguments in your `.mod` file

The first thing to decide is how you want to change the default settings that were described in Section 3.1. You would do this in the same way as you would define/change the commands in the `settings.alis` file. For example, if you wanted to change the number of CPUs that ALIS uses for the calculation to 8, and you would like to output the best-fitting model fits and be sure that old fits were overwritten (without being prompted by ALIS), you would need to include the following three parameter commands at the top of your `.mod` file:

```
run ncpus 8
out fits True
out overwrite True
```

3.3.2 How to read in a data file

Once you've set the details of how ALIS should run, you need to specify the data, and a list of the corresponding properties of that data. The first argument on every new line must be the pathname to the datafile (you can either specify the absolute path [e.g. `/home/data/datafile.fits`] or the relative path [e.g. `../work/datafile.fits`] from the `.mod` file). If you specify nothing more on this line, ALIS will assign this datafile the default properties (which is probably not what you want!). It is recommended that you manually assign the properties of this datafile on the remainder of the line. The allowed properties you can set (along with a brief description — see later for more details on these properties) are the following:

- `specid` — An ID number that links the data and model (ALIS will fit data of a given `specid` with the corresponding model of the same `specid`)
- `fitrange` — The wavelength range of the input data that should be used for the fit
- `loadrange` — The wavelength range of the input data that should be loaded and used to generate the model (**NOTE:** this command specifies the wavelength range where the model should generated, whereas `fitrange` tells ALIS where to calculate the chi-squared)
- `resolution` — Defines the model for the instrumental broadening profile
- `shift` — Defines the model for the relative shift between two different spectra
- `systematics` — define where ALIS should obtain systematics information from (only if you want to calculate systematics – see Section **??**)
- `systmodule` — A user-defined module to deal with systematics (only if you're doing systematics – see Section **??**)
- `columns` — The columns of the datafile that should be read by ALIS
- `loadall` — Load all of the data (not just the fitted region)
- `bintype` — This parameter allows you to override the default `run+bintype` for a given set of data

- `nsubpix` — This parameter allows you to override the default `run+nsubpix` for a given set of data.
- `plotone` — Set to True if you want to plot this data in its own panel (rather than what is specified by `plot+dims`)
- `label` — A label that you would like to have plotted on the lower left corner of the plot
- `yrange` — An optional command if you're unsatisfied with the automatic y-axis range of the plotted data

Note that every line in your `.mod` file between the `data read` and `data end` commands is considered independently of all previous lines. This means that you need to specify the data properties on every line (or you will just get the default settings). In general, to set a keyword property type the keyword followed by an '=' sign, followed by the value to give that keyword with no spaces! For example, to set `specid` to 1, you would type `specid=1`. All of the above options are now described in more detail.

specid — Any model with the corresponding `specid` parameter will be applied to this data. ALIS reads this parameter as a character string, so it can be any value. Note that you can specify the same `specid` for several datafiles. In this instance, the model with the corresponding `specid` will be applied to both datafiles. The only instance where this parameter is not needed is when you want every specified model to be applied to all of the datafiles.

fitrange — This is a required (... well, not really, but it is highly recommended that you do!!) parameter which tells ALIS where to fit the model to the data. It can take three arguments. The first is the string 'all' (which is the default – without quotation marks), and will fit the model to all of the data. This is not recommended if you are convolving the data, since your model will contain spurious edge effects. If you want to fit only a certain wavelength interval, you can specify the minimum (e.g. 3500.0 Å) and the maximum (e.g. 3600.0 Å) by typing `fitrange=[3500.0,3600.0]` where the comma is required (no spaces!). The final argument you are allowed to pass in is the string 'columns' (without quotation marks), which tells ALIS to obtain the wavelengths to be fitted from the datafile itself. If you specify `fitrange=columns`, you will need to also specify the column number in the parameter `columns` (see below). The column data needs to be a series of 1's and 0's, where a 1 indicates that you would like to include a pixel in the fit, and a 0 tells ALIS not to use that pixel during the fitting process.

loadrange — This is a highly recommended input parameter which tells ALIS over what wavelength range should the model be generated. It can take two arguments. The first is the string 'all' (which is the default – without quotation marks, i.e. `loadrange=all`), and will generate a model over the entire wavelength range (this can be slow if you're reading in a large file!). This option is not recommended if you are convolving the data, since your model will contain spurious edge effects. If you want to generate the model within a certain wavelength interval, you can alternatively specify the minimum (e.g. 3480.0 Å) and the maximum (e.g. 3620.0 Å) by typing `loadrange=[3480.0,3620.0]` where the comma is required (no spaces!).

resolution — This keyword is required, and you should always explicitly define it. ALIS presently has three built-in functions for an instrumental profile (`Afwhm`, `vfwhm` and `vsigma`). These functions convolve the model with a Gaussian profile: Use `Afwhm` if you want to specifiy the full-width at half-maximum resolution in Angstroms, or use `vfwhm` if you want to specifiy the full-width at half-maximum velocity resolution (i.e. c/R), or use `vsigma` is you want to specify the standard deviation. Since the argument of this parameter is a function, you will also need to specify the parameters of the function. At present, these functions take a single argument (which is a constant). For example, if the instrumental resolution is known to have FWHM velocity of 7.0 km s^{-1}, you should use the command `resolution=vfwhm(7.0)` with no spaces. Since this is a function, the parameters of this function are allowed to be a free

parameter of the model. In Section 3.3.3, you can find out how to fix, tie, or limit the parameters of this function. If you need an instrumental profile that is not built-in, (e.g. if the resolution changes linearly with wavelength), or you want to define some arbitrary function for the profile, you can do this by (straightforwardly) writing your own function, and loading it into ALIS. For more details on writing your own functions, see Section 9. The `resolution` parameter can also be set to the string 'columns', to read the instrumental resolution from a column in the input datafile. In this case, you must specify the velocity full-width at half-maximum at every pixel. If you do not want to convolve the data with an instrumental profile, just use either `vfwhm` or `vsigma` with an argument of 0.0 (and be sure to fix the resolution to this value later when specifying the model — see Section 3.3.3). In Section 3.3.3, there is a description for how you can fix and tie the parameters of the instrumental resolution profile with uppercase and lowercase letters respectively.

shift — This keyword is not required, but is used to specify (or fit) a relative shift between two spectra (for example, if you want to fit for the heliocentric correction). ALIS presently has two built-in functions that allow you to specify the shift (`Ashift`, `vshift` and `vsigma`). Use `Ashift` if you want to shift this spectrum by a constant amount in Angstroms, or use `vshift` if you want to set the shift to be a constant velocity shift. Since the argument of this parameter is a function, you will also need to specify the parameters of the function. At present, these functions take a single argument (which is a constant). For example, if the velocity shift is estimated to be +30.0 km s^{-1}, you should use the command `shift=vshift(30.0)` with no spaces. Since this is a function, the parameters of this function are allowed to be a free parameter of the model. In Section 3.3.3, you can find out how to fix, tie, or limit the parameters of this function. If you need to specify a shift that is not built-in, (e.g. if the shift changes as a function of wavelength), or you want to define some arbitrary function for the shift, you can do this by (straightforwardly) writing your own function, and loading it into ALIS. For more details on writing your own functions, see Section 9. In Section 3.3.3, there is a description for how you can fix and tie the parameters of the shift with uppercase and lowercase letters respectively.

systematics — This keyword is to be used if you want to quantify the systematics that affect your data. It can (essentially) take one of three arguments for this version of ALIS. Specifying the string 'columns' will tell ALIS that the relevant systematics information is located in one of the columns of the datafile. To set the column number of the systematics information, use the `columns` keyword below. You can also specify the string 'None' which implies that no 'extra' information is required. Finally, you can specify the path (absolute or relative), where a datafile with the relevant systematics can be found.

systmodule — If none of the built-in systematics routines fit your purpose, you can (fairly easily) write your own module to deal with systematics. If you want to do this, the argument should provide the name of the `.py` file followed by a comma, followed by a string which you can use as an identifier. For example, you would type `systmodule=mysystfile.py,myidstring` to load `mysystfile.py` with an identifier string `myidstring`. Two arguments, separated by a comma – with no spaces – are *required* for this parameter. If you want to write your own systematics module, please see Section 7 for the relevant details.

columns — The columns keyword is required, and informs ALIS which columns of the datafile contain the relevant information. Within square brackets, you can include the following information: `wave`, `flux`, `error`, `continuum`, `fitrange`, `systematics`. ALIS uses zero-indexed data, so if your wavelength array is the first column of data, you will need to specify `wave:0`. The minimum keywords that you need to specify are `wave`, `flux`, and `error` (if you're using `.fits` files, you only need to specify the flux and error columns, since the wavelength array is read from the header). If you gave the keyword 'columns' to either `fitrange` or `systematics` (see above), then you should specify here which column of data to obtain it from. For example, if the wavelength, flux, and error arrays are in the first, second and third columns, whilst the fitrange is in the fifth column, you would need to set (without

any spaces!) `columns=[wave:0,flux:1,error:2,fitrange:4]`. The keyword `continuum` will multiply the final model by a fractional amount of the continuum. I *do not* recommend that you use your own continuum fits, but rather, you should model the continuum self-consistently with ALIS (yes, you can do this!). The `continuum` option should only be used if you *really* need it.

loadall — ALIS will try to load only the data that it needs for accurate fitting (i.e. it will take the fitted regions you specify plus a bit more so that edge effects from convolution does not affect your model profile). If you want to, you can force ALIS to load all of the data in this file (it will slow things down only slightly — depending on how big your data are!). `True` and `False` are the only two arguments that are allowed for this parameter. To load all of the data, use the command `loadall=True`.

bintype — This parameter accepts the strings (without quotation marks) 'km/s' (if your wavelength bins have a constant velocity) or 'A' (if your wavelength bins have a constant Angstroms). This is an optional keyword which, if not specified, will assume the default value set by `run+bintype`. Conversely, if you choose to specify `bintype` for a given datafile, this will override the default setting for *just this datafile*.

nsubpix — This parameter is an optional keyword which is used by ALIS to subpixelate the spectrum (one standard deviation for the model is sampled by the number of pixels specified with this keyword). If you do not specify this keyword, ALIS will assume the default value given by `run+nsubpix`. If you choose to specify `nsubpix` for a given datafile, this will override the default setting for *just this datafile*.

plotone — If you are plotting your results, ALIS will use the default `plot+dims` arguments toplot your data in different panels. `plotone` is an optional parameter which allows you to override the default plotting panels, and instead plot this datafile and the best-fitting model with a single plot. This parameter takes `True` or `False` arguments; to plot a given datafile in its own window, use the command `plotone=True`.

label — If you would like to plot a label on a given panel to easily recognize the plot, provide a label here, with no spaces and no quotation marks. If you wanted spaces, use the underscore character (i.e. _). For example, if you want to highlight that a given panel is Si II λ1808, you would use the command `label=Si_II_1808`.

yrange — ALIS will usually derive a reasonable y-axis range for plotting, but you can also specify it manually if you prefer. For example, you can specify the minimum value of the y plotting axis (e.g. -0.3) and the maximum (e.g. 1.5) by typing `yrange=[`$-0.3, 1.5$`]` where the comma is required (and there are no spaces!).

3.3.3 How to specify your model

Now that the data have been specified, you need to provide the model that is fitted to the data. The entire model must be specified between the `model read` and `model end` commands. There are several sections that you need to fill out in order to appropriately specify the model. The first step is defines the global properties of the model (i.e. what parameters should be globally fixed (or not), and what the appropriate limits are for the parameters etc.) In most cases, the default values for the models are sufficient, but if you want to change them for this `.mod` file, you can do it here. The next step is to define the model for emission, then absorption (if you want to fit absorption features!), and finally, you can perform a fit to the zero-level of the data (but this feature should probably not be used unless you know it well (and can fix it to a known value), or you have strong (saturated) absorption features where you can accurately determine, or fit to, the zero-level). These sections are now described separately.

A quick note on specifying model parameters — Each model contains a number of parameters, and a number of keywords. Not all parameters and keywords are required, but some are. If you don't specify all parameters for a given model, ALIS will take the default values (which are sensible –

but may not be what you're after!). Some keywords are absolutely required (don't worry, ALIS will flag an error if you don't specify them!). If you want to run a blind analysis (i.e. you only see the fits and do not see the results) you can globally specify the `run+blind` command (this will force the entire run to be blind — see Table 1), or you can specify `blind=True` on a given model line to blind just that model.

A quick note on fixing and tying model parameters — ALIS has the ability to tie and/or fix parameters that should be the same. A good example is a series of Gaussian emission lines that you want to tie together so that they have the same redshift, and this redshift is a free parameter. In order to tie the redshift of these two Gaussians together, ALIS requires a text string that immediately follows the parameter. If your initial model guess of the redshift is, say, 0.5 then the parameter you would use could be 0.5tieit (where 'tieit' is an arbitrary length (user-defined) string of lowercase letters only — you could also use 'skfg'). Every model parameter (with the string 'tieit' after the model parameter) that is encountered by ALIS after this first instance, will inherit the first instance of 'tieit' (in this case 0.5 — regardless of the model type/parameter). In other words, 'tieit' is thereafter reserved as 0.5, and once fitting has commenced, all values with 'tieit' will change together. If you instead want to fix parameters, use UPPERCASE letters only. Note that uppercase letters, while fixed, are also tied. For example, if you had instead set the redshift above to 0.5R, and you later had defined a model with the parameter 10.0R, this second model parameter will still be a fixed number, but it will fixed at 0.5, rather than 10.0. More details (including examples) for fixing and tying parameters are provided below.

Specifying the emission model — For every set of data that you want to fit, you must specify a model that describes the emission. For this example, I will use three Gaussian functions and a constant function. Since it is emission, it doesn't matter what order we specify these models. Also, we decide (for a good reason) that we want to tie the redshift for these three gaussians, and we know that the data are offset by a constant of 1.0. The first thing that needs to specified is that all of these models are emission. This is achieved by issuing a line with the text `emission` (and nothing else). Every subsequent line of the model will be interpreted as emission. The model definition would be (explicitly) given by:

```
model read
emission
constant value=1.0CONST
gaussian amplitude=9.0 redshift=0.5tval dispersion=1000.0 wave=H_I_1215
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1238
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1242
model end
```

In this example, the value of 1.0 used for the constant is fixed with the text string 'CONST', and the redshift of the three Gaussians are tied (but free) with the string 'tval'. For more details on all of the built-in models available with ALIS, see Section 8. For now, note that every model has a number of keywords (some are absolutely required) in addition to the "required" parameters (whilst not formally required, if you don't specify the "required" parameters, defaults will be used, which is not advised!). Default values cannot (or rather, will not) be used for the keyword arguments. The function called `constant` in the above example has a single parameter (called `value`), and one keyword argument (called `specid`).

In the above example, ALIS thinks you have not specified a `specid` for the data, and will therefore apply all four of these models to all of the data. If you do not wish for this to happen, simply set the keyword argument `specid=1` (with no spaces!) Commas can be used to separate different values of `specid`. Therefore, to apply a given model to the data where `specid` is 1, 2, and 3, you would use the

keyword argument `specid=1,2,3` again with no spaces.

To avoid confusion, I have explicitly defined the single parameter for the `constant` model as `value=1.0CONST`. I could have instead simply used `1.0CONST` and remove the 'value=' part of the parameter definition. This is because ALIS expects to read in the model parameters in a certain order. If you do not explicity tell ALIS what the parameter name is that a given number should be assigned to, it will do this automatically. Note that keywords do not work in the same way. You must specify the entire keyword as shown above for `specid`. For further detail, see Section 8 for the order of parameters that is used in the built-in functions. If you explicitly give the parameter name, order is not a problem. If you only provide some of the parameter names explicitly, ALIS will assign these first, and then assign the remaining keywords in the appropriate order.

Specifying the absorption model — Once the emission model is defined, you can then specify the absorption that is superimposed on top of this emission. The first step is to write the keyword 'absorption' on a new line by itself, followed by the models you wish to define as absorption. In the following example I will assume there is a damped Lyman-α system providing the absorption on top of the Gaussian emission that I defined above

```
model read
emission
constant 1.0CONST
gaussian amplitude=9.0 redshift=0.5tval dispersion=1000.0 wave=H_I_1215
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1238
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1242
absorption
voigt ion=1H_I 20.5 redshift=0.47567RA 4.0DA 1.0E4TA
model end
```

In this particular instance, a DLA with an H I column density of $10^{20.5}$ atoms cm^{-2} at redshift 0.47567, with a turbulent Doppler parameter of 4.0 km s^{-1} and kinetic temperate of 10^4 K will be used. The only free parameter is the column density (which I haven't explicitly given the parameter name). I have only explicitly given the parameter name for the redshift (although this wasn't necessary).

In principle, I can now continue to define an emission feature on top of this absorption feature (for example, Lyα emission in the DLA's trough), and absorption on top of this emission+absorption+emission (for example, from another nearby DLA), and so forth. This can be done as follows:

```
model read
emission
constant 1.0CONST
gaussian amplitude=9.0 redshift=0.5tval dispersion=1000.0 wave=H_I_1215
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1238
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1242
absorption
voigt ion=1H_I 20.5 redshift=0.47567RA 4.0DA 1.0E4TA
emission
gaussian 0.1 redshift=0.47567RA dispersion=10.0 wave=H_I_1215
absorption
```

```
...
model end
```

You might think that this functionality (of being able to indefinitely specify the emission then absorption etc.) has limited application (and I agree, if you use it for the above application!). However, a simple (common) example where this functionality might prove to be useful is if you have several data with different `specid`. You might want to use the first set of emission+absorption commands to define the model for `specid=1`, the second set of emission+absorption commands to define the model for `specid=2`, and so forth. You could equally well define the entire model for all `specid` in a single emission+absorption command (although it would be less well-organised). Usually, I would recommend sticking to a single set of emission+absorption commands – it's more guaranteed to work as you expect!

Specifying the zero-level — The zero-level can be fitted for in the same way that the emission and absorption models are fitted. Simply issue the command `zerolevel` on a new line, and follow the same procedures above. For the current example, we can fit a constant to the zerolevel as follows:

```
model read
emission
constant 1.0CONST
gaussian amplitude=9.0 redshift=0.5tval dispersion=1000.0 wave=H_I_1215
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1238
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1242
absorption
voigt ion=1H_I 20.5 redshift=0.47567RA 4.0DA 1.0E4TA
emission
gaussian 0.1 redshift=0.47567RA dispersion=10.0 wave=H_I_1215
absorption
...
zerolevel
constant 0.01
model end
```

change the default model specifications — You can easily adjust the limiting values of the models you wish to use for a given fit. To do this, you can use a series of four parameter arguments. To fix a given parameter from a model, the first argument is `fix`. To limit a given parameter from a model, the first argument is `lim`. For the second argument, you need to provide the name of the model function. ALIS comes built-in with a series of useful functions (for a full list see Section **??**, and to write your own, see Section **??**). For the current example we are following, I will use the `gaussian` function. This name is used as the second argument. The `gaussian` function (as defined by ALIS) takes three parameters which are given the (fairly obvious) name identifiers, `amplitude`, `redshift`, and `dispersion`. Using the above example, let's assume that we had good reason to expect the redshift of the emission lines to be exactly (or very close to) 0.5 and we do not want this to be a free parameter anymore. If you want this parameter to be fixed (without explicitly changing 'tval' to uppercase letters), you can issue the four argument command `fix gaussian redshift True` just after the `model read` command. Similarly, if you wanted to allow the parameter `value` of the function `constant` to vary (without explicitly changing 'CONST' to lowercase), you would issue `fix constant value False`. Other than 'True' or 'False', you can also use 'None' (which will keep the parameter as is), or you can specify a floating point number

that should be used in place of the current value.

For this example, I'm going to assume that the instrumental resolution profile was defined by the user as the function `vfwhm` in the `data read` section. This function takes one parameter, called `value`. To fix (and change) this value to 4.3, you could issue the command `fix vfwhm value 4.3`, without having to explicitly change the `vfwhm` parameter arguments specified in the `data read` section.

As a side note, although the function used as the instrumental profile (see Section 3.3.2 for more details) should not be defined in the `model read` section, you can still fix or limit it's value in this section (in fact, if you want to *change* whether the model parameters from the instrumental resolution function are fixed or limited, you can only do it here — after the `model read` command). In this case, I will assume

Finally, if you want to limit the amplitude of the Gaussian emission lines to be between 0.0 and 10.0 you would issue the command `lim gaussian amplitude [0.0,10.0]` with no spaces for the final argument. If you instead feel that the emission should *only* be limited from below, such that the minimum value is 0.0 and there is no maximum value, you would instead issue the command `lim gaussian amplitude [0.0,None]`, where 'None' indicates that you do not want to specify a limiting value. As such, the model that we have defined up until this point is the following:

```
model read
fix gaussian redshift True
fix constant value False
lim gaussian amplitude [0.0,None]
emission
constant 1.0CONST
gaussian amplitude=9.0 redshift=0.5tval dispersion=1000.0 wave=H_I_1215
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1238
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1242
absorption
voigt ion=1H_I 20.5 redshift=0.47567RA 4.0DA 1.0E4TA
model end
```

Note that all models of a given type after the `fix` *and* `lim` *declaration will inherit the* `fix` *and* `lim` *commands you issue.* For example, if we now decide that we want to fix the `dispersion` parameter for the model function called `gaussian`, but we only want to fix the value for the first Gaussian (i.e. with `dispersion=1000.0`), without fixing the dispersion for the other two Gaussians (i.e. with `dispersion=500.0`), we can place the fix command at the appropriate location in the `model read` section, to free the `dispersion` parameter for all subsequent definitions of the `gaussian` model.

```
model read
fix gaussian redshift True
fix gaussian dispersion True
fix constant value False
lim gaussian amplitude [0.0,None]
emission
constant 1.0CONST
gaussian amplitude=9.0 redshift=0.5tval dispersion=1000.0 wave=H_I_1215
```

```
fix gaussian dispersion False
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1238
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1242
absorption
voigt ion=1H_I 20.5 redshift=0.47567RA 4.0DA 1.0E4TA
model end
```

If you instead want to place a limit on just a *single* parameter, you can do so by giving that parameter an ID tag ('`jc`' in the example below), and specifying the limit with the command `lim param jc [20.0,21.0]` which is of the same form as described above. Similarly, you can fix a single parameter with the command `fix param tval True` or free a single parameter with `fix param DA False` or fix a parameter to a given value with `fix param tval 0.51`.

```
model read
fix gaussian redshift True
fix param tval True
fix gaussian dispersion True
fix constant value False
lim param jc [20.0,21.0]
lim gaussian amplitude [0.0,None]
emission
constant 1.0CONST
gaussian amplitude=9.0 redshift=0.5tval dispersion=1000.0 wave=H_I_1215
fix gaussian dispersion False
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1238
gaussian amplitude=6.0 redshift=0.5tval dispersion=500.0 wave=N_V_1242
absorption
voigt ion=1H_I 20.5jc redshift=0.47567RA 4.0DA 1.0E4TA
model end
```

NOTE: Be careful when limiting or fixing parameters with the same ID tag. If you want to place the same limit on *two* parameters that are supposed to have different values, you will need to specify this separately, as follows

```
model read
lim param jc [20.0,21.0]
lim param jd [20.0,21.0]
⋮
voigt ion=1H_I 20.7jc redshift=0.47567RA 4.0DA 1.0E4TA
voigt ion=1H_I 20.2jd redshift=0.52324RB 3.0DB 1.0E4TA
⋮
model end
```

3.3.4 How to specify your links

Links are useful if you know that there is some well-defined relation between two of your model parameters. For example if you have a good physical reason to believe that the ratio between two parameters should be fixed, or you want the sum of several parameters to be a constant value. Linking works very similar to tying parameters, so you should attach lowercase letters to the right of the parameters in your model that you want to link.

As a first example, consider the two emission lines [O III] λ5007 Å and [O III] λ4959 Å. In some environments, it is reasonable to assume that their integrated flux ratio is locked 3:1. To enforce this criteria, we introduce the variable '`va`' after specifying the value of the integrated flux for [O III] λ5007 Å, and we also place a *different* variable, in this case '`vb`', after specifying the value of the integrated flux for [O III] λ4959 Å. Running such a model without including the `link` section of the model, would cause both integrated fluxes to be free parameters. To force their ratio to be 3:1, one must include the link section, explicitly telling ALIS that the parameter `vb` as a function of `va` is equal to `va` divide by 3 (i.e. `vb(va) = va / 3.0`). The model will therefore look something like the following:

```
model read
emission
line_emission ion=16O_III_5007 IntFlux=9.0va 0.03ra 7.0ba
line_emission ion=16O_III_4959 IntFlux=3.0vb 0.03ra 7.0ba
model end

link read
vb(va) = va / 3.0
link end
```

As another example, suppose you wanted to make sure the total integrated flux from 3 emission lines is equal to 10, you would need to write something like the model below. In this case, both `vb` and `vc` are allowed to vary, but the variable `va` will be adjusted such that `va=10-vb-vc`.

```
model read
emission
line_emission ion=1H_I_6563 IntFlux=5.0va 0.03ra 7.0ba
line_emission ion=1H_I_4861 IntFlux=3.0vb 0.03ra 7.0ba
line_emission ion=1H_I_4341 IntFlux=2.0vc 0.03ra 7.0ba
model end

link read
va(vb,vc) = 10.0 - vb - vc
link end
```

ALIS is fairly able to interpret the expression to the right of the '`=`' sign, provided that it contains only numbers (with and without a floating decimal point), and any combination of the following symbols:

'+', '-', '*', '/', '**', '(', and ')', where the standard order of operations applies (note that '**' means 'to the power of'), and any number of variable strings that are defined anywhere in the model section. Finally, note that any variable listed on the left hand side of any linking equation will not be assigned an error. This is because it is related to other parameters, and by imposing some restriction on this parameter's value, you are removing 1 degree of freedom from the minimization process. If you specify a link twice for a given parameter (e.g. for the model above you specified `va(vb,vc) = 10.0-vb-vc` on one line and `va(vc) = 5.0-vc` on another line), the first equation read by ALIS will be used, and the remaining cases will be ignored.

In the context of linking, the function '`variable`' can be very useful if you want to calculate the value and associated error on a combination of model parameters. The '`variable`' function takes just one parameter, and this should be tied to a value in the links section of the model specification. Using the links section can also be useful if you want to start a parameter value with a random value (drawn from a user-specified distribution). See Section 8 for more details on this functionality.

Note: you cannot tie '`resolution`' variables within ALIS.

4 The ALIS plotting environment

If you're unfamiliar with the `matplotlib` plotting environment, I recommend that you have a quick look at the following url, in order to take full advantage of the plotting package:
`http://matplotlib.org/users/navigation_toolbar.html`

4.1 Setting plot parameters

FILL THIS IN LATER

4.2 Description of the interface

FILL THIS IN LATER

5 Example `.mod` files and fitting

The detail for how to prepare a `.mod` file is described in Section 3. In this Section, a few standard examples will be introduced, and fitted with ALIS. These examples (both the data and the `.mod` files) are provided in the 'examples' directory.

5.1 Your first fit – The example from Section 3

5.2 A metal-poor DLA

5.3 Fine-structure constant variation

5.4 Isotope ratios

5.5 Quasar spectrum

6 Generate Fake Data

One of the functions that I find very useful with ALIS is the ability to straightforwardly generate perfect data (i.e. without noise) and fake data (with noise). To do so, you specify the desired model as described in Section 3.3.3.

There are essentially two more steps in order to generate perfect/fake data. The first step involves telling ALIS that you want to generate data from the model you've specified. You can do this with the three parameter arguments that are specified at the beginning of your `.mod` file. For a description of how to do this, refer to their corresponding entries in Table 1.

The final step is to specify some details in the `data read` section. Again, this works in the same way as described in Section 3.3.2. If you provide a filename that exists, ALIS will attempt to read in the wavelength array and the error spectrum. The model will be generated and noise will be applied to the model based on the error spectrum that is read in from the specified file. The output file is given the suffix '_model' before the extension type.

If you instead provide a filename that doesn't exist, ALIS will generate a wavelength array based on the wavelength range provided by `fitrange`, in steps of `generate+pixelsize` and in units of `run+bintype`. If noise is to be applied to the model, ALIS will derive the appropriate error spectrum based on `generate+peaksnr` and `generate+skyfrac`. Finally, ALIS will output the generated data to the specified filename.

NOTE: If you are generating perfect/fake data, and you are convolving your data with an instrumental resolution profile, the model may be subject to edge effects. If you are loading a file that already exists and you want to avoid these edge effects, make sure `fitrange` is well within the wavelength range of the generated data, and you probably want to specify `loadall=True`. If you ask ALIS to generate data from scratch (i.e. the filename you specify does not exist), make sure that `fitrange` covers a larger wavelength range than the range you want, so that you can later reject the edge effects, if that's what you want to do!

7 Monte Carlo Simulations

It is also straightforward to perform Monte Carlo simulations with ALIS. You can tell ALIS to perform Monte Carlo simulations by using the appropriate three parameter arguments in your `.mod` file (see Section 3.3 and Table 1).

7.1 random

If you want to perform 1000 Monte Carlo simulations to test the random error of your data, simply use the three parameter argument: `sim random 1000` at the beginning of your `.mod` file, ALIS will do the

rest and save the output appropriately. **NOTE:** The output will contain 1001 lines in this case; the first line of the output will always be the input model parameters (which you can discard if you don't want them).

In the above example, the Monte Carlo runs will be assigned the ID number 0–999. If you want to start from a different ID number, say 10, you would also need to specify `sim startid 10` (and the simulations will now run over the ID numbers 10–1010).

For each of the simulations, the best-fitting model parameter file is output into a directory specified by `sim+dirname`. If you want to output the results of your simulations to a directory called "my_simulations", you would issue the following three parameter argument at the beginning of your `.mod` file: `sim dirname my_simulations`.

If you decide to find the best-fitting model, and after this you decide that some Monte Carlo simulations are needed, you do not need to refit the real data, provided that you have output the best-fitting model file and covariance matrix. To begin from the best-fitting model, issue the following three parameter argument `sim beginfrom myfit.mod.out`. In this example, from the command line you would need to issue the command `alis myfit.mod`. In the same directory that you run this command, you must have the best-fitting model file (myfit.mod.out), and the corresponding covariance matrix (myfit.mod.out.covar).

Since fake data is being generated during the Monte Carlo process, your data may suffer from edge effects if you have convolved your data with the instrumental resolution. A warning message will be issued if the generated data suffer from edge effects. You can change the threshold of this warning using the three parameter argument `sim edgecut 4.0`, where 4.0 can be any floating point number. The number you specify represents the number of standard deviations for the instrumental profile that the fitrange values need to be from the wavelength edges. If you get these warnings, do not trust the random simulations. To avoid such warnings, you can either reduce the `fitrange`, or provide more wavelength coverage for the input file. Similarly, you can reduce this threshold (but don't trust your results, unless you can convince yourself that the edge effects are minimal enough).

7.2 systematics

If you have 'massaged' your data before running it through ALIS (for example, you normalise your continuum), and you want to know how this might systematically affect your results, you can test this with ALIS by setting the three parameter argument `sim systematics True`. In order to run systematics, you must also run random simulations. The idea here is that you want to perform the same 'massaging' on the data generated in the random simulations, as you did to the real data. You can either use one of the built-in systematics modules (in this case, you must match your 'massaging' to the routine that you select), or you can write your own systematics module (described below). At present, ALIS only comes built-in with one systematics modile, which accounts for continuum normalisation with a polynomial. In order to use this module you will need to issue the three parameter argument `sim systmodule default` or you can also issue `sim systmodule continuumpoly` for the same routine. In this case, you must specify an additional column of data to be read in called `systematics`. This column of data must contain a -1 for every pixel that is not used in the polynomial fit, and n (where n is unique, and is the order of the polynomial — e.g. 1 would be linear) for the pixels that were used to normalise the continuum. Simlar warnings will be supplied if edge effects (see above) are important (for example, if you use pixels at the extremeties of the data to estimate the continuum).

If you would not like to apply systematics to a given datafile that is read in from the `data read` section, you can issue `systmodule=None` on the appropriate line where you are reading in that datafile.

It is very likely that no other built-in systematics functions will be implemented (unless you can

come up with a fairly common use of this functionality that I can implement!). I've therefore created the flexibility for you to write your own systematics module to deal with the systematics that your data may suffer from. Here are a few details for how to write your own systematics module. You need to write your module in PYTHON in order for it to work.

First, create a PYTHON script in the same directory as your `.mod` file, for example `mysystmod.py`. In this file, you can do what you like (!) but you must create a function definition called `loader` which accepts several arguments. The first argument is an ID string, which is discussed below. The second, third, fourth and fifth arguments are respectively the wavelength array, flux spectrum, error spectrum and the column of data read in from file that has information on the systematics involved (see above). The sixth argument contains a two element array with the minimum (zeroth element) and maximum (first element) wavelengths that are free from edge effects (see above for more details). The final argument is the name of the data file (just incase you need it for reference or whatever).

To tell ALIS that you want to use your module for a given data file, you need to specify the name of your module on the corresponding line in the `date read` section of your `.mod` file. You can do this by giving the command `systmodule=mysystmod.py,idstring`, where '`,idstring`' is optional, and can be any text string that you desire. This text string will be passed to the first argument of your systematics module. If you don't provide an ID string on this line, your systematics module will be passed a black string with zero characters (i.e. '').

8 Built-in functions

There are several built-in functions that come as standard with ALIS. If this suite of functions is not sufficient, then you can (easily) write your own and read it into ALIS (for more details on how to do this, refer to Section 9). Please, if you write a function that you believe will be useful to the wider community, let me know and I'll include it in the next release! It is worth noting that all functions in the source code directory have the prefix `alfunc_<function name>.py`.

It is worth reiterating at this point that you do not have to specify every parameter of the model. However, if you decide to not set a parameter, ALIS will use the default value, and will fix this value.

In the following subsections, each built-in function is described in alphabetical order in more detail. Here is a summary of the current model functions:

- Afwhm
- base
- brokenpowerlaw
- chebyshev
- constant
- gaussian
- legendre
- linear
- polynomial
- powerlaw
- random
- tophat
- vfwhm
- voigt
- vsigma

8.1 Afwhm

The `Afwhm` function is a model that describes the instrumental broadening as a Normal Distribution function with standard deviation (in Angstroms) given by:

$$\sigma(\text{Å}) = 2\sqrt{2\ \ln\ 2}\, p_0 \tag{1}$$

where the description of this single parameter is:

0. **value** — The full-width at half-maximum in Angstroms (Default = 0.0).

The corresponding keywords are:

- **blind** — If you would like to blind yourself from this model, set `blind=True`.

This function is specifically designed for convolving the model spectrum with the instrumental broadening function. The convolution is performed with a fast fourier transform.

8.2 base

You should not change anything in this file. Moreover, this function should not be used for any model definition. It contains the guts for defining a model, and as the name suggests, it is the base from which all other models are defined. The benefit of this file is that you can easily write your own functions by importing some definitions that have already been written for you!

8.3 brokenpowerlaw

The `brokenpowerlaw` model is given by the following equation:

$$\text{model} = \frac{p_0}{(\lambda/p_3)^{p_1}(1.0 + (\lambda/p_3)^{p_4})^{(p_2-p_1)/p_4}} \tag{2}$$

I'm not sure if (or where) this style of function has been defined before, but the idea is that this model will have a power-law form at both blue and red wavelengths, where the power-law index is different for the blue and red. There are five parameters which are given the names:

0. **coefficient** — A scaling coefficient for the model (Default = 0.0).
1. **blueindex** — The power-law index for blue wavelengths (Default = 0.0).
2. **redindex** — The power-law index for red wavelengths (Default = 0.0).
3. **location** — The location of the break (Default = 5000.0).
4. **strength** — The sharpness of the break. Higher values provide a sharper break, whereas lower values provide a more gradual transition (Default = 1.0).

The corresponding keywords are:

- **specid** — The set of specid's that this model should be applied to.
- **blind** — If you would like to blind yourself from this model, set `blind=True`.

8.4 chebyshev

The `chebyshev` model is a Chebyshev polynomial of the first kind:

$$\text{model} = p_0 + p_1\lambda + p_2(2\lambda^2 - 1) + \ldots \tag{3}$$

You can specify as many parameters as you like (well, up to 10 — you can only fit a Chebyshev polynomial of order 9 and below). The 'downside' of having an arbitrary number of parameters is that you cannot specify the `parid`, `fixpar`, `limited`, or `limits` parameters. Although these commands will work, the limits you place will be applied to all coefficients, which may not be so bad, if you use the `scale` keyword to scale your coefficients to be of the same magnitude. If you want the ability to limit your polynomial coefficients, another way around this would be to write your own Chebyshev function for the order that interests you. Every coefficient is given a non-unique `parid`:

0. **coefficient** — The coefficient for the n^{th} Chebyshev polynomial (Default = 0.0).

The corresponding keywords are:

- **specid** — The set of specid's that this model should be applied to.
- **blind** — If you would like to blind yourself from this model, set `blind=True`.
- **scale** — An array that scales each of the coefficients If you use this keyword, you must provide an array of comma-separated numbers, with the same number of elements as coefficients, of the form: `scale=[1.0,0.1,0.001,1.0E-6]`. This example would yield a third order Chebyshev polynomial of the first kind.

8.5 constant

The `constant` model is just that — a constant. The model equation is given by:

$$\text{model} = p_0 \tag{4}$$

where the description of this parameter is:

0. **value** — The value of the constant (Default = 1.0).

The corresponding keywords are:

- **specid** — The set of specid's that this model should be applied to.
- **blind** — If you would like to blind yourself from this model, set `blind=True`.

8.6 gaussian

The `gaussian` model is a Gaussian function of the form:

$$\text{model} = p_0 \exp\left(-\frac{(\lambda - \text{wave} \times (1.0 + p_1))^2}{2.0p_2^2}\right) \tag{5}$$

where the parameters are given by:

0. **amplitude** — The amplitude of the Gaussian feature (Default = 0.0).

1. **redshift** — The emission redshift (Default = 0.0).

2. **dispersion** — One standard deviation (in km s^{-1}) (Default = 100.0).

The corresponding keywords are:

- **specid** — The set of specid's that this model should be applied to.
- **blind** — If you would like to blind yourself from this model, set `blind=True`.
- **wave** — Specify the ion+wavelength (or just a numerical wavelength) of the transition. This is a required parameter since it is used in the model calculation.

8.7 legendre

The `legendre` model is a Legendre polynomial of the first kind:

$$\text{model} = p_0 + p_1\lambda + p_2\frac{3\lambda^2 - 1}{2} + \ldots \tag{6}$$

You can specify as many parameters as you like (well, up to 10 — you can only fit a Legendre polynomial of order 10 and below). The 'downside' of having an arbitrary number of parameters is that you cannot specify the `parid`, `fixpar`, `limited`, or `limits` parameters. Although these commands will work, the limits you place will be applied to all coefficients, which may not be so bad, if you use the `scale` keyword to scale your coefficients to be of the same magnitude. If you want the ability to limit your polynomial coefficients, another way around this would be to write your own Legendre function for the order that interests you. Every coefficient is given a non-unique `parid`:

0. **coefficient** — The coefficient for the n^{th} Legendre polynomial (Default = 0.0).

The corresponding keywords are:

- **specid** — The set of specid's that this model should be applied to.
- **blind** — If you would like to blind yourself from this model, set `blind=True`.
- **scale** — An array that scales each of the coefficients If you use this keyword, you must provide an array of comma-separated numbers, with the same number of elements as coefficients, of the form: `scale=[1.0,0.1,1.0E-3]`. This example would yield a second order Legendre polynomial of the first kind.

8.8 linear

The `linear` function is a straight line. The model equation is given by:

$$\text{model} = p_0 + p_1\lambda \tag{7}$$

where the description of these parameters are:

0. **intercept** — The value of the function when $\lambda = 0$ (Default = 1.0).

1. **gradient** — The slope of the line (Default = 0.0).

The corresponding keywords are:

- **specid** — The set of specid's that this model should be applied to.
- **blind** — If you would like to blind yourself from this model, set `blind=True`.

8.9 polynomial

The `polynomial` model is a standard polynomial of the form:

$$\text{model} = p_0 + p_1\lambda + p_2\lambda^2 + \ldots \tag{8}$$

You can specify as many parameters as you like. The 'downside' of having an arbitrary number of parameters is that you cannot specify the `parid`, `fixpar`, `limited`, or `limits` parameters. Although these commands will work, the limits you place will be applied to all coefficients, which may not be so bad, if you use the `scale` keyword to scale your coefficients to be of the same magnitude. If you want the ability to limit your polynomial coefficients, another way around this would be to write your own Polynomial function for the order that interests you. Every coefficient is given a non-unique `parid`:

0. **coefficient** — The coefficient for the n^{th} term of the polynomial (Default = 0.0).

The corresponding keywords are:

- **specid** — The set of specid's that this model should be applied to.
- **blind** — If you would like to blind yourself from this model, set `blind=True`.
- **scale** — An array that scales each of the coefficients If you use this keyword, you must provide an array of comma-separated numbers, with the same number of elements as coefficients, of the form: `scale=[10.0,0.1,1.0E-3]`. This example would yield a second order polynomial (i.e. a quadratic).

8.10 powerlaw

The `powerlaw` function is a single powerlaw. The model equation is given by:

$$\text{model} = p_0\lambda^{p_1} \tag{9}$$

where the description of these parameters are:

0. **coefficient** — A multiplicative (or scaling) constant (Default = 0.0).
1. **index** — The index of the powerlaw (Default = 0.0).

The corresponding keywords are:

- **specid** — The set of specid's that this model should be applied to.
- **blind** — If you would like to blind yourself from this model, set `blind=True`.

8.11 random

The `random` function forces the starting parameter value to be a random number drawn from a distribution specified by the '`command`' keyword argument. You will need to do some linking as well, to tell the `random` function which variable to apply the randomly generated value to. For example, if you know that the logarithm of the column density in your voigt profile is somewhere between 13.0 and 14.0, you might specify the following model:

```
model read
emission
constant 1.0CONST
absorption
voigt ion=1Ly_a 13.2lra redshift=0.47567 4.0 1.0E4TA
random 0.0lrb command=uniform(13.0,14.0)
model end

link read
lra(lrb) = lrb
link end
```

The `uniform` distribution is just one of the allowed values. You could also choose, for example, a normal distribution with centroid 13.5 and width 0.3. In this case, you would have `command=normal(13.5,0.3)`. A list of all the permitted distributions are specified on the following website:

`http://docs.scipy.org/doc/numpy/reference/routines.random.html`

NOTE: Be warned, placing a prior on a starting parameter's value should be considered carefully; you should almost always use a uniform prior with two extreme boundaries (that are considered as the limits of all possible values), unless you know what you're doing.

8.12 tophat

The `tophat` function is a rectangular function. The model equation is given by:

$$\text{model} = \begin{cases} 0, & \text{if } \lambda < p_1 - p_2/2. \\ p_0, & \text{if } p_1 - p_2/2 \leq \lambda < p_1 + p_2/2. \\ 0, & \text{if } \lambda \geq p_1 + p_2/2. \end{cases} \tag{10}$$

where the description of these parameters are:

0. **height** — The amplitude of the tophat function in the specified interval (Default = 1.0).
1. **centroid** — The centroid of the tophat function (Default = 0.0).
2. **width** — The width of the tophat function (Default = 1.0).

The corresponding keywords are:

- **specid** — The set of specid's that this model should be applied to.

- **blind** — If you would like to blind yourself from this model, set `blind=True`.
- **hstep** — The step size to be used when calculating the partial derivative of the model with respect to the `centroid` (should be of order `width`).
- **wstep** — The step size to be used when calculating the partial derivative of the model with respect to the `width` (should be of order `width`).

8.13 variable

The `variable` function creates a dummy variable that can be used to connect two parameters through the `link` command. Suppose you know that a parameter is exactly a constant times another parameter (but you don't know what either parameter is, nor the cosntant scaling). For example, suppose you want to fit a two component Voigt model, where silicon and sulphur are tied to have the same relative abundance, but you don't know the column density of Si II or S II or the relative abundance. In this case, you could specify the model

```
model read
emission
constant 1.0CONST
absorption
voigt ion=28Si_II 13.2colsia redshift=0.475686ra 4.0da 1.0E4TA
voigt ion=28Si_II 13.0colsib redshift=0.475705rb 3.0db 1.0E4TB
voigt ion=32S_II 12.2colsa redshift=0.475686ra 4.0da 1.0E4TA
voigt ion=32S_II 12.0colsb redshift=0.475705rb 3.0db 1.0E4TB
variable -1.0lnksis
model end

link read
colsa(colsia,lnksis) = colsia+lnksis
colsb(colsib,lnksis) = colsib+lnksis
link end
```

which forces the S II column densities to be set by the Si II column densities + some fitted constant value called `lnksis`. Another example, is that you can use a variable to find the error on the summed column density for a given ion. Here is an example of that for Si:

```
model read
emission
constant 1.0CONST
absorption
voigt ion=28Si_II 13.2colsia redshift=0.475686ra 4.0da 1.0E4TA
voigt ion=28Si_II 13.0colsib redshift=0.475705rb 3.0db 1.0E4TB
variable 13.5lnksis
model end
```

```
link read
colsib(lnksis,colsia) = numpy.log10(10.0**lnksis - 10.0**colsia)
link end
```

In this last case, `lnksis` is the total (summed) column density. **NOTE:** It is generally a good idea to specify `colsib` as the lowest column density component of the absorption.

8.14 vfwhm

The `vfwhm` function is a model that describes the instrumental broadening as a Normal Distribution function with standard deviation (in velocity) given by:

$$\sigma(v) = 2\sqrt{2\,\ln\,2}\,p_0 \tag{11}$$

where the description of this single parameter is:

0. **value** — The velocity full-width at half-maximum (Default = 0.0).

The corresponding keywords are:

- **blind** — If you would like to blind yourself from this model, set `blind=True`.

This function is specifically designed for convolving the model spectrum with the instrumental broadening function. The convolution is performed with a fast fourier transform.

8.15 voigt

The `voigt` function describes the absorption line profile for a group of atoms that obey a Maxwell-Boltzmann distribution. The functional form is the convolution of the intrinsic line profile (i.e. a Lorentzian) with a Maxwell-Boltzmann distribution. This profile involves a very numerically expensive calculation. To avoid this time-consuming operation, ALIS uses a tabulated form of the Voigt profile which expands the following functional form into a Taylor series. The implementation that is used in ALIS is the same as that used in `vpfit`, prepared by Julian King (which is an extension from the work by others), and is correct to within 1 part in 10^5. For reference, the functional form of the Voigt profile is as follows:

$$\text{model} = I(\lambda)_0 e^{-p_0\,\sigma_\lambda} \tag{12}$$

where $I(\lambda)_0$ is the continuum intensity, and σ_λ is given by:

$$\sigma_\lambda = a_0\,H(a,x) \tag{13}$$

For a given transition, a_0 contains the atomic parameters and $H(a,x)$ is known as the Voigt integral. These are given by:

$$H(a,x) = \frac{a}{\pi}\int_{-\infty}^{\infty}\frac{\exp(-y^2)\,\mathrm{d}y}{(x-y)^2+a^2} \tag{14}$$

$$a_0 = \frac{\sqrt{\pi}\,e^2}{m_e c^2}\frac{f}{\Delta\nu_D} \tag{15}$$

where the oscillator strength, f, is read in from the `atomic.xml` file. The damping parameter of the intrinsic line shape, a, and the Doppler frequency, $\Delta\nu_D$, have the form

$$a = \frac{\Gamma}{4\pi\,\Delta\nu_D} \tag{16}$$

$$\Delta\nu_D = \frac{b}{\lambda_0} = \frac{1}{\lambda_0}\sqrt{b_{th}^2 + p_2^2}, \tag{17}$$

where Γ and λ_0 are respectively the transition rate and the rest wavelength of the transition (both are read in from `atomic.xml`). To convert the rest wavelength into the observed frame, the observed wavelength if $\lambda_{\rm obs} = \lambda_0(1+p_1)$. b_{th} is the thermal Doppler parameter (describing the thermal broadening of the line profile) which is given by

$$b_{th} = \sqrt{\frac{2\,k\,p_3}{m_{atom}}} \tag{18}$$

Finally, the dimensionless parameter x in the Voigt integral describes the frequency offset from the line centre, in units of the Doppler frequency,

$$x = \frac{\nu - \nu_0}{\Delta\nu_D}. \tag{19}$$

and y is the convolution parameter. If you are estimating a possible variation in the fine-structure constant, the rest wavenumber, $w_0 \equiv 1/\lambda_0$ is shifted by:

$$w_z = w_0 + q([1 + p_4]^2 - 1) \tag{20}$$

where q is as a number that describes how sensitive a given atomic transition is to changes in the fine-structure constant, and is read in from the `atomic.xml` file (for a small handful of transitions). **DESCRIBE VARIATION IN PROTON-TO-ELECTRON MASS RATIO HERE** In summary, the parameters are given by:

0. **ColDens** — The column density (in cm^{-2}), where by default the logarithmic value of the column density is calculated, but see keywords (Default = 8.1).
1. **redshift** — The absorption redshift (Default = 0.0).
2. **bturb** — The turbulent Doppler parameter (in km s^{-1}) (Default = 7.0).
3. **temperature** — The kinetic temperature of the gas (in K) (Default = 100.0).
4. **DELTAa/a** — The relative variation of the fine-structure constant (Default = 0.0).
5. **DELTAmu/mu** — The relative variation of the proton-to-electron mass ratio (Default = 0.0).

The corresponding keywords are:

- **specid** — The set of specid's that this model should be applied to.
- **blind** — If you would like to blind yourself from this model, set `blind=True`.

- **ion** — Specify the element+ionization stage (separated by an underscore). The first letter of the element should be a uppercase letter. This is a required parameter since it is used in the model calculation.

- **logn** — A True or False argument will respectively calculate the log or linear value of the column density for this one model. For weak lines with large errors, you should always use linear (i.e. `logn=False`), it will give you a more reliable estimate of your errors. Any column density that is well-measured (with a low error), can use either, but in this case, I recommend using the logarithmic value of the column density (i.e. `logn=True`).

8.16 vsigma

The `vsigma` function is a model that describes the instrumental broadening as a Normal Distribution function with standard deviation given by the only parameter of this model, p_0, which is called `value`. The keywords for this function are:

- **blind** — If you would like to blind yourself from this model, set `blind=True`.

This function is specifically designed for convolving the model spectrum with the instrumental broadening function. The convolution is performed with a fast fourier transform.

9 Writing your own function

XXX — Ryan, describe here the separate module that should be used to load user functions.

9.1 The Base function

When writing a function from the bare bones, you need to understand how ALIS is written. If you don't care for that (and I don't blame you!), you can (and should) use the `Base` functionality to write your own functions.

9.2 Writing your own arbitrary function

Before getting started, if you wish to write a polynomial function, or a function with an arbitrary number of parameters, you should read Section 9.3.

The best way to learn how to use the `Base` function is to look at the `gaussian` model function that comes built-in with ALIS. In summary, by specifying `alfunc_base.Base` as the first and only argument in the `python class` environment (see line 6 of the `gaussian` model function code), you're function will use all of the predefined routines that are outlined in `alfunc_base.Base`, meaning that all you need to do is write only a few lines of `python` code, as discussed below.

The first thing you need to do is change the setup for the model. This includes changing the text string for what your model function will be called, how many parameters you wish the model to have, the names and default values for any keywords that are specified, the names and default values of the model parameters (including their default limits and if they should be fixed by default), and the format for how your parameters or keywords should be printed to screen. You can also specify which keywords should be printed before the parameters. For an example of the values you could enter, see lines 14–25

of the `gaussian` model function. **NOTE:** If you are writing your own model function, you cannot use the keyword `input` as one of your keywords. This is reserved by ALIS and cannot be changed.

You will also need to change the models functional form in the `def model(par)` section (see lines 41-45 of the `gaussian` model function code). Use the variable `x` as the wavelength array, and the array `par` to define the parameters (`par[0]` is the first model parameter, `par[1]` is the second model parameter and so forth).

Apart from specifying the functional form of your model, you also need to tell ALIS how the input parameters (i.e. the parameters specified in a user's `.mod` file) relate to the model parameters you just specified above in your model function. You should do this for each of the parameters in your model (see line 63–65 of the `gaussian` model function for an example). If you want to combine keywords with model parameters, or you want to combine several parameters into a single parameter, you should use the `parb` variable which is specified in `def set_vars` (see line 78 of the `gaussian` model function for an example of passing both a keyword [`wave`] and a parameter [the redshift]).

You may also need to change the `nexbin` parameters (see the example provided on lines 86–90 of the `gaussian` model function) if you want to make sure any subpixellation that's applied (where `nexbin[0]`≡`run+bintype` and `nexbin[1]`≡`run+nsubpix`) samples your model function accurately. For the `gaussian` model function that we are using as our example here, two arguments are returned: The first argument is `params`, and this should not be changed. The second argument is the number of subpixels to use for this model. The definition of `run+nsubpix` is the number of subpixels per standard deviation, so `nexbin[1]/params[2]`, with some corrections to rounding, gives the desired return value. If you don't wish to implement this for your function, you can simply return the integer value 1 after `params` (i.e. `return params, 1`).

9.3 Writing your own polynomial function

This section will help you to write your own arbitrary polynomial function of your interest. Since polynomials in ALIS have the added functionality of specifying an arbitrary number of coefficients, you can very easily use the `Polynomial` base, instead of the `Base` base. All you need to do then is specify the form of the function that should be multiplied by each of the coefficients in a definition called `call_CPU`.

If you are planning to write your own polynomial function, I recommend you copy the design of the `chebyshev` or `legendre` model functions that come built-in with ALIS, rather than writing your own polynomial function from scratch.

10 Troubleshooting

10.1 List of Error Messages

To be completed

10.2 List of Warning Messages

To be completed

10.3 Frequently Asked Questions

Based on the questions I've received so far (asked by other users and myself!),I've compiled a list of 'frequently asked questions'. If I've ever encountered a problem, it is usually listed here to remind me

how to solve it! Please look through here to find if you're problem already has a solution.

10.3.1 What should the tolerances be?

Whatever you like! The general rule, is that an *absolute* change in the χ^2 value of 0.001 means that the solution is converged, and any change in the parameter values are not physical. Given that ALIS finds a local minimum, rather than a global minimum, I would suggest you aim for the above level in χ^2. Therefore, if you want nothing more, set `xtol` and `gtol` to be equal to `0.0`, and set `ftol` to be something small — of order (or slightly less than) `0.001/dof`, where `dof` is the number of degrees of freedom. If you're running a convergence check, you might want to set `ftol` to be `0.01/dof`, since the convergence check will decrease `ftol` by an order of magnitude. Another common practise is to set `xtol`, `gtol`, and `ftol` to be equal to `0.0`, and set `atol` to be 0.001 (which is the tolerance for the absolute difference between iterations)

10.3.2 My eye can see a better fit than ALIS, what's going wrong?

There's no simple, gauranteed solution to this problem, but here are a list of possibilities that you might want to try.

- Check the reason for convergence (this should be printed in the output file). If you're doing a blind analysis, the reason for convergence is printed on screen. If ALIS has only taken 1 iteration, and then returned, you can be almost certain that the solution is bad.
- Check the value of 'fstep'. ALIS works by calculating the numerical derivatives with the finite difference method. If `fstep` is close to 1.0, then the parameters are adjusted by the value of the parameter multiplied by the machine precision (i.e. hardly at all). Low values of `fstep` are preferred, since the derivatives are better, and the convergence is quicker. If it is too low, some parameters (depending on how the model is specified), aren't very sensitive to such a low change (the code thinks that $f(x)$ and $f(x+\delta x)$ are equal). Try increasing the value of `fstep` by an order of magnitude or more.
- Check the maximum number of iterations is > 1, and set the minimum number of iterations to > 1.
- Check the maximum number of iterations has not been reached. If you've reached the maximum number of iterations, the solution has probably not converged.
- If ALIS returns before the minimum number of iterations is reached, then the machine precision was reached first (unless you received an error message!). (you probably have a lot of parameters and/or a large dynamic range in these parameters). If you have a large number of parameters, try a slightly different set of starting parameters. Sometimes, if the fit is too good for some parameters and terrible for others, ALIS may think it's closer to convergence than it really is.
- If your model is discretized (i.e. a tabulated model), then the software can get stuck between two grid points. This can be overcome by interpolation, or define a model with finer grid sizes.
- ALIS does not derive the global minimum, but rather, the local minimum that is based on the user starting parameters. Try a different set of starting parameters that is a closer representation to the data.

10.3.3 ALIS is taking too long to converge

In general, I really wouldn't expect that your fit will need more than a couple of hundred iterations (for extreme examples!). More generally, the number of iterations should be less than about 100 (and certainly less than 1000!). If ALIS takes a long time to converge and you're at less than 100 iterations, you must either have a complicated model function (or your model is poorly defined – make sure you are using `numpy` to speed up array operations), or you have a large number of parameters. Try increasing the number of cpus that ALIS uses to speed things up. However, if you are really doing such a calculation, you must have a higher-than-average level of patience!

10.3.4 My model function is not working

In general, this is something that I'm not able to help you with, other than suggesting that you read carefully through Section 9, and make sure that your models are well-defined (i.e. no discontinuities), and the derivatives can be numerically calculated fairly easily (for example, absolute values of free parameters will probably have trouble).

11 Comments/Questions/Additions?

I would love to hear any comments you have of how to improve ALIS, and/or features that you would want or feel limited by. If you find a bug (there probably is one!) then let me know and I'll do my best to find a quick solution. If you have a snippet of code that will fix the bug, then please send it to me!

Did you write a useful function? Please send it to me and I'll include it in the next version of ALIS.

My current contact details are: rcooke@ucolick.org

References

Markwardt C. B., 2009, ASPC, 411, 251