SPEX Help Center Documentation

Release 3.06.01

Jelle de Plaa, Jelle Kaastra, SPEX Development Team

Dec 22, 2020

CONTENTS

CHAPTER

ONE

GETTING STARTED

1.1 How to install SPEX

1.1.1 Download

SPEX install files can be downloaded from our Zenodo page.

1.1.2 Linux

Unpack the tar file in the destination directory (for example /usr/local/):

```
tar xvfz spex-3.06.01-Linux-x86_64.tar.gz
```

The tar file will create a directory called SPEX-3.05.00-Linux in which the program will be installed.

Set the environment variable SPEX90 to the installation directory (for example /usr/local/SPEX-3.06.01-Linux):

```
export SPEX90=/usr/local/SPEX-3.06.01-Linux (bash shell)
setenv SPEX90 /usr/local/SPEX-3.06.01-Linux (C-type shell)
```

Source the script provided by the distribution:

```
source $SPEX90/spexdist.sh (bash shell)
source $SPEX90/spexdist.csh (C-type shell)
```

Remove the tar file: rm SPEX-3.06.01-Linux-x86_64.tar.gz

1.1.3 Mac OS

Administrator install

If you have administrator rights on your Mac, this option is the easiest one. Download the DMG file from Zenodo (SPEX-3.06.01-MacOS.dmg), open it in Mac OS and follow the instructions on the screen.

Non-administrator install

Unpack the tar file in the destination directory (for example /usr/local/):

```
tar xvfz spex-3.06.01-MacOS.tar.gz
```

The tar file will create a directory called SPEX-3.06.01-MacOS in which the program will be installed.

Set the environment variable SPEX90 to the installation directory (for example /usr/local/SPEX-3.06.01-MacOS):

```
export SPEX90=/usr/local/SPEX-3.06.01-MacOS (bash shell)
setenv SPEX90 /usr/local/SPEX-3.06.01-MacOS (C-type shell)
```

Source the script provided by the distribution:

```
source $SPEX90/spexdist.sh (bash shell)
source $SPEX90/spexdist.csh (C-type shell)
```

Remove the tar file: rm SPEX-3.06.01-MacOS.tar.gz

1.1.4 Windows 10

SPEX can be installed in Windows 10 using the new linux subsystem in Windows. SPEX can run in an Ubuntu Linux environment that can be installed through the Microsoft Store.

How to install Ubuntu in Windows 10

In addition to the Ubuntu terminal, you also need a graphical X server. We recommend the Vcxsrv server. Otherwise, the XMING server is an alternative. This will make sure that you can plot in SPEX.

Install SPEX in the Ubuntu environment

1. On the Ubuntu terminal, first install the OpenBlas library: apt-get install libopenblas-base
2. Download spex-3.05.00-Linux-gfortran.tar.gz from our FTP server. This file is not (yet) available on Zenodo. Unpack the tar file in the destination directory (for example /usr/local/):

   ```
   tar xvfz spex-3.05.00-Linux-gfortran.tar.gz
   ```

   The tar file will create a directory called SPEX-3.05.00-Linux in which the program will be installed.
3. Set the environment variable SPEX90 to the installation directory (for example /usr/local/SPEX-3.05.00-Linux):

   ```
   export SPEX90=/usr/local/SPEX-3.05.00-Linux (bash shell)
   setenv SPEX90 /usr/local/SPEX-3.05.00-Linux (C-type shell)
   ```
4. Source the script provided by the distribution:

   ```
   source $SPEX90/spexdist.sh (bash shell)
   source $SPEX90/spexdist.csh (C-type shell)
   ```
5. Remove the tar file: rm SPEX-3.05.00-Linux-gfortran.tar.gz
6. Set the DISPLAY variable in ~/.bashrc:

   ```
   echo "export DISPLAY=localhost:0.0" >> ~/.bashrc
   . ~/.bashrc
   ```
7. Run SPEX:

```
spex
```

If you get an error about a missing library when running SPEX, please run the following command:

```
sudo apt-get install libopenblas-base libreadline8 x11-common libx11-6 gfortran
```

The command above makes sure that all the packages that SPEX needs are installed.

1.2 How to run SPEX

SPEX is a spectral fitting program used to fit high-resolution X-ray spectra. The code contains several simple and detailed models that are able to deal with the radiative processes observed in the X-ray band. Because SPEX has a command-line interface, a first-time user should get familiar with the syntax of the commands to be able to work with it. This chapter provides some basic commands and threads to fit X-ray spectra.

1.2.1 The SPEX data format

The data files containing the spectrum of the source and the response need to be in the correct format. In the SPEX installation, we provide a program called *Trafo* (page 181) to convert OGIP standard fits files into SPEX format (see *How to convert spectra to SPEX format* (page 9) for an explanation of how to use trafo). In this chapter, we assume that you already have spectra in SPEX format. For example, the demo spectra from the SPEX web site.

SPEX needs two files per spectrum:

- `<filename>.spo` – This file contains the countrate per energy bin for the source (D_i), as well as the background countrate and the errors (σ_i).
- `<filename>.res` – This file contains the instrumental response: the energy redistribution and effective area ($R_{ij}\ A_j$).

In order to calculate the observed model spectrum, SPEX uses this integral equation to account for the imperfections caused by the instrument:

$$D(c) = \int R(c,E)A(E)S(E)dE \tag{1.1}$$

$$D_i = \sum_{j=1}^{n} R_{ij}A_jS_j$$

The `.res` and `.spo` files are so-called FITS files. This is a data format widely used in Astronomy. FITS files can contain images as well as data tables. The software package FTOOLS provided by NASA contains a large number of tools to manipulate FITS files. If you are interested, then you can launch `flaunch` to see which tools are available. For more information about the SPEX spectrum and response format see *Optimal definition of respons matrices* (page 288).

1.2.2 Loading spectra into SPEX

The SPEX program is started by entering `spex` in a linux terminal window. In the following sections we describe one run of the program. To start SPEX do this:

```
user@linux:~> spex
 Welcome user to SPEX version 3.00.00

SPEX>
```

First, we have to load the data files. This is done using the `data` command (*Data: read response file and spectrum* (page 83)). It is a general thing in SPEX that filename extensions are not typed explicitly when issuing a command. If you have a file called `filename.spo` and `filename.res` then you type:

```
SPEX> data filename filename
```

The responsefile (.res) is entered first and then the file containing the spectrum (.spo). You can avoid confusion by giving the same filename to both .res and .spo files. Remember that the order of the words in the commands is very important!

To save you from typing a lot, many commands can be abbreviated by typing just the first few characters. For example, `da` is equivalent to `dat` and `data`.

1.2.3 Plotting the data

If the `data` command was successful, we can now have a look at the spectra. SPEX offers a lot of different plot commands (see *Plot: Plotting data and models* (page 106)). Using default settings, the easiest way of plotting a spectrum is as follows:

```
SPEX> plot dev xs
SPEX> plot type data
SPEX> plot
```

The sequence above opens a PGPLOT window (`plot dev xs`) and tells SPEX that we want to plot the spectral data (`plot type data`). This will create a linear-linear plot in keV units.

The plot can be tailored to your wishes. Below is an example to change the plot to a linear-linear plot in Å and add a title to the plot:

```
SPEX> plot x lin
SPEX> plot y lin
SPEX> plot ux a
SPEX> plot uy a
SPEX> plot rx 8.:35.
SPEX> plot ry 0.:0.05
SPEX> plot set 1
SPEX> plot cap ut text "This is my plot"
SPEX> plot cap lt disp false
SPEX> plot cap id disp false
SPEX> plot
```

To make sure the axes are linear, we give the commands (`plot x lin` and `plot y lin`) and change the axes to unit Å (`plot ux a` and `plot uy a`). The commands `plot rx 8.:35.` and `plot ry 0.:0.05` change the ranges on the x and y axes, respectively. Then the color of the data, background spectrum and model are set. The last commands beginning with `plot cap` remove some standard titles and other text around the plot. After you define the plot like in the example above, you can plot it with a single `plot` command.

The y-axis in this plot is in counts s^{-1} $^{-1}$. Ångstrom is not the only unit used in high-energy astrophysics. Usually, the energy of the photons is expressed in keV. In SPEX you can use keV by writing `k` instead of `a` in all commands. For example, `plot ux k` to use keV for the x-axis. An overview of possible units is provided in *Plot axis units and scales* (page 129).

1.2.4 Ignoring and rebinning

High-resolution X-ray spectra from Chandra and XMM-Newton are usually oversampled (e.g. the energy bins are much smaller than the spectral resolution) and contain a lot more channels then is useful. Therefore, it is necessary to remove wavelength intervals which contain bad data and rebin your spectrum. The SPEX command to ignore parts of the spectrum is called `ignore` (*Ignore: ignoring part of the spectrum* (page 95)) and the command to rebin is called `bin` (*Bin: rebin the spectrum* (page 81)). In the next example we bin the spectrum over the 8–35 Å range with a factor of 5 and ignore the rest of the spectrum:

```
ign 0:8 unit a
ign 35:100 unit a
bin 8:35 5 unit a
```

The words `unit a` tells SPEX that the ranges (for example 8.0:35.0) are given in Å. If you work with more than one spectrum (from more than one instrument), you can add an extra instrument statement:

```
ign ins 1:2 0:8 unit a
ign ins 1:2 35:100 unit a
bin ins 1:2 8:35 5 unit a
```

Here, instrument 1 to 2 are binned with a factor of 5 over the 8–35 Å range.

1.2.5 Defining a model

Now we have a clean and rebinned spectrum that is ready to fit. Before we can start fitting, we first need to define a model. It's equivalent to $S(E)$ in Eq. (1.1). The model can contain one or more of these components:

- `absm` Model for interstellar absorption.
- `reds` Redshift.
- `po` Powerlaw.
- `ga` Gaussian.

And there are more (see *Overview of spectral components* (page 137))! The following command sequence defines a simple powerlaw model at a certain redshift and absorbed by the interstellar medium. The individual components of the model are loaded one-by-one with the `com` command (*Comp: create, delete and relate spectral components* (page 82)):

```
SPEX> com reds
SPEX> com absm
SPEX> com po
SPEX> com rel 3 1,2
```

The last command (`com rel 3 1,2`) tells SPEX that component 3, the powerlaw, is first redshifted by component 1 and then absorbed by component 2. The order of the 1 and the 2 is important! Always think what happens in which order on the way from the source to the telescope.

For most sources the distance is more or less known. To get a right luminosity estimate for the source, the expected distance has to be provided to SPEX. This is done with the `distance` command (*Distance: set the source distance* (page 86)):

```
SPEX> dist 0.1 z
Distances assuming H0 =  50.0 km/s/Mpc and q0 = 0.500
Sector    m      A.U.        ly        pc       kpc       Mpc  redshift        cz
-------------------------------------------------------------------------------
1 1.894E+25 1.266E+14 2.002E+09 6.139E+08 6.139E+05  613.8689    0.1000   29979.2
-------------------------------------------------------------------------------
```

With this command, the distance to the source is set to a redshift of 0.1. The derived distances for this cosmology are in the output of the `dist` command.

Now we have to estimate the initial parameters. With the command `par show` we can see which parameters there are:

```
SPEX> par show
-------------------------------------------------------------------------------
sect comp mod  acro parameter with unit     value      status    minimum   maximum

   1    1 reds z    Redshift              0.000000     frozen   -1.0      1.00E+10
```

(continues on next page)

(continued from previous page)

```
   1    2 absm nh   Column (1E28/m**2)   9.9999997E-05 thawn     0.0      1.00E+20
   1    2 absm f    Covering fraction     1.000000     frozen    0.0       1.0

   1    3 pow  norm Norm (1E44 ph/s/keV)  1.000000     thawn     0.0      1.00E+20
   1    3 pow  gamm Photon index          2.000000     thawn    -10.       10.
   1    3 pow  dgam Photon index break    0.000000     frozen   -10.       10.
   1    3 pow  e0   Break energy (keV)   1.0000000E+10 frozen    0.0      1.00E+20
   1    3 pow  b    Break strength        0.000000     frozen    0.0       10.
   1    3 pow  type Type of norm          0.000000     frozen    0.0       1.0
   1    3 pow  elow Low flux limit (keV)  2.000000     frozen   1.00E-20  1.00E+10
   1    3 pow  eupp Upp flux limit (keV)  10.00000     frozen   1.00E-20  1.00E+10
   1    3 pow  lum  Luminosity (1E30 W)   1.000000     frozen    0.0      1.00E+20

-------------------------------------------------------------------------------
Fluxes and restframe luminosities between   2.0000     and    10.000     keV

 sect comp mod   photon flux   energy flux nr of photons    luminosity
            (phot/m**2/s)      (W/m**2)   (photons/s)           (W)
    1    3 pow    0.00000       0.00000       0.00000       0.00000
```

We can set the parameters using the `par` command (*Par: Input and output of model parameters* (page 104)). The first "1" in column "sect" can usually be ignored. The commands then look like this:

```
SPEX> par 1 z val 0.1
SPEX> par 2 nh val 2.E-3
SPEX> par 3 norm val 1.E+10
SPEX> par gamm val 1.5
```

The last component number used is saved, so in the last line we can skip typing the number 3 after `par`. Then, we run `par show` again to see what happened:

```
SPEX> par show
-------------------------------------------------------------------------------
sect comp mod  acro parameter with unit     value      status    minimum   maximum

   1    1 reds z    Redshift               0.100000     frozen   -1.0      1.00E+10

   1    2 absm nh   Column (1E28/m**2)   2.0000001E-03 thawn     0.0      1.00E+20
   1    2 absm f    Covering fraction     1.000000     frozen    0.0       1.0

   1    3 pow  norm Norm (1E44 ph/s/keV)  1.000000E+10 thawn     0.0      1.00E+20
   1    3 pow  gamm Photon index          1.500000     thawn    -10.       10.
   1    3 pow  dgam Photon index break    0.000000     frozen   -10.       10.
   1    3 pow  e0   Break energy (keV)   1.0000000E+10 frozen    0.0      1.00E+20
   1    3 pow  b    Break strength        0.000000     frozen    0.0       10.
   1    3 pow  type Type of norm          0.000000     frozen    0.0       1.0
   1    3 pow  elow Low flux limit (keV)  2.000000     frozen   1.00E-20  1.00E+10
   1    3 pow  eupp Upp flux limit (keV)  10.00000     frozen   1.00E-20  1.00E+10
   1    3 pow  lum  Luminosity (1E30 W)  5.6014867E+08 frozen    0.0      1.00E+20

-------------------------------------------------------------------------------
Fluxes and restframe luminosities between   2.0000     and    10.000     keV

 sect comp mod   photon flux   energy flux nr of photons    luminosity
            (phot/m**2/s)      (W/m**2)   (photons/s)           (W)
    1    3 pow    0.00000       0.00000       0.00000       0.00000
```

Finding the right initial values for the parameters is a game of trial and error. To see whether you are going in the right direction, you can calculate the model with the command `calc` and `plot` again (*Calculate: evaluate the spectrum* (page 81)). If you see the model appear in your screen, then the model is close enough to be fitted.

Especially the normalization of the powerlaw (`3 norm`) can vary a lot depending on the count rate of the source.

1.2.6 Fitting the data

We are ready to fit the data! SPEX has a nice feature to look at the progress of the fit. To activate this feature you have to give the command `fit print 1` (see *Fit: spectral fitting* (page 91)). If your initial parameters were acceptable, you can see the model converge to the data in the plot window after you entered the `fit` command. When the fit is done, then the parameters and C-stat are printed on screen. If the C-stat value is close to the expected C-stat value, then your fit is acceptable. Sometimes more runs of the command `fit` are necessary after changing some initial parameters. This is especially true when using complex models. Again this is a game of trial and error.

You also might want to fix or free certain parameters to see if they can be constrained. In SPEX fixing is `f` (frozen) and freeing is `t` (thawn). You can free the redshift and fix the $N_{\rm H}$ by the following commands:

```
SPEX> par 1 z stat t
SPEX> par 2 nh stat f
```

1.2.7 Calculating errors

When the fit is acceptable, you might want to know the uncertainties on your fitted parameters. Errors are determined one-by-one by fixing the parameter to some value and calculate the Δ C-stat with respect to the best fit. If you want to know the 1σ error on the parameter, you need to know its values at Δ C-stat = 1. This is done by the `error` command (*Error: Calculate the errors of the fitted parameters* (page 89)). You can calculate the error for each parameter. For example redshift:

```
SPEX> error 1 z
```

If you need another Δ C-stat limit (not recommended), then you can set the desired Δ C-stat in SPEX using the command: `error dchi 1.`

1.2.8 Making life easier

In this short manual you have seen a lot of commands, but to avoid typing too much you want to use some identical series of commands every time you fit a certain spectrum. For example, you don't want to type all plot commands again when making a plot. Therefore, the program has a command to solve this problem called `log` (see *Log: Making and using command files* (page 100)). With the command `log exe filename` you can execute a number of commands at the same time. The numbers are read from a normal text file with (in this case) the name `filename.com`. Again the extension `.com` should not be typed explicitly. Below is an example to setup a plot for an EPIC spectrum (range 0.2–10.0 keV) with a small frame that shows residuals. Note that you can put any command in such a command file and you can make comment lines by putting a # sign in front of the line.

```
# This is a command file that creates a plot with residuals.
plot dev xs
plot type data
plot x log
plot y log
plot rx 0.2:10.
plot ry 0.0001:10.
plot back disp t
plot set 1
plot data col 1
plot model col 2
plot back col 1
plot set all
plot frame new
plot frame 2
```

(continues on next page)

(continued from previous page)

```
plot type chi
plot uy rel
plot x log
plot rx 0.2:10.
plot ry -0.5:0.5
plot view def f
plot view x 0.08:0.92
plot view y 0.1:0.3
plot cap y text "Rel. Error"
plot cap ut disp f
plot cap lt disp f
plot cap id disp f
plot frame 1
plot view def f
plot view x 0.08:0.92
plot view y 0.3:0.9
plot cap x disp f
plot cap id disp f
plot cap ut disp f
plot box numlab bot f
```

1.2.9 Saving your work

There are several ways in SPEX you can save your work. Below you find a few examples to save your commands, output or plots.

Saving a plot

These commands open a PostScript plot device with filename `filename.ps`, then they plot your figure in the PS file and closes the device:

```
SPEX> plot dev cps filename.ps
SPEX> plot
SPEX> plot close 2
```

Saving commands

If you want to save all commands that you execute to an ASCII file (`filename.com`), then type `log save filename` (see also *Log: Making and using command files* (page 100)). Do not forget to close the file at the end of the session by typing `log close save`. The saved commands in the textfile can be executed again by the `log exe filename` command.

Saving output

In the same way as in the previous example, you can also save the output on your screen by typing `log out filename` (the file will be an ASCII file called `filename.out`). You can close the file with `log close out`. This command is very useful to save your parameters and errors.

1.2.10 Quitting the program

Just type `quit` (see *Quit: finish the program* (page 111)).

1.2.11 Tips & Tricks

- If you make a typo in a command or you want to do the same command again, then push the `arrow-up` button on your keyboard. There is an entire history of your commands there.
- The `Tab` key is able to automatically complete the command you are typing. In case there are more possibilities, it shows them all.

1.3 How to convert spectra to SPEX format

Trafo is a program to convert OGIP spectra and responses into SPEX format. Before trafo is started, you need the OGIP spectra and responses first. Please read the documentation regarding your dataset for more information on how to create OGIP files. The minimum that trafo needs is a spectrum (.pi or .pha) and a response matrix (.rmf or .rsp). If you want to subtract background spectra or if you have an additional arf file, then please also collect these files in your working directory.

Note: There is also an alternative for the trafo program called ogip2spex. This program is part of the SPEX Python tools (PYSPEX). This program takes the necessary input from the command line and is easy for scripting. More information about ogip2spex can be found on the Pyspex Github documentation page.

1.3.1 Starting trafo

It is best to run trafo in the directory where your OGIP files are. It is an interactive program, so it will ask the user for information when the program is run.

```
user@linux:~> trafo
Program trafo: transform data to SPEX 2.0 format
This is version 1.04, of trafo

Are your data in OGIP format            : type=1
Old (Version 1.10 and below) SPEX format: type=2
New (Version 2.00 and above) SPEX format: type=3

Enter the type:
```

The first questions are quite straightforward. In the case of OGIP spectra, the type is always 1. In principle, it is possible to put more than one spectrum in a spo and res file, but for most simple cases transforming a single spectrum is sufficient.

```
Enter the type: 1
Enter the number of spectra you want to transform: 1
Enter the maximum number of response groups per energy per spectrum: 100000
```

The maximum number of response groups should just be a large number in nearly all cases. In special cases, it can put a limit on the number of response groups that will be saved in the .res file.

1.3.2 Optimizing the response matrix

The following feature is present in trafo since version 1.02 (SPEX version 2.02). It allows the user to re-arrange the response matrix to make more optimal use of parallel processing. There are three options: 1. Keep the matrix as provided. 2. Try to re-arrange the matrix into contiguous groups. The program tries to identify physically distinctive components and avoids overlapping data. 3. Split the matrix into N equal-sized components. This is particularly useful for grating spectra (RGS) and allows for efficient matrix multiplication on multi-core processors. Any power of 2 between 8 and 32 should provide a fast response matrix. In the terminal, this option is provided in the following way:

```
How should the matrix be partioned?
Option 1: keep as provided (1 component, no re-arrangements)
Option 2: rearrange into contiguous groups
Option 3: split into N roughly equal-sized components
Enter your preferred option (1,2,3): 1
```

Option number 1 is the safest option to choose, but also the slowest. Option 2 and 3 can provide a significant increase in performance, but results should be carefully checked. More information about re-arranging response matrices can be found in the SPEX Manual.

1.3.3 Reading the spectra

Then, trafo asks for the filenames of the source and background spectra. First, provide the file name of the source spectrum. trafo will return some of the basic properties of the spectral file, like exposure time and values of the most important FITS keywords.

```
Enter filename spectrum to be read: PN-source.pi
Exposure time (s): 2.10571992E+04
Assuming Poissonian Errors
Areascal: 1.00000000E+00
Backscal: 1.00000000E+00
No BACKFILE keyword found
Corrscal: 1.00000000E+00
No CORRFILE keyword found
No RESPFILE keyword found
No ANCRFILE keyword found
No background specified in pha-file.
```

A background spectrum can be provided (optional), which will be subtracted from the source spectrum. If a background file is already specified in the FITS header of the source spectrum, this question will not be asked.

```
Read nevertheless a background file? (y/n) [no]: y
Enter filename background spectrum to be read: PN-background.pi
Exposure time (s): 2.10572832E+04
Assuming Poissonian Errors
Areascal: 1.00000000E+00
Backscal: 1.00000000E+00
No BACKFILE keyword found
Corrscal: 1.00000000E+00
No CORRFILE keyword found
No RESPFILE keyword found
No ANCRFILE keyword found
```

1.3.4 Bad channels and grouping

Depending on the instrument used, there is a chance that the spectrum contains bad channels. This is especially true for grating spectra. Sometimes the background spectrum can have a different number of bad channels than the source spectrum. It is therefore important that a particular bad channel in either of the two spectra is ignored. In this example, there are no bad channels, so either yes or no will do.

```
Checking data quality and grouping ...
Ogip files have quality flags. Quality 0 means okay
Your spectrum file has          0 bins with bad quality
Your background file has        0 bins with bad quality
Your combined file has          0 bins with bad quality
Shall we ignore bad channels? (y/n) [no]:y
```

If grppha has been used on the spectrum, trafo will also ask whether the spectra should be binned according to the groups defined in the PHA file.

Important note: We do not recommend the use of grppha for binning spectra. For spectra with Poisson statistics (most X-ray spectra), it is much better to use C-statistics and use an optimal binning algorithm in SPEX based on the spectral resolution of the instrument.

1.3.5 Read response and effective area files

In the next step, the response matrix is read. Sometimes, the response matrices start at channel 0, which can be somewhat confusing. Especially when some arrays start at channel 0 and others at channel 1. If both data sets start at zero, it is best to shift the channel numbers with 1 unit. For most instruments this is fine, however, there are situations when this does not apply. In that case, please check your energy grid by loading a delta line component in SPEX and check the energy of the line manually. Then, compare the output with a delta line defined in XSPEC.

```
Determining background subtracted spectra ...
No response matrix file specified in pha-file.
Enter filename response matrix to be read: PN.rmf
Reading response matrix ...
Warning, ebounds data started at channel    0
Warning, response data started at channel 0
Possible response conflict; check xspec/spex with delta line!
Enter shift to response array (1 recommended, but some cases may be 0):1
No effective area file specified in pha-file.
```

Sometimes, also an effective area file needs to be provided separately:

```
Read nevertheless an effective area file? (y/n) [no]: y
Enter filename arf-file to be read: PN.arf
Reading effective area ...
Determining zero response data  ...
Total number of channels with zero response:      373
Original number of data channels                                :         4096
Channels after passing mask and omitting zero response channels:         3723
Rebinning data where necessary ...
Rebinning response where necessary ...
old number of response elements:       435950
new number of response elements:       435950
old number of response groups  :         1481
new number of response groups  :         1481
Correcting for effective area ...

Determine number of components ...
Found     1 components
Enter any shift in bins (0 recommended): 0
order will not be swapped ...
```

If there are bins with zero response, then they are excluded from the resulting file. Also here a shift in bins can be set, but the recommended value is 0.

1.3.6 Writing res and spo files

The final step is writing the spectra in SPEX format. The file names should be provided without an extension. The .spo and .res extension will be added automatically.

```
Enter filename spectrum to be saved (without .spo): PN
Enter filename response to be saved (without .res): PN
Final number of response elements:   435950
```

The PN.spo and PN.res file have been saved in the current directory.

1.4 Install the SPEX Python interface

The SPEX python interface depends on a quite specific python environment. Using conda, you can create this environment pretty quickly. Please make sure you have (mini)conda installed and initialized on your machine before you continue.

1.4.1 Binary installation

The easiest option to install the Python interface for SPEX is by downloading the SPEX binary version for your platform. See and follow the instruction in *How to install SPEX* (page 1).

Once the standard SPEX environment is set up using the `spexdist.sh` or `spexdist.csh` files, it is time to build the conda environment. This can be done using the `spex.yml` file provided in the SPEX directory. Please enter the following command:

```
(base) unix:~/SPEX> conda env create -f $SPEX90/python/spex.yml
```

This creates the `spex` conda environment for you. This step should only be done once.

Note: If you installed SPEX through the Mac package installer, then `spex.yml` is not located in a writeable directory. Please copy `spex.yml` first to your home directory (`cp /opt/spex/python/spex.yml ~/`) and then create the conda environment like: `conda env create -f ~/spex.yml`.

If successful, you can from now on activate the environment with the command:

```
(base) unix:~/SPEX> conda activate spex
```

And from now on, you can use the python interface in SPEX:

```
(spex) unix:~/SPEX> python
Python 3.5.6 |Anaconda, Inc.| (default, Aug 26 2018, 21:41:56)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pyspex.spex import Session
>>> s=Session()
 Welcome user to SPEX version 3.06.01

 NEW in this version of SPEX:
22-07-2020 Bugfix: neij gives line emission while abundance is zero.
18-08-2020 Changed reference density from electron density to hydrogen density.
04-09-2020 Added pyroxene back to the amol model.
23-10-2020 Update of H I collision strengths (IMPORTANT)
```

(continues on next page)

(continued from previous page)

```
24-11-2020 Added ascdump and par_show functions for the Python interface.

 Currently using SPEXACT version 2.07.00. Type `help var calc` for details.
>>>
```

If you need other python packages for your project, you can install them through conda in your new `spex` environment.

1.4.2 Compile Python interface for SPEX

If you do not want to use conda, you can also compile the Python interfaces for SPEX. This can be done either through a compile script called `compile.py` or manually with CMake by adding the option `-DPYTHON=2` or `-DPYTHON=3` depending on the python version. The use of Python 3 is strongly recommended. See *Compile SPEX from source* (page 13) for details.

1.4.3 Integration into iPython and Jupyter Notebook

Next to the dependencies installed in the conda environment `spex`, the Python interface also depends on the SPEX environment variables set with the spexdist.(c)sh files. So before running iPython or Jupyter notebook, it is necessary to source the SPEX environment (also make sure the conda `spex` environment is activated):

```
(spex) unix:~> source /opt/spex/spexdist.sh
```

(replace /opt/spex in this path with the location of SPEX on your machine, or $SPEX90 if this variable is already set).

It may be that iPython and Jupyter are not yet installed in your conda environment. If not, please install them using the command:

```
(spex) unix:~> conda install jupyter_client ipykernel
```

With iPython and Jupyter notebook, it can be helpful to install the spex conda environment explicitly for your project:

```
(spex) unix:~> ipython kernel install --user --name=spex
```

In your Jupyter notebook, you can now select `spex` from the drop-down menu if you are creating a new project. The `spex` conda environment should now be linked to your Jupyter project.

1.5 Compile SPEX from source

Table of Contents:

1.5.1 Getting started

The SPEX source code can be compiled using the multi-platform Cmake build system. See http://www.cmake.org/ for more information and downloads, or check the package manager of your Linux distribution. The SPEX install needs CMake version 3.0 or higher.

Since SPEX is programmed mostly in Fortran 90, it is recommended to use a recent Fortran compiler. SPEX has been tested with GFortran (version 4 and above) and the Intel Fortran Compiler.

This Zenodo contains a tar.bz2 file containing the source code of SPEX: spex-3.06.01-Source.tar.bz2. Unpack it in a convenient directory:

```
unix:~/Software> tar xvfj spex-3.06.01-Source.tar.bz2
```

In the top-level directory, a script called `compile.py` is available to guide the user through the compilation process. Please read the section about library dependencies first and install what is needed. Otherwise, the script will tell you what is missing. To run the script call:

```
unix:~/Software/spex> python compile.py
```

Please note that the compile script requires Python 3 to be installed.

1.5.2 Library dependencies

SPEX depends on a few external libraries to function. For some of those, the library source code has been included in the SPEX source code package. By default, CMake will look for system libraries to link to. If they are not there, then the version in the source package will be used.

The following libraries and packages are required to compile SPEX:

- CMake
- X11
- Readline
- CFITSIO (*)
- BLAS (*)
- LAPACK (*)
- PGPLOT (*)

(*) The SPEX source tree also contains the library if necessary.

All these libraries are commonly available in Linux distributions, so please read the documentation of your distribution to find out how to install these libraries. Please note that some distributions require you to also install the 'development' package of a library to be able to use them during compilation. In the Debian repository, for example, the development package of readline is called 'libreadline-dev'.

Below, we list some library-specific comments that can be helpful in case of problems.

Readline

Note for Mac OSX users: The OSX readline library is NOT compatible with the GNU readline library. You need to compile your own readline library from source or find a GNU readline library elsewhere on your system to link to. Compilation may work in, for example, a MacPorts environment, although this has not been tested. The official Mac version of SPEX statically links to a compiled version of readline downloaded from:

https://tiswww.case.edu/php/chet/readline/rltop.html

CFITSIO

The SPEX source tree contains an old fortran version of fitsio, which is sufficient for the vast majority of applications. However, in exceptional cases, the old library cannot handle very large fits files. In those cases, it is better to link to a system CFITSIO library.

BLAS and LAPACK

Some of the SPEX models depend heavily on the BLAS and LAPACK linear algebra packages. The default routines are available in the SPEX source tree, but compiling those will not provide the best performance. The performance improves substantially if an optimized BLAS or LAPACK library is used. There are two tested options:

- Intel Math Kernel Library (MKL)
- OpenBLAS

When compiling with the Intel Fortran compiler, using MKL is quite obvious. To link the MKL library, add the following option to the cmake command:

```
cmake . -DMKL=YES
```

If MKL is not set, cmake will look for other options, like OpenBLAS, if they are installed on your machine. If nothing is found, the non-optimized code in the SPEX source tree is used. On Mac, CMake could find the MacOS Accelerator framework.

1.5.3 MacOS instructions

The compilation of SPEX on MacOS is slightly more demanding. SPEX can run natively on MacOS (without ports), but then it needs a few pre-installed programs:

- Xcode (Through the App store)
- CMake
- XQuartz
- GNU readline Compile and install readline with 'clang' and install in /usr/local.
- Fortran compiler. For example GCC/GFORTRAN.

1.5.4 General Compilation Instructions

When all library dependencies are installed, the compilation process can begin. Execute cmake in the root directory of the SPEX source tree, where CMakeLists.txt is located (mind the dot):

```
unix:~/Software/SPEX-3.06.01-Source> cmake .
```

If no errors occured and all libraries were found, then type 'make':

```
unix:~/Software/SPEX-3.06.01-Source> make
```

When the program needs to be installed system wide, then execute:

```
unix:~/Software/SPEX-3.06.01-Source> sudo make install
```

The program will be installed to /opt/spex by default. Usually, administrator rights are necessary to copy the files to the right location.

Before you can run SPEX, the environment needs to be set. This can be done with the source command:

```
source /opt/spex/spexdist.sh (bash shell)
source /opt/spex/spexdist.csh (C-type shell)
```

In case you used another prefix for the SPEX installation directory, you can find spexdist.sh or spexdist.csh in the prefix directory that you set. To load the SPEX environment automatically, add the relevant source line to your ~/.cshrc or ~/.bashrc file.

1.5.5 Optional features

There are several options that can be passed to CMake to influence the build process through the -D operator. Of course, all options can be combined in a single cmake call. See the cmake documentation and the CMakeLists.txt file for details.

Compiler selection

Select a different fortran compiler:

```
unix:~/Software/SPEX-3.06.01-Source> cmake . -DCMAKE_Fortran_COMPILER=ifort
```

Install prefix

Install SPEX at a different location in the 'make install' step:

```
unix:~/Software/SPEX-3.06.01-Source> cmake . -DCMAKE_INSTALL_PREFIX=/home/user/
↪software
```

Force use of SPEX libraries

The use of the SPEX libraries in the source tree can be forced:

```
unix:~/Software/SPEX-3.06.01-Source> cmake . -DCFITSIO=YES -DPGPLOT=YES
```

The command above will compile these libraries from the SPEX source tree. See the CMakeLists.txt file for more options.

1.6 Use SPEX through Docker

SPEX is written mostly in Fortran 90 and depends on a few system libraries. This makes it difficult to provide a few binary versions that will continue to run on multiple platforms over many years. Therefore, we have also created a Docker image for SPEX that can be run on the Docker platform, which is available for Linux, Mac OS and Windows.

1.6.1 Step 1: Download and install Docker on your computer

To run a Docker image, please install Docker on your computer. See the Docker website for details and look for the Docker Engine community edition. Once you have downloaded and installed Docker, you can continue with this tutorial.

1.6.2 Step 2: Download the SPEX docker image from Zenodo

The SPEX Docker image is available on this Zenodo page as a tar.gz file. Please download the file called spex-[version]-Docker.tar.gz, for example:

```
spex-3.06.01-Docker.tar.gz
```

Or, alternatively, you can pull the image from Docker hub with the command: `docker pull spexxray/spex:latest`.

1.6.3 Step 3: Import the SPEX image into Docker

Before you can run the docker file, it should be imported into the docker system. This can be done on the command line:

```
user@linux:~> docker load -i spex-3.06.01-docker.tar.gz
```

The image will be named spexxray/spex with the tag 3.05.00 and can be found with the command:

```
user@linux:~> docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
spexxray/spex 3.06.01 0a0a0a0a0a0 1 minute ago 996MB
```

1.6.4 Step 4: Run SPEX on Docker

Using the docker run command, the image can be executed and SPEX can be run in a so-called container. To enjoy all the capabilities of SPEX, two things need to be arranged in the docker run command: access to local directories and a graphical X11 connection. To arrange this, additional flags need to be specified on the command line.

1.6.5 Mounting local directories to the container

If you would like to mount your own home directory into the SPEX container such that you can use some spo and res files there or save the output files, then you need the following flags:

```
-e LOCAL_USER_ID=`id -u $USER`
-v /home/myusername:/home/user
```

The first flag arranges that the user in the container will have the same user ID as you. This will allow you to read and write to your home directory from within the container.

The second flag arranges that your true local home directory called /home/myusername is mounted to /home/user inside the SPEX container.

MAC users: Please note that on OSX your home directory is in /Users/myusername.

1.6.6 Arranging the X11 connection

To make sure PGPLOT can connect to the X11 server on the host, we need to make a few connections from the container to the host machine. This is done with the following flags:

```
-e DISPLAY=$DISPLAY
-v /tmp/.X11-unix:/tmp/.X11-unix
```

The first flag sets the DISPLAY variable inside the container to the DISPLAY variable of the host machine. The second flag mounts the X11 temporary directory of the host to the same directory inside the container.

MAC users: To use X11 on Mac, you need to install XQuartz (or a similar X11 server) and set it to 'Allow connections from network clients' in the XQuartz settings. In addition, the X server should be set to allow incoming connections from localhost on the command line:

```
user@macos:~> xhost +127.0.0.1
```

Then the DISPLAY variable on the docker run line should be set to host.docker.internal:0

1.6.7 The complete docker run command

The full run commands for docker now look like below, where -w means that the container will start in working directory /home/user.

For Linux:

```
docker run -it \
-e DISPLAY=$DISPLAY \
-e LOCAL_USER_ID=`id -u $USER` \
-v /tmp/.X11-unix:/tmp/.X11-unix \
-v /home/myusername:/home/user \
-w /home/user \
spexxray/spex:3.06.01
```

For Mac:

```
docker run -it \
-e DISPLAY=host.docker.internal:0 \
-e LOCAL_USER_ID=`id -u $USER` \
-v /tmp/.X11-unix:/tmp/.X11-unix \
-v /Users/myusername:/home/user \
-w /home/user \
spexxray/spex:3.06.01
```

The docker run command above will provide you with a prompt that will allow you to run spex:

```
user@linux:~> docker run -it -e DISPLAY=$DISPLAY -e LOCAL_USER_ID=`id -u $USER` \
-v /tmp/.X11-unix:/tmp/.X11-unix -v /home/myusername:/home/user -w /home/user \
spexxray/spex:3.06.01

Welcome to the SPEX Docker Container!
Just type 'spex' to start the program.
user@0922f2e4ff85:~>
```

In this environment, you can just run spex or trafo:

```
user@0922f2e4ff85:~> spex
 Welcome user to SPEX version 3.06.01

 NEW in this version of SPEX:
22-07-2020 Bugfix: neij gives line emission while abundance is zero.
18-08-2020 Changed reference density from electron density to hydrogen density.
04-09-2020 Added pyroxene back to the amol model.
23-10-2020 Update of H I collision strengths (IMPORTANT)
24-11-2020 Added ascdump and par_show functions for the Python interface.

 Currently using SPEXACT version 2.07.00. Type `help var calc` for details.

SPEX>
```

CHAPTER

TWO

ANALYSIS THREADS

2.1 Fitting a CCD spectrum

2.1.1 Goal

To load and fit a CCD spectrum (e.g. MOS1 aboard XMM-Newton) into SPEX.

2.1.2 Preparation

To be able to fit a spectrum, you need a spectrum and response file in SPEX format. To convert spectra from OGIP format (e.g. pha, arf, rmf) to SPEX format, please use the `trafo` program like described in *How to convert spectra to SPEX format* (page 9). If you have the Pyspextools package installed, then you can use ogip2spex as an alternative to `trafo`.

Using the commands above, you can obtain a .spo and .res file containing your spectrum. To follow this thread, you can also download example files here: `mos.spo` and `mos.res`.

2.1.3 Starting SPEX

The SPEX program is started by entering `spex` in a linux terminal window. In the following sections we describe one run of the program. To start SPEX do this:

```
user@linux:~> spex
 Welcome user to SPEX version 3.06.00

SPEX>
```

If SPEX does not run, please check if you set the `SPEX90` variable right and sourced `spexdist.(c)sh` as explained in *How to install SPEX* (page 1).

2.1.4 Loading data

Then, we have to load the data files. This is done using the `data` command (*Data: read response file and spectrum* (page 83)). It is a general thing in SPEX that filename extensions are not typed explicitly when issuing a command. If you have a file called `mos.spo` and `mos.res` then you type:

```
SPEX> data mos mos
```

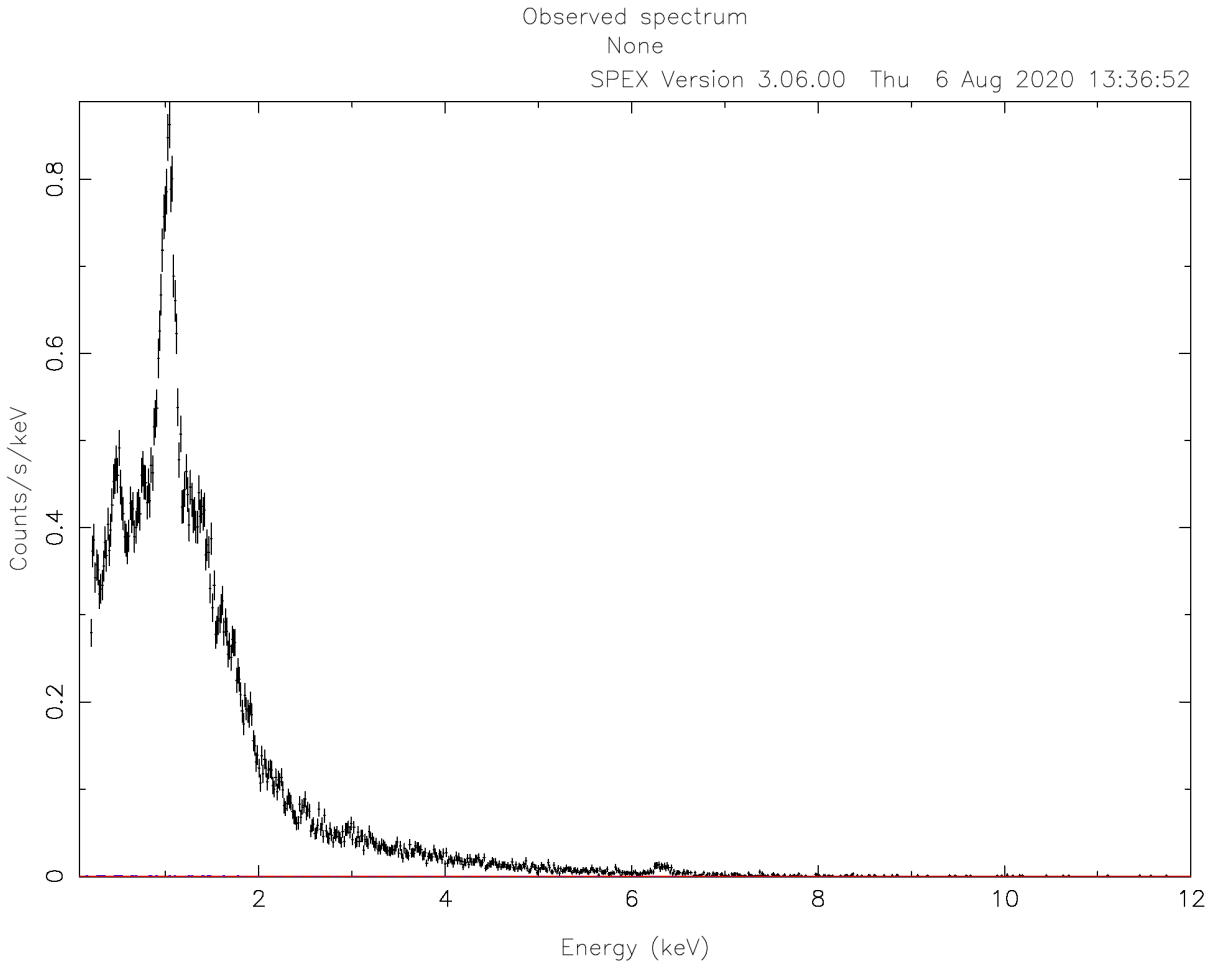
The response file (`mos.res`) is entered first and then the file containing the spectrum (`mos.spo`). You can avoid confusion by giving the same filename to both .res and .spo files. Remember that the order of the words in the commands is very important!

2.1.5 Plot the data

It really helps to see the spectrum while we are working on it, so we need a plot window and tell spex that we want to see a plot of the data. (Note that the text after the # is a comment and should not be copied on the command line).

```
SPEX> plot dev xs              # Open plot device `X server`
SPEX> plot type data           # Choose the plot type `data`
SPEX> plot
```

The commands above open a PGPLOT window and set the plot type to `data`, which means that the observed spectrum and the model (folded through the response matrix) will be shown. The last `plot` command tells SPEX to refresh the plot.

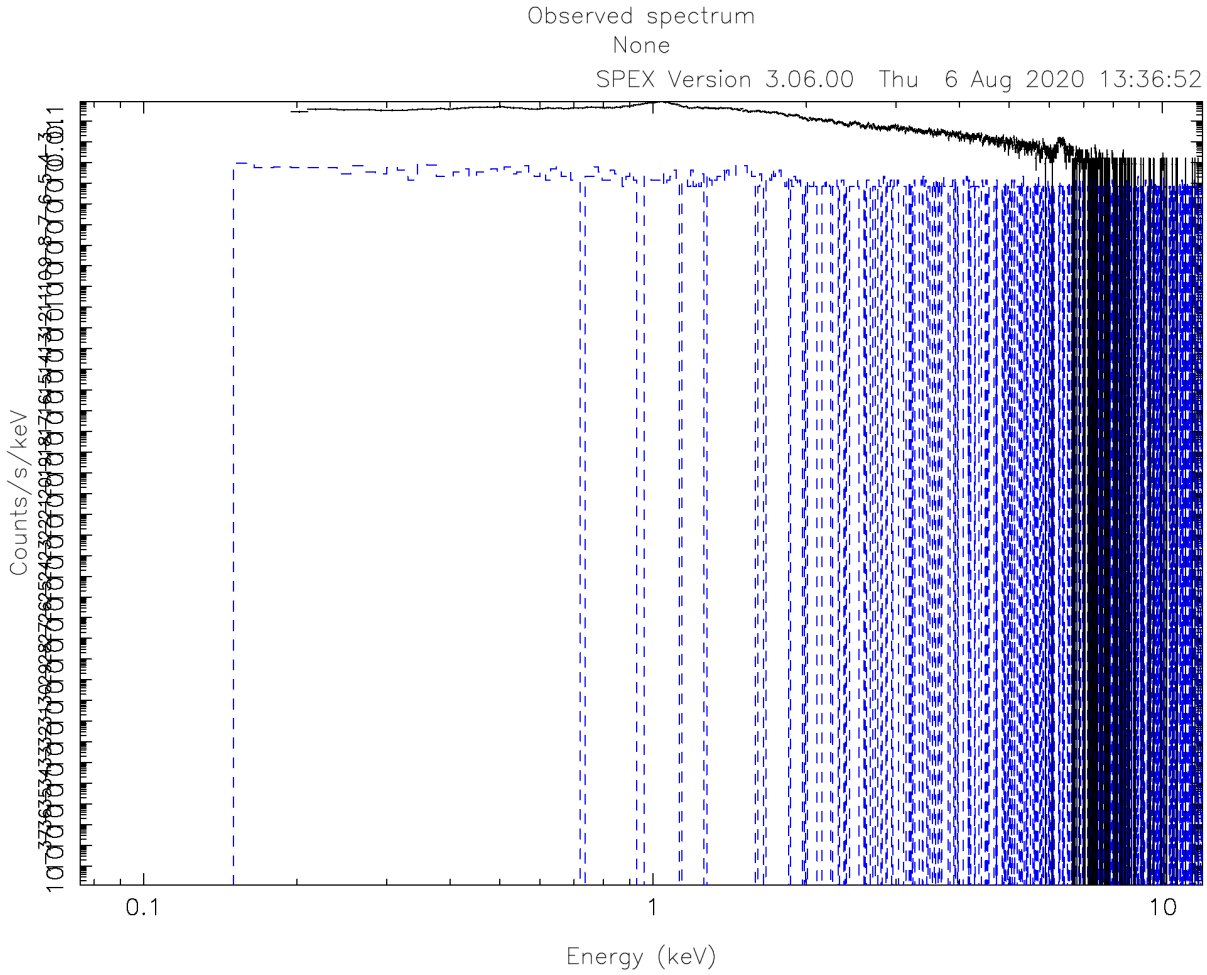


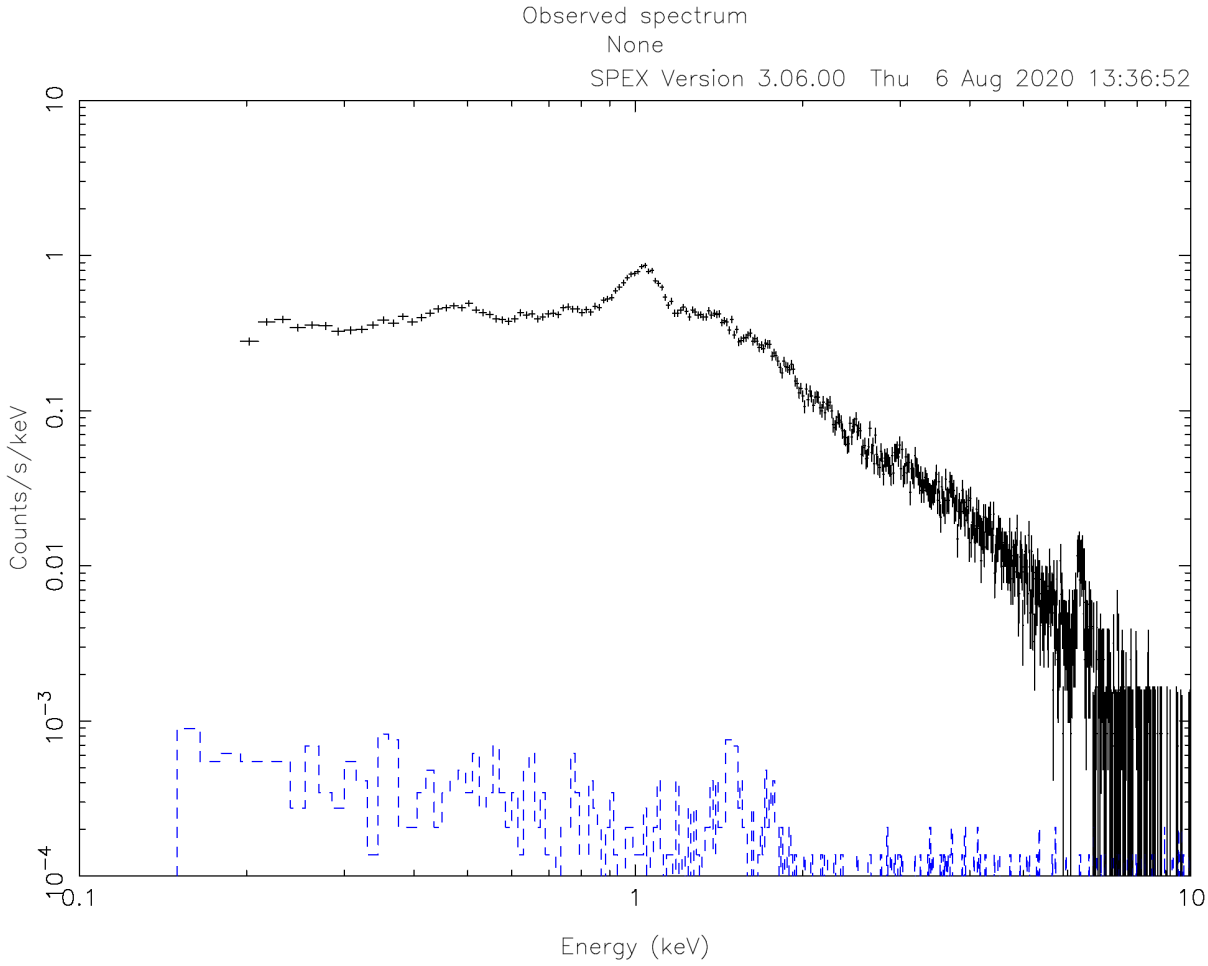
Usually CCD spectra benefit from a logarithmic scale on both the X and Y axes:

```
SPEX> plot x log
SPEX> plot y log
```

By default, the ranges of the axes are usually too broad. For this spectrum, the X axis range is good between 0.1 and 10 keV and the Y axis range between 1E-4 and 10.

```
SPEX> plot rx 0.1:10.
SPEX> plot ry 1E-4:10.
```


Observed spectrum
None
SPEX Version 3.06.00 Thu 6 Aug 2020 13:36:52
Counts/s/keV
0.1
1
10
Energy (keV)

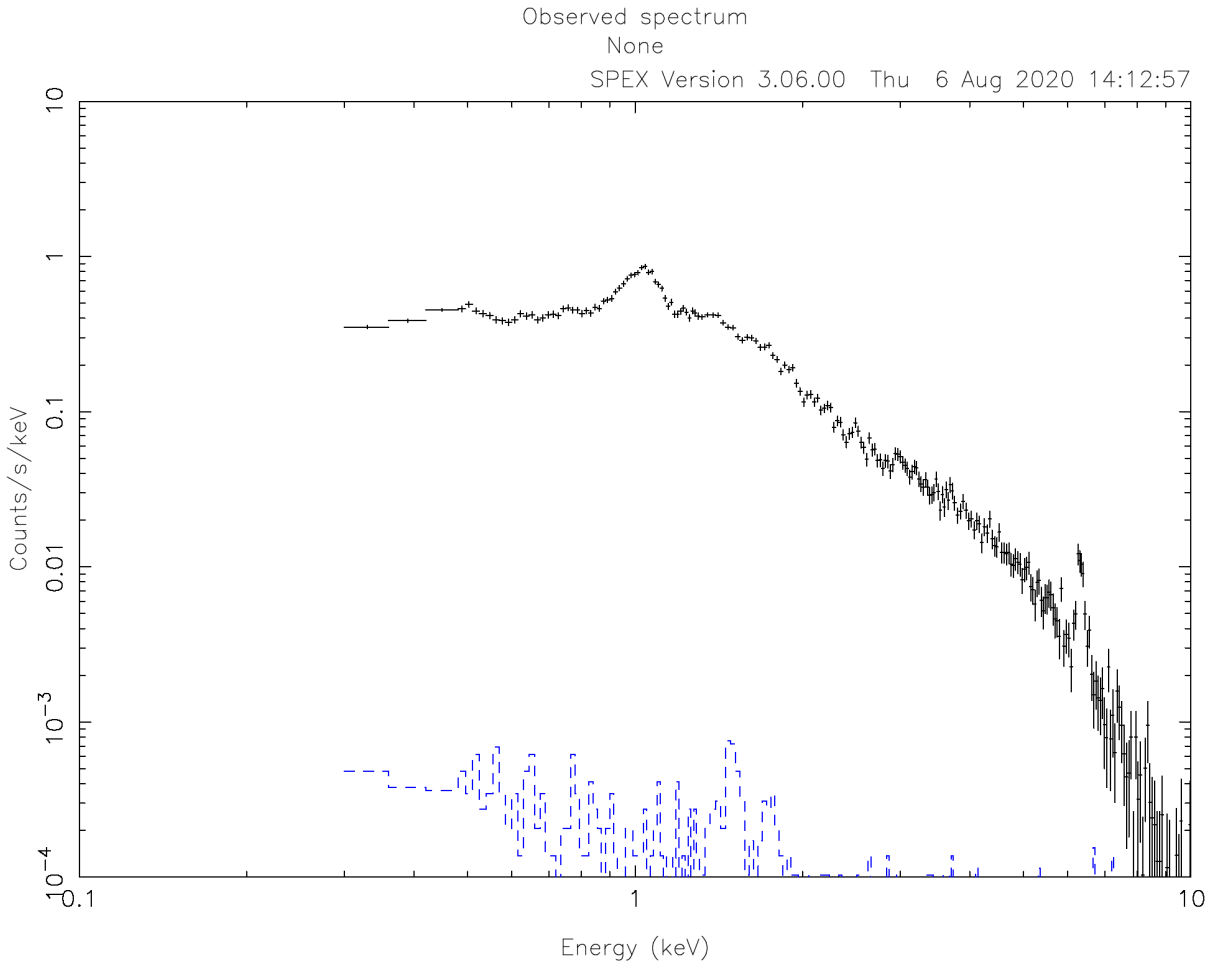

Observed spectrum
None
SPEX Version 3.06.00 Thu 6 Aug 2020 13:36:52
Counts/s/keV
10
1
0.1
0.01
10^{-3}
10^{-4}
0.1
1
10
Energy (keV)

2.1.6 Select good data and bin the spectrum

Before we start fitting, we can make sure that only trustworthy data is fitted. For MOS1, for example, we know that the calibration is valid for the energy range between roughly 0.3 keV and 10 keV. To fit only the good spectral interval, we need to ignore the parts at low and high energies. This is done using the `ignore` command.

In addition, we can bin the spectrum. To automatically rebin to the recommended (optimal) bin size, one can use the `obin` command. This command bins the spectrum optimally based on the instrument resolution and statistics.

```
SPEX> ignore 0:0.3 unit kev
SPEX> ignore 10:100 unit kev
SPEX> obin 0.3:10 unit kev
SPEX> plot
```



2.1.7 Define the model

Next we can define the model that we want to fit. In this case, we are looking at a MOS spectrum of a galaxy cluster. The simplest model that we can try is a single temperature spectrum absorbed by gas in the ISM. We also add a redshift component `reds` (*Reds: redshift model* (page 169)) to shift the energy of the model spectrum with the right amount:

```
SPEX> com reds
You have defined    1 component.
SPEX> com hot
You have defined    2 components.
SPEX> com cie
You have defined    3 components.
```

The `hot` model (*Hot: collisional ionisation equilibrium absorption model* (page 153)) is actually a gas in equilibrium in absorption, which is a fair representation of the neutral gas phase of the ISM. Later we will put the temperature of this component to 5×10^{-4} keV to emulate a neutral plasma.

The `cie` model (*Cie: collisional ionisation equilibrium model* (page 142)) represents a single temperature plasma in collisional ionisation equilibrium, which is commonly used for clusters.

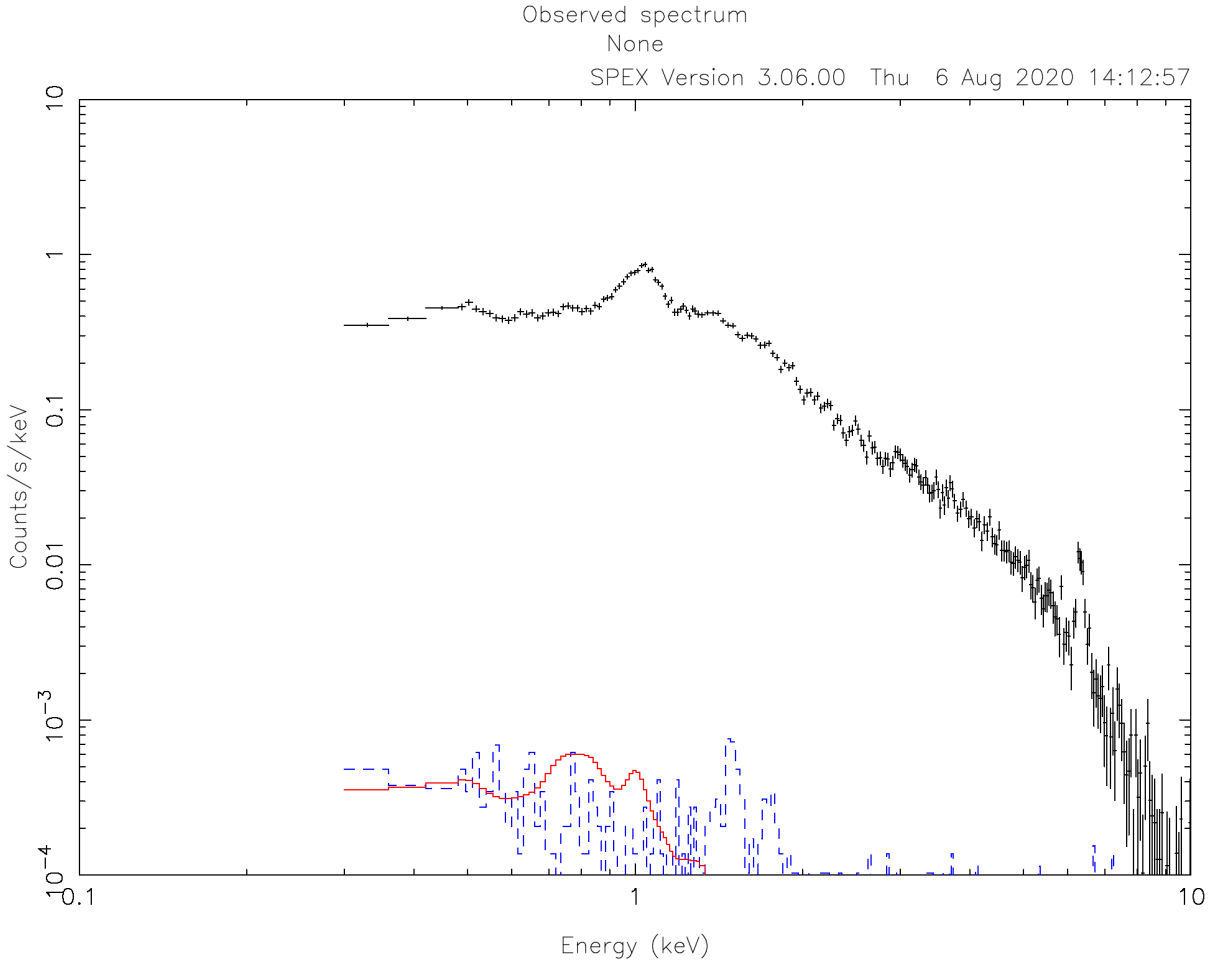
Then the components need to be related to each other, which means you need to specify how the multiplicative models should be applied to the additive models. The multiplicative components should be listed in order from the source to the observer (see also *Comp: create, delete and relate spectral components* (page 82)):

```
SPEX> com rel 3 1,2
```

This means that the emitted CIE component (#3) will be first redshifted by component #1 and then absorbed by component #2. If you have multiple additive components, this should be done for each one. It is possible to supply a range of components.

```
SPEX> calc
SPEX> plot
```

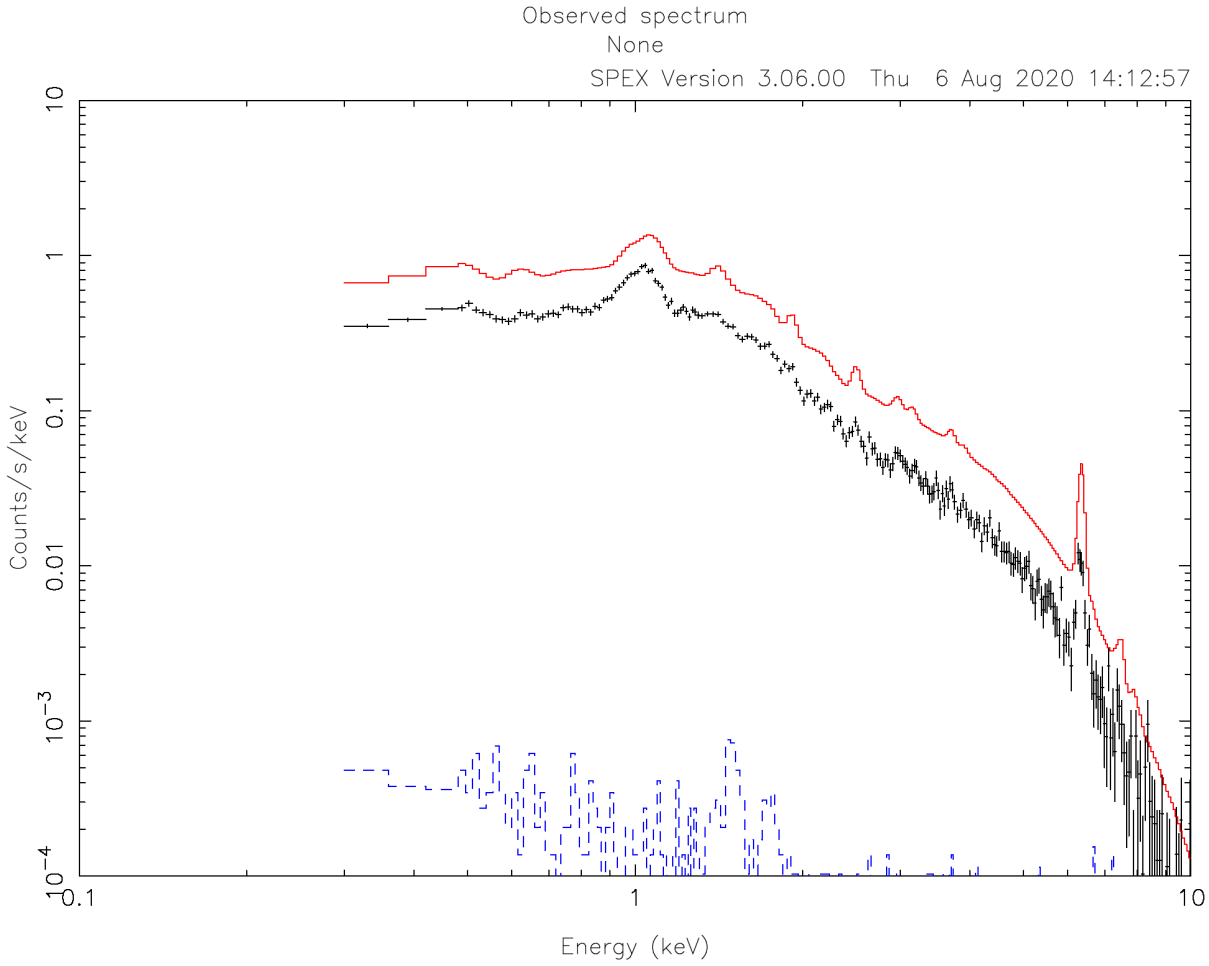
Calculating and plotting the model unsurprisingly results in a curve that is not near to the data.



2.1.8 Initial guess of parameters

To help the spectral fitting process, it is good to provide initial guesses for the model parameters. This way, the spectral fit starts already with a model that is in the right direction. As a more experienced user, you usually have a rough idea what the parameters should be by looking at the raw spectrum. For this cluster, for example, the redshift is around 0.05, the absorption column is small, and the temperature of the cluster is around 3 keV. We can set the guess parameters as follows:

```
SPEX> par 1 1 z v 0.05
SPEX> par 1 1 z s t
SPEX> par 1 2 t v 8E-6
SPEX> par 1 2 t s f
SPEX> par 1 2 nh v 1E-4
SPEX> par 1 3 norm v 1000.
SPEX> par 1 3 t v 3.0
SPEX> ca
SPEX> pl
```



As we can see from the image, the first guess of the model is already in the right direction.

In this example, we assume that the exact redshift is unknown. However, if you do have an accurate measurement of the distance, it is wise to set that distance in SPEX (*Distance: set the source distance* (page 86)):

```
SPEX> dist 0.05 z
```

The command above sets the distance to 0.05 z and makes sure that the luminosities are correctly calculated. Note that this distance change also affects the values of the normalisation of the models! So, we increase the normalisation to keep the model reasonably close to the data:

```
SPEX> par 1 3 norm v 1E+8
```

2.1.9 Fit the model

We are now ready to fit the spectrum. To see the fitting steps, we can give the command `fit print 1`. This needs to be set only once per session. A subsequent `fit` command (*Fit: spectral fitting* (page 91)) starts to optimize the parameters:

```
SPEX> fit print 1
SPEX> fit
   42102.6              5  5.000E-02  1.000E-04  1.000E+08   3.00
   2461.87             10  5.404E-02  9.087E-05  2.021E+08   2.04
   428.533             15  5.439E-02  1.041E-04  2.412E+08   2.31
   414.925             20  5.437E-02  9.425E-05  2.429E+08   2.37
   414.651             25  5.426E-02  9.229E-05  2.428E+08   2.37
   414.532             30  5.408E-02  9.269E-05  2.428E+08   2.37
   414.525             35  5.412E-02  9.281E-05  2.428E+08   2.37
   414.524             42  5.411E-02  9.270E-05  2.428E+08   2.37
   414.524             48  5.411E-02  9.265E-05  2.428E+08   2.37
-------------------------------------------------------------------------------
↪---------------
sect comp mod  acro parameter with unit     value      status    minimum   maximum␣
↪lsec lcom lpar

   1    1 reds z    Redshift              5.4108180E-02 thawn    -1.0      1.00E+10
   1    1 reds flag Flag: cosmo=0, vel=1  0.000000     frozen    0.0        1.0

   1    2 hot  nh   X-Column (1E28/m**2) 9.2653019E-05 thawn     0.0      1.00E+20
   1    2 hot  t    Temperature (keV)    1.9999999E-04 frozen   2.00E-04  1.00E+03
   1    2 hot  rt   T(balance) / T(spec)  1.000000     frozen   1.00E-04  1.00E+04
   1    2 hot  fcov Covering fraction     1.000000     frozen    0.0       1.0
   1    2 hot  v    RMS Velocity (km/s)   100.0000     frozen    0.0      3.00E+05
   1    2 hot  rms  RMS blend    (km/s)   0.000000     frozen    0.0      1.00E+05
   1    2 hot  dv   Vel. separ. (km/s)    100.0000     frozen    0.0      1.00E+05
   1    2 hot  zv   Average vel. (km/s)   0.000000     frozen  -1.00E+05  1.00E+05
   1    2 hot  ref  Reference atom        1.000000     frozen    1.0       30.
   1    2 hot  01   Abundance H           1.000000     frozen    0.0      1.00E+10
   1    2 hot  02   Abundance He          1.000000     frozen    0.0      1.00E+10
   1    2 hot  03   Abundance Li          1.000000     frozen    0.0      1.00E+10
   1    2 hot  04   Abundance Be          1.000000     frozen    0.0      1.00E+10
   1    2 hot  05   Abundance B           1.000000     frozen    0.0      1.00E+10
   1    2 hot  06   Abundance C           1.000000     frozen    0.0      1.00E+10
   1    2 hot  07   Abundance N           1.000000     frozen    0.0      1.00E+10
   1    2 hot  08   Abundance O           1.000000     frozen    0.0      1.00E+10
   1    2 hot  09   Abundance F           1.000000     frozen    0.0      1.00E+10
   1    2 hot  10   Abundance Ne          1.000000     frozen    0.0      1.00E+10
   1    2 hot  11   Abundance Na          1.000000     frozen    0.0      1.00E+10
   1    2 hot  12   Abundance Mg          1.000000     frozen    0.0      1.00E+10
   1    2 hot  13   Abundance Al          1.000000     frozen    0.0      1.00E+10
   1    2 hot  14   Abundance Si          1.000000     frozen    0.0      1.00E+10
   1    2 hot  15   Abundance P           1.000000     frozen    0.0      1.00E+10
   1    2 hot  16   Abundance S           1.000000     frozen    0.0      1.00E+10
   1    2 hot  17   Abundance Cl          1.000000     frozen    0.0      1.00E+10
   1    2 hot  18   Abundance Ar          1.000000     frozen    0.0      1.00E+10
   1    2 hot  19   Abundance K           1.000000     frozen    0.0      1.00E+10
   1    2 hot  20   Abundance Ca          1.000000     frozen    0.0      1.00E+10
   1    2 hot  21   Abundance Sc          1.000000     frozen    0.0      1.00E+10
   1    2 hot  22   Abundance Ti          1.000000     frozen    0.0      1.00E+10
   1    2 hot  23   Abundance V           1.000000     frozen    0.0      1.00E+10
   1    2 hot  24   Abundance Cr          1.000000     frozen    0.0      1.00E+10
```

(continues on next page)

(continued from previous page)

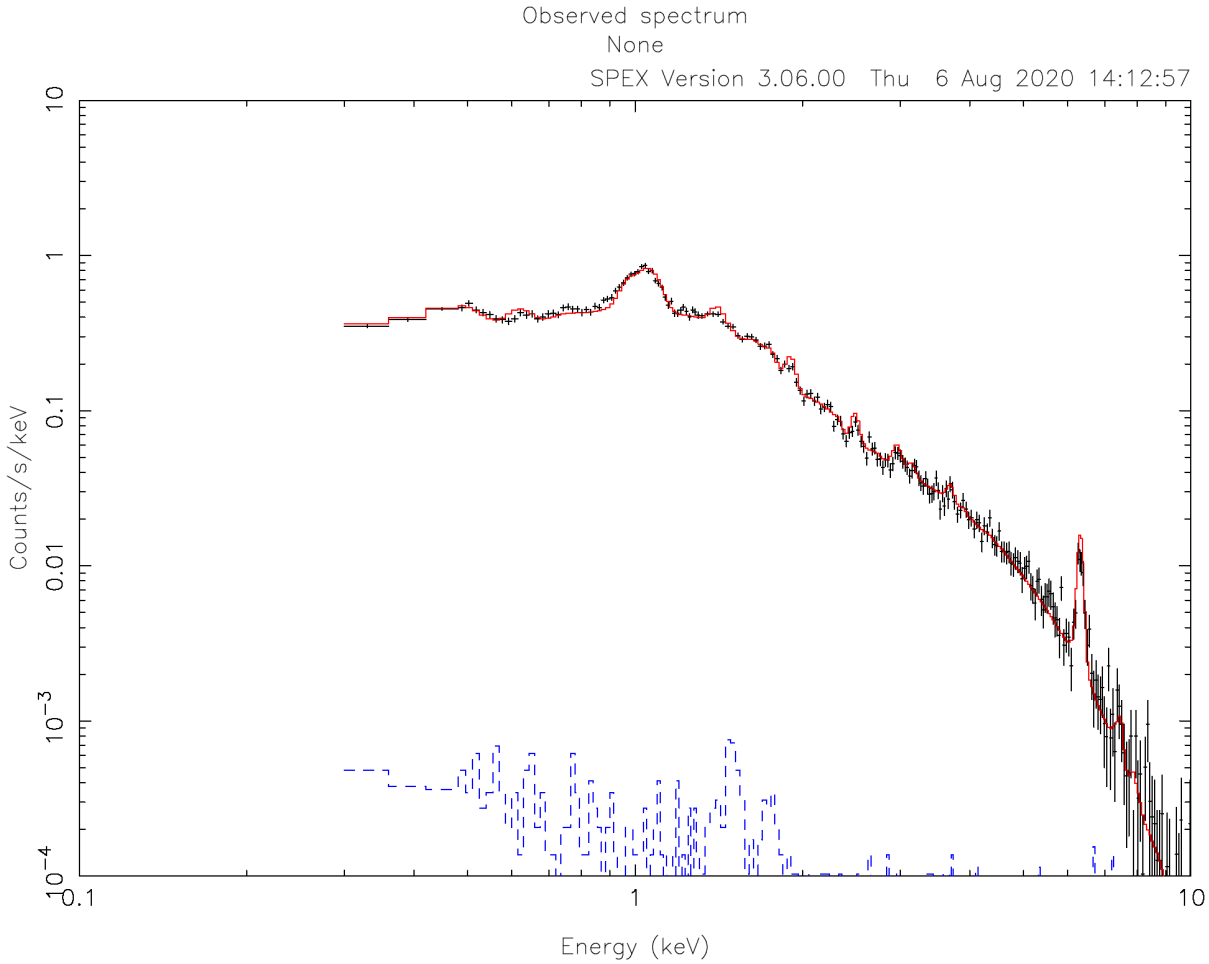
```
   1    2 hot  25   Abundance Mn          1.000000     frozen    0.0       1.00E+10
   1    2 hot  26   Abundance Fe          1.000000     frozen    0.0       1.00E+10
   1    2 hot  27   Abundance Co          1.000000     frozen    0.0       1.00E+10
   1    2 hot  28   Abundance Ni          1.000000     frozen    0.0       1.00E+10
   1    2 hot  29   Abundance Cu          1.000000     frozen    0.0       1.00E+10
   1    2 hot  30   Abundance Zn          1.000000     frozen    0.0       1.00E+10
   1    2 hot  file File electr.distrib.

   1    3 cie  norm ne nX V (1E64/m**3)  2.4279381E+08 thawn     0.0       1.00E+20
   1    3 cie  t    Temperature (keV)     2.372914     thawn    5.00E-04  1.00E+03
   1    3 cie  sig  Sigma                 0.000000     frozen    0.0       1.00E+04
   1    3 cie  sup  Sigma up              0.000000     frozen    0.0       1.00E+04
   1    3 cie  logt T grid (lin/log)      1.000000     frozen    0.0        1.0
   1    3 cie  ed   El  dens (1E20/m**3) 9.9999998E-15 frozen   1.00E-22  1.00E+10
   1    3 cie  it   Ion temp  (keV)       1.000000     frozen   1.00E-04  1.00E+07
   1    3 cie  rt   T(balance) / T(spec)  1.000000     frozen   1.00E-04  1.00E+04
   1    3 cie  vrms RMS Velocity (km/s)   0.000000     frozen    0.0       3.00E+05
   1    3 cie  ref  Reference atom        1.000000     frozen    1.0        30.
   1    3 cie  01   Abundance H           1.000000     frozen    0.0       1.00E+10
   1    3 cie  02   Abundance He          1.000000     frozen    0.0       1.00E+10
   1    3 cie  03   Abundance Li          1.000000     frozen    0.0       1.00E+10
   1    3 cie  04   Abundance Be          1.000000     frozen    0.0       1.00E+10
   1    3 cie  05   Abundance B           1.000000     frozen    0.0       1.00E+10
   1    3 cie  06   Abundance C           1.000000     frozen    0.0       1.00E+10
   1    3 cie  07   Abundance N           1.000000     frozen    0.0       1.00E+10
   1    3 cie  08   Abundance O           1.000000     frozen    0.0       1.00E+10
   1    3 cie  09   Abundance F           1.000000     frozen    0.0       1.00E+10
   1    3 cie  10   Abundance Ne          1.000000     frozen    0.0       1.00E+10
   1    3 cie  11   Abundance Na          1.000000     frozen    0.0       1.00E+10
   1    3 cie  12   Abundance Mg          1.000000     frozen    0.0       1.00E+10
   1    3 cie  13   Abundance Al          1.000000     frozen    0.0       1.00E+10
   1    3 cie  14   Abundance Si          1.000000     frozen    0.0       1.00E+10
   1    3 cie  15   Abundance P           1.000000     frozen    0.0       1.00E+10
   1    3 cie  16   Abundance S           1.000000     frozen    0.0       1.00E+10
   1    3 cie  17   Abundance Cl          1.000000     frozen    0.0       1.00E+10
   1    3 cie  18   Abundance Ar          1.000000     frozen    0.0       1.00E+10
   1    3 cie  19   Abundance K           1.000000     frozen    0.0       1.00E+10
   1    3 cie  20   Abundance Ca          1.000000     frozen    0.0       1.00E+10
   1    3 cie  21   Abundance Sc          1.000000     frozen    0.0       1.00E+10
   1    3 cie  22   Abundance Ti          1.000000     frozen    0.0       1.00E+10
   1    3 cie  23   Abundance V           1.000000     frozen    0.0       1.00E+10
   1    3 cie  24   Abundance Cr          1.000000     frozen    0.0       1.00E+10
   1    3 cie  25   Abundance Mn          1.000000     frozen    0.0       1.00E+10
   1    3 cie  26   Abundance Fe          1.000000     frozen    0.0       1.00E+10
   1    3 cie  27   Abundance Co          1.000000     frozen    0.0       1.00E+10
   1    3 cie  28   Abundance Ni          1.000000     frozen    0.0       1.00E+10
   1    3 cie  29   Abundance Cu          1.000000     frozen    0.0       1.00E+10
   1    3 cie  30   Abundance Zn          1.000000     frozen    0.0       1.00E+10
   1    3 cie  file File electr.distrib.
   1    3 cie  x1   T1/T0                 1.000000     frozen    1.0       1.00E+10
   1    3 cie  y1   N1/N0                 0.000000     frozen    0.0       1.00E+10


Instrument     1 region    1 has norm    1.00000E+00 and is frozen

-------------------------------------------------------------------------------
 Fluxes and restframe luminosities between   2.0000     and    10.000     keV

 sect comp mod   photon flux   energy flux nr of photons    luminosity
              (phot/m**2/s)      (W/m**2)   (photons/s)           (W)
    1    3 cie    4.13667      2.229383E-15  2.655905E+51  1.422295E+36
```

(continues on next page)

(continued from previous page)

```
------------------------------------------------------------------------------
 Fit method          : Classical Levenberg-Marquardt
 Fit statistic       : C-statistic
 C-statistic         :       414.52
 Expected C-stat     :       247.13 +/-        21.92
 Chi-squared value   :       528.16
 Degrees of freedom  :       239
 W-statistic         :         0.00
```

At the end of the optimization step, SPEX prints out an overview of the fit parameters and the fit statistics. Here we can see that the fit improved, but there is still room for improvement.



2.1.10 Fitting abundances

Although the C-statistics are not too bad, there are still residuals in the spectrum, especially around the strongest spectral lines. This is because the metal abundances in the gas are still fixed to 1.0. We can let the abundances vary in the optimization by setting them to thawn:

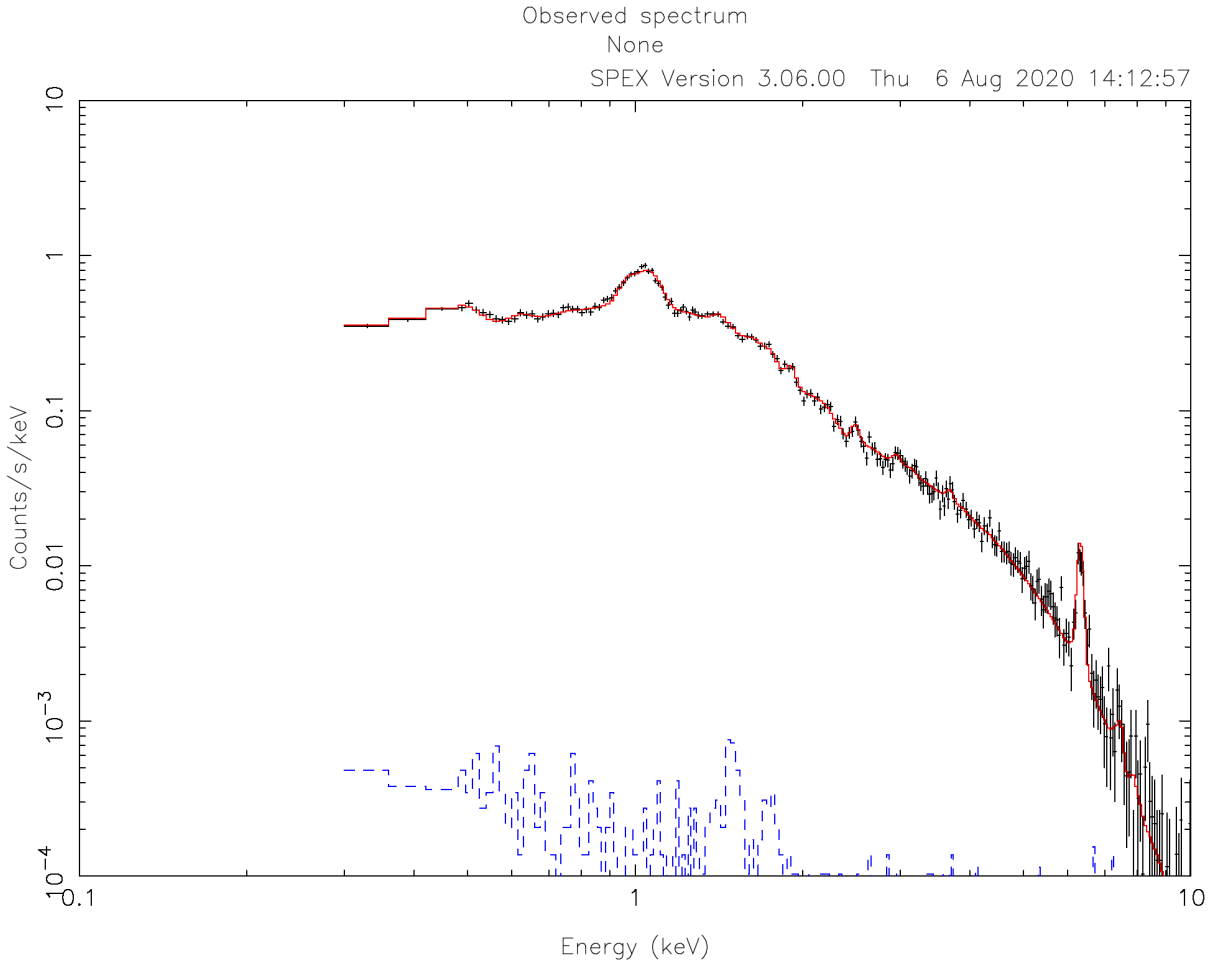
```
SPEX> par 1 3 08 s t
SPEX> par 1 3 12 s t
SPEX> par 1 3 14 s t
SPEX> par 1 3 16 s t
SPEX> par 1 3 18 s t
SPEX> par 1 3 20 s t
```

(continues on next page)

(continued from previous page)

```
SPEX> par 1 3 26 s t
SPEX> par 1 3 28 s t
SPEX> fit
```

The optimization leads to an even better fit:




```
Fit method          : Classical Levenberg-Marquardt
Fit statistic       : C-statistic
C-statistic         :       266.80
Expected C-stat     :       247.12 +/-        21.91
Chi-squared value   :       366.77
Degrees of freedom:       231
W-statistic         :         0.00
```

2.1.11 Calculating errors

When we have the best fit, we can calculate the errors (*Error: Calculate the errors of the fitted parameters* (page 89)). This has to be done per parameter. Below we calculate, for example, the error on the best fit temperature:

```
SPEX> error 1 3 t
 parameter          C-stat      Delta          Delta
   value            value       parameter     C-stat
-------------------------------------------------------
   2.33348          267.68     -2.246547E-02    0.89
```

(continues on next page)

(continued from previous page)

```
   2.31101           270.23      -4.493093E-02    3.44
   2.33348           267.68      -2.246547E-02    0.89
   2.33247           267.69      -2.346587E-02    0.90
   2.32425           268.52      -3.169394E-02    1.73
   2.32836           268.05      -2.758002E-02    1.26
   2.33042           267.87      -2.552295E-02    1.07
   2.33127           267.79      -2.466774E-02    1.00
   2.37841           267.71       2.246547E-02    0.92
   2.40087           270.28       4.493093E-02    3.49
   2.37841           267.71       2.246547E-02    0.92
   2.37914           267.77       2.319646E-02    0.98
   2.37940           267.78       2.345586E-02    0.99
   2.37974           267.81       2.379751E-02    1.02
 Parameter   1    3 t   :    2.3559     Errors:  -2.46677E-02 ,   2.37975E-02
```

The error command reports the best fit value for the temperature and the lower and upper 1 sigma (68%) confidence level.

Usually, the error calculation stage is the end point of a spectral analysis. In this example, we can quit SPEX now:

```
SPEX> quit
Thank you for using SPEX!
```

2.2 Modeling particle background

2.2.1 Goal

Time dependent particle backgrounds in X-ray spectra are very difficult to correctly subtract, especially for extended sources. Many times, a quiescent particle background remains present in the spectrum after flare filtering. In this example, we show how to model the quiescent soft-proton contribution in a spectrum extracted from an annulus around the core of a cluster of galaxies. The difficulty here is that the effective area for soft protons is very different from the effective area for X-rays. Please note that the example provided is not necessarily scientifically correct. The goal of this example is to show a general method to deal with these kind of problems in SPEX. The choice of models probably needs to be different in other cases.

2.2.2 SPEX solution

The basic problem we have to solve here, is that we need a number of model components that are folded through the mirror effective area (the cosmic X-rays) and a few components describing the particle background, which are not folded through the ARF. In SPEX, this can be solved using sectors (*Sectors and regions* (page 285)). Sectors are essentially model groups representing different areas or different components on the sky. In this case, we will create two sectors: one for the cosmic X-rays and one for the particle background. The second sector should not be folded through the ARF. To achieve this, we have to create a special spectrum and response file with `trafo` (*Trafo* (page 181)) in which we define the sectors.

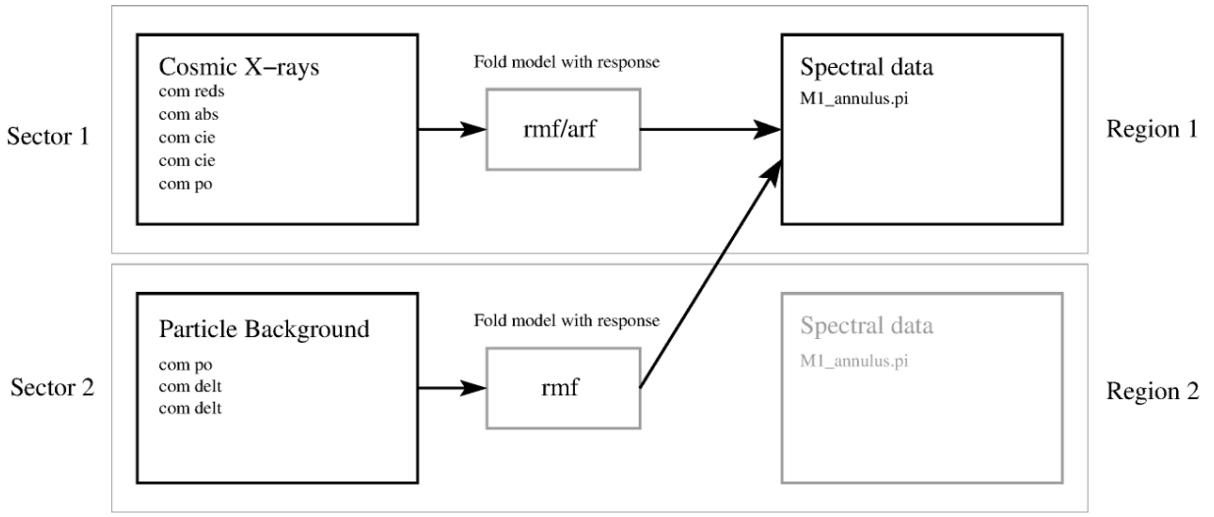


Fig. 1: Schematic representation of the sectors and regions in this example. We load two spectra with trafo and we define two sectors (left). The exact model components are later defined in SPEX. The models for sectors 1 and 2 are folded through the response matrix separately. The result of the folding is added and applied to the first spectrum (region 1 on the right) only.

Running trafo

In this trafo run, we will actually load the same spectrum twice. One for every sector. Here we use a MOS1 spectrum extracted from an annulus between 6 and 9 arcmin from the cluster core. The background spectrum was extracted using the XMM Extended Source Analysis Software by Snowden & Kuntz. After starting `trafo` we have to tell it that we want to transform two spectra in two sectors:

```
Program trafo: transform data to SPEX 2.0 format
This is version 1.02, of trafo

 Are your data in OGIP format            : type=1
 Old (Version 1.10 and below) SPEX format: type=2

 Enter the type: 1
 Enter the number of spectra you want to transform: 2
 Enter the maximum number of response groups per energy per spectrum: 1000000
 Enter the number of sectors you want to create: 2
```

The region number respresents the spectral data that we will fit. Because we want to add the cosmic X-ray spectrum and the particle background spectrum, we want both sectors to point to region 1. First, we enter the spectra for the first sector. We only show the most relevant input/output lines here.

```
Enter the sector and region number: 1 1
How should the matrix be partioned?
Option 1: keep as provided (1 component, no re-arrangements)
Option 2: rearrange into contiguous groups
Option 3: split into N roughly equal-sized components
Enter your preferred option (1,2,3): 1
Enter filename spectrum to be read: M1_annulus.pi
Read nevertheless a background file? (y/n) [no]: y
Enter filename background spectrum to be read: M1_annulus_bkg.pi
Shall we ignore bad channels? (y/n) [no]:y
Enter filename response matrix to be read: M1_annulus.rmf
Enter new bin boundary values manually: 3.E-5 5.E-3
Enter shift to response array (1 recommended, but some cases may be 0):1
Read nevertheless an effective area file? (y/n) [no]: y
Enter filename arf-file to be read: M1_annulus.arf
```

The first spectrum is now read in, including an ARF file. Now we enter the same spectrum again, but now without ARF. The region number here is 1, because we want the models in this sector to be added to the models of sector 1.

```
Enter the sector and region number: 2 1
Enter your preferred option (1,2,3): 1
Enter filename spectrum to be read: M1_annulus.pi
Read nevertheless a background file? (y/n) [no]: y
Enter filename background spectrum to be read: M1_annulus_bkg.pi
Enter filename response matrix to be read: M1_annulus.rmf
Read nevertheless an effective area file? (y/n) [no]: n
```

Save the spectrum by providing convenient names for the res and spo files.

```
Enter filename spectrum to be saved (without .spo): M1_annulus
Enter filename response to be saved (without .res): M1_annulus
```

Running SPEX

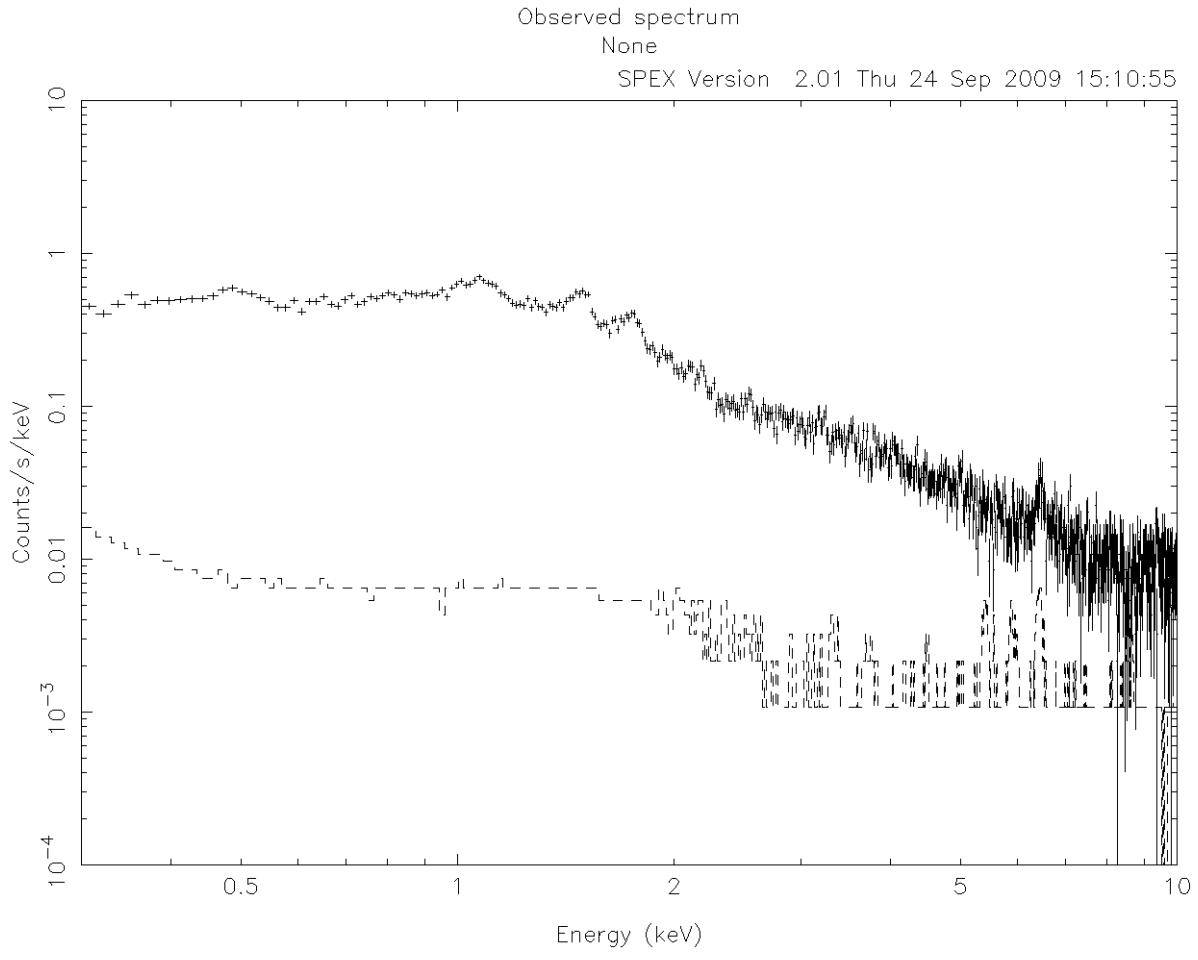


Fig. 2: XMM-Newton MOS1 spectrum extracted from a 6–9 arcmin annulus around a cluster of galaxies.

If the res and spo files are created, we are ready to run `spex`. In this description, we skip some very basic commands about, for example, plotting. See *How to run SPEX* (page 3) for an overview of a basic SPEX session. First, we load the spectrum and plot it:

```
Welcome user to SPEX version 3.00.00
```

(continues on next page)

(continued from previous page)

```
SPEX> data M1_annulus M1_annulus
...
SPEX> plot
```

Figure *XMM-Newton MOS1 spectrum extracted from a 6–9 arcmin annulus around a cluster of galaxies.* (page 34) shows a plot of the spectrum. For presentation purposes we rebin the spectrum here with the `obin` command (*Obin: optimal rebinning of the data* (page 103)). If C-statistics are used, binning is not strictly necessary. An important thing to remember at this point is to ignore the spectrum in region number 2:

```
SPEX> ign reg 2 1:1000000
```

We ignore region 2 from channel 1 to 1000000, which should be more then enough to make sure no data is left in the region. Of course, some data at very low and high energies also need to be ignored in region 1.

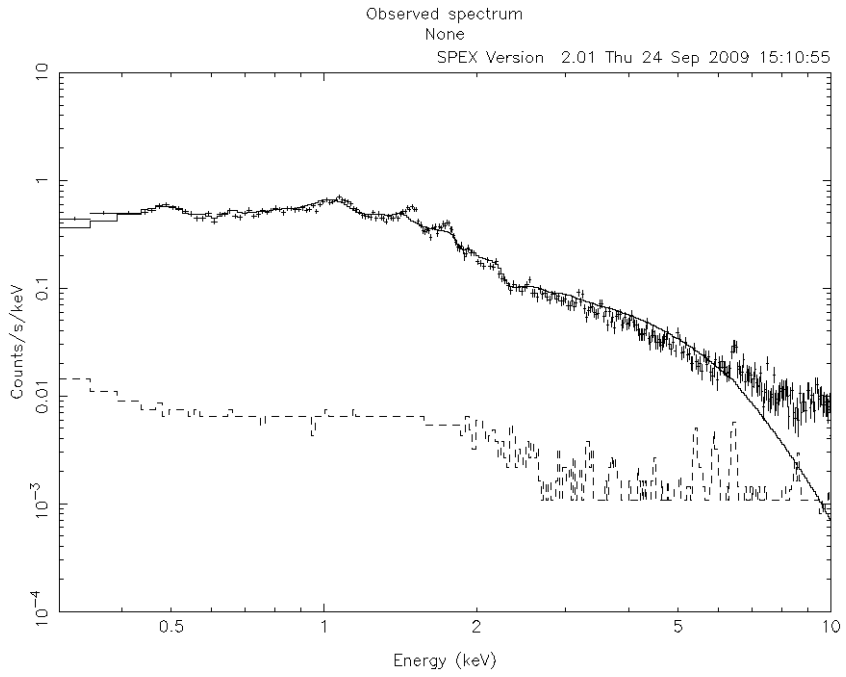


Fig. 3: A fit without modeling the particle background is not successful. Especially, the high-energy region in the spectrum is not fitted well due to soft protons.

Now, we set up the cosmic X-ray model for sector 1. We can just load the components normally, because they are automatically added to the first sector:

```
SPEX> com reds
SPEX> com abs
SPEX> com cie
SPEX> com cie
SPEX> com po
SPEX> com rel 3 1,2
SPEX> com rel 5 1,2
```

In this model, we put a cosmological redshift, interstellar absorption, and a single-temperature model to describe the cluster emission. In addition, we put a single-temperature model with a fixed temperature of 0.2 keV to model the emission from the local hot bubble, and a power law with a gamma value of 1.41 to account for the Cosmic X-ray Backgound (CXB) due to unresolved point sources.

```
SPEX> par 1 4 t v 0.2
SPEX> par 1 4 t s f
SPEX> par 1 5 gamm v 1.41
SPEX> par 1 5 gamm s f
```

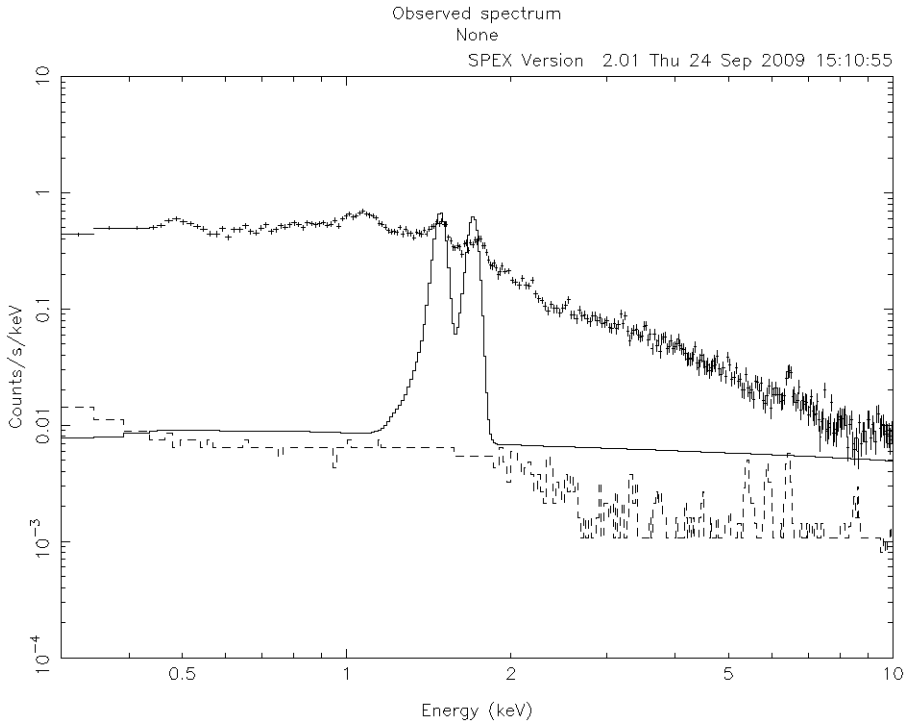


Fig. 4: Here, we plot the particle background model. We ignore the cluster model components for now. It is clear to see that the power law is not folded through the arf.

Just to show what happens if we fit the data now, we plot the result in Figure *A fit without modeling the particle background is not successful. Especially, the high-energy region in the spectrum is not fitted well due to soft protons.* (page 35). It is clear that the spectrum is not well fitted at low and high energies. A contribution of soft protons is visible at the high-energy end of the spectrum. In addition, we see that the instrumental fluoresence lines of Al and Si at ∼1.49 and ∼1.75 keV are not fitted. To model these features, we need to use the second sector and define an additional model there.

```
SPEX> sector new
SPEX> com 2 po
SPEX> com 2 delt
SPEX> com 2 delt
SPEX> par 2 1 gamm v 0.2
SPEX> par 2 2 e v 1.49
SPEX> par 2 2 e s f
SPEX> par 2 3 e v 1.75
SPEX> par 2 3 e s f
```

In this sequence of commands, we define a new sector (number 2) and add a power-law and two delta-line components to it. The slope of the gamma-parameter is initially set to ∼0.2. In Figure *Here, we plot the particle background model. We ignore the cluster model components for now. It is clear to see that the power law is not folded through the arf.* (page 36), we put the components in sector 1 to zero to show the particle background model that we have just defined. The flat shape of the power-law model confirms that these components are not folded through the arf.

When we reset the components in sector 1 to their initial values we can start fitting. In Figure *Best fit model to our example spectrum. The particle background model has been able to fit the discrepancies at high energies.* (page 37), we show the best fit using this model. The contribution of soft-protons at high energies is now being accounted for by the power law.

Warning: The example above uses a simplified model of the X-ray background. Background subtraction for extended sources is complicated and subject of continuous research. Please be very careful in selecting model

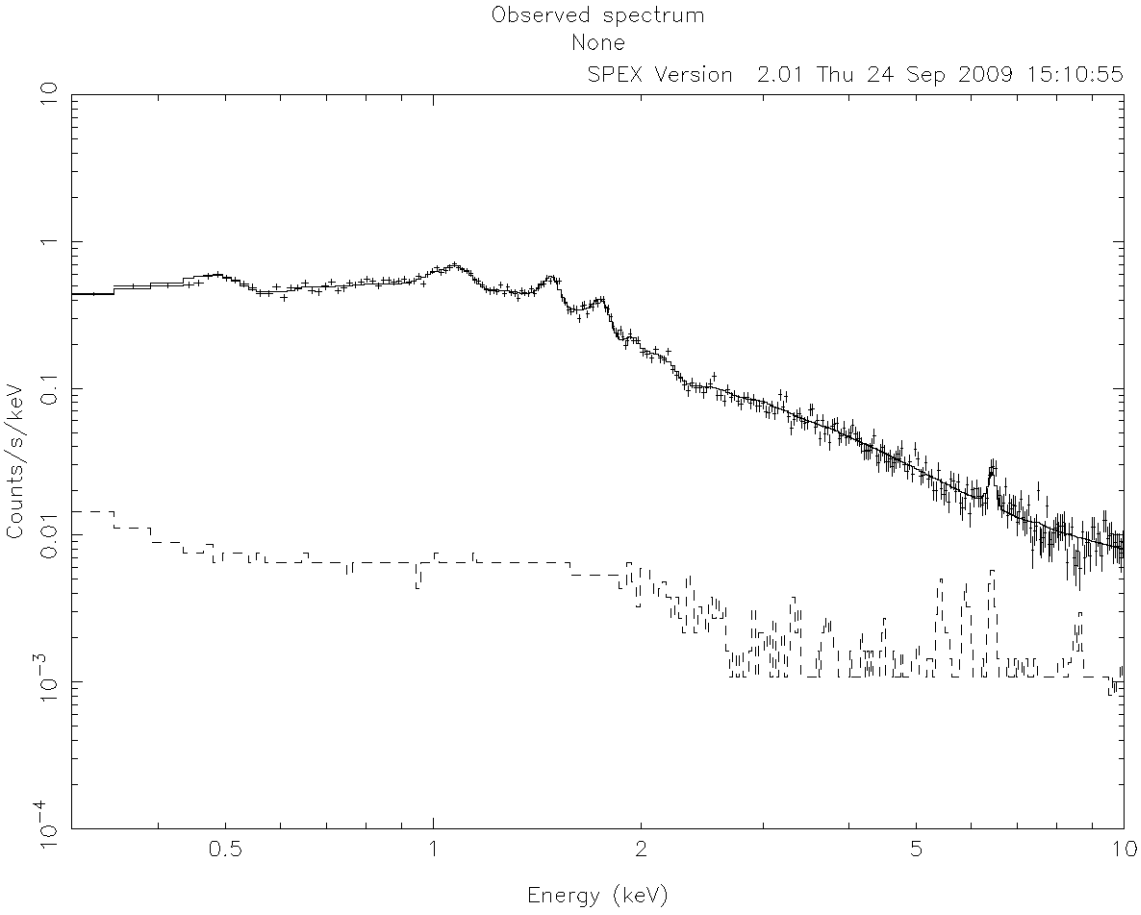


Fig. 5: Best fit model to our example spectrum. The particle background model has been able to fit the discrepancies at high energies.

components and deciding which parameters can be left free.

2.3 Fitting interstellar dust absorption

By: Daniele Rogantini

2.3.1 Goal

Characterise the extinction of the interstellar dust along the line of sight of a bright low-mass X-ray binary observed with Chandra HETG.

Note: This thread merely intends to show the fit of the magnesium and silicon K edges using the `amol` model. The simulated dataset with 250 ks exposure time is based on the model used to fit the source GX 3+1 in Rogantini et al. 2019.

2.3.2 Preparation

To follow this thread, it is necessary to download the simulated spectrum and its responsive matrix: `data_sim.spo` and `data_sim.res`.

2.3.3 Starting SPEX

Start SPEX in a linux terminal window:

```
user@linux:~> spex
Welcome user to SPEX version 3.05.00

SPEX>
```

2.3.4 Loading data

A command file tailored for this thread to load data is available here `data_gx.com`:

```
user@linux:~> cat data_gx.com

# Simulated data
#---------------
# HETG DATA
data data_sim data_sim
bin inst 1 reg 1:2 0:10000 2 unit ang
ignore inst 1 reg 1:2 0:3 unit ang
ignore inst 1 reg 1:2 11:1000 unit ang
```

Load the above command file into SPEX:

```
SPEX> log exe data_gx
```

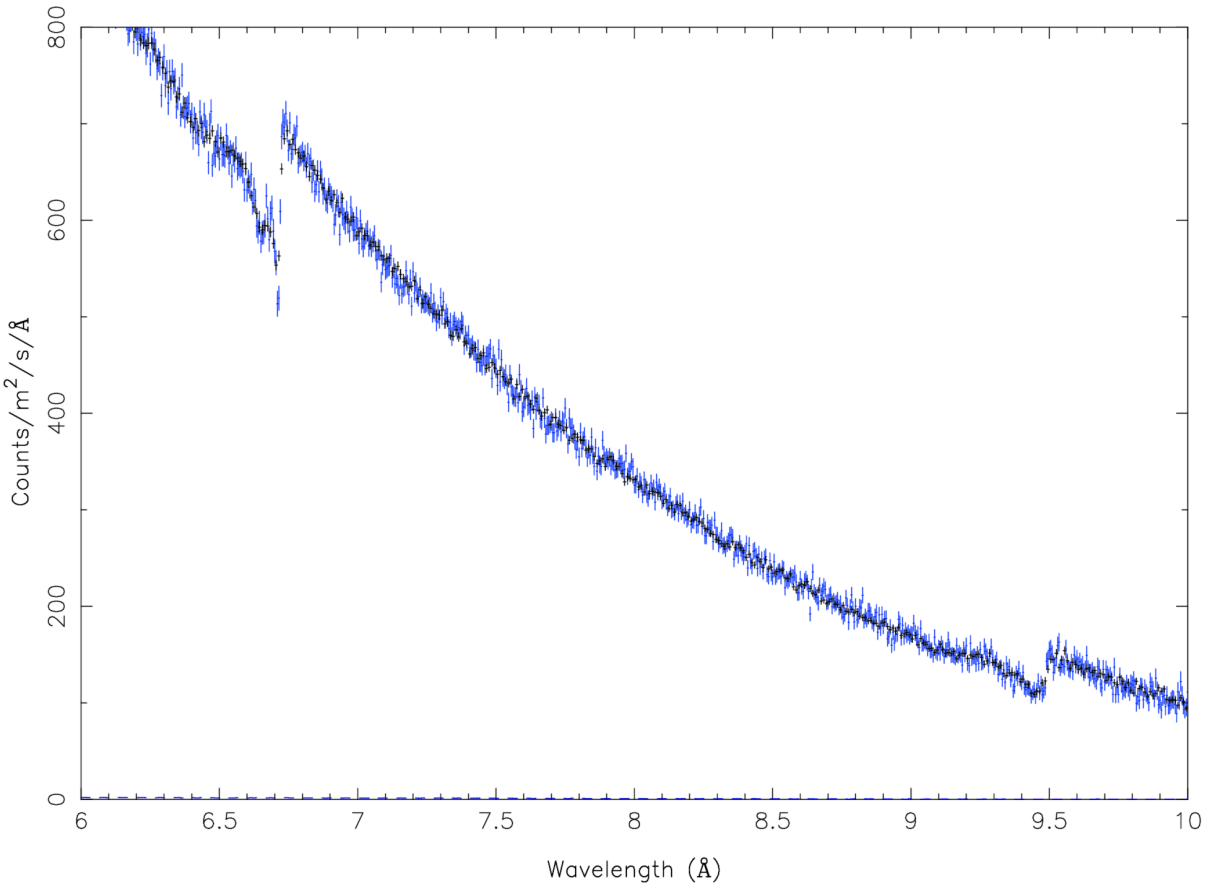
2.3.5 Plotting data

A command file tailored for this thread to plot the data is available here `plot_edges.com`:

```
user@linux:~> cat plot_edges.com
# plot setting
plot dev xw
plot type data
plot x lin
plot y lin
plot ux a
plot uy fa
plot rx 6:10
plot ry 0:900
plot set 1
# HEG color blue
plot data col 11
plot mod lw 3
plot fill disp f
plot back disp f
plot cap id disp f
plot cap ut disp f
plot cap lt disp f
plot
```

Load the above command file into SPEX:

```
SPEX> log exe plot_edges
```



2.3.6 Defining the broadband model

We are studying the interstellar dust along the line of sight of a bright low-mass X-ray binary located near the Galactic bulge (distance 6.1 kpc).

Setting the distance of the source

```
SPEX> distance 6.1 kpc
 Distances assuming H0 =  70.0 km/s/Mpc, Omega_m = 0.300 Omega_Lambda = 0.700␣
↪Omega_r = 0.000
Sector       m      A.U.        ly        pc       kpc       Mpc  redshift       ␣
↪cz   age(yr)
-------------------------------------------------------------------------------
↪-----------
   1 1.882E+20 1.258E+09 1.990E+04 6100.0000    6.1000 6.100E-03    0.0000       0.
↪4 1.990E+04
-------------------------------------------------------------------------------
↪-----------
```

Setting the SED

Set the intrinsic spectral-energy-distribution (SED) of the low-mass X-ray binary. For a typical X-ray binary, the SED between 0.1 and 10 keV is described by two components (Mitsuda et al. 1984): a thermal component, e.g. a black-body (*Bb: blackbody model* (page 141)), and a non-thermal component, e.g. a power-law (*Pow: power law model* (page 167)):

```
SPEX> com pow
 You have defined    1 component.
SPEX> par 1 1 norm value 30
SPEX> par 1 1 gamm value 1.1
SPEX> com bb
 You have defined    2 components.
SPEX> par 1 2 norm value 3.e-7
SPEX> par 1 2 t value 0.8
```

Setting the Galactic cold neutral absorption

```
SPEX> com hot
 You have defined    3 components.
SPEX> par 1 3 nh value 1.9e-2
SPEX> par 1 3 t value 8e-6
SPEX> par 1 3 t status frozen
```

2.3.7 Defining the dust absorption

Here we introduce the `amol` components (*Amol: interstellar dust absorption model* (page 137)) to characterise the interstellar dust extinction. In this example we add four arbitrary dust compounds: a-olivine (index=4230, $MgFeSiO_4$), a-quartz (index=2234, SiO_2), c-forsterite (index=3230, Mg_2SiO_4), and a-enstatite (index=3231, $MgSiO_3$). The full list of all compounds is reported in Table *Compounds list* (page 138) and Table *Additional compounds list* (page 138) in the *Amol: interstellar dust absorption model* (page 137) section of the manual.

Setting the interstellar dust models

Defining `amol` with the initial guess for the column densities of the dust compounds:

```
SPEX> com amol
 You have defined    4 components.
SPEX> par 1 4 i1 value 4230
SPEX> par 1 4 i2 value 2234
SPEX> par 1 4 i3 value 3230
SPEX> par 1 4 i4 value 3231
SPEX> par 1 4 n1 value 1e-7
SPEX> par 1 4 n2 value 1e-7
SPEX> par 1 4 n3 value 1e-7
SPEX> par 1 4 n4 value 1e-7
SPEX> par 1 4 n1 status thawn
SPEX> par 1 4 n2 status thawn
SPEX> par 1 4 n3 status thawn
SPEX> par 1 4 n4 status thawn
```

Warning: It is necessary to change and let free to vary the relative abundances of the cold gas elements (*Hot: collisional ionisation equilibrium absorption model* (page 153) in this case) which are also contained in the dust compounds. In this example, the dust models contain oxygen (`08`), magnesium (`12`), silicon (`14`) and iron (`26`). We let them to vary within a limited range according to the depletion intervals defined by Whittet et al. (2002) and Jenkins et al. (2009).

```
SPEX> par 1 3 08 value 0.7
SPEX> par 1 3 12 value 0.10
SPEX> par 1 3 14 value 0.10
SPEX> par 1 3 26 value 0.05
SPEX> par 1 3 08 range 0.4 1
SPEX> par 1 3 12 range 0 0.4
SPEX> par 1 3 14 range 0 0.4
SPEX> par 1 3 26 range 0 0.2
SPEX> par 1 3 08 status thawn
SPEX> par 1 3 12 status thawn
SPEX> par 1 3 14 status thawn
SPEX> par 1 3 26 status thawn
```

Setting the component relations

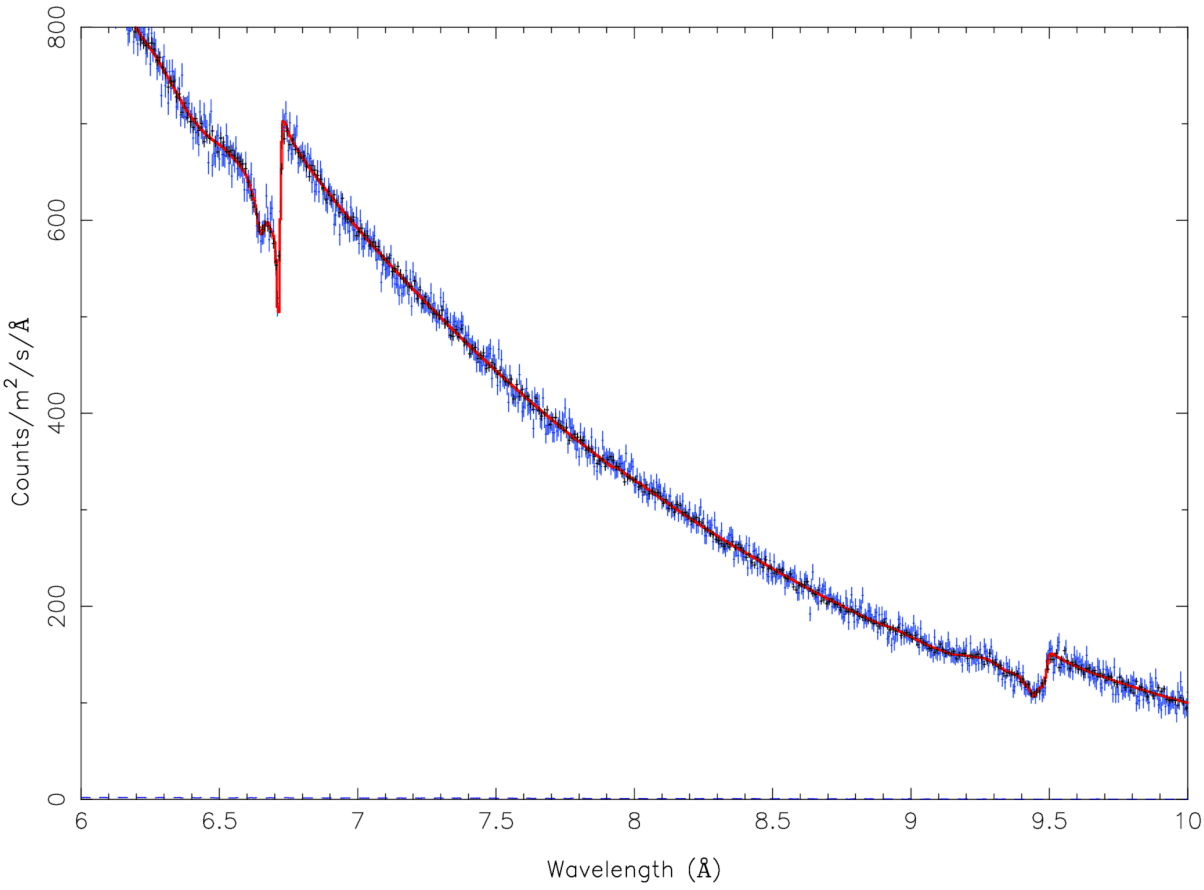
Adding the multiplicative components `hot` and `amol` to the broad-band model:

```
SPEX> com rel 1:2 4,3
SPEX> model show
-------------------------------------------------------------------------------
 Number of sectors         :     1
 Sector:     1 Number of model components:     4
    Nr.     1: pow [4,3 ]
    Nr.     2: bb  [4,3 ]
    Nr.     3: hot
    Nr.     4: amol
```

2.3.8 Fitting

We fit the model to the data and print the free parameters:

```
SPEX> calc
SPEX> fit print 1
SPEX> fit
SPEX> fit
SPEX> plot
SPEX> par show free

-------------------------------------------------------------------------------
↪---------------
sect comp mod  acro parameter with unit     value      status    minimum   maximum␣
↪lsec lcom lpar

   1    1 pow  norm Norm (1E44 ph/s/keV)  23.14066     thawn      0.0      1.00E+20
   1    1 pow  gamm Photon index         0.9320605     thawn    -10.        10.

   1    2 bb   norm Area (1E16 m**2)     3.5883755E-07 thawn      0.0      1.00E+20
   1    2 bb   t    Temperature (keV)    0.7793768     thawn    1.00E-04  1.00E+03

   1    3 hot  nh   X-Column (1E28/m**2) 2.0304110E-02 thawn      0.0      1.00E+20
   1    3 hot  08   Abundance O          0.5010648     thawn    0.40       1.0
   1    3 hot  12   Abundance Mg         0.1016048     thawn      0.0      0.40
   1    3 hot  14   Abundance Si         0.1060375     thawn      0.0      0.40
   1    3 hot  26   Abundance Fe          0.000000     thawn      0.0      0.20

   1    4 amol n1   Column 1 (1E28/m**2) 5.6286910E-07 thawn      0.0      1.00E+20
   1    4 amol n2   Column 1 (1E28/m**2) 1.1466740E-07 thawn      0.0      1.00E+20
   1    4 amol n3   Column 1 (1E28/m**2) 1.3037014E-07 thawn      0.0      1.00E+20
   1    4 amol n4   Column 1 (1E28/m**2) 9.8849377E-08 thawn      0.0      1.00E+20


Instrument     1 region    1 has norm    1.00000E+00 and is frozen
Instrument     1 region    2 has norm    1.00000E+00 and is frozen

-------------------------------------------------------------------------------
 Fluxes and restframe luminosities between   2.0000     and    10.000     keV

 sect comp mod   photon flux   energy flux nr of photons    luminosity
              (phot/m**2/s)      (W/m**2)   (photons/s)          (W)
    1    1 pow    7877.93      6.754451E-12  4.125361E+45  3.331746E+30
    1    2 bb     3030.35      1.681985E-12  1.818740E+45  9.538203E+29

-------------------------------------------------------------------------------
 Fit method        : Classical Levenberg-Marquardt
 Fit statistic     : C-statistic
 C-statistic       :      2388.02
 Expected C-stat   :      2402.60 +/-        69.35
 Chi-squared value :      2406.15
 Degrees of freedom:      2388
 W-statistic       :         0.00
```



2.3.9 Final remarks

This is the end of this analysis thread. If you want, you can save the parameters and quit SPEX:

```
SPEX> par write parameters
SPEX> log out fit_result
SPEX> par show
SPEX> log close output
    SPEX> quit
     Thank you for using SPEX!
```

2.4 Import UV/Optical data

For some applications it would be helpful to include non-X-ray data into the spectral fit. For example, this can be an optical/UV spectrum that the user wants to fit simultaneously with the X-ray spectrum using the models of SPEX.

For some instruments, namely the photometric filters of XMM-Newton's OM and Swift's UVOT, the data products are available in the same format as the X-ray data (i.e. the OGIP-standard PHA and response files). Therefore, for OM and UVOT filters, the user is recommended to convert the PHA and response files to SPEX format using TRAFO as normally done for the X-ray data.

Information about the instrumental response files of OM and UVOT filters can be found here:

https://www.cosmos.esa.int/web/xmm-newton/om-response-files

https://swift.gsfc.nasa.gov/proposals/swift_responses.html

However, for some other instruments the user may not know, or have access, to the counts and instrumental response information. For example, the user obtained optical/UV fluxes from NED or a table in a paper. In such

cases the user is recommended to first use the FTOOLS program called `ftflx2xsp` to convert their data to the OGIP-standard PHA and response files. The webpage of `ftflx2xsp` provides all the instructions with some useful examples:

https://heasarc.nasa.gov/lheasoft/ftools/fhelp/ftflx2xsp.html

This program reads a text file containing the spectrum with the specified units, and creates the corresponding PHA and RSP files. These files can be then converted to SPEX format using TRAFO as usual.

The SPEX equivalent of `ftflx2xsp` is *Uvtospex* (page 193). Please follow the link to find a brief manual for this tool. It creates a SPEX format spectrum and response matrix based on the fluxes as function of wavelength stored in a text file. The instrument resolution is needed in units of km/s (FWHM).

Note: There are some OM response matrices around that do not conform to the OGIP standard. The extension of the response matrix is then called 'SPECTRESP MATRIX' instead of the standard 'SPECRESP MATRIX'. `trafo` and `ogip2spex` will exit with an error when this name error is encountered. This can be solved by renaming the extension to 'SPECRESP MATRIX' in the rmf file. This can be done, for example, by using the HEASOFT task `fv`.

2.5 PION setup for AGN warm absorber

By: Junjie Mao, Missagh Mehdipour, and Jelle Kaastra

2.5.1 Goal

Setup the PION model for the warm absorber in a nearby Seyfert 1 galaxy observed with Chandra HRC/LETGS.

Note: A simulated spectrum was used because this thread merely intends to show the setup of the PION model.

2.5.2 Preparation

To follow this thread, you need to download the example files here: `chl.spo` and `chl.res`.

2.5.3 Start SPEX

Start SPEX in a linux terminal window:

```
user@linux:~> spex
 Welcome user to SPEX version 3.05.00

SPEX>
```

2.5.4 Load data

A command file tailored for this thread to load data is available here `data.com`

```
user@linux:~> cat data.com
# Simulated data
#---------------
# HRC/LETGS DATA
data chl chl
bin inst 1 reg 1 0:10000 2 unit ang
ignore inst 1 reg 1 0:1.5 unit ang
ignore inst 1 reg 1 60:1000 unit ang
```

Load the above command file into SPEX:

```
SPEX> log exe data
```

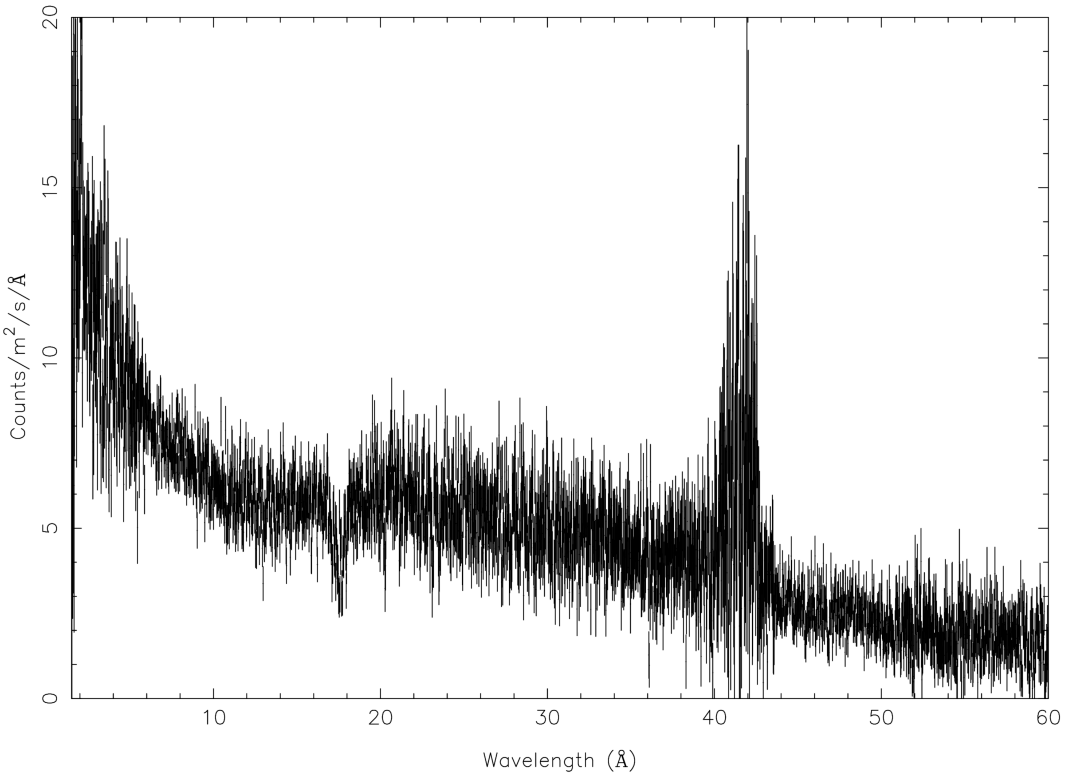
2.5.5 Plot data

A command file tailored for this thread to plot the data is available here `plot.com`

```
user@linux:~> cat plot.com
# plot setting
plot dev xw
plot type data
plot x lin
plot ux a
plot rx 1.5:60
plot y lin
plot uy fa
plot ry 0:20
plot set 1
plot mo lw 3
plot fill disp f
plot back disp f
plot cap id disp f
plot cap ut disp f
plot cap lt disp f
plot
```

Load the above command file into SPEX:

```
SPEX> log exe plot
```



2.5.6 Define model components and component relations (step-by-step)

Here we are looking at the warm absorber in a nearby (z = 0.07) Seyfert 1 galaxy.

Set the distance of the source

```
SPEX> dist 0.07 z
Distances assuming H0 =  70.0 km/s/Mpc, Omega_m = 0.300 Omega_Lambda = 0.700 Omega_
↪r = 0.000
Sector       m      A.U.        ly        pc       kpc       Mpc  redshift
↪cz   age(yr)
-------------------------------------------------------------------------------
↪-----------
1 9.740E+24 6.511E+13 1.030E+09 3.157E+08 3.157E+05  315.6554    0.0700   20985.5
↪9.302E+08
-------------------------------------------------------------------------------
↪-----------
SPEX> com reds
You have defined    1 component.
SPEX> par 1 1 z val 0.07
```

Set the redshift component

```
SPEX> com reds
You have defined    1 component.
SPEX> par 1 1 z val 0.07
```

Set the galactic absorption

```
SPEX> com hot
You have defined    2 components.
SPEX> par 1 2 nh val 2.0e-4
```

(continued from previous page)

```
SPEX> par 1 3 t1 s f
SPEX> par 1 3 tau val 20
SPEX> par 1 3 tau s f
SPEX> com pow
You have defined    4 components.
SPEX> par 1 4 norm val 1.E+09
SPEX> par 1 4 norm s t
SPEX> par 1 4 gamm val 1.7
SPEX> par 1 4 gamm s t
SPEX> com refl
You have defined    5 components.
SPEX> par 1 5 norm couple 1 4 norm
SPEX> par 1 5 gamm couple 1 4 gamm
SPEX> par 1 5 ecut val 300
SPEX> par 1 5 ecut s f
SPEX> par 1 5 pow:fgr v 0
SPEX> par 1 5 scal val 1.
SPEX> par 1 5 scal s f
```

Apply an exponential cut-off to the power-law

Apply exponential cut-off to the power-law component of the SED both below the Lyman limit and above the high-energy cut-off.

Note: The `ecut` parameter in the `refl` component applies to itself only.

```
SPEX> com etau
You have defined    6 components.
SPEX> par 1 6 a val -1
SPEX> par 1 6 a s f
SPEX> par 1 6 tau val 1.3605E-2
SPEX> par 1 6 tau s f
SPEX> com etau
You have defined    7 components.
SPEX> par 1 7 a val 1
SPEX> par 1 7 a s f
SPEX> par 1 7 tau val 3.3333E-3
SPEX> par 1 7 tau s f
```

Set the PION (absorption) components

Here we introduce three PION components (*Pion: SPEX photoionised plasma model* (page 163)). The parameters of the PION components are restricted to improve the efficiency of a realistic fitting process. `fcov=1` refers to the PION component fully covers the line-of-sight. `omeg=1.E-7` refers to a negligible solid angle (Ω) subtended by the PION component with respect to the nucleus (omeg = $\Omega/4\pi$).

Note: The third `pion` component is a spare one with `fcov=0` and `omeg=0`. This is practical when analyzing real data without any prior knowledge of the number of PION components required.

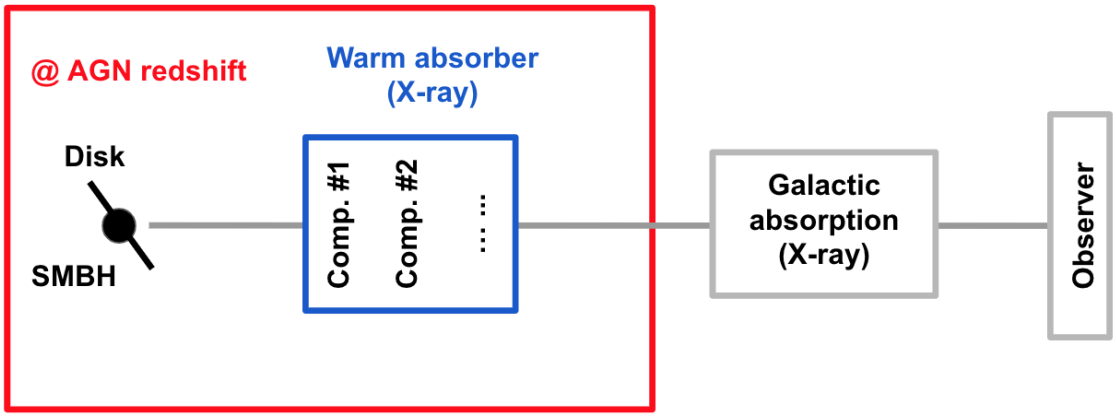
Note: To see the density effect of the absorption features, it is necessary to set a non-zero `omeg` value.

```
SPEX> com pion
You have defined    8 components.
** Pion model: take care about proper COM REL use: check manual!
SPEX> com pion
You have defined    9 components.
** Pion model: take care about proper COM REL use: check manual!
SPEX> com pion
You have defined    10 components.
** Pion model: take care about proper COM REL use: check manual!
SPEX> par 1 8:10 nh range 1.E-7:1.E1
SPEX> par 1 8:10 xil range -5:5
SPEX> par 1 8:10 omeg range 0:1
SPEX> par 1 8 nh val 5.E-03
SPEX> par 1 8 xil val 2.7
SPEX> par 1 8 zv val -500
SPEX> par 1 8 zv s t
SPEX> par 1 8 v val 100
SPEX> par 1 8 v s t
SPEX> par 1 8 omeg val 1.E-7
SPEX> par 1 9 nh val 2.E-03
SPEX> par 1 9 xil val 1.6
SPEX> par 1 9 zv val -100
SPEX> par 1 9 zv s t
SPEX> par 1 9 v val 50
SPEX> par 1 9 v s t
SPEX> par 1 9 omeg val 1.E-7
SPEX> par 1 10 nh val 1.E-7
SPEX> par 1 10 xil val 0
SPEX> par 1 10 fcov val 0
SPEX> par 1 10 omega val 0
```

Set component relation along our line of sight

Set the component relation for the intrinsic AGN SED above the Lyman limit along our line-of-sight.

Note: Photons from both the Comptonized disk and power-law components are screened by the warm absorber components at the redshift of the target, as well as the galactic absorption before reaching the detector. Photons from the neutral reflection component is assumed not to be screened by the warm absorber for simplicity. It is still redshifted and requires the galactic absorption.



```
SPEX> com rel 3 8,9,10,1,2
SPEX> com rel 4 6,7,8,9,10,1,2
SPEX> com rel 5 1,2
```

Set the component relation for the PION components

Assuming that the warm absorber components closer to the central engine are defined first (with a smaller component index), photons transmitted from the inner PION components (with a nonzero `omeg` value) are screened by all the outer PION components at the redshift of the target, as well as the galactic absorption before reaching the detector:

```
SPEX> com rel 8 9,10,1,2
SPEX> com rel 9 10,1,2
SPEX> com rel 10 1,2
```

Check the model settings and calculate

We check the setting of the component relation:

```
SPEX> model show
-------------------------------------------------------------------------------
Number of sectors         :     1
Sector:    1 Number of model components:    10
Nr.    1: reds
Nr.    2: hot
Nr.    3: comt[8,9,10,1,2 ]
Nr.    4: pow [6,7,8,9,10,1,2 ]
Nr.    5: refl[1,2 ]
Nr.    6: etau
Nr.    7: etau
Nr.    8: pion[9,10,1,2 ]
Nr.    9: pion[10,1,2 ]
Nr.   10: pion[1,2 ]
```

We check the setting of the free parameters and calculate the 1–1000 Ryd ionizing luminosity:

```
SPEX> elim 1.E0:1.E3 ryd
SPEX> calc
SPEX> plot
SPEX> par show free
-------------------------------------------------------------------------------
 ↪---------------
sect comp mod  acro parameter with unit     value      status    minimum   maximum␣
 ↪lsec lcom lpar



   1    3 comt norm Norm (1E44 ph/s/keV) 3.0000001E+12 thawn     0.0      1.00E+20
   1    3 comt t0   Wien temp (keV)      5.0000002E-04 thawn    1.00E-05  1.00E+10
   1    3 comt t1   Plasma temp (keV)    0.1500000     thawn    1.00E-05  1.00E+10
   1    3 comt tau  Optical depth         20.00000     thawn    1.00E-03  1.00E+03

   1    4 pow  norm Norm (1E44 ph/s/keV) 1.0000000E+09 thawn     0.0      1.00E+20
   1    4 pow  gamm Photon index          1.700000     thawn    -10.       10.




   1    8 pion nh   X-Column (1E28/m**2) 4.9999999E-03 thawn    1.00E-07   10.
```

(continues on next page)

(continued from previous page)

```
   1    8 pion xil  Log xi (1E-9 Wm)       2.700000     thawn    -5.0        5.0
   1    8 pion v    RMS Velocity (km/s)    100.0000     thawn     0.0       3.00E+05
   1    8 pion zv   Average vel. (km/s)   -500.0000     thawn   -1.00E+05  1.00E+05

   1    9 pion nh   X-Column (1E28/m**2) 2.0000001E-03 thawn    1.00E-07   10.
   1    9 pion xil  Log xi (1E-9 Wm)       1.600000     thawn    -5.0        5.0
   1    9 pion v    RMS Velocity (km/s)    50.00000     thawn     0.0       3.00E+05
   1    9 pion zv   Average vel. (km/s)   -100.0000     thawn   -1.00E+05  1.00E+05



Instrument     1 region    1 has norm    1.00000E+00 and is frozen

-------------------------------------------------------------------------------
 Fluxes and restframe luminosities between  1.36057E-02 and    13.606     keV

 sect comp mod   photon flux   energy flux nr of photons    luminosity
              (phot/m**2/s)      (W/m**2)   (photons/s)           (W)
    1    3 comt    9.90871      4.310500E-16  1.447224E+54  7.988849E+36
    1    4 pow     243.883      6.246949E-14  2.869709E+54  1.021577E+38
    1    5 refl    5.98565      7.190691E-15  6.284842E+51  7.467485E+36
    1    8 pion  1.711172E-07  3.011339E-23  3.540241E+45  8.248552E+28
    1    9 pion  1.753678E-06  1.559744E-22  1.947966E+47  1.168208E+30
    1   10 pion   0.00000       0.00000       0.00000       0.00000

 Fit method        : Classical Levenberg-Marquardt
 Fit statistic     : C-statistic
 C-statistic       :      2424.54
 Expected C-stat   :      2348.67 +/-        68.66
 Chi-squared value :      2531.73
 Degrees of freedom:         0
 W-statistic       :      2353.74
```

2.5.7 Final remarks

This is the end of this analysis thread. If you want, you can quit SPEX now:

```
SPEX> quit
Thank you for using SPEX!
```

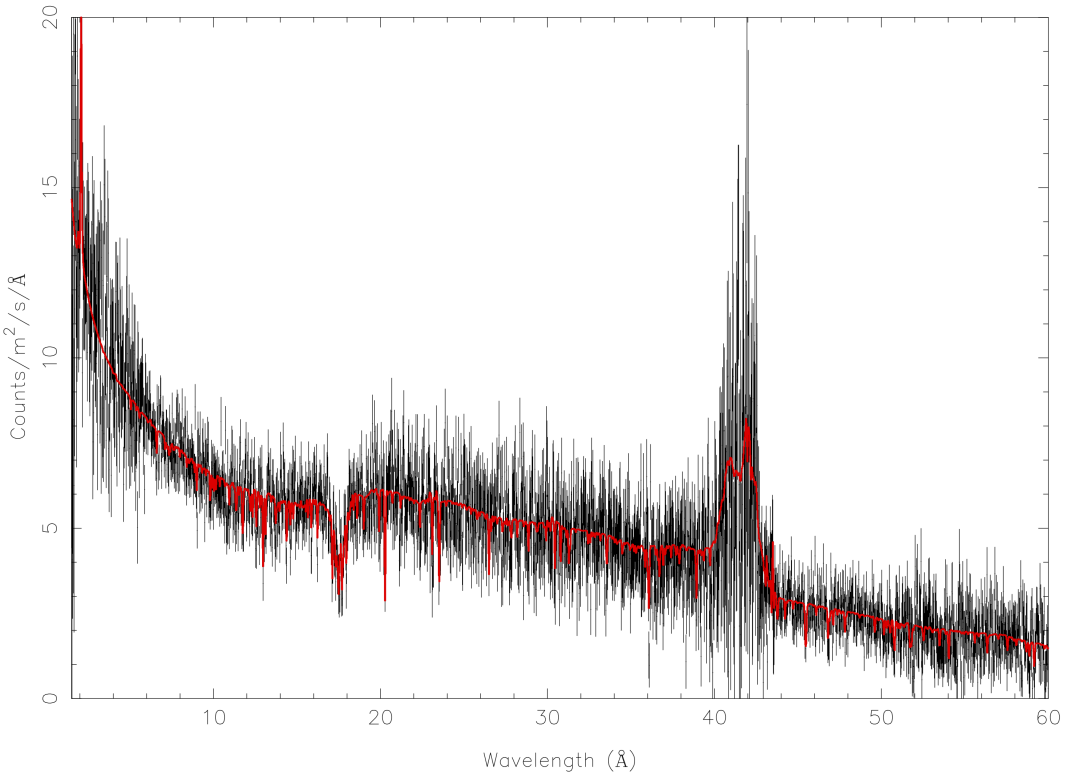
Below, we provide a useful command file.

Define model components and component relations (running scripts)

A command file tailored for this thread to setup the model components and parameters is available here `mdl_pa.com`.

Load the above command file into SPEX:

```
SPEX> log exe mdl_pa
```



2.6 PION setup for emission and absorption features in AGN

By: Junjie Mao, Missagh Mehdipour, and Jelle Kaastra

2.6.1 Goal

Setup the PION model (*Pion: SPEX photoionised plasma model* (page 163)) for the emission and absorption features in a nearby Seyfert 1 galaxy observed with XMM-Newton (OM, RGS, and EPIC-pn).

Note: A simulated spectrum was used because this thread merely intends to show the setup of the PION model.

2.6.2 Preparation

To follow this thread, you need to download the example files here: `pionena.tar.gz`:

```
user@linux:~> cd /path/to/your/folder/
user@linux:~> mv ~/Downloads/pionena.tar.gz ./
user@linux:~> tar -xvf pionena.tar.gz
user@linux:~> cd pionena
```

2.6.3 Start SPEX

Start SPEX in a linux terminal window:

```
user@linux:~> spex
 Welcome user to SPEX version 3.05.00

SPEX>
```

2.6.4 Load data

`data.com` is the command file tailored for this thread to load data:

```
user@linux:~> cat data.com
# RGS (inst 1)
data rgs rgs
# EPIC-pn (inst 2)
data pn pn
# OM (inst 3:8)
data om_UVW2 om_UVW2
data om_UVM2 om_UVM2
data om_UVW1 om_UVW1
data om_U om_U
data om_B om_B
data om_V om_V
# ign/use, binning
# RGS (inst 1)
bin inst 1 reg 1 0:1000 2 unit ang
ign inst 1 reg 1 0:7 unit ang
ign inst 1 reg 1 37:1000 unit ang
# EPIC-pn (inst 2)
obin inst 2 reg 1 1:100000
ign inst 2 reg 1 0:0.3 unit kev
ign inst 2 reg 1 8:1000 unit ang
ign inst 2 reg 1 10:1000 unit kev
```

Load the above command file into SPEX:

```
SPEX> log exe data
```

2.6.5 Plot data

`plot.com` is the command file tailored for this thread to plot data:

```
user@linux:~> cat plot.com
# plotting
plot dev xw
plot type data
plot ux a
plot uy fa
plot x log
plot rx 1.2E0 1.E4
plot y lin
plot ry 0 50
plot set 1
plot mo lw 3
plot set 2
plot da col 11
plot mo col 3
```

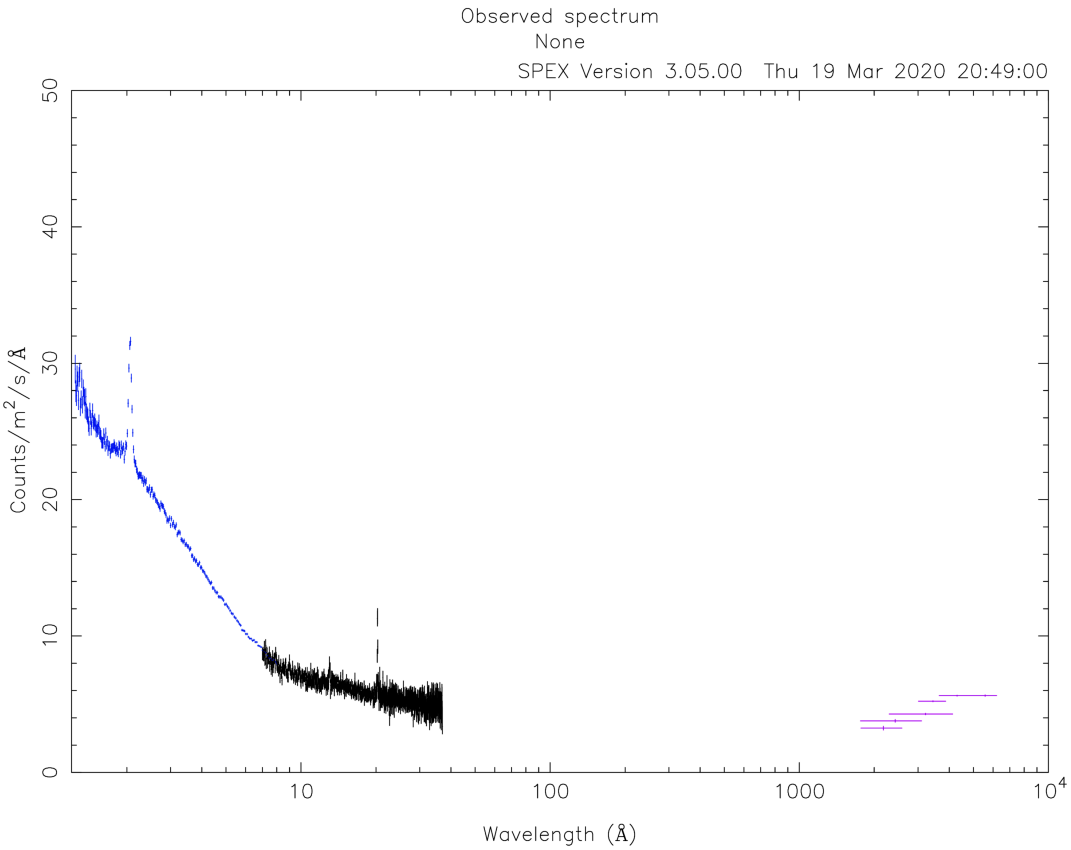
(continues on next page)

(continued from previous page)

```
plot mo lw 3
plot set 3:8
plot da col 6
plot set all
plot back disp f
plot
```

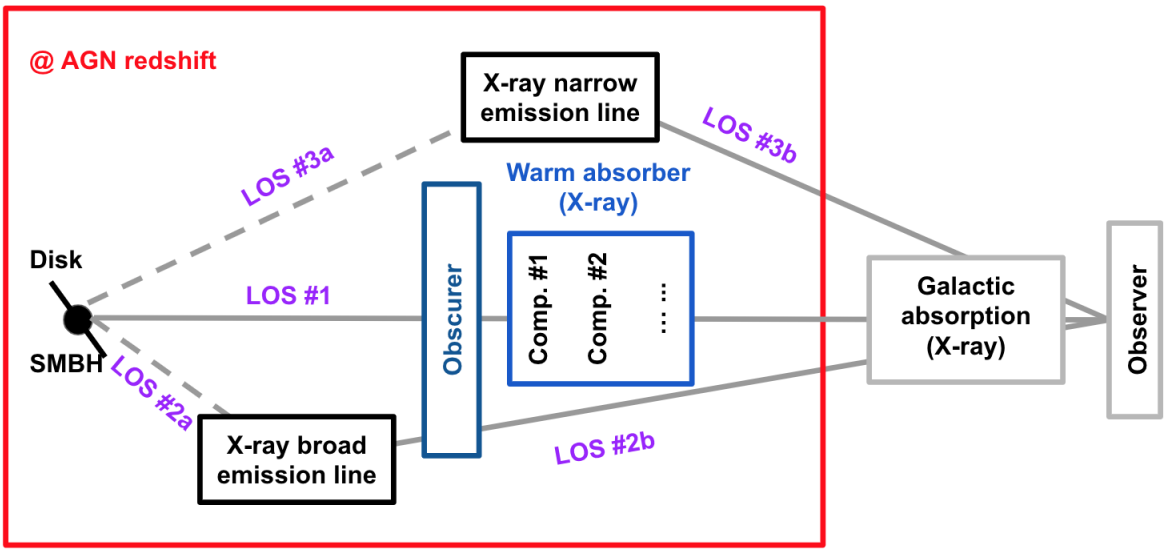
Load the above command file into SPEX:

```
SPEX> log exe plot
```



2.6.6 Define model components and component relations (step-by-step)

Here we are receiving photons from three line-of-sights in a nearby (z = 0.07) Seyfert 1 galaxy.



Set the distance of the source

```
SPEX> dist 0.07 z
Distances assuming H0 =  70.0 km/s/Mpc, Omega_m = 0.300 Omega_Lambda = 0.700 Omega_
↪r = 0.000
Sector       m      A.U.        ly        pc       kpc       Mpc  redshift       ␣
↪cz   age(yr)
-------------------------------------------------------------------------------------
↪-----------
1 9.740E+24 6.511E+13 1.030E+09 3.157E+08 3.157E+05  315.6554    0.0700   20985.5␣
↪9.302E+08
-------------------------------------------------------------------------------------
↪-----------
```

Set the redshift component

```
SPEX> com reds
You have defined    1 component.
SPEX> par 1 1 z val 0.07
```

Set the galactic absorption

```
SPEX> com hot
You have defined    2 components.
SPEX> par 1 2 nh val 2.0e-4
SPEX> par 1 2 t val 8E-6
SPEX> par 1 2 t s f
SPEX> par 1 2 nh s f
```

Set the components and component relations for line-of-sight #1

(A) Set the intrinsic spectral-energy-distribution (SED) of the AGN above the Lyman limit along line-of-sight #1.

For a typical Seyfert 1 galaxy, the SED has three components (Mehdipour et al. 2015):

- A Comptonized disk component (*Comt: comptonisation model* (page 145)) for optical to soft X-rays data
- A power-law component (*Pow: power law model* (page 167)) for X-ray data
- A neutral reflection component (*Refl: reflection model* (page 169)) for hard X-rays data. Usually, the reflection component has an exponential cut-off energy (300 keV here).

```
SPEX> com comt
You have defined    3 components.
SPEX> par 1 3 norm val 0.
SPEX> par 1 3 norm s f
SPEX> par 1 3 t0 val 5e-4
SPEX> par 1 3 t0 s f
SPEX> par 1 3 t1 val 0.15
SPEX> par 1 3 t1 s f
SPEX> par 1 3 tau val 20
SPEX> par 1 3 tau s f
SPEX> com pow
You have defined    4 components.
SPEX> par 1 4 norm val 1.E+09
SPEX> par 1 4 norm s t
SPEX> par 1 4 gamm val 1.7
SPEX> par 1 4 gamm s t
SPEX> com refl
You have defined    5 components.
SPEX> par 1 5 norm couple 1 4 norm
SPEX> par 1 5 gamm couple 1 4 gamm
SPEX> par 1 5 ecut val 300
SPEX> par 1 5 ecut s f
SPEX> par 1 5 pow:fgr v 0
SPEX> par 1 5 scal val 1.
SPEX> par 1 5 scal s f
```

(B) Apply exponential cut-off to the power-law component of the SED both below the Lyman limit and above the high-energy cut-off.

Note: The `ecut` parameter in the `refl` component applies to itself only.

```
SPEX> com etau
You have defined    6 components.
SPEX> par 1 6 a val -1
SPEX> par 1 6 a s f
SPEX> par 1 6 tau val 1.3605E-2
SPEX> par 1 6 tau s f
SPEX> com etau
You have defined    7 components.
SPEX> par 1 7 a val 1
SPEX> par 1 7 a s f
SPEX> par 1 7 tau val 3.3333E-3
SPEX> par 1 7 tau s f
```

(C) Set the PION (obscuring wind) components.

Here we introduce two PION components for the obscuring wind (Kaastra et al. 2014). The parameters of the PION components are restricted to improve the efficiency of a realistic fitting process.

Note: The second `pion` component is a spare one with `fcov=0` and `omeg=0`. This is practical when analyzing real data without any prior knowledge of the number of PION components required.

```
SPEX> com pion
You have defined    8 components.
** Pion model: take care about proper COM REL use: check manual!
SPEX> com pion
You have defined    9 components.
** Pion model: take care about proper COM REL use: check manual!
SPEX> par 1 8:9 nh range 1.E-7:1.E1
SPEX> par 1 8:9 xil range -5:5
SPEX> par 1 8 nh val 5.E-02
SPEX> par 1 8 xil val 0.0
SPEX> par 1 8 zv val -3000
SPEX> par 1 8 zv s t
SPEX> par 1 8 v val 1100
SPEX> par 1 8 v s t
SPEX> par 1 9 nh val 1.E-7
SPEX> par 1 9 nh s f
SPEX> par 1 9 xil val 0
SPEX> par 1 9 xil s f
SPEX> par 1 9 fcov val 0
SPEX> par 1 9 omega val 0
```

(D) Set the PION (warm absorber) components.

Here we introduce three PION components for the X-ray warm absorber. `omeg=1.E-7` refers to a negligible solid angle (Ω) subtended by the PION component with respect to the nucleus (omeg = $\Omega/4\pi$).

Note: To see the density effect of the absorption features, it is necessary to set a non-zero `omeg` value.

```
SPEX> com pion
You have defined    10 components.
** Pion model: take care about proper COM REL use: check manual!
SPEX> com pion
You have defined    11 components.
** Pion model: take care about proper COM REL use: check manual!
SPEX> com pion
You have defined    12 components.
** Pion model: take care about proper COM REL use: check manual!
SPEX> par 1 10:12 nh range 1.E-7:1.E1
SPEX> par 1 10:12 xil range -5:5
SPEX> par 1 10:12 omeg range 0:1
SPEX> par 1 10 nh val 5.E-03
SPEX> par 1 10 xil val 2.7
SPEX> par 1 10 zv val -500
SPEX> par 1 10 zv s t
SPEX> par 1 10 v val 100
SPEX> par 1 10 v s t
SPEX> par 1 10 omeg val 1.E-7
SPEX> par 1 11 nh val 2.E-03
SPEX> par 1 11 xil val 1.6
SPEX> par 1 11 zv val -100
SPEX> par 1 11 zv s t
SPEX> par 1 11 v val 50
SPEX> par 1 11 v s t
SPEX> par 1 11 omeg val 1.E-7
SPEX> par 1 12 nh val 1.E-7
SPEX> par 1 12 xil val 0
```

(continues on next page)

(continued from previous page)

```
SPEX> par 1 12 fcov val 0
SPEX> par 1 12 omega val 0
```

(E) Set the component relation for line-of-sight #1.

Note: Photons from both the Comptonized disk and power-law components are screened by the obscuring wind and warm absorber components at the redshift of the target, as well as the galactic absorption before reaching the detector. Photons from the neutral reflection component is assumed not to be screened by the obscuring wind and warm absorber for simplicity. It is still redshifted and requires the galactic absorption.

```
SPEX> com rel 3 8,9,10,11,12,1,2
SPEX> com rel 4 6,7,8,9,10,11,12,1,2
SPEX> com rel 5 1,2
```

(F) Set the component relation for the PION components. Assuming that the obscuring wind and warm absorber components closer to the central engine are defined first (with a smaller component index), photons transmitted from the inner PION components (with a nonzero `omeg` value) are screened by all the outer PION components at the redshift of the target, as well as the galactic absorption before reaching the detector.

```
SPEX> com rel 8 9,10,11,12,1,2
SPEX> com rel 9 10,11,12,1,2
SPEX> com rel 10 11,12,1,2
SPEX> com rel 11 12,1,2
SPEX> com rel 12 1,2
```

Set the components and component relations for line-of-sights #2 and #3

(A) Set the AGN SED above the Lyman limit along line-of-sights #2a and #3a.

Note: Here we assume that the photoionizing SED for the X-ray broad emission PION component(s) is set to be the same as that for the obscuring wind and warm absorber. This simplification assumes that the X-ray broad-line region respond to the photoionizing SED instantaneously. Because the X-ray broad-line region is typically a few lightdays away from the central engine and it has a relatively high density. On the other hand, the photoionizing SED for the X-ray narrow emission PION component(s) is set to a long-term averaged SED. This simplification assumes that the X-ray narrow-line region is in a steady state, i.e. it varies slightly around a mean value corresponding to the mean flux level over time. Because the X-ray narrow-line region is typically a few parsecs away from the central engine and it has a relatively low density. Readers are referred to Silva et al. 2016 for a detailed spectral timing study.

```
SPEX> com comt
You have defined    13 components.
SPEX> par 1 13 norm:type couple 1 3 norm:type
SPEX> com pow
You have defined    14 components.
SPEX> par 1 14 norm:lum couple 1 4 norm:lum
SPEX> com comt
You have defined    15 components.
SPEX> par 1 15 norm val 1.E12
SPEX> par 1 15 norm s f
SPEX> par 1 15 t0 val 3.E-4
SPEX> par 1 15 t0 s f
SPEX> par 1 15 t1 val 0.125
SPEX> par 1 15 t1 s f
SPEX> par 1 15 tau val 20
SPEX> par 1 15 tau s f
```

(continues on next page)

(continued from previous page)

```
SPEX> com pow
You have defined    16 components.
SPEX> par 1 16 norm val 6.E9
SPEX> par 1 16 norm s f
SPEX> par 1 16 gamm val 1.6
SPEX> par 1 16 gamm s f
```

(B) Apply exponential cut-off to the above AGN SEDs at all energies because these photons do not reach us (dashed gray lines in Figure 1).

```
SPEX> com etau
You have defined    17 components.
SPEX> par 1 17 tau val 1.E3
SPEX> par 1 17 tau s f
SPEX> par 1 17 a val 0
SPEX> par 1 17 a s f
```

(C) Set the PION (emission) components.

Here we introduce three PION components. The parameters of the PION components are restricted to improve the efficiency of a realistic fitting process. `fcov=0` for the emission PION components.

Note: The first `pion` component refers to the X-ray broad-line region. The second `pion` component refers to the X-ray narrow-line region. The third `pion` component is a spare one with `fcov=0` and `omeg=0`. This is practical when analyzing real data without any prior knowledge of the number of PION components required.

```
SPEX> com pion
You have defined    18 components.
** Pion model: take care about proper COM REL use: check manual!
SPEX> com pion
You have defined    19 components.
** Pion model: take care about proper COM REL use: check manual!
SPEX> com pion
You have defined    20 components.
** Pion model: take care about proper COM REL use: check manual!
SPEX> par 1 16:18 nh range 1.E-7:1.E1
SPEX> par 1 16:18 xil range -5:5
SPEX> par 1 16:18 omeg range 0:1
SPEX> par 1 16 nh val 8.E-02
SPEX> par 1 16 xil val 0.8
SPEX> par 1 16 zv val 0
SPEX> par 1 16 zv s f
SPEX> par 1 16 v val 100
SPEX> par 1 16 v s f
SPEX> par 1 16 omeg val 3.E-2
SPEX> par 1 16 omeg s t
SPEX> par 1 17 nh val 5.E-02
SPEX> par 1 17 xil val 2.3
SPEX> par 1 17 zv val 0
SPEX> par 1 17 zv s f
SPEX> par 1 17 v val 240
SPEX> par 1 17 v s t
SPEX> par 1 17 omeg val 5.E-2
SPEX> par 1 17 omeg s t
SPEX> par 1 18 nh val 1.E-7
SPEX> par 1 18 nh s f
SPEX> par 1 18 xil val 0
SPEX> par 1 18 xil s f
SPEX> par 1 18 fcov val 0
SPEX> par 1 18 omeg val 0
```

(D) Set the broadening due to macroscopic motion for the PION (emission) components.

Note: The `v` parameter in PION components refer to the microscopic (i.e. turbulent) motion. The macroscopic motion refers to the rotation around the black hole. For the X-ray broad emission lines, the macroscopic motion dominates the broadening. For the X-ray narrow emission lines, the microscopic and macroscopic motion are often degenerate (Mao et al. 2018). The second and third `vgau` components are spare.

```
SPEX> com vgau
You have defined    21 components.
par 1 21 sig val 7.E3
par 1 21 sig s t
SPEX> com vgau
You have defined    22 components.
SPEX> com vgau
You have defined    23 components.
```

(E) Set the component relation for line-of-sights #2a and #3a.

Note: Photons from both the Comptonized disk and power-law (with exponential low- and high-energy cut-offs) components are the photoionizing source of the PION emission components at the redshift of the target. While (reflected/reprocessed) photons from the PION emission components reach us.

```
SPEX> com rel 13 18,1,17
SPEX> com rel 14 6,7,18,1,17
SPEX> com rel 15 19,20,1,17
SPEX> com rel 16 6,7,19,20,1,17
```

(F) Set the component relation for the PION (emission) components.

Note: Here we assume that the obscuring wind is outside the X-ray broad-line region and it screens photons emitted from the X-ray broad-line region before it reaches us. On the other hand, since the obscuring wind is closer to the central engine than the X-ray narrow-line region, photons emitted from the X-ray narrow-line region are not screened by the obscuring wind.

```
SPEX> com rel 18 21,8,9,1,2,26
SPEX> com rel 19 22,1,2,26
SPEX> com rel 20 23,1,2,26
```

(G) Set the component relation for the AGN SED below the Lyman limit (optical/UV) along line-of-sight #1.

```
SPEX> com rel 24 30,1,31,27
SPEX> com rel 25 6,7,30,1,31,27
SPEX> com rel 28 1
SPEX> com rel 29 1
```

2.6.7 Check settings and calculate

We check the setting of the component relation:

```
SPEX> model show
------------------------------------------------------------------------------
 Number of sectors         :     1
 Sector:    1 Number of model components:    31
    Nr.    1: reds
    Nr.    2: hot
    Nr.    3: comt[8,9,10,11,12,1,2,26 ]
    Nr.    4: pow [6,7,8,9,10,11,12,1,2,26 ]
    Nr.    5: refl[1,2,26 ]
    Nr.    6: etau
    Nr.    7: etau
    Nr.    8: pion[9,10,11,12,1,2,26 ]
    Nr.    9: pion[10,11,12,1,2,26 ]
    Nr.   10: pion[11,12,1,2,26 ]
    Nr.   11: pion[12,1,2,26 ]
    Nr.   12: pion[1,2,26 ]
    Nr.   13: comt[18,1,17 ]
    Nr.   14: pow [6,7,18,1,17 ]
    Nr.   15: comt[19,20,1,17 ]
    Nr.   16: pow [6,7,19,20,1,17 ]
    Nr.   17: etau
    Nr.   18: pion[21,8,9,1,2,26 ]
    Nr.   19: pion[22,1,2,26 ]
    Nr.   20: pion[23,1,2,26 ]
    Nr.   21: vgau
    Nr.   22: vgau
    Nr.   23: vgau
    Nr.   24: comt[30,1,31,27 ]
    Nr.   25: pow [6,7,30,1,31,27 ]
    Nr.   26: etau
    Nr.   27: etau
    Nr.   28: file[1 ]
    Nr.   29: file[1 ]
    Nr.   30: ebv
    Nr.   31: ebv
```

We check the setting of the free parameters and calculate the 1–1000 Ryd ionizing luminosity:

```
SPEX> elim 1.E0:1.E3 ryd
SPEX> calc
SPEX> plot
SPEX> par show free
------------------------------------------------------------------------------------------
↪---------------
sect comp mod  acro parameter with unit     value      status    minimum   maximum␣
↪lsec lcom lpar



   1    3 comt norm Norm (1E44 ph/s/keV) 3.0000001E+12 thawn     0.0      1.00E+20
   1    3 comt t0   Wien temp (keV)      5.0000002E-04 thawn    1.00E-05  1.00E+10
   1    3 comt t1   Plasma temp (keV)    0.1500000     thawn    1.00E-05  1.00E+10
   1    3 comt tau  Optical depth         20.00000     thawn    1.00E-03  1.00E+03

   1    4 pow  norm Norm (1E44 ph/s/keV) 1.0000000E+09 thawn     0.0      1.00E+20
   1    4 pow  gamm Photon index          1.700000     thawn    -10.       10.
```

(continues on next page)

(continued from previous page)

```
   1    8 pion nh   X-Column (1E28/m**2) 5.0000001E-02 thawn    1.00E-07   10.
   1    8 pion xil  Log xi (1E-9 Wm)      0.000000     thawn    -5.0        5.0
   1    8 pion v    RMS Velocity (km/s)   1100.000     thawn     0.0       3.00E+05
   1    8 pion zv   Average vel. (km/s)  -3000.000     thawn   -1.00E+05  1.00E+05


   1   10 pion nh   X-Column (1E28/m**2) 4.9999999E-03 thawn    1.00E-07   10.
   1   10 pion xil  Log xi (1E-9 Wm)      2.700000     thawn    -5.0        5.0
   1   10 pion v    RMS Velocity (km/s)   100.0000     thawn     0.0       3.00E+05
   1   10 pion zv   Average vel. (km/s)  -500.0000     thawn   -1.00E+05  1.00E+05

   1   11 pion nh   X-Column (1E28/m**2) 2.0000001E-03 thawn    1.00E-07   10.
   1   11 pion xil  Log xi (1E-9 Wm)      1.600000     thawn    -5.0        5.0
   1   11 pion v    RMS Velocity (km/s)   50.00000     thawn     0.0       3.00E+05
   1   11 pion zv   Average vel. (km/s)  -100.0000     thawn   -1.00E+05  1.00E+05









   1   18 pion nh   X-Column (1E28/m**2) 7.9999998E-02 thawn    1.00E-07   10.
   1   18 pion xil  Log xi (1E-9 Wm)     0.8000000     thawn    -5.0        5.0
   1   18 pion omeg Scaling factor emis. 2.9999999E-02 thawn     0.0        1.0

   1   19 pion nh   X-Column (1E28/m**2) 5.0000001E-02 thawn    1.00E-07   10.
   1   19 pion xil  Log xi (1E-9 Wm)      2.300000     thawn    -5.0        5.0
   1   19 pion v    RMS Velocity (km/s)   240.0000     thawn     0.0       3.00E+05
   1   19 pion omeg Scaling factor emis. 9.9999998E-03 thawn     0.0        1.0


   1   21 vgau sig  Sigma (km/s)          7000.000     thawn     0.0       3.00E+05









   1   28 file norm Flux scale factor    0.3000000     thawn     0.0       1.00E+20

   1   29 file norm Flux scale factor    0.4000000     thawn     0.0       1.00E+20

   1   30 ebv  ebv  E(B-V)  (mag)        0.1000000     thawn     0.0       1.00E+20

   1   31 ebv  ebv  E(B-V)  (mag)        0.1200000     thawn     0.0       1.00E+20


Instrument     1 region    1 has norm    1.00000E+00 and is frozen
Instrument     2 region    1 has norm    1.00000E+00 and is frozen
Instrument     3 region    1 has norm    1.00000E+00 and is frozen
Instrument     4 region    1 has norm    1.00000E+00 and is frozen
Instrument     5 region    1 has norm    1.00000E+00 and is frozen
Instrument     6 region    1 has norm    1.00000E+00 and is frozen
Instrument     7 region    1 has norm    1.00000E+00 and is frozen
Instrument     8 region    1 has norm    1.00000E+00 and is frozen

------------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

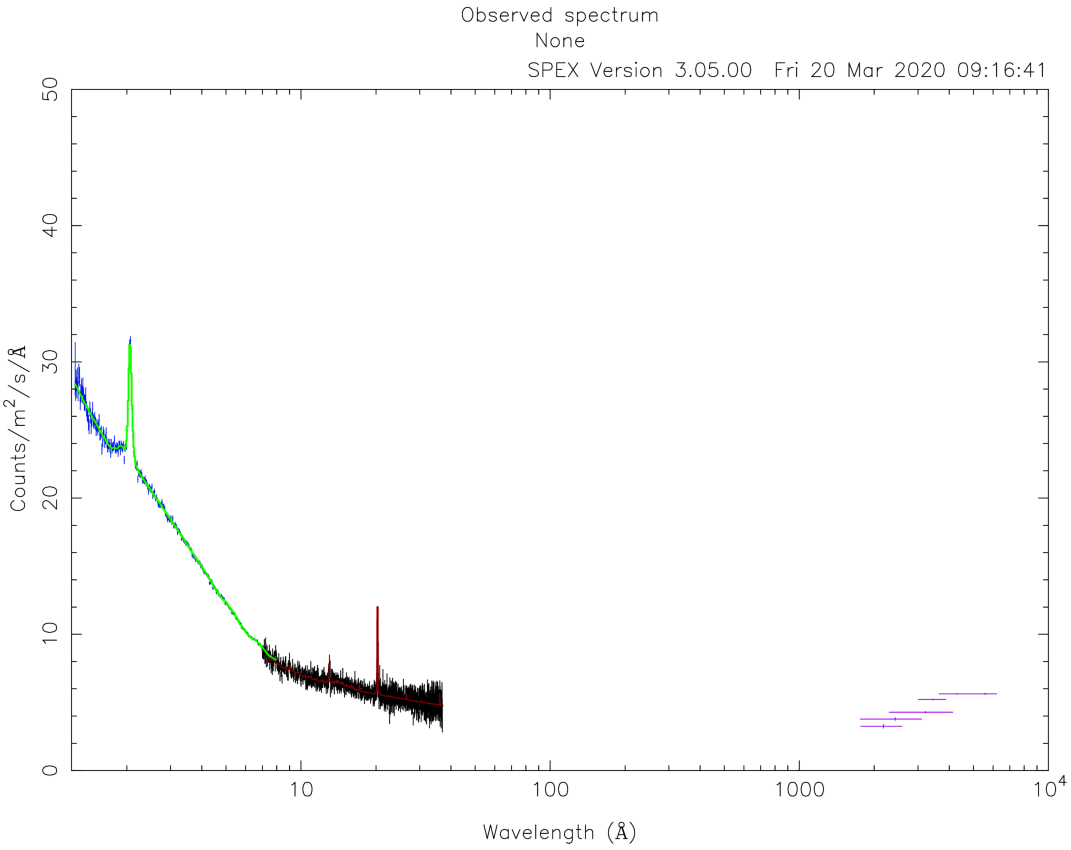
```
Fluxes and restframe luminosities between  1.36057E-02 and    13.606     keV

sect comp mod   photon flux   energy flux nr of photons    luminosity
             (phot/m**2/s)      (W/m**2)   (photons/s)          (W)
   1    3 comt  7.891731E-04  1.775058E-19  1.447225E+54  7.988903E+36
   1    4 pow    38.8452      3.366349E-14  2.869709E+54  1.021578E+38
   1    5 refl   5.98573      7.190706E-15  6.284845E+51  7.467510E+36
   1    8 pion   0.00000       0.00000       0.00000       0.00000
   1    9 pion   0.00000       0.00000       0.00000       0.00000
   1   10 pion  1.755872E-08  5.460370E-24  2.240611E+44  1.101832E+28
   1   11 pion  7.849879E-10  9.871699E-26  3.169252E+45  7.940836E+27
   1   12 pion   0.00000       0.00000       0.00000       0.00000
   1   13 comt   1213.94      6.701157E-15  1.447225E+54  7.988903E+36
   1   14 pow    1657.30      8.033095E-14  2.869709E+54  1.021578E+38
   1   15 comt   0.00000       0.00000      1.106767E+53  5.268881E+35
   1   16 pow    0.00000       0.00000      1.296679E+55  6.397146E+38
   1   18 pion  2.157629E-03  5.832195E-19  1.541392E+54  9.503085E+36
   1   19 pion   3.30138      4.647512E-16  5.174083E+52  1.025305E+36
   1   20 pion   0.00000       0.00000       0.00000       0.00000
   1   24 comt  0.501314      1.089752E-18  1.447225E+54  7.988903E+36
   1   25 pow   0.193548      4.207327E-19  2.869709E+54  1.021578E+38
   1   28 file   0.00000       0.00000       0.00000       0.00000
   1   29 file   0.00000       0.00000       0.00000       0.00000

Fit method        : Classical Levenberg-Marquardt
Fit statistic     : C-statistic
C-statistic       :      1215.69
Expected C-stat   :      1212.71 +/-        49.26
Chi-squared value :      1221.23
Degrees of freedom:         0
W-statistic       :         0.00
Contributions of instruments and regions:
  Ins   Reg    Bins      C-stat  Exp C-stat  Rms C-stat      chi**2      W-stat
    1     1     996     1007.73      996.70       44.66     1012.35        0.00
    2     1     210      197.87      210.01       20.49      198.61        0.00
    3     1       1        3.06        1.00        1.41        3.22        0.00
    4     1       1        0.01        1.00        1.41        0.01        0.00
    5     1       1        0.31        1.00        1.41        0.32        0.00
    6     1       1        0.20        1.00        1.41        0.20        0.00
    7     1       1        4.62        1.00        1.41        4.67        0.00
    8     1       1        1.89        1.00        1.41        1.87        0.00
```

2.6.8 Final remarks

This is the end of this analysis thread. If you want, you can quit SPEX now:

```
SPEX> quit
Thank you for using SPEX!
```

Below, we provide a useful command file.



Define model components and component relations (running scripts)

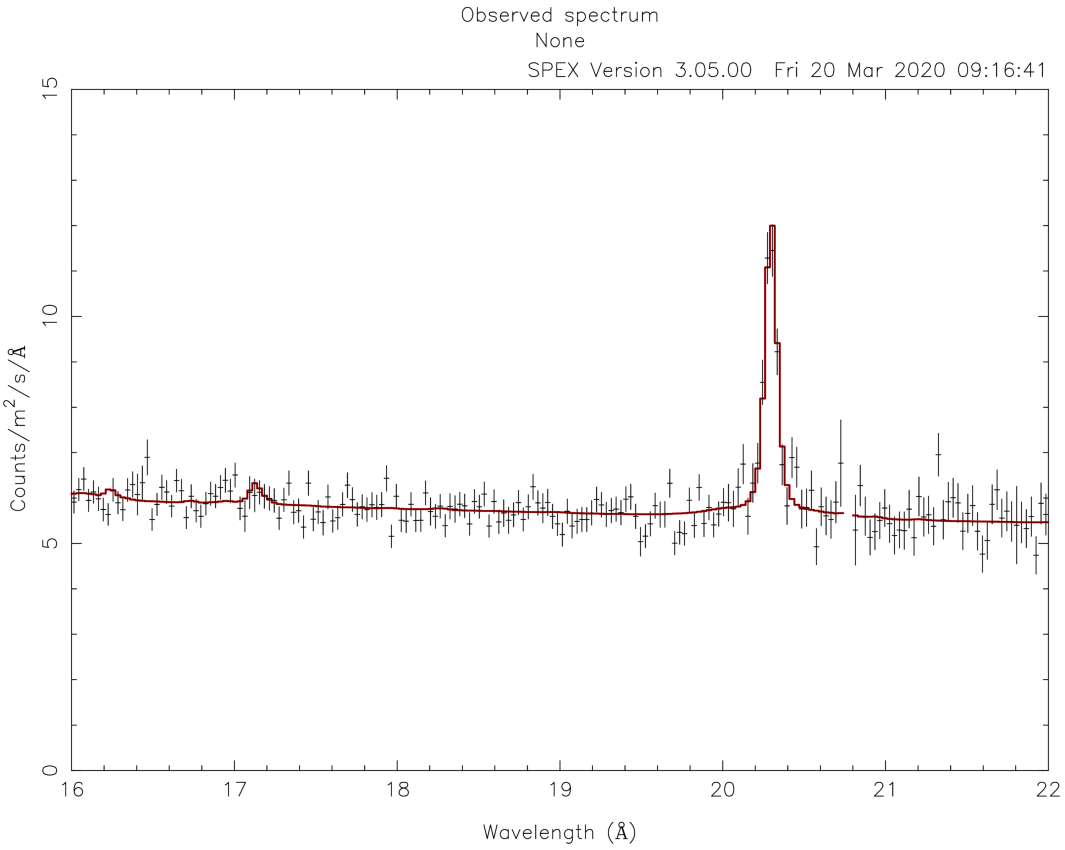
`calc.com` is the command file tailored for this thread.

Load the above command file into SPEX:

```
user@linux:~> spex
Welcome user to SPEX version 3.05.00

SPEX> log exe calc
```

2.7 Fitting two different spectra simultaneously

Suppose one wants to fit two observed spectra from different sources (or areas on the sky) simultaneously with two different models, but with certain parameters coupled to each other. This is possible with SPEX! In this thread, we will guide you through the necessary steps to make this work.



2.7.1 Goal

Suppose we have two point sources absorbed by the same molecular cloud in front of them. The elemental abundances in the cloud appear to be non-solar, but are assumed to be constant trough the cloud. Let's say, we want to fit both sources, while we couple the abundance in the absorber. This requires us to load two different spectra and apply two different source models to them, while constraining some parameters between the models.

2.7.2 Sectors and regions

Before we begin, it is good to understand the SPEX definitions of 'sectors' and 'regions'. These SPEX concepts are very useful when creating more complicated fitting setups, but are often considered to be confusing. In this context, it is also useful to know the definition of 'instruments', because this is very close to the definition of 'region'.

Sectors are used in SPEX to create multiple separate models, for example to model two different sources or areas of an extended source. Key is that this is a division in **model space**. Each sector is assigned an integer number such that each model can be identified.

Regions are used in SPEX to distinguish different observed spectra, for example observed spectra from different sources or spectra extracted from different extraction regions on the detector. Key is that this is a division in **data space**. Each observed spectrum is assigned its own integer number to be identified. Regions are usually defined when the spectra and response are combined into one .spo and .res file.

Instruments are another division in **data space**. Spectra from each .spo and .res file combination that are read in, are assigned an integer instrument number. Because one set of .spo and .res files can contain multiple regions, one can imagine a setup with multiple instruments which each a number of regions with an observed spectrum in them. In this analysis thread, we will only use the regions.

When fitting, SPEX needs to know which model (sector) needs to be applied to which observed spectrum (region). This is already defined when spectra are converted to SPEX format with `trafo`, which is explained in the next section.

2.7.3 Step 1: Trafo

For this science case, we have two sets of XMM-Newton MOS spectra: `M1_src1.pha` and `M1_src2.pha` and their respective arf and rmf files. One spectrum for each point source.

To enable that we create two distinct models for each source, we need to set the relationship between the model sectors that we will create later in SPEX and the spectra that we read in. In this case, we want the model in sector #1 to be fitted to the observed spectrum #1 and, obviously, the model in sector #2 fitted to observed spectrum #2. In SPEX, this needs to be defined already in the spectral file. For this, we use the `trafo` program:

```
Linux:~> trafo
Program trafo: transform data to SPEX 2.0 format
This is version 1.04, of trafo

 Are your data in OGIP format            : type=1
 Old (Version 1.10 and below) SPEX format: type=2
 New (Version 2.00 and above) SPEX format: type=3

 Enter the type: 1
```

In this example, we have OGIP files, so we enter `1` for the type. The next question is how many spectra we want to transform and there we enter `2`:

```
Enter the number of spectra you want to transform: 2
Enter the maximum number of response groups per energy per spectrum: 100000
```

Answer the next question with a large number, like `100000`. The question after that is important for our purpose:

```
Enter the number of sectors you want to create: 2
```

Since we want to create two separate models, we enter `2` here. Then we can start to load the data one by one, while defining the relation between the model number and observed spectrum at the same time. We start with source number 1. For this source we define that sector (model) #1 should be fitted to spectrum (region) #1:

```
Enter the sector and region number: 1 1
```

After this, the usual `trafo` questions are asked about the input files. Provide `trafo` with the files for source #1:

```
How should the matrix be partioned?
Option 1: keep as provided (1 component, no re-arrangements)
Option 2: rearrange into contiguous groups
Option 3: split into N roughly equal-sized components
Enter your preferred option (1,2,3): 1
Enter filename spectrum to be read: M1_src1.pha
Exposure time (s): 1.00000000E+04
Assuming Poissonian Errors
Areascal: 1.00000000E+00
Backscal: 1.00000000E+00
Backfile:
Corrscal: **************
```

(continues on next page)

(continued from previous page)

```
Corrfile:
Respfile:
Ancrfile:
No background specified in pha-file.

Read nevertheless a background file? (y/n) [no]:
Checking data quality and grouping ...
Ogip files have quality flags. Quality 0 means okay
Your spectrum file has          0 bins with bad quality
Your combined file has          0 bins with bad quality
Shall we use these quality flags to ignore bad channels? (y/n) [no]:y
Your spectrum has grouping flags. Do you want your
Spectrum to be binned according to these groups? (y/n) [no]:n
Determining background subtracted spectra ...
No response matrix file specified in pha-file.
Enter filename response matrix to be read: M1_src1.rmf
Reading response matrix ...
Lower model bin boundary for bin        1 must be positive; current values:   0.
↪000000E+00  5.000000E-03
Enter new bin boundary values manually: 3E-5 5E-3
Warning, ebounds data started at channel    0
Warning, response data started at channel 0
Possible response conflict; check xspec/spex with delta line!
Enter shift to response array (1 recommended, but some cases may be 0):1
No effective area file specified in pha-file.

Read nevertheless an effective area file? (y/n) [no]: y
Enter filename arf-file to be read: M1_src1.arf
Reading effective area ...
Determining zero response data  ...
Total number of channels with zero response:        0
Original number of data channels                                :          800
Channels after passing mask and omitting zero response channels:          800
Rebinning data where necessary ...
Rebinning response where necessary ...
old number of response elements:      1280442
new number of response elements:      1280442
old number of response groups  :         2394
new number of response groups  :         2394
Correcting for effective area ...

Determine number of components ...
Found     1 components
Enter any shift in bins (0 recommended): 0
order will not be swapped ...
```

Now, we can do the same for source #2, but now we want model #2 to be fitted to spectrum #2:

```
Enter the sector and region number: 2 2

How should the matrix be partioned?
Option 1: keep as provided (1 component, no re-arrangements)
Option 2: rearrange into contiguous groups
Option 3: split into N roughly equal-sized components
Enter your preferred option (1,2,3): 1
Enter filename spectrum to be read: M1_src2.pha
Exposure time (s): 1.00000000E+04
Assuming Poissonian Errors
Areascal: 1.00000000E+00
Backscal: 1.00000000E+00
Backfile:
```

(continues on next page)

(continued from previous page)

```
Corrscal: **************
Corrfile:
Respfile:
Ancrfile:
No background specified in pha-file.

Read nevertheless a background file? (y/n) [no]: n
Checking data quality and grouping ...
Ogip files have quality flags. Quality 0 means okay
Your spectrum file has          0 bins with bad quality
Your combined file has          0 bins with bad quality
Shall we use these quality flags to ignore bad channels? (y/n) [no]:n
Your spectrum has grouping flags. Do you want your
Spectrum to be binned according to these groups? (y/n) [no]:n
Determining background subtracted spectra ...
No response matrix file specified in pha-file.
Enter filename response matrix to be read: M1_src2.rmf
Reading response matrix ...
Lower model bin boundary for bin        1 must be positive; current values:   0.
↪000000E+00  5.000000E-03
Enter new bin boundary values manually: 3E-5 5E-3
Warning, ebounds data started at channel    0
Warning, response data started at channel 0
Possible response conflict; check xspec/spex with delta line!
Enter shift to response array (1 recommended, but some cases may be 0):1
No effective area file specified in pha-file.

Read nevertheless an effective area file? (y/n) [no]: y
Enter filename arf-file to be read: M1_src2.arf
Reading effective area ...
Determining zero response data  ...
Total number of channels with zero response:        0
Original number of data channels                                :          800
Channels after passing mask and omitting zero response channels:          800
Rebinning data where necessary ...
Rebinning response where necessary ...
old number of response elements:      1280442
new number of response elements:      1280442
old number of response groups  :         2394
new number of response groups  :         2394
Correcting for effective area ...

Determine number of components ...
Found     1 components
Enter any shift in bins (0 recommended): 0
order will not be swapped ...
```

Finally, we need to provide the file names for the .spo and .res files (M1):

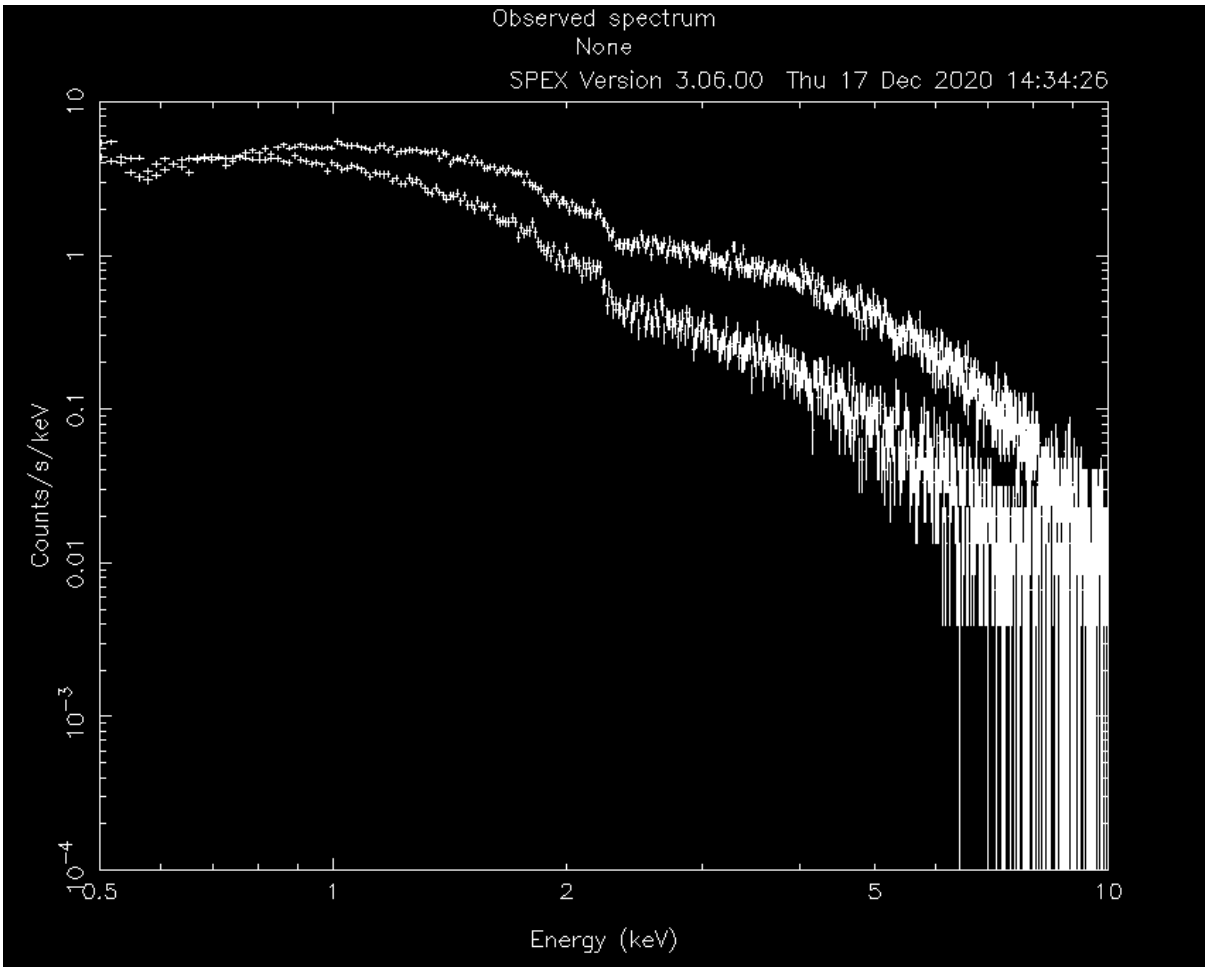
```
Enter filename spectrum to be saved (without .spo): M1
Enter filename response to be saved (without .res): M1
Final number of response elements:  2560884
```

We now have the observed spectra in a data file that we can use in SPEX.

2.7.4 Step 2: SPEX

Let's first load the data into SPEX and see how they look:

```
Linux:~> spex
 Welcome user to SPEX version 3.06.00

 Currently using SPEXACT version 2.07.00. Type `help var calc` for details.

SPEX> data M1 M1
SPEX> plot dev xs
SPEX> plot type data
SPEX> plot x log
SPEX> plot y log
SPEX> pl rx 0.4:10
SPEX> pl ry 1E-4:10.
SPEX> pl
```



Now, we ignore the parts of the spectra that are not well calibrated or contain too little photons:

```
SPEX> ign 0.0:0.4 un k
SPEX> ign 10:100 un k
```

To keep this thread focused on the method to fit two different spectra, we chose to keep the models simple. Both sources consist of an absorbed powerlaw (with different slope and normalisation), but both absorbed by the same molecular cloud. Now, we need to define the model for both the sectors we intent to use:

```
SPEX> com hot
 You have defined    1 component.
SPEX> com po
 You have defined    2 components.
SPEX> com rel 2 1
SPEX> sector new
There are 2 sectors
SPEX> com 2 hot
 You have defined    1 component.
SPEX> com 2 po
 You have defined    2 components.
SPEX> com rel 2 2 1
SPEX> model show
------------------------------------------------------------------------------
 Number of sectors         :     2
 Sector:    1 Number of model components:     2
    Nr.    1: hot
    Nr.    2: pow [1 ]
 Sector:    2 Number of model components:     2
    Nr.    1: hot
    Nr.    2: pow [1 ]
```

The absorption by the molecular cloud is done with the `hot` component and the power law using `pow`. Using the `model show` command, we can see how the models are built up for each sector.

Note that in the case above, where the components in both sectors are the same, we could have used the command `sector copy` to copy the contents of sector #1 into sector #2. This would have saved us a couple of lines of commands.

Let's have a look at how this shows in the `par show` command:

```
SPEX> par show
------------------------------------------------------------------------------
↪---------------
sect comp mod  acro parameter with unit     value      status    minimum   maximum␣
↪lsec lcom lpar

   1    1 hot  nh   X-Column (1E28/m**2)  1.000000     thawn     0.0      1.00E+20
   1    1 hot  t    Temperature (keV)     1.000000     thawn    2.00E-04  1.00E+03
   1    1 hot  rt   T(balance) / T(spec)  1.000000     frozen   1.00E-04  1.00E+04
   1    1 hot  fcov Covering fraction     1.000000     frozen    0.0       1.0
   1    1 hot  v    RMS Velocity (km/s)   100.0000     frozen    0.0      3.00E+05
   1    1 hot  rms  RMS blend    (km/s)   0.000000     frozen    0.0      1.00E+05
   1    1 hot  dv   Vel. separ. (km/s)    100.0000     frozen    0.0      1.00E+05
   1    1 hot  zv   Average vel. (km/s)   0.000000     frozen  -1.00E+05  1.00E+05
   1    1 hot  ref  Reference atom        1.000000     frozen    1.0       30.
   1    1 hot  01   Abundance H           1.000000     frozen    0.0      1.00E+10
   1    1 hot  02   Abundance He          1.000000     frozen    0.0      1.00E+10
   1    1 hot  03   Abundance Li          1.000000     frozen    0.0      1.00E+10
   1    1 hot  04   Abundance Be          1.000000     frozen    0.0      1.00E+10
   1    1 hot  05   Abundance B           1.000000     frozen    0.0      1.00E+10
   1    1 hot  06   Abundance C           1.000000     frozen    0.0      1.00E+10
   1    1 hot  07   Abundance N           1.000000     frozen    0.0      1.00E+10
   1    1 hot  08   Abundance O           1.000000     frozen    0.0      1.00E+10
   1    1 hot  09   Abundance F           1.000000     frozen    0.0      1.00E+10
   1    1 hot  10   Abundance Ne          1.000000     frozen    0.0      1.00E+10
   1    1 hot  11   Abundance Na          1.000000     frozen    0.0      1.00E+10
   1    1 hot  12   Abundance Mg          1.000000     frozen    0.0      1.00E+10
   1    1 hot  13   Abundance Al          1.000000     frozen    0.0      1.00E+10
   1    1 hot  14   Abundance Si          1.000000     frozen    0.0      1.00E+10
   1    1 hot  15   Abundance P           1.000000     frozen    0.0      1.00E+10
   1    1 hot  16   Abundance S           1.000000     frozen    0.0      1.00E+10
```

(continues on next page)

(continued from previous page)

```
   1    1 hot  17   Abundance Cl          1.000000     frozen    0.0       1.00E+10
   1    1 hot  18   Abundance Ar          1.000000     frozen    0.0       1.00E+10
   1    1 hot  19   Abundance K           1.000000     frozen    0.0       1.00E+10
   1    1 hot  20   Abundance Ca          1.000000     frozen    0.0       1.00E+10
   1    1 hot  21   Abundance Sc          1.000000     frozen    0.0       1.00E+10
   1    1 hot  22   Abundance Ti          1.000000     frozen    0.0       1.00E+10
   1    1 hot  23   Abundance V           1.000000     frozen    0.0       1.00E+10
   1    1 hot  24   Abundance Cr          1.000000     frozen    0.0       1.00E+10
   1    1 hot  25   Abundance Mn          1.000000     frozen    0.0       1.00E+10
   1    1 hot  26   Abundance Fe          1.000000     frozen    0.0       1.00E+10
   1    1 hot  27   Abundance Co          1.000000     frozen    0.0       1.00E+10
   1    1 hot  28   Abundance Ni          1.000000     frozen    0.0       1.00E+10
   1    1 hot  29   Abundance Cu          1.000000     frozen    0.0       1.00E+10
   1    1 hot  30   Abundance Zn          1.000000     frozen    0.0       1.00E+10
   1    1 hot  file File electr.distrib.

   1    2 pow  norm Norm (1E44 ph/s/keV) 1.000000     thawn     0.0       1.00E+20
   1    2 pow  gamm Photon index         2.000000     thawn    -10.        10.
   1    2 pow  dgam Photon index break   0.000000     frozen   -10.        10.
   1    2 pow  e0   Break energy (keV)   1.0000000E+10 frozen    0.0       1.00E+20
   1    2 pow  b    Break strength       0.000000     frozen    0.0        10.
   1    2 pow  type Type of norm         0.000000     frozen    0.0        1.0
   1    2 pow  elow Low flux limit (keV) 2.000000     frozen   1.00E-20  1.00E+10
   1    2 pow  eupp Upp flux limit (keV) 10.00000     frozen   1.00E-20  1.00E+10
   1    2 pow  lum  Luminosity (1E30 W)  2.5786482E-02 frozen    0.0       1.00E+20


   2    1 hot  nh   X-Column (1E28/m**2) 1.000000     thawn     0.0       1.00E+20
   2    1 hot  t    Temperature (keV)    1.000000     thawn    2.00E-04  1.00E+03
   2    1 hot  rt   T(balance) / T(spec) 1.000000     frozen   1.00E-04  1.00E+04
   2    1 hot  fcov Covering fraction    1.000000     frozen    0.0        1.0
   2    1 hot  v    RMS Velocity (km/s)  100.0000     frozen    0.0       3.00E+05
   2    1 hot  rms  RMS blend    (km/s)  0.000000     frozen    0.0       1.00E+05
   2    1 hot  dv   Vel. separ. (km/s)   100.0000     frozen    0.0       1.00E+05
   2    1 hot  zv   Average vel. (km/s)  0.000000     frozen  -1.00E+05  1.00E+05
   2    1 hot  ref  Reference atom       1.000000     frozen    1.0        30.
   2    1 hot  01   Abundance H           1.000000     frozen    0.0       1.00E+10
   2    1 hot  02   Abundance He          1.000000     frozen    0.0       1.00E+10
   2    1 hot  03   Abundance Li          1.000000     frozen    0.0       1.00E+10
   2    1 hot  04   Abundance Be          1.000000     frozen    0.0       1.00E+10
   2    1 hot  05   Abundance B           1.000000     frozen    0.0       1.00E+10
   2    1 hot  06   Abundance C           1.000000     frozen    0.0       1.00E+10
   2    1 hot  07   Abundance N           1.000000     frozen    0.0       1.00E+10
   2    1 hot  08   Abundance O           1.000000     frozen    0.0       1.00E+10
   2    1 hot  09   Abundance F           1.000000     frozen    0.0       1.00E+10
   2    1 hot  10   Abundance Ne          1.000000     frozen    0.0       1.00E+10
   2    1 hot  11   Abundance Na          1.000000     frozen    0.0       1.00E+10
   2    1 hot  12   Abundance Mg          1.000000     frozen    0.0       1.00E+10
   2    1 hot  13   Abundance Al          1.000000     frozen    0.0       1.00E+10
   2    1 hot  14   Abundance Si          1.000000     frozen    0.0       1.00E+10
   2    1 hot  15   Abundance P           1.000000     frozen    0.0       1.00E+10
   2    1 hot  16   Abundance S           1.000000     frozen    0.0       1.00E+10
   2    1 hot  17   Abundance Cl          1.000000     frozen    0.0       1.00E+10
   2    1 hot  18   Abundance Ar          1.000000     frozen    0.0       1.00E+10
   2    1 hot  19   Abundance K           1.000000     frozen    0.0       1.00E+10
   2    1 hot  20   Abundance Ca          1.000000     frozen    0.0       1.00E+10
   2    1 hot  21   Abundance Sc          1.000000     frozen    0.0       1.00E+10
   2    1 hot  22   Abundance Ti          1.000000     frozen    0.0       1.00E+10
   2    1 hot  23   Abundance V           1.000000     frozen    0.0       1.00E+10
   2    1 hot  24   Abundance Cr          1.000000     frozen    0.0       1.00E+10
   2    1 hot  25   Abundance Mn          1.000000     frozen    0.0       1.00E+10
```

(continues on next page)

(continued from previous page)

```
   2    1 hot  26   Abundance Fe          1.000000     frozen    0.0       1.00E+10
   2    1 hot  27   Abundance Co          1.000000     frozen    0.0       1.00E+10
   2    1 hot  28   Abundance Ni          1.000000     frozen    0.0       1.00E+10
   2    1 hot  29   Abundance Cu          1.000000     frozen    0.0       1.00E+10
   2    1 hot  30   Abundance Zn          1.000000     frozen    0.0       1.00E+10
   2    1 hot  file File electr.distrib.

   2    2 pow  norm Norm (1E44 ph/s/keV)  1.000000     thawn     0.0       1.00E+20
   2    2 pow  gamm Photon index          2.000000     thawn    -10.        10.
   2    2 pow  dgam Photon index break    0.000000     frozen   -10.        10.
   2    2 pow  e0   Break energy (keV)   1.0000000E+10 frozen    0.0       1.00E+20
   2    2 pow  b    Break strength        0.000000     frozen    0.0        10.
   2    2 pow  type Type of norm          0.000000     frozen    0.0        1.0
   2    2 pow  elow Low flux limit (keV)  2.000000     frozen   1.00E-20   1.00E+10
   2    2 pow  eupp Upp flux limit (keV)  10.00000     frozen   1.00E-20   1.00E+10
   2    2 pow  lum  Luminosity (1E30 W)  2.5786482E-02 frozen    0.0       1.00E+20


Instrument     1 region    1 has norm    1.00000E+00 and is frozen
Instrument     1 region    2 has norm    1.00000E+00 and is frozen

-------------------------------------------------------------------------------
 Fluxes and restframe luminosities between   2.0000     and    10.000     keV

 sect comp mod   photon flux   energy flux nr of photons    luminosity
              (phot/m**2/s)      (W/m**2)   (photons/s)           (W)
    1    2 pow   2.963082E-03  2.752802E-18  4.000000E+43  2.578606E+28

    2    2 pow   2.963082E-03  2.752802E-18  4.000000E+43  2.578606E+28

-------------------------------------------------------------------------------
 Fit method        : Classical Levenberg-Marquardt
 Fit statistic     : C-statistic
 C-statistic       :   2929139.50
 Expected C-stat   :        64.11 +/-        16.31
 Chi-squared value :    158042.30
 Degrees of freedom:          0
 W-statistic       :          0.00
```

Obviously, the initial model parameters are quite far off. We also need to fix the hot model to a low temperature to mimic neutral gas:

```
SPEX> par 1 1 t v 2E-4
SPEX> par 2 1 t v 2E-4
SPEX> par 1 1 t stat f
SPEX> par 2 1 t stat f
SPEX> par 1 1 nh v 1E-3
SPEX> par 2 1 nh v 1E-3
SPEX> par 1 2 norm v 1000
SPEX> par 2 2 norm v 1000
SPEX> calc
SPEX> plot
```

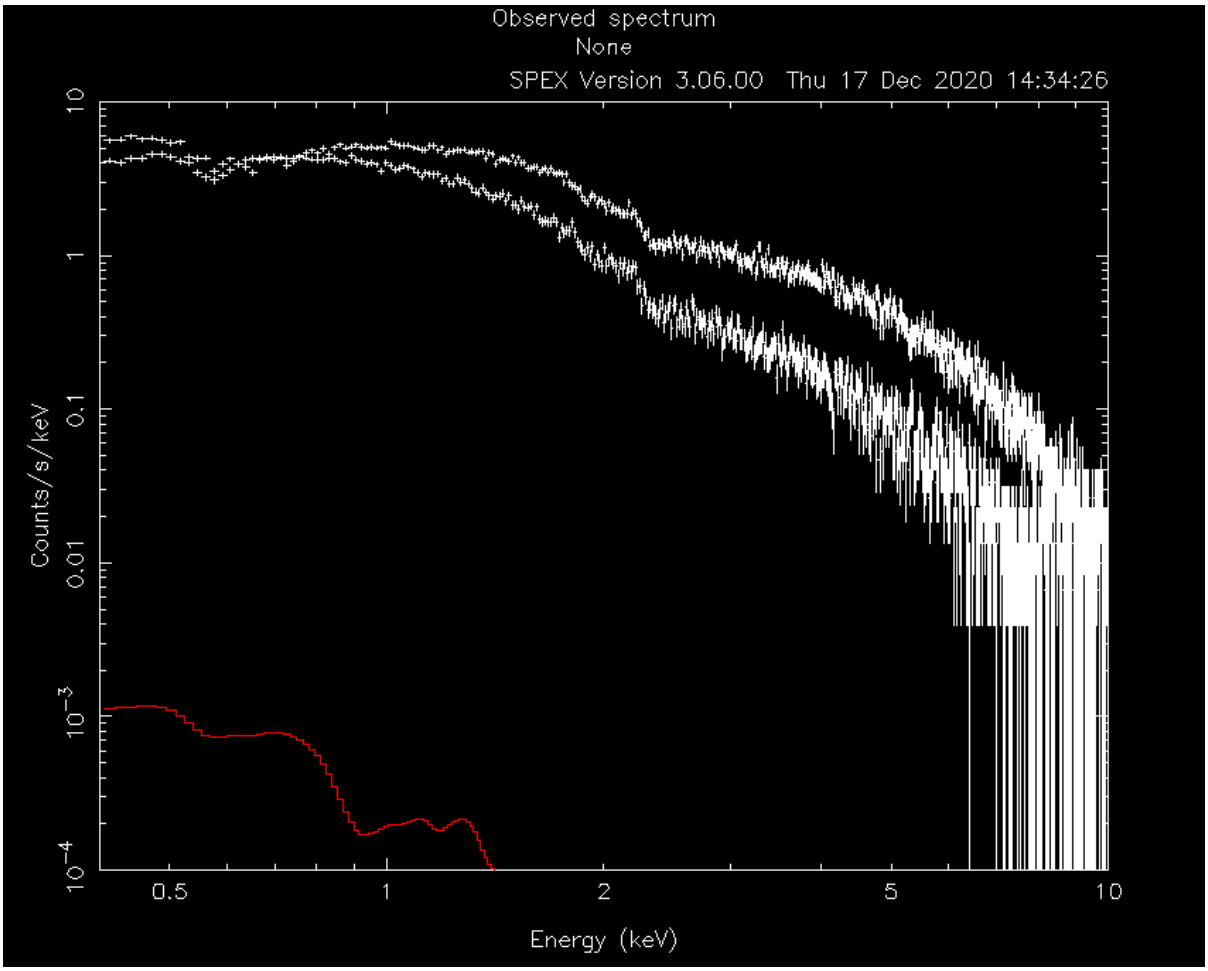
Now, the model spectrum is much closer to the best solution, so we can attempt a fit:

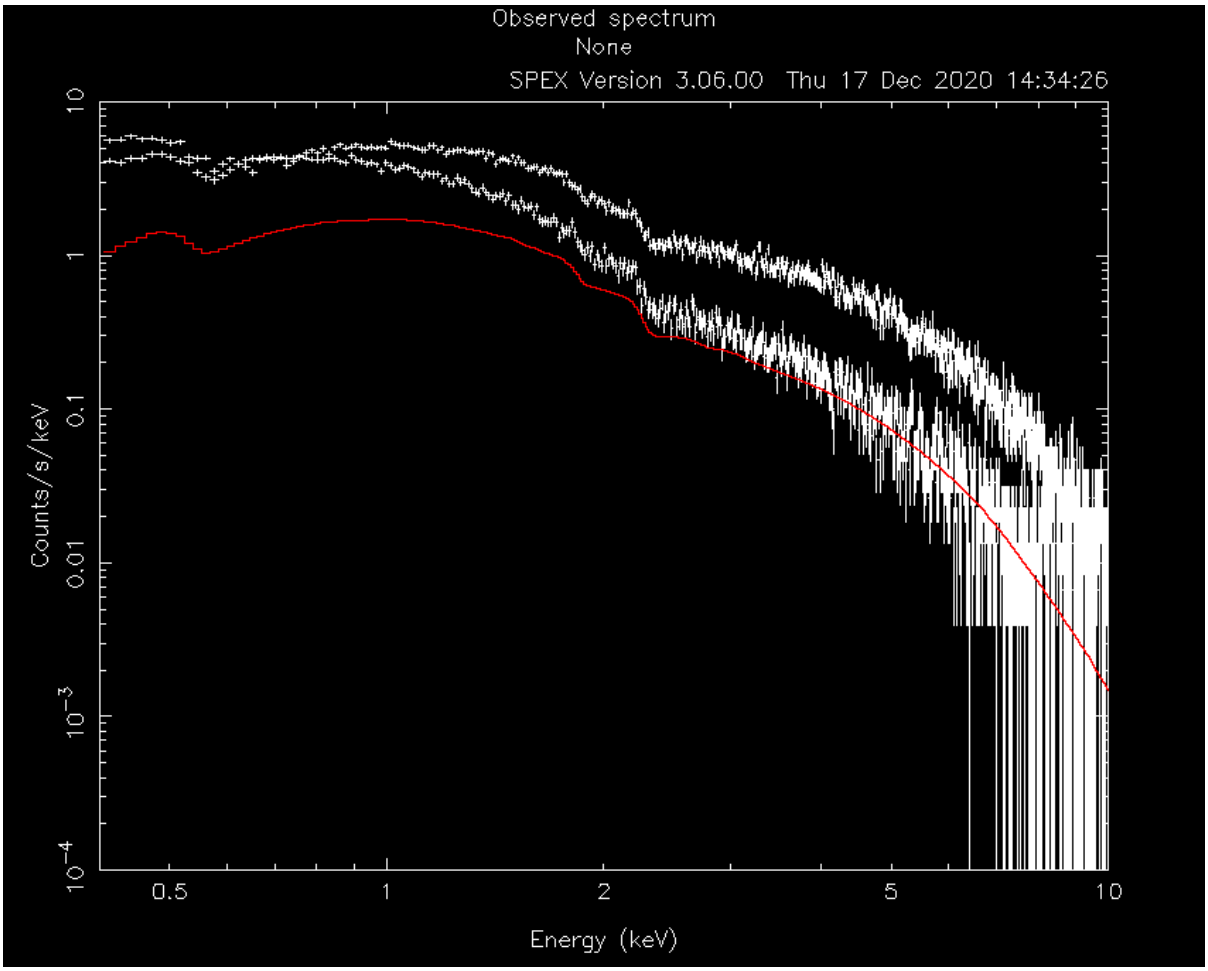
```
SPEX> fit
```

which provides the following best fit parameters:

```
SPEX> par show free
-------------------------------------------------------------------------------
 ↪---------------
```

(continues on next page)


Observed spectrum
None
SPEX Version 3.06.00 Thu 17 Dec 2020 14:34:26
Counts/s/keV
10
1
0.1
0.01
10^-3
10^-4
0.5
1
2
5
10
Energy (keV)


Observed spectrum
None
SPEX Version 3.06.00 Thu 17 Dec 2020 14:34:26
Counts/s/keV
10
1
0.1
0.01
10^{-3}
10^{-4}
0.5
1
2
5
10
Energy (keV)

(continued from previous page)

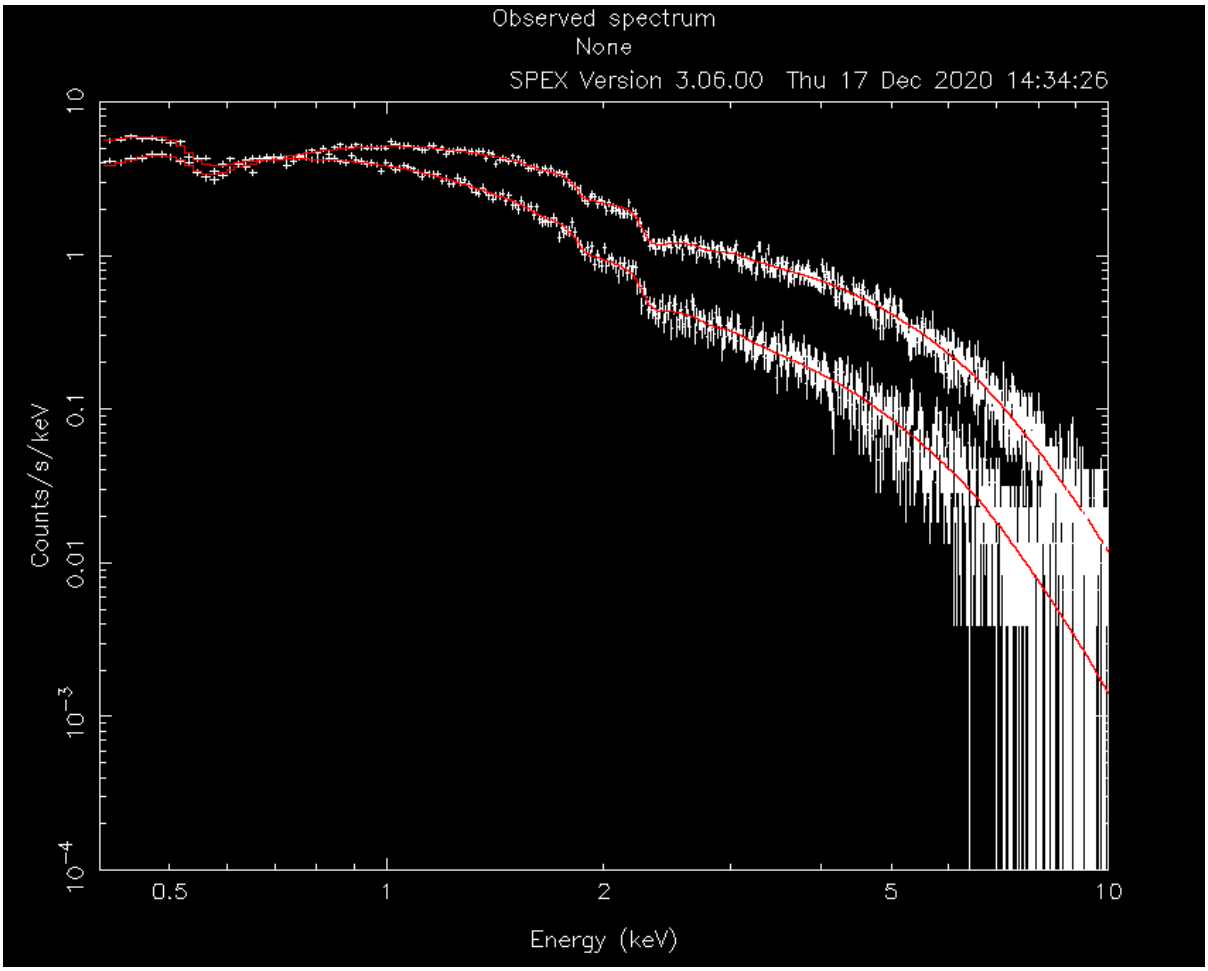
```
sect comp mod  acro parameter with unit     value      status    minimum   maximum␣
↪lsec lcom lpar

   1    1 hot  nh   X-Column (1E28/m**2) 2.1649615E-04 thawn     0.0      1.00E+20

   1    2 pow  norm Norm (1E44 ph/s/keV)  2507.844     thawn     0.0      1.00E+20
   1    2 pow  gamm Photon index          1.496156     thawn    -10.       10.


   2    1 hot  nh   X-Column (1E28/m**2) 1.5288954E-04 thawn     0.0      1.00E+20

   2    2 pow  norm Norm (1E44 ph/s/keV)  1878.148     thawn     0.0      1.00E+20
   2    2 pow  gamm Photon index          2.292902     thawn    -10.       10.


Instrument     1 region    1 has norm    1.00000E+00 and is frozen
Instrument     1 region    2 has norm    1.00000E+00 and is frozen

-------------------------------------------------------------------------------
 Fluxes and restframe luminosities between   2.0000     and    10.000     keV

 sect comp mod   photon flux   energy flux nr of photons    luminosity
              (phot/m**2/s)      (W/m**2)   (photons/s)          (W)
    1    2 pow    156.468      1.123012E-13  1.971041E+47  1.413442E+32

    2    2 pow    41.1973      2.504328E-14  5.188680E+46  3.152038E+31

-------------------------------------------------------------------------------
 Fit method        : Classical Levenberg-Marquardt
 Fit statistic     : C-statistic
 C-statistic       :      1334.89
 Expected C-stat   :      1306.19 +/-        49.94
 Chi-squared value :      1240.11
 Degrees of freedom:      1272
 W-statistic       :         0.00
```

And plot:

This fit already looks acceptable, but let's assume that we want to test if the oxygen abundance in the absorber is consistent with solar. To do this, we can couple the oxygen abundances in the two hot models to each other:

```
SPEX> par 2 1 08 couple 1 1 08
SPEX> par 1 1 08 s t
SPEX> fit

SPEX> par show free
-------------------------------------------------------------------------------
↪---------------
sect comp mod  acro parameter with unit     value      status    minimum   maximum␣
↪lsec lcom lpar

   1    1 hot  nh   X-Column (1E28/m**2) 2.1453765E-04 thawn     0.0      1.00E+20
   1    1 hot  08   Abundance O           1.083166     thawn     0.0      1.00E+10

   1    2 pow  norm Norm (1E44 ph/s/keV)  2509.920     thawn     0.0      1.00E+20
   1    2 pow  gamm Photon index          1.496786     thawn    -10.       10.


   2    1 hot  nh   X-Column (1E28/m**2) 1.5155401E-04 thawn     0.0      1.00E+20
   2    1 hot  08   Abundance O           1.083166     frozen    0.0      1.00E+10␣
↪   1    1 08
```

(continues on next page)


Observed spectrum
None
SPEX Version 3.06.00 Thu 17 Dec 2020 14:34:26
Counts/s/keV
10
1
0.1
0.01
10^-3
10^-4
0.5
1
2
5
10
Energy (keV)

(continued from previous page)

```
   2    2 pow  norm Norm (1E44 ph/s/keV)  1879.305     thawn     0.0      1.00E+20
   2    2 pow  gamm Photon index           2.293384     thawn    -10.       10.


Instrument     1 region    1 has norm    1.00000E+00 and is frozen
Instrument     1 region    2 has norm    1.00000E+00 and is frozen

-------------------------------------------------------------------------------
 Fluxes and restframe luminosities between   2.0000     and    10.000     keV

 sect comp mod   photon flux   energy flux nr of photons    luminosity
              (phot/m**2/s)      (W/m**2)   (photons/s)           (W)
    1    2 pow    156.456      1.122785E-13  1.970944E+47  1.413182E+32

    2    2 pow    41.1969      2.504092E-14  5.188782E+46  3.151799E+31

-------------------------------------------------------------------------------
 Fit method        : Classical Levenberg-Marquardt
 Fit statistic     : C-statistic
 C-statistic       :      1334.81
 Expected C-stat   :      1306.19 +/-        49.94
 Chi-squared value :      1239.62
 Degrees of freedom:      1273
 W-statistic       :         0.00
```

The example above shows that we can couple parameters across sectors to each other and fit them. Although this may not be a realistic science case, it shows how fitting two different spectra simultaneously can be done and how parameters of different models can be coupled to each other.

By the way, the oxygen abundance in this example did not turn out to be significantly different from solar when one calculates the error on oxygen:

```
SPEX> error 1 1 08
 parameter          C-stat      Delta          Delta
   value            value       parameter     C-stat
-------------------------------------------------------
  0.838682         1335.33     -0.244484        0.52
  0.594198         1336.85     -0.488968        2.03
  0.838682         1335.33     -0.244484        0.52
  0.761005         1335.70     -0.322161        0.89
  0.739144         1335.82     -0.344022        1.01
  0.741589         1335.81     -0.341577        0.99
   1.32765         1335.17      0.244484        0.36
   1.57213         1336.28      0.488968        1.46
   1.41755         1335.50      0.334387        0.69
   1.48019         1335.78      0.397026        0.97
   1.48650         1335.82      0.403330        1.00
 Parameter   1    1 08  :    1.0832     Errors:  -0.34158     ,   0.40333
```

2.8 How to use the SPEX user model

Some users would like to add their own models to the SPEX program, similar to the local model option in Xspec. Since the developers of SPEX do not have the man power to implement all possible models out there in a SPEX variety, we have devised a quick way of running external models in SPEX.

SPEX contains two user models `user` (*User: User defined model* (page 173)) and `musr` (*Musr: User defined multiplicative model* (page 159)) for additive and multiplicative models respectively. These model components communicate with the external model through an input and output file containing the model parameters, the energy grid and the calculated spectrum. By creating a program that reads in the input parameters and energy grid, calculates the spectrum, and writes the result to an output file in the right format, users have a very flexible way of using any model in SPEX.

Since this setup offers many possibilities, we can only show a few examples of how these user model components can be used. We have prepared a helper library for Python and Fortran.

2.8.1 Python

The Python helper library for the user model is part of the pyspextools package. A general example of how to user the Python interface to the user model can be found here.

As a more specific example we also provide a method to calculate APEC models and import them to SPEX using the Python interface.

2.8.2 Fortran

Using the Fortran module moduser, we list a number of example user models below:

Use any Xspec model in SPEX

The most general way to incorporate Xspec models in SPEX is to call Xspec itself directly by a SPEX user model executable. Here we provide a fortran+tcl example to use an Xspec built-in comptt model. The program `comptt-xspec.f90` can be modified to calculate different models.

It requires the following files:

- Xspec as provided in HEASARC web site.
- Module moduser.f90 from our Github site.
- Example program comptt-xspec.f90 as provided on our Github page.

Modify the commtt-xspec program (optional)

The key part of the model can be found in line 18-21 of `comptt-xspec.f90`. There, the TCL script sets up the model in Xspec. If you need another model than comptt, then change the model name and modify the number of parameters accordingly!

Compile the executable

The program can be compiled in a few simple steps:

```
linux:~/lmodel>  gfortran -g -c -o moduser.o moduser.f90
linux:~/lmodel>  gfortran -g -c -o comptt-xspec.o comptt-xspec.f90
linux:~/lmodel>  gfortran -g -o comptt-xspec comptt-xspec.o moduser.o
```

Use the Xspec model in SPEX

Here are the SPEX commands to calculate the Xspec-version comptt model. If you are using a different Xspec model, please adapt the number of parameters to the number of parameters in the Xspec model:

```
SPEX> com user
SPEX> par 1 exec av ./comptt-xspec
SPEX> par 1 npar v 6
# The number of parameter needed by the Xspec model
SPEX> par 1 p01 v 0
...
SPEX> calc
```

Use local Xspec model ismabs in SPEX

In this example, we include the Xspec local model called ismabs (Gatuzz et al., 2015) into a SPEX user model executable. The local model consists of a FITS file with atomic data, a `ismabs.f90` Fortran file, and a parameter definition file called `lmodel_ismabs.dat` where the parameters of the model are defined. Since the local model is written in Fortran 90, the easiest way to use it in a SPEX user model is to write a Fortran 90 program that will be the interface between SPEX and the model.

In this case, we need the following files and libraries:

- The cfitsio library should be installed on the system.
- Module moduser.f90 from our Github site.
- Example program loc-xspec.f90 as provided by our Github site.
- The ismabs local model files as provided on the XSPEC web site.

Write a small user program

The `moduser.f90` file is a library with user callable functions to make it easy to write a user model. This file does not need any editing. The user should only edit the example program `loc-xspec.f90` and sometimes also the local XSPEC model, which will be explained later. The example program `loc-xspec.f90` looks like this:

```
program locxspec
use moduser
implicit none

integer             :: i

! Get input and output filenames
call getfilenames(fin,fout,ier)

! Read input file
call readprm(trim(fin))

! Allocate output arrays
```

(continues on next page)

(continued from previous page)

```
call allopar()

! Call local XSPEC model
call ismabs(ipar%eg,ipar%neg,ipar%par,1,opar%sener)

! Do not use wener values for now
do i=1,opar%neg
  opar%wener(i)=0.0
enddo

! Write result to output file
call writespc(fout)

! Clean up memory
call deallpar()

end program
```

Explanation of the called routines

getfilenames(fin,fout,ier) is a routine to read the file names from the command line. The format of the files and the order on the command line is defined by the user model in SPEX. This routine returns the file names the program needs to read the input parameters and write the result.

readprm(trim(fin)) is the routine that reads the input parameters and the input energy grid. It allocates and fills the structure ipar with the needed numbers.

allopar() allocates the memory for the output arrays based on the input file.

ismabs(ipar%eg,ipar%neg,ipar%par,1,opar%sener) is the actual call of the XSPEC local model. The parameters from the ipar structure contain the input parameters and the opar structure contains the output spectrum. For other local models than ismabs, simply change the name of the routine on this line.

writespc(fout) writes the resulting spectrum to the output file.

deallpar() deallocates all the allocated variables from the moduser module.

Compile the executable

In principle, the source files can now be compiled into an executable that the SPEX user model can use. Make sure you have all the necessary files in one directory (see above) and execute the following commands in a terminal:

```
linux:~/lmodel>  gfortran -g -c -o moduser.o moduser.f90
linux:~/lmodel>  gfortran -g -c -o ismabs.o ismabs.f90
linux:~/lmodel>  gfortran -g -c -o loc-xspec.o loc-xspec.f90
linux:~/lmodel>  gfortran -g -o loc-xspec loc-xspec.o ismabs.o moduser.o
```

In the last step, it will be clear whether the executable has access to all the necessary functions. In this case, the ismabs model needs cfitsio to read the fits file with atomic data. It also needs a few XSPEC internal functions to read the path for the FITS file. The cfitsio library can be easily linked by adding -lcfitsio to the last command in the sequence above. For the internal Xspec calls, we need to adapt ismabs.f90 slightly. The few calls to the XSPEC routines can be removed and with a slight modification we can also make sure it finds the fits file. This step needs a little programming experience to do it right. Always keep a backup of the original routine.

When you are done, repeat the following commands to create the executable:

```
linux:~/lmodel>  gfortran -g -c -o ismabs.o ismabs.f90
linux:~/lmodel>  gfortran -g -o loc-xspec loc-xspec.o ismabs.o
                    moduser.o -L/path/to/cfitsio -lcfitsio
```

The '-L/path/to/cfitsio' is optional. You may need to adapt it in case the compiler cannot find libcfitsio.so in the library path. In this flag, you can specify the correct path to libcfitsio.so.

Use the Xspec local model in SPEX

Start SPEX in a directory where the loc-xspec executable that we just made is located. Since the ismabs model is a multiplicative model, we need to load the `musr` component. In the example below, we show how a power-law model is absorbed by ismabs in SPEX:

```
SPEX> com po
SPEX> com musr
SPEX> com rel 1 2
# Link the new loc-xspec executable to the musr component
SPEX> par 1 2 exec av ./loc-xspec
# The number of parameters is found in 'local_ismabs.dat', supplied by the Xspec␣
↪model
SPEX> par 1 2 npar v 31
```

The file `local_ismabs.dat` also describes the parameters and their limits in order. It is advisable to write a SPEX command file to set the parameters and their ranges to their default values. The order of the parameters should be the same in the `musr` model and in the `local_ismabs.dat` file. If the bookkeeping is right, you should be able to issue a calculate command in SPEX and show the absorbed power law in a plot.

CHAPTER

THREE

COMMAND OVERVIEW

3.1 Command syntax

This chapter contains a brief explanation of the most relevant syntax used in SPEX. Each subchapter is divided in an overview, an explanation of the use of syntax followed by some practical examples.

3.1.1 Abundance: standard abundances

Overview

For the plasma models, a default set of abundances is used. All abundances are calculated relative to those standard values. The current default abundance set are the proto-Solar abundances of Lodders et al. (2009). Note that in older versions of SPEX, Anders & Grevesse (1989) was the default. In particular, we recommend to use the proto-Solar (= Solar system) abundances for most applications, as the Solar photospheric abundance has been affected by nucleosynthesis (burning of H to He) and settlement in the Sun during its lifetime. The following abundances (*see the Table below* (page **??**)) can be used in SPEX:

Table 1: Standard abundance sets

Abbrevation	Reference
reset	default Lodders et al. (2009)
ag	Anders & Grevesse (1989)
allen	Allen (1973)
ra	Ross & Aller (1976)
grevesse	Grevesse et al. (1992)
gs	Grevesse & Sauval (1998)
lodders	Lodders proto-Solar Lodders
solar	Lodders Solar photospheric Lodders

For the case of Grevesse & Sauval (1998) we adopted their meteoritic values (in general more accurate than the photospheric values, but in most cases consistent), except for He (slightly enhanced in the solar photosphere due to element migration at the bottom of the convection zone), C, N and O (largely escaped from meteorites) Ne and Ar (noble gases).

In *the table below* (page **??**) we show the values of the standard abundances. They are expressed in logarithmic units, with hydrogen by definition 12.0.

Table 2: Abundances for the standard sets

Z	elem	Default	AG	Allen	RA	Grevesse	GS	Lodders	solar
1	H	12.000	12.00	12.00	12.00	12.00	12.00	12.00	12.00
2	He	10.987	10.99	10.93	10.80	10.97	10.99	10.98	10.90
3	Li	3.331	1.16	0.70	1.00	1.16	3.31	3.35	3.28
4	Be	1.373	1.15	1.10	1.15	1.15	1.42	1.48	1.41

continues on next page

Table 2 – continued from previous page

Z	elem	Default	AG	Allen	RA	Grevesse	GS	Lodders	solar
5	B	2.860	2.6	<3.0	2.11	2.6	2.79	2.85	2.78
6	C	8.443	8.56	8.52	8.62	8.55	8.52	8.46	8.39
7	N	7.912	8.05	7.96	7.94	7.97	7.92	7.90	7.83
8	O	8.782	8.93	8.82	8.84	8.87	8.83	8.76	8.69
9	F	4.491	4.56	4.60	4.56	4.56	4.48	4.53	4.46
10	Ne	8.103	8.09	7.92	7.47	8.08	8.08	7.95	7.87
11	Na	6.347	6.33	6.25	6.28	6.33	6.32	6.37	6.30
12	Mg	7.599	7.58	7.42	7.59	7.58	7.58	7.62	7.55
13	Al	6.513	6.47	6.39	6.52	6.47	6.49	6.54	6.46
14	Si	7.586	7.55	7.52	7.65	7.55	7.56	7.61	7.54
15	P	5.505	5.45	5.52	5.50	5.45	5.56	5.54	5.46
16	S	7.210	7.21	7.20	7.20	7.21	7.20	7.26	7.19
17	Cl	5.299	5.5	5.60	5.50	5.5	5.28	5.33	5.26
18	Ar	6.553	6.56	6.90	6.01	6.52	6.40	6.62	6.55
19	K	5.161	5.12	4.95	5.16	5.12	5.13	5.18	5.11
20	Ca	6.367	6.36	6.30	6.35	6.36	6.35	6.41	6.34
21	Sc	3.123	1.10	1.22	1.04	3.20	3.10	3.15	3.07
22	Ti	4.979	4.99	5.13	5.05	5.02	4.94	5.00	4.92
23	V	4.042	4.00	4.40	4.02	4.00	4.02	4.07	4.00
24	Cr	5.703	5.67	5.85	5.71	5.67	5.69	5.72	5.65
25	Mn	5.551	5.39	5.40	5.42	5.39	5.53	5.58	5.50
26	Fe	7.514	7.67	7.60	7.50	7.51	7.50	7.54	7.47
27	Co	4.957	4.92	5.10	4.90	4.92	4.91	4.98	4.91
28	Ni	6.276	6.25	6.30	6.28	6.25	6.25	6.29	6.22
29	Cu	4.319	4.21	4.50	4.06	4.21	4.29	4.34	4.26
30	Zn	4.700	4.60	4.20	4.45	4.60	4.67	4.70	4.63

Warning: For Allen (1973) the value for boron is just an upper limit.

The current active solar abundance table can be shown using the command `abundance show`.

Syntax

The following syntax rules apply:

`abundance #a` : Set the standard abundances to the values of reference #a in the table above.
`abundance show` : Show the currently active abundance table.

Examples

`abundance gs` : change the standard abundances to the set of Grevesse & Sauval (1998)
`abundance reset` : reset the abundances to the standard set
`abundance show` : show the currently active abundance table

3.1.2 Ascdump: ascii output of plasma properties

Overview

One of the drivers in developing SPEX is the desire to be able to get insight into the astrophysics of X-ray sources, beyond merely deriving a set of best-fit parameters like temperature or abundances. The general user might be interested to know ionic concentrations, recombination rates etc. In order to facilitate this SPEX contains options for ascii-output.

Ascii-output of plasma properties can be obtained for any spectral component that uses the basic plasma code of SPEX; for all other components (like power law spectra, gaussian lines, etc.) this sophistication is not needed and therefore not included. There is a choice of properties that can be displayed, and the user can either display these properties on the screen or write it to file.

The possible output types are listed below. Depending on the specific spectral model, not all types are allowed for each spectral component. The keyword in front of each item is the one that should be used for the appropriate syntax.

plas: basic plasma properties like temperature, electron density etc.

heat: heating and cooling rates for various processes (for photoionised models only)

abun: elemental abundances and average charge per element.

icon: ion concentrations, both with respect to Hydrogen and the relevant elemental abundance.

rion: ionization rates per atomic subshell, specified according to the different contributing processes.

rate: total ionization, recombination and charge-transfer rates specified per ion.

time: recombination time scale per ion according to Bottorf et al. (2000) definition, and relative ion concentrations. Note that the recombination time scale depends upon the hydrogen density, so do not forget to set the relevant density in the model.

grid: the energy and wavelength grid used in the last evaluation of the spectrum.

con: list of the ions that contribute to the free-free, free-bound and two-photon continuum emission, followed by the free-free, free-bound, two-photon and total continuum spectrum, for the last plasma layer of the model.

clin: the continuum, line and total spectrum for each energy bin for the last plasma layer of the model.

line: the line energy and wavelength, as well as the total line emission (photons/s) for each line contributing to the spectrum, for the last plasma layer of the model. Also given is the natural line width and the Doppler broadening (including thermal and turbulent broadening), expressed as a FWHM in keV. Optionally, the results can be sorted according to various columns as follows (first description, between brackets the acronym): energy (ener), wavelength (wav), ion (ion), line power (powe), natural line width (wid).

ebal: the energy balance contributions of each layer (only for photoionized plasmas).

nei: the history of ionisation parameter and temperature in NEI calculations.

snr: hydrodynamical and other properties of the supernova remnant (only for supernova remnant models such as Sedov, Chevalier etc.).

tcon: list of the ions that contribute to the free-free, free-bound and two-photon continuum emission, followed by the free-free, free-bound, two-photon and total continuum spectrum, added for all plasma layers of the model.

tcl: the continuum, line and total spectrum for each energy bin added for all plasma layers of the model.

tlin: the line energy and wavelength, as well as the total line emission (photons/s) for each line contributing to the spectrum, added for all plasma layers of the model.

pop: the occupation numbers as well as upwards/downwards loss and gain rates to all quantum levels included.

lev: the contributions to the population of the energy levels by various processes: positive for gain, negative for loss

elex: the collisional excitation and de-excitation rates for each level, due to collisions with electrons.

prex: the collisional excitation and de-excitation rates for each level, due to collisions with protons.

rad: the radiative transition rates from each level.

two: the two-photon emission transition rates from each level.

rec: writes for each atomic level the populating contributions from radiative, dielectronic and charge exchange recombination, as well as inner-shell ionisation

dem: the emission measure distribution (for the pdem model)

col: the ionic column densities for the hot, pion, slab, xabs and warm models

tran: In two subsequent tables, the transmission and equivalent width of absorption lines (first table) and absorption edges (second table) are listed for the hot, pion, slab, xabs and warm models. Optionally, the results (lines only) can be sorted according to various columns as follows (first description, between brackets the acronym): energy (ener), wavelength (wav), ion (ion), optical depth at line center (tau), equivalent width in keV (ewk), equivalent width in Å (ewa), Voigt a parameter (avo).

warm: the column densities, effective ionization parameters and temperatures of the *warm* model

Syntax

The following syntax rules apply for ascii output:

`ascdump terminal #i1 #i2 #a` : Dump the output for sky sector #i1 and component #i2 to the terminal screen; the type of output is described by the parameter #a which is listed in the table above.
`ascdump file #a1 #i1 #i2 #a2` : As above, but output written to a file with its name given by the parameter #a1. The suffix ".asc" will be appended automatically to this filename.
`ascdump terminal #i1 #i2 #a1 sort #a2` : Dump the output for sky sector #i1 and component #i2 to the terminal screen; the type of output is described by the parameter #a1 which is listed in the table above; the results are sorted according to parameter #a2. Sorting is only possible for the "line" and "tran" options; see there for allowed entries.
`ascdump file #a1 #i1 #i2 #a2` : As above, but output written to a file with its name given by the parameter #a1. The suffix ".asc" will be appended automatically to this filename.
`ascdump set range #r1: unit #a1` : Set the output energy range for the ascii output of: con, clin, lev, line, pop, tcon, tcl, tlin, tran.
`ascdump set flux #r1` : Set the minimum line flux/tau to display in the ascii output of: line, tlin and tran.

Warning: Any existing files with the same name will be overwritten.

Warning: Sorting only possible for the line and tran options.

Examples

`ascdump terminal 3 2 icon` : dumps the ion concentrations of component 2 of sky sector 3 to the terminal screen.
`ascdump file mydump 3 2 icon` : dumps the ion concentrations of component 2 of sky sector 3 to a file named mydump.asc.
`ascdump terminal 3 2 line sort pow` : dumps the emission line power of component 2 of sky sector 3 to the terminal screen, sorted according to line strength.
`ascdump set range 0.1:1.0 unit kev` : sets the output energy range to 0.1 to 1.0 keV.
`ascdump set flux 1E+35` : sets the minimum line strength for the line and tlin output to 1E+35.

3.1.3 Bin: rebin the spectrum

Overview

This command rebins the data (thus both the spectrum file and the response file) in a manner as described in Section *Data: read response file and spectrum* (page 83). The range to be rebinned can be specified either as a channel range (no units required) or in either any of the following units: keV, eV, Rydberg, Joule, Hertz, Å, nanometer, with the following abbreviations: kev, ev, ryd, j, hz, ang, nm.

Note: Usually, the ignore command and the bin command are used around the same time. The safest order is to ignore parts of the spectrum first and then re-bin it. If the bin command is done first and then parts of the spectrum are ignored using a different energy range, then bins at the edges of the energy range could end up being reduced in width.

Syntax

The following syntax rules apply:

`bin #r #i` : This is the simplest syntax allowed. One needs to give the range, #r, over at least the input data channels one wants to rebin. If one wants to rebin the whole input file the range must be at least the whole range over data channels (but a greater number is also allowed). #i is then the factor by which the data will be rebinned.
`bin [instrument #i1] [region #i2] #r #i` : Here one can also specify the instrument and region to be used in the binning. This syntax is necessary if multiple instruments or regions are used in the data input.
`bin [instrument #i1] [region #i2] #r #i [unit #a]` : In addition to the above here one can also specify the units in which the binning range is given. The units can be eV, Å, or any of the other units specified above.

Examples

`bin 1:10000 10` : Rebins the input data channel 1:10000 by a factor of 10.
`bin instrument 1 1:10000 10` : Rebins the data from the first instrument as above.
`bin 1:40 10 unit a` : Rebins the input data between 1 and 40 Å by a factor of 10.

3.1.4 Calculate: evaluate the spectrum

Overview

This command evaluates the current model spectrum. When one or more instruments are present, it also calculates the model folded through the instrument. Whenever the user has modified the model or its parameters manually, and wants to plot the spectrum or display model parameters like the flux in a given energy band, this command should be executed first (otherwise the spectrum is not updated). On the other hand, if a spectral fit is done (by typing the `fit` command) the spectrum will be updated automatically and the calculate command needs not to be given.

Syntax

The following syntax rules apply:

`calc` : Evaluates the spectral model.

Examples

`calc` : Evaluates the spectral model.

3.1.5 Comp: create, delete and relate spectral components

Overview

In fitting or evaluating a spectrum, one needs to build up a model made out of at least 1 component. This set of commands can create a new component in the model, as well as delete any component. Usually we distinguish two types of spectral components in SPEX.

The *additive* components correspond to emission components, such as a power law, a Gaussian emission line, a collisional ionization equilibrium (CIE) component, etc.

The second class (dubbed here *multiplicative* components for ease) consists of operations to be applied to the additive components. Examples are truly multiplicative operations, such as the Galactic absorption, where the model spectrum of any additive component should be multiplied by the transmission of the absorbing interstellar medium, warm absorbers etc. Other operations contained in this class are redshifts, convolutions with certain velocity profiles, etc.

The user needs to define in SPEX which multiplicative component should be applied to which additive components, and in which order. The order is important as operations are not always communative. This setting is also done with this component command.

If multiple sectors are present in the spectral model or response matrix (see Section *Sectors and regions* (page **??**)) then one has to specify the spectral components and their relation for each sector. The possible components to the model are listed and described in Section *Spectral models* (page **??**).

Note that the order that you define the components is not important. However, for each sector, the components are numbered starting from 1, and these numbers should be used when relating the multiplicative components to the additive components.

If you want to see the model components and the way they are related, type "model show".

Warning: If in any of the commands as listed above you omit the sector number or sector number range, the operation will be done for all sectors that are present. For example, having 3 sectors, the "comp pow" command will define a power law component for each of the three sectors. If you only want to define/delete/relate the component for one sector, you should specify the sector number(s). In the very common case that you have only one sector you can always omit the sector numbers.

Warning: After deleting a component, all components are re-numbered! So if you have components 1,2,3 for example as pow, cie, gaus, and you type "comp del 2", you are left with 1=pow, 2=gaus.

Syntax

The following syntax rules apply:

`comp [#i:] #a` : Creates a component #a as part of the model for the (optional) sector range #i:
`comp delete [#i1:] #i2:` : Deletes the components with number from range #i2: for sector range (optional) #i1. See also the warning above
`comp relation [#i1:] #i2: #i3,...,#in` : Apply multiplicative components #i3, . . . , #in (numbers) in this order, to the additive components given in the range #i2: of sectors in the range #i1 (optional). Note that the multiplicative components must be separated by a ","

Examples

`comp pow` : Creates a power-law component for modeling the spectrum for all sectors that are present.
`comp 2 pow` : Same as above, but now the component is created for sector 2.
`comp 4:6 pow` : Create the power law for sectors 4, 5 and 6
`com abs` : Creates a Morrison & McCammon absorption component.
`comp delete 2` : Deletes the second created component. For example, if you have 1 = pow, 2 = cie and 3 = gaus, this command delets the cie component and renumbers 1 = pow, 2 = gaus
`comp del 1:2` : In the example above, this will delete the pow and cie components and renumbers now 1 = gaus
`comp del 4:6 2` : If the above three component model (pow, cie, gaus) would be defined for 8 sectors (numbered 1–8), then this command deletes the cie component (nr. 2) for sectors 4–6 only.
`comp rel 1 2` : Apply component 2 to component 1. For example, if you have defined before with "comp pow" and "comp "abs" a power law and galactic absorption, the this command tells you to apply component 2 (abs) to component 1 (pow).
`comp rel 1 5,3,4` : Taking component 1 a power law (pow), component 3 a redshift operation (reds), component 4 galactic absorption (abs) and component 5 a warm absorber (warm), this command has the effect that the power law spectrum is multiplied first by the transmission of the warm absorber (5=warm), then redshifted (3=reds) and finally multiplied by the transmission of our galaxy (4=abs). Note that the order is always from the source to the observer!
`comp rel 1:2 5,3,4` : Taking component 1 a power law (pow), component 2 a gaussian line (gaus), and 3–5 as above, this model applies multiplicative components 5, 3 and 4 (in that orer) to the emission spectra of both component 1 (pow) and 2 (cie).
`comp rel 7:8 1:2 5,3,4` : As above, but only for sectors 7 and 8 (if those are defined).
`comp rel 3 0` : Remove all relations from component 3.

3.1.6 Data: read response file and spectrum

Overview

In order to fit an observed spectrum, SPEX needs a spectral data file and a response matrix. These data are stored in FITS format, tailored for SPEX (see section *Response formats* (page **??**)). The data files need not necessarily be located in the same directory, one can also give a pathname plus filename in this command.

Warning: Filenames should be entered in the data command without their extension. For the files response.res and spectrum.spo, the data command would look like: data response spectrum.

Syntax

The following syntax rules apply:
`data #a1 #a2` : Read response matrix #a1 and spectrum #a2
`data delete instrument #i` : Remove instrument #i from the data set
`data merge sum #i:` : Merge instruments in range #i: to a single spectrum and response matrix, by adding the data and matrices
`data merge aver #i:` : Merge instruments in range #i: to a single spectrum and response matrix, by averaging the data and matrices
`data save #i #a [overwrite]` : Save data #a from instrument #i with the option to overwrite the existent file. SPEX automatically tags the .spo extension to the given file name. No response file is saved.
`data show` : Shows the data input given, as well as the count (rates) for source and background, the energy range for which there is data the response groups and data channels. Also the integration time and the standard plotting options are given.

Examples

`data mosresp mosspec` : read a spectral response file named mosresp.res and the corresponding spectral file mosspec.spo. Hint, although 2 different names are used here for ease of understanding, it is eased if the spectrum and response file have the same name, with the appropriate extension.
`data delete instrument 1` : delete the first instrument
`data merge aver 3:5` : merge instruments 3–5 into a single new instrument 3 (replacing the old instrument 3), by averaging the spectra. Spectra 3–5 could be spectra taken with the same instrument at different epochs.
`data merge sum 1:2` : add spectra of instruments 1–2 into a single new instrument 1 (replacing the old instrument 1), by adding the spectra. Useful for example in combining XMM-Newton MOS1 and MOS2 spectra.
`data save 1 mydata` : Saves the data from instrument 1 in the working directory under the filename of mydata.spo
`data /mydir/data/mosresp /mydir/data/mosspec` : read the spectrum and response from the directory /mydir/data/

3.1.7 DEM: differential emission measure analysis

Overview

SPEX offers the opportunity to do a differential emission measure analysis. This is an effective way to model multi-temperature plasmas in the case of continuous temperature distributions or a large number of discrete temperature components.

The spectral model can only have one additive component: the DEM component that corresponds to a multi-temperature structure. There are no restrictions to the number of multiplicative components. For a description of the DEM analysis method see document SRON/SPEX/TRPB05 (in the documentation for version 1.0 of SPEX), Mewe et al. (1994), Mewe et al. (1995) and Kaastra et al. (1996).

SPEX has 5 different dem analysis methods implemented, as listed shortly below. We refer to the above papers for more details.

1. reg – Regularization method (minimizes second order derivative of the DEM; advantage: produces error bars; disadvantage: needs fine-tuning with a regularization parameter and can produce negative emission measures.
2. clean – Clean method: uses the clean method that is widely used in radio astronomy. Useful for "spiky emission measure distributions.
3. poly – Polynomial method: approximates the DEM by a polynomial in the $\log T - \log Y$ plane, where Y is the emission measure. Works well if the DEM is smooth.

4. mult – Multi-temperature method: tries to fit the DEM to the sum of Gaussian components as a function of $\log T$. Good for discrete and slightly broadened components, but may not always converge.
5. gene – Genetic algorithm: using a genetic algorithm try to find the best solution. Advantage: rather robust for avoiding local subminima.Disadvantage: may require a lot of cpu time, and each run produces slightly different results (due to randomization).

In practice to use the DEM methods the user should do the following steps:

1. Read and prepare the spectral data that should be analysed
2. Define the dem model with the "comp dem" command
3. Define any multiplicative models (absorption, redshifts, etc.) that should be applied to the additive model
4. Define the parameters of the dem model: number of temperature bins, temperature range, abundances etc.
5. Give the "dem lib" command to create a library of isothermal spectra.
6. Do the dem method of choice (each one of the five methods outlined above)
7. For different abundances or parameters of any of the spectral components, first the "dem lib" command must be re-issued!

Syntax

The following syntax rules apply:

`dem lib` : Create the basic DEM library of isothermal spectra

`dem reg auto` : Do DEM analysis using the regularization method, using an automatic search of the optimum regularization parameter. It determines the regularisation parameter R in such a way that $\chi^2(R) = \chi^2(0)[1 + s\sqrt{2/(n - n_{\rm T}}]$ where the scaling factor $s = 1$, n is the number of spectral bins in the data set and $n_{\rm T}$ is the number of temperature components in the DEM library.

`dem reg auto #r` : As above, but for the scaling factor s set to #r.

`dem reg #r` : Do DEM analysis using the regularization method, using a fixed regularization parameter $R =$#r.

`dem chireg #r1:#r2 #i` : Do a grid search over the regularization parameter R, with #i steps and R distributed logarithmically between #r1 and #r2. Useful to scan the $\chi^2(R)$ curve whenever it is complicated and to see how much "penalty' (negative DEM values) there are for each value of R.

`dem clean` : Do DEM analysis using the clean method

`dem poly #i` : Do DEM analysis using the polynomial method, where #i is the degree of the polynomial

`dem mult #i` : Do DEM analysis using the multi-temperature method, where #i is the number of broad components

`dem gene #i1 #i2` : Do DEM analysis using the genetic algorithm, using a population size given by #i1 (maximum value 1024) and #i2 is the number of generations (no limit, in practice after $\sim$100 generations not much change in the solution. Experiment with these numbers for your practical case.

`dem read #a` : Read a DEM distribution from a file named #a which automatically gets the extension ".dem". It is an ascii file with at least two columns, the first column is the temperature in keV and the second column the differential emission measure, in units of 10^{64} m:math:*^{-3}* keV:math:*^{-1}*. The maximum number of data points in this file is 8192. Temperature should be in increasing order. The data will be interpolated to match the temperature grid defined in the dem model (which is set by the user).

`dem save #a` : Save the DEM to a file #a with extension ".dem". The same format as above is used for the file. A third column has the corresponding error bars on the DEM as determined by the DEM method used (not always relevant or well defined, exept for the regularization method).

`dem smooth #r` : Smoothes a DEM previously determined by any DEM method using a block filter/ Here #r is the full width of the filter expressed in $^{10}\log T$. Note that this smoothing will in principle worsen the χ^2 of the solution, but it is sometimes useful to "wash out" some residual noise in the DEM distribution, preserving total emission measure.

Examples

`dem lib` : create the DEM library
`dem reg auto` : use the automatic regularization method
`dem reg 10.` : use a regularization parameter of $R = 10$ in the regularization method
`dem chireg 1.e-5:1.e5 11` : do a grid search using 11 regularisation parameters R given by 10^{-5}, 10^{-4}, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10^4, 10^5.
`dem clean` : use the clean method
`dem poly 7` : use a 7th degree polynomial method
`dem gene 512 128` : use the genetic algorithm with a population of 512 and 128 generations
`dem save mydem` : save the current dem on a file named mydem.dem
`dem read modeldem` : read the dem from a file named modeldem.dem
`dem smooth 0.3` : smooth the DEM to a temperature width of 0.3 in $^{10}\log T$ (approximately a factor of 2 in temperature range).
Recommended citation: Mewe et al. (1995).

3.1.8 Distance: set the source distance

Overview

One of the main principles of SPEX is that spectral models are in principle calculated at the location of the X-ray source. Once the spectrum has been evaluated, the flux received at Earth can be calculated. In order to do that, the distance of the source must be set.

SPEX allows for the simultaneous analysis of multiple sky sectors. In each sector, a different spectral model might be set up, including a different distance. For example, a foreground object that coincides partially with the primary X-ray source has a different distance value.

The user can specify the distance in a number of different units. Allowed distance units are shown in the table below.

Table 3: SPEX distance units

Abbrevation	Unit
spex	internal SPEX units of 10^{22} m (this is the default)
m	meter
au	Astronomical Unit, 1.49597892 10^{11} m
ly	lightyear, 9.46073047 10^{15} m
pc	parsec, 3.085678 10^{16} m
kpc	kpc, kiloparsec, 3.085678 10^{19} m
mpc	Mpc, Megaparsec, 3.085678 10^{22} m
z	redshift units for the given cosmological parameters
cz	recession velocity in km/s for the given cosmological parameters

The default unit of 10^{22} m is internally used in all calculations in SPEX. The reason is that with this scaling all calculations ranging from solar flares to clusters of galaxies can be done with single precision arithmetic, without causing underflow or overflow. For the last two units (z and cz), it is necessary to specify a cosmological model. Currently this model is simply described by H_0, Ω_m (matter density), Ω_Λ (cosmological constant related density), and Ω_r (radiation density). At startup, the values are:

H_0: 70 km/s/Mpc ,

Ω_m: 0.3 ,

Ω_Λ: 0.7 ,

Ω_r: 0.0

i.e. a flat model with cosmological constant. However, the user can specify other values of the cosmological parameters. Note that the distance is in this case the luminosity distance.

Note that the previous defaults for SPEX ($H_0 = 50$, $q_0 = 0.5$) can be obtained by putting $H_0 = 50$, $\Omega_m = 1$, $\Omega_\Lambda = 0$ and $\Omega_r = 0$.

Warning: When H_0 or any of the Ω is changed, the luminosity distance will not change, but the equivalent redshift of the source is adjusted. For example, setting the distance first to z=1 with the default H_0=70 km/s/Mpc results into a distance of $2.039\,10^{26}$ m. When H_0 is then changed to 100 km/s/Mpc, the distance is still $2.168\,10^{26}$ m, but the redshift is adjusted to 1.3342.

Warning: In the output also the light travel time is given. This should not be confused with the (luminosity) distance in light years, which is simply calculated from the luminosity distance in m!

Syntax

The following syntax rules apply to setting the distance:

`distance [sector #i:] #r [#a]` : set the distance to the value #r in the unit #a. This optional distance unit may be omittted. In that case it is assumed that the distance unit is the default SPEX unit of 10^{22} m. The distance is set for the sky sector range #i:. When the optional sector range is omitted, the distance is set for all sectors.
`distance show` : displays the distance in various units for all sectors.
`distance h0 #r` : sets the Hubble constant H_0 to the value #r.
`distance om #r` : sets the Ω_m parameter to the value #r.
`distance ol #r` : sets the Ω_Λ parameter to the value #r.
`distance or #r` : sets the Ω_r parameter to the value #r.

Examples

`distance 2` : sets the distance to 2 default units, i.e. to 2E22 m.
`distance 12.0 pc` : sets the distance for all sectors to 12 pc.
`distance sector 3 0.03 z` : sets the distance for sector 3 to a redshift of 0.03.
`distance sector 2 : 4 50 ly` : sets the distance for sectors 2-4 to 50 lightyear.
`distance h0 50.` : sets the Hubble constant to 50 km/s/Mpc.
`distance om 0.27` : sets the matter density parameter Ω_m to 0.27
`distance show` : displays the distances for all sectors, see the example below for the output format.

```
SPEX> di 100 mpc
 Distances assuming H0 =  70.0 km/s/Mpc, Omega_m = 0.300 Omega_Lambda = 0.700␣
↪Omega_r = 0.000
Sector       m      A.U.        ly        pc       kpc       Mpc  redshift       ␣
↪cz   age(yr)
------------------------------------------------------------------------------
↪-----------
   1 3.086E+24 2.063E+13 3.262E+08 1.000E+08 1.000E+05  100.0000    0.0229    6878.
↪7 3.152E+08
------------------------------------------------------------------------------
↪-----------
```

3.1.9 Egrid: define model energy grids

Overview

SPEX operates essentially in two modes: with an observational data set (read using the data commands), or without data, i.e. theoretical model spectra. In the first case, the energy grid needed to evaluate the spectra is taken directly from the data set. In the second case, the user can choose his own energy grid.

The energy grid can be a linear grid, a logarithmic grid or an arbitrary grid read from an ascii-file. It is also possible to save the current energy grid, whatever that may be. In case of a linear or logarithmic grid, the lower and upper limit, as well as the number of bins or the step size must be given.

The following units can be used for the energy or wavelength: keV (the default), eV, Ryd, J, Hz, Å, nm.

When the energy grid is read or written from an ascii-file, the file must have the extension ".egr", and contains the bin boundaries in keV, starting from the lower limit of the first bin and ending with the upper limit for the last bin. Thus, the file has 1 entry more than the number of bins! In general, the energy grid must be increasing in energy and it is not allowed that two neighbouring boundaries have the same value.

Finally, the default energy grid at startup of SPEX is a logarithmic grid between 0.001 and 100 keV, with 8192 energy bins.

Syntax

The following syntax rules apply:

`egrid lin #r1:#r2 #i [#a]` : Create a linear energy grid between #r1 and #r2, in units given by #a (as listed above). If no unit is given, it is assumed that the limits are in keV. The number of energy bins is given by #i

`egrid lin #r1:#r2 step #r3 [#a]` : as above, but do not prescribe the number of bins, but the bin width #r3. In case the difference between upper and lower energy is not a multiple of the bin width, the upper boundary of the last bin will be taken as close as possible to the upper boundary (but cannot be equal to it).

`egrid log #r1:#r2 #i [#a]` : Create a logarithmic energy grid between #r1 and #r2, in units given by #a (as listed above). If no unit is given, it is assumed that the limits are in keV. The number of energy bins is given by #i

`egrid log #r1:#r2 step #r3 [#a]` : as above, but do not prescribe the number of bins, but the bin width (in log E) #r3.

`egrid read #a` : Read the energy grid from file #a.egr

`egrid save #a` : Save the current energy grid to file #a.egr

Warning: The lower limit of the energy grid must be positive, and the upper limit must always be larger than the lower limit.

Examples

`egrid lin 5:38 step 0.02 a` : create a linear energy grid between $5 - 38$ Å, with a step size of 0.02 Å.

`egrid log 2:10 1000` : create a logarithmic energy grid with 1000 bins between $2 - 10$ keV.

`egrid log 2:10 1000 ev` : create a logarithmic energy grid with 1000 bins between $0.002 - 0.010$ keV.

`egrid read mygrid` : read the energy grid from the file mygrid.egr.

`egrid save mygrid` : save the current energy grid to file mygrid.egr.

3.1.10 Elim: set flux energy limits

Overview

SPEX offers the opportunity to calculate the model flux in a given energy interval for the current spectral model. This information can be viewed when the model parameters are shown (see section *Par: Input and output of model parameters* (page 104)).

For each additive component of the model, the following quantities are listed:

- the observed photon number flux ($\mathrm{photons\ m^{-2}\ s^{-1}}$) at earth, including any effects of galactic absorption etc.
- the observed energy flux ($\mathrm{Wm^{-2}}$) at earth, including any effects of galactic absorption etc.
- the intrinsic number of photons emitted at the source, not diluted by any absorption etc., in $\mathrm{photons\ s^{-1}}$.
- the intrinsic luminosity emitted at the source, not diluted by any absorption etc., in W.

The following units can be used to designate the energy or wavelength range: keV (the default), eV, Ryd, J, Hz, Å, nm.

Syntax

The following syntax rules apply:

`elim #r1:#r2 [#a]` : Determine the flux between #r1 #r2, in units given by #a, and as listed above. The default at startup is $2 - 10$ keV. If no unit is given, it is assumed that the limits are in keV.

Warning: When new units or limits are chosen, the spectrum must be re-evaluated (e.g. by giving the "calc" command) in order to determine the new fluxes.

Warning: The lower limit must be positive, and the upper limit must always be larger than the lower limit.

Examples

`elim 0.5:4.5` : give fluxes between 0.5 and 4.5 keV
`elim 500:4500 ev` : give fluxes between 500 and 4500 eV (same result as above)
`elim 5:38 a` : give fluxes in the 5 to 38 Å wavelength band

3.1.11 Error: Calculate the errors of the fitted parameters

Overview

This command calculates the error on a certain parameter or parameter range, if that parameter is free (thawn). Standard the 1σ error is calculated, which is equivalent to a 68 % confidence level. So $\Delta\chi^2$ is equal to 1 for a single parameter error. The $\Delta\chi^2$ value can be set, such that for instance 90 % errors are determined (see the table at the bottom of this page for $\Delta\chi^2$ values with their corresponding confidence levels).

SPEX determines the error bounds by iteratively modifying the parameter of interest and calculating χ^2 as a function of the parameter. During this process the other free parameters of the model may vary. The iteration stops when $\chi^2 = \chi^2_{\mathrm{min}} + \Delta\chi^2$, where $\Delta\chi^2$ is a parameter that can be set separately. The iteration steps are displayed. It is advised to check them, because sometimes the fit at a trial parameter converges to a different

solution branch, therefore creating a discontinuous jump in χ^2. In those situations it is better to find the error bounds by varying the search parameter by hand.

Note that SPEX remembers the parameter range for which you did your last error search. This saves you often typing in sector numbers or component numbers if you keep the same spectral component for your next error search.

If the error search reveals a new minimum in χ^2 space that was not found in the fit, SPEX will save the parameters of this new minimum in the file `spex_lower_chi.com`. After the error search these parameters can be set through the command `log exe spex_lower_chi`, in order to direct the model to the new minimum. Note that in the file `spex_lower_chi.com` the parameter for which the error was calculated is "frozen".

Warning: A parameter must have the status "thawn" in order to be able to determine its errors.

Warning: The first time that you use the error command, you need to provide the sector number before it is remembered by SPEX. For example, error 1 1 norm.

Syntax

The following syntax rules apply:

`error [[[#i1:] #i2:] #a:]` : Determine the error bars for the parameters specified by the sector range #i1: (optional), component range #i2 (optional) and parameter range #a: (optional). If not specified, the range for the last call will be used. On startup, this is the first parameter of the first component of the first sector.

`error dchi #r` : This command changes the $\Delta\chi^2$, to the value #r. Default at startup and recommended value to use is 1, for other confidence levels see the Table below.

`error start #r` : This command gives an initial guess of the error bar, from where to start searching the relevant error. This can be helpful for determining the errors on normalization parameters, as otherwise SPEX may from a rather small value. To return to the initial situation, put #r=0 (automatic error search).

Examples

`error norm` : Find the error for the normalization of the current component

`error 2:3 norm:gamm` : determines the error on all free parameters between "norm" and "gamm" for components 2:3

`error start 0.01` : Start calculating the error beginning with an initial guess of 0.01

`error dchi 2.71` : Calculate from now onwards the 90 % error, for 1 degree of freedom. (Not recommended, use standard 68 % errors instead!)

Table 4: Table of $\Delta\chi^2$ as a function of confidence level, P, and degrees of freedom, ν.

P	1	2	3	4	5	6
68.3%	1.00	2.30	3.53	4.72	5.89	7.04
90%	2.71	4.61	6.25	7.78	9.24	10.6
95.4%	4.00	6.17	8.02	9.70	11.3	12.8
99%	6.63	9.21	11.3	13.3	15.1	16.8
99.99%	15.1	18.4	21.1	13.5	25.7	27.8

3.1.12 Fit: spectral fitting

Overview

With this command one fits the spectral model to the data. Only parameters that are thawn are changed during the fitting process. Options allow you to do a weighted fit (according to the model or the data), have the fit parameters and plot printed and updated during the fit, and limit the number of iterations done.

Fit methods

There are three methods implemented that can provide the best fit. All these methods are described in detail in the book of Press et al. (Numerical Recipes). We refer to that book for more details (our algorithms, however, are taken from other sources).

The main and default algorithm is the Levenberg-Marquardt minimisation algorithm. This is recommended for most cases, but it takes advantage of functions with continuous derivatives with respect to the parameters. Also, it will not always find the true minimum, unless the initial values are chosen with care. For those cases, we have as alternatives the simplex method and the simulated annealing method. Both methods are better suited to obtain the true minimum in case of multiple minima, but again here choice of initial parameters is important, and there is no guarantee that the true minimum is reached. In addition, they require significantly more function evaluations compared with the Levenberg-Marquardt algorithm. It is advised to turn back to the Levenberg-Marquardt method when error search is done.

Simulated annealing details

It is beyond the scope of this paper to describe the method in detail. More can be found in the Numerical Recipes book mentioned above, and for our implementation in Goffe et al. (1994). The method basically moves step by step from one set of parameters to another, accepting always better solutions but every now and then accepting poorer solutions (to get out of sub-minima). The degree to which poorer solutions are accepted depends on the Metropolis criterion. During the process, a fake temperature is slowly decreased, until the absolute minimum is reached. This temperature should NOT be confused with the true temperature in case the spectral model contains plasma components. The default parameters should work, but several other parameters can be modified when needed. Important here are the start temperature T, the relative temperature decrease RT after each set of iterations, the stop criterion ϵ, the step size scaling factor vm, the scaling for the number of function evaluations at each temperature ns, the maximum number of function evaluations max, and a flag controlling the printing of intermediate results.

Fit statistic

At the moment SPEX uses two types of fit statistic, χ^2 and C-stat. We first treat the χ^2 statistic, because historically that has been most widely used. However, in the present version of SPEX C-stat is the default because in the far majority of the cases it gives more robust results. Both statistics can be used for all of the fitting methods.

Chi-squared fitting

First we make a few remarks about proper data weighting. χ^2 is usually calculated as the sum over all data bins i of $(N_i - s_i)^2/\sigma_i^2$, i.e.

$$\chi^2 = \sum_{i=1}^{n} \frac{(N_i - s_i)^2}{\sigma_i^2},$$

where N_i is the observed number of source plus background counts, s_i the expected number of source plus background counts of the fitted model, and for Poissonian statistics usually one takes $\sigma_i^2 = N_i$. Take care that the spectral bins contain sufficient counts (either source or background), recommended is e.g. to use at least ~ 10 counts per bin. If this is not the case, first rebin the data set whenever you have a "continuum" spectrum. For line

spectra you cannot do this of course without loosing important information! Note however that this method has inaccuracies if N_i is less than ~ 100.

Wheaton et al. (1995) have shown that the classical χ^2 method becomes inaccurate for spectra with less than ~ 100 counts per bin. This is *not* due to the approximation of the Poisson statistic by a normal distribution, but due to using the *observed* number of counts N_i as weights in the calculation of χ^2. Wheaton et al. (1995) showed that the problem can be resolved by using instead $\sigma_i^2 = s_i$, i.e. the *expected* number of counts from the best fit model.

The option "fit weight model" allows to use these modified weights. By selecting it, the expected number of counts (both source plus background) of the current spectral model is used onwards in calculating the fit statistic. Wheaton et al. (1995) suggest to do the following 3-step process, which we also recommend to the user of SPEX who uses this option:

1. first fit the spectrum using the data errors as weights (the default of SPEX).
2. After completing this fit, select the "fit weight model" option and do again a fit
3. then repeat this step once more by again selecting "fit weight model" in order to replace s_i of the first step by s_i of the second step in the weights. The result should now have been converged (under the assumption that the fitted model gives a reasonable description of the data, if your spectral model is way off you are in trouble anyway!).

C-stat

There is yet another option to try for spectral fitting with low count rate statistics and that is maximum likelyhood fitting. It can be shown that a good alternative to χ^2 in that limit is

$$C = 2\sum_{i=1}^{n} s_i - N_i + N_i \ln(N_i/s_i).$$

This is strictly valid in the limit of Poissonian statistics. If you have a background subtracted spectrum, take care that the subtracted number of background counts is properly stored in the spectral data file, so that raw number of counts can be reconstructed.

This statistic was originally proposed in some other form by Cash (1979) and in the present form sometimes attributed to Castor. However, it appears that it was already introduced and well explained by Baker et al. (1994) .

Warning: Note that for a spectrum with many counts per bin $C \rightarrow \chi^2$, but if the predicted number of counts per bin is small, the expected value for C can be substantially smaller than the number of bins n.

To help the user to see if a C-value corresponds to an acceptable fit, SPEX gives, after spectral fitting, the expected value of C and its r.m.s. spread, based on the best-fit model. Both quantities are simply determined by adding the expected contributions and their variances over all bins. See Kaastra et al. (2017) for more details.

The expected value C_e for C in a bin i and its variance C_v are given by:

$$C_e = 2\sum_{k=0}^{\infty} P_k(\mu)(\mu - k + k\ln(k/\mu)),$$

$$S_v = 4\sum_{k=0}^{\infty} P_k(\mu)(\mu - k + k\ln(k/\mu))^2,$$

$$C_v = S_v - C_e^2,$$

with $P_k(\mu)$ the Poisson distribution:

$$P_k(\mu) = e^{-\mu}\mu^k/k!$$

and μ the expected number of counts. We show both quantities in Fig. *1* (page **??**).

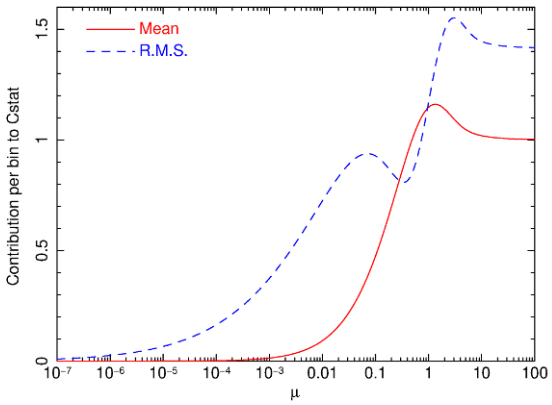


Fig. 1: Expected value of the contribution per bin to C, and its r.m.s. uncertainty, as a function of the mean expected number of counts μ.

Warning: For a proper use of C-stat, it is needed that the background (if present) is also a model for the background, not a scaled background observation. Unfortunately, the fast majority of instrument software packeges provide spectra with such a scaled (and therefore noisy) background). By experimenting it can be shown that in situations where the source is (much) weaker that the subtracted background, this can give bias in the fitted flux (it will be over-estimated). Rebinning the spectrum resolves the problem (because it is some kind of smoothing) but at the expense of spectral resolution. This is undesired. We therefore offer an auxiliary program called backfilter that can filter the subtracted background. It works on a .spo file and creates an improved .spo file. See the documentation of backfilter for more details.

W-stat

The above problem is mitigated in the Xspec package by introducing the so-called W-statistic. See the Xspec manual for more details. We provide here the option to fit using W-stat for compatibility reasons,

Warning: But we do not recommend to use it, but instead use C-stat with background filtering (see above) where needed.

The W-stat first calculates a background estimate for each bin using maximum likelihood techniques. This background depends on the number of counts in the background region, the number of counts in the source region, the predicted number of source counts from the spectral model, and the exposure times of the source region and background region (or, equivalently, incorporating any background area scaling ratio). Using these background estimates, W-stat is then evaluated, and this can be used in the algorithm to find the best-fit set of source parameters.

The Xspec manual notes that for weak sources it can generate an obviously wrong best fit", and they advice to rebin to at least one count per bin to mitigate. This however may degrade the spectral resolution too much. Moreover, for a simple case (blackbody fit to an isolated neutron star), we found that the fitting procedure can show non-monotonous behaviour of W-stat versus iteration, with annoying oscillatory behaviour. Also, a full fit with error search of that spectrum required four times more model evaluations compared with C-stat fitting with filtered background.

Syntax

The following syntax rules apply:

`fit` : Execute a spectral fit to the data.

`fit print #i` : Printing the intermediate results during the fitting to the screen for every n-th step, with n=#i (most useful for $n = 1$). Default value: 0 which implies no printing of intermediate steps.

`fit iter #i` : Stop the fitting process after #i iterations, regardless convergence or not. This is useful to get a first impression for very cpu-intensive models. To return to the default stop criterion, type fit iter 0.

`fit weight model` : Use the current spectral model as a basis for the statistical weight in all subsequent spectral fitting.

`fit weight data` : Use the errors in the spectral data file as a basis for the statistical weight in all subsequent spectral fitting. This is the default at the start of SPEX.

`fit statistic chi2` : Use the χ^2 statistic for the minimisation.

`fit statistic cstat` : Use the C-statistics for the minimisation. This is the default at start-up.

`fit statistic wstat` : Use the W-statistics for the minimisation.

`fit method classical` : Use the classical Levenberg-Marquardt minimisation as the fitting method.

`fit method simplex` : Use simplex minimisation as the fitting method.

`fit method anneal` : Use simulated annealing minimisation as the fitting method.

`fit ann rt #r` : Change the temperature reduction factor. Default value is 0.85.

`fit ann t #r` : Change the start temperature. Default value: 5.

`fit ann eps #r` : Change the convergence criterion ϵ. If the final function values from the last 4 temperatures differ from the corresponding value at the current temperature by less than ϵ and the final function value at the current temperature differs from the current optimal function value by less than ϵ, execution terminates. Default value is 0.10.

`fit ann vm #r` : The step length vector. On input it should encompass the region of interest given the starting value X. For point X(I), the next trial point is selected is from X(I) - VM(I) to X(I) + VM(I). Since VM is adjusted so that about half of all points are accepted, the input value is not very important (i.e. if the value is off, the algorithm adjusts VM to the correct value). Default value: 1.

`fit ann ns #i` : Number of cycles. After $ns * n$ function evaluations, where n is the number of free parameters, each element of the vector VM is adjusted so that approximately half of all function evaluations are accepted. The vector VM controls the relative step size for the free parameters. Default value for ns is 20.

`fit ann max #i` : The maximum number of function evaluations. If during iteration more than this maximum nunber of evaluations is used, the process terminates with an error message (not converged). Default value: 100000.

`fit ann print #r` : Controls the printing of details of th simulated annealing process during the fit. Allowed values 0 to 3. Only relevant for debugging your problem, may give a lot of output depending on its value. Default value: 0 (no printing). This is overruled by the fit print ... command, which for the simulated annealing method prints every new set of parameters and plots its spectrum if a new minimum is found.

Examples

`fit` : Performs a spectral fit. At the end the list of best fit parameters is printed, and if there is a plot this will be updated.

`fit print 1` : If followed by the above fit command, the intermediate fit results are printed to the screen, and the plot of spectrum, model or residuals is updated (provided a plot is selected).

`fit iter 10` : Stop the after 10 iterations or earlier if convergence is reached before ten iterations are completed.

`fit iter 0` : Stop fitting only after full convergence (default).

`fit weight model` : Instead of using the data for the statistical weights in the fit, use the current model.

`fit weight data` : Use the data instead for the statistical weights in the fit.

`fit method clas` : Use the classical Leveberg-Marquardt method to find minima.

`fit ann rt 0.5` : changes the temperature reduction factor for simulated annealing to 0.5.
`fit statistic chi2` : Switch from C-statistics to χ^2.
`fit statistic cstat` : Switch back to C-statistics.

3.1.13 Ibal: set type of ionisation balance

Overview

For the plasma models, different ionisation balance calculations are possible.

Currently, the default set is Urdampilleta et al. (2017) The Table below lists the possible options.

Table 5: Ionisation balance modes

Abbrevation	Reference
reset	default (=u17)
ar85	Arnaud & Rothenflug (1985)
ar92	Arnaud & Raymond (1992) for Fe,
	Arnaud & Rothenflug (1985) for the other elements
bryans09	Bryans et al. (2009)
u17	Urdampilleta et al. (2017) (default)

Syntax

The following syntax rules apply:

`ibal #a` : Set the ionisation balance to set #a with #a in the table above.
`ibal show` : Show the currently active ionisation balance.

Examples

`ibal reset` : Take the standard ionisation balance
`ibal ar85` : Take the Arnaud & Rothenflug ionisation balance

3.1.14 Ignore: ignoring part of the spectrum

Overview

If one wants to ignore part of a data set in fitting the input model, as well as in plotting this command should be used. The spectral range one wants to ignore can be specified as a range in data channels or a range in wavelength or energy. Note that the first number in the range must always be smaller or equal to the second number given. If multiple instruments are used, one must specify the instrument as well. If the data set contains multiple regions, one must specify the region as well. So per instrument/region one needs to specify which range to ignore. The standard unit chosen for the range of data to ignore is data channels. To undo ignore, see the `use` command (*Use: reuse part of the spectrum* (page 118)).

The range to be ignored can be specified either as a channel range (no units required) or in either any of the following units: keV, eV, Rydberg, Joule, Hertz, Å, nanometer, with the following abbrevations: kev, ev, ryd, j, hz, ang, nm.

Syntax

The following syntax rules apply:

`ignore [instrument #i1] [region #i2] #r` : Ignore a certain range #r given in data channels of instrument #i1 and region #i2. The instrument and region need to be specified if there are more than 1 data sets in one instrument data set or if there are more than 1 data set from different instruments.

`ignore [instrument #i1] [region #i2] #r unit #a` : Same as the above, but now one also specifies the unites #a in which the range #r of data points to be ignored are given. The units can be either Å (ang) or (k)eV.

Examples

`ignore 1000:1500` : Ignores data channels 1000 till 1500.

`ignore region 1 1000:1500` : Ignore data channels 1000 till 1500 for region 1.

`ignore instrument 1 region 1 1000:1500` : Ignores the data channels 1000 till 1500 of region 1 from the first instrument.

`ignore instrument 1 region 1 1:8 unit ang` : Same as the above example, but now the range is specified in units of Å instead of in data channels.

`ignore 1:8 unit ang` : Ignores the data from 1 to 8 Å, only works if there is only one instrument and one region included in the data sets.

3.1.15 Ion: select ions for the plasma models

Overview

For the plasma models, it is possible to include or exclude specific groups of ions from the line calculations. This is helpful if a better physical understanding of the (atomic) physics behind the spectrum is requested.

In addition, there is an option to mute particular lines in the spectrum for specific analysis purposes.

There are two main reasons why the user may use this option: computational speed and educational reasons.

Computational speed

The first reason is **accelerating the calculations**. For complex spectral models, the computational time may be long due to the large number of ions and transitions that need to be taken into account, in particular for the line calculations.

By reducing the number of ions in the calculation, or using a maximum principal quantum number, or other reductions, allow to make the calculations faster by simply skipping the line emission from those transitions. Note that obviously this leads to less accurate spectra as compared to the full calculation. Technically, it is done by using the "ions ignore ..." or "ions use ..." commands (for getting rid of, or including line emission from specific ions), or the "ions nmax ..." or "ions lmax ..." commands, to reduce the maximum principal quantum number n and the maximum orbital quantum number l, respectively. Also, by using the "ions old ..." and "ions new ..." commands one may switch between the default (new) spex calculations and the (old) *mekal* calculations. See *Optimizing model performance* (page 178) for more details.

A minor note must be made here: when excluding a single ion, the calculations becomes less accurate, because level populations of ions depend also on how many ionisations or recombinations occur from levels of neighbouring ions. By ignoring an ion, it also cannot contribute to its neighbours.

Currently these settings *only* affect the line emission; in the calculation of the ionisation balance as well as the continuum always all ions are taken into account (unless of course the abundance is put to zero).

Diagnostics & education

The second reason to include or exclude ions is for **diagnostic or educational reasons**. It may be of interest to know how the spectrum of a single ion looks, or how the total spectrum would look without an ion. For such cases, SPEX has the "ions mute ..." or its inverse "ions unmute ..." commands.

Contrary to the use/ignore commands, with this command the full spectrum is calculated, and only at the last step the contribution of the ion is muted or unmuted. Also, contrary to the use/ignore commands, this option works on the full spectrum (continuum and lines, both in emission and absorption). Finally, take care when combining the use/ignore with the mute/unmute commands. Whenever you ignore an ion, it will not be calculated and in those cases the mute or unmute commands are not effective.

Finally, when you use the mute/unmute commands, it will also affect the ascii output for a few important output options, like the "line" and "tra" options for line emission or absorption.

Warning: When using the pion model with this option, you will get a different solution, because it affects the heating and cooling rates, and thus the ionisation balance (equilibrium temperature). Exception is when you use the tmod=1 option for pion, which forces the temperature to be equal to what you prescribe through parameter tinp. For diagnosing the heating/cooling contributions of ions or elements, it is therefore recommended to run first the model with all ions, make an ascii-output of the plasma parameters, take the temperature from there as the "tinp" parameter, and set tmod=1. You can play then with the mute/unmute command.

Quicklook

A new **quicklook mode** is introduced in SPEX 3.0. This mode can greatly reduce computation time by excluding the atomic levels of outer shells that barely affect the obtained spectrum. The maximum quantum numbers n and l of Hydrogen-like ions are provided in the table below.

The quicklook mode is enabled by the command:

```
SPEX> ions ql
```

To undo the quicklook mode, just type `ions use all`.

Warning: This mode will not work for CX model, since electron captured by charge exchange usually populate the outer shells.

Table 6: Preset maximum n and l for quicklook mode (H-like)

Ion	max. n	max. l	Ion	max. n	max. l	Ion	max. n	max. l
C VI	16	3	N VII	16	3	O VIII	16	5
F IX	2	1	Ne X	16	4	Na XI	9	2
Mg XII	16	4	Al XIII	9	2	Si XIV	16	4
P XV	5	1	S XVI	16	4	Cl XVII	4	1
Ar XVIII	13	2	K XIX	4	1	Ca XX	9	2
Sc XXI	2	1	Ti XXII	4	1	V XXIII	2	1
Cr XXIV	5	1	Mn XXV	4	1	Fe XXVI	16	4
Co XXVII	2	1	Ni XXVIII	8	2	Cu XXIX	2	1
Zn XXX	2	1						

Mute lines

A new feature since version 3.06.01 is the line mute command. In some cases, when users what to study a particular line, they want to remove the line from the spectrum to replace it, for example, with a delta line or Gaussian.

The `ions mute line` command allows to mute up to 10 spectral lines identified from the `asc ter line` output. Please note that this command only works for SPEXACT v3 (`var calc new`). When SPEXACT v3 is enabled, the `ascdump line` command (*Ascdump: ascii output of plasma properties* (page 79)) will show a line list with line id numbers:

```
1   O  VIII   1s 2S1/2                          - 2p 2P1/2                          0.
↪6534939       18.97252      1.418E+42  1.689E-06  3.957E-04
2   O  VIII   1s 2S1/2                          - 2s 2S1/2                          0.
↪6535030       18.97225      1.011E+39  1.419E-12  3.957E-04
3   O  VIII   1s 2S1/2                          - 2p 2P3/2                          0.
↪6536802       18.96711      2.834E+42  1.691E-06  3.958E-04
```

In the example of O VIII above, the line id is listed as an integer in the first column before O VIII. To not show the O VIII Lya lines in the spectrum, they can be muted with the command:

```
SPEX> ions mute line 1 ion 8 8
SPEX> ions mute line 3 ion 8 8
```

Where the number behind `line` is the line id, and the numbers behind `ion` are the atomic number and ionisation stage, respectively. If necessary, the lines can be unmuted with the `ions unmute line` command which has a very similar syntax.

Syntax

The following syntax rules apply:

`ions show` : Display the list of ions currently taken into account

`ions use all` : Use all possible ions in the calculation of the line spectrum. This is the default at startup of the program.

`ions use iso #i:` : Use ions of the iso-electronic sequences indicated by #i: in the line spectrum

`ions use z #i:` : Use ions with the atomic numbers indicated by #i: in the line spectrum

`ions use ion #i1 #i2:` : Use ions with the atomic number indicated by #i1 and ionisation stage indicated by #i2: in the line spectrum

`ions ignore all` : Ignore all possible ions in the calculation of the line spectrum

`ions ignore iso #i:` : Ignore ions of the iso-electronic sequences indicated by #i: in the line spectrum

`ions ignore z #i:` : Ignore ions with the atomic numbers indicated by #i: in the line spectrum

`ions ignore ion #i1 #i2:` : Ignore ions with the atomic number indicated by #i1 and ionisation stage indicated by #i2: in the line spectrum

`ions unmute all` : Display the contributions of all possible ions in the final spectrum. This is the default at startup of the program.

`ions unmute iso #i:` : Display ions of the iso-electronic sequences indicated by #i: in the spectrum

`ions unmute z #i:` : Display ions with the atomic numbers indicated by #i: in the spectrum

`ions unmute ion #i1 #i2:` : Display ions with the atomic number indicated by #i1 and ionisation stage indicated by #i2: in the spectrum

`ions mute all` : Ignore all possible ions in the display of the spectrum; will create a zero emission or transmission spectrum! Most useful when followed immieditately by a "ions unmute ..." command

`ions mute iso #i:` : Do not display ions of the iso-electronic sequences indicated by #i: in the spectrum

`ions ignore z #i:` : Do not display ions with the atomic numbers indicated by #i: in the spectrum

`ions ignore ion #i1 #i2:` : Do not display ions with the atomic number indicated by #i1 and ionisation stage indicated by #i2: in the spectrum
`ions nmax all #i:` : Set maximum n for all ions
`ions nmax iso #i1: #i2` : Set maximum n to #i2 for isoelectronic sequence indicated by #i1
`ions nmax z #i1: #i2` : Set maximum n to #i2 for atomic number indicated by #i1
`ions nmax ion #i1 #i2: #i3` : Set maximum n to #i3 for atomic number indicated by #i1 and ionisation stage indicated by #i2.
`ions lmax all #i:` : Set maximum l for all ions
`ions lmax iso #i1: #i2` : Set maximum l to #i2 for isoelectronic sequence indicated by #i1
`ions lmax z #i1: #i2` : Set maximum l to #i2 for atomic number indicated by #i1
`ions lmax ion #i1 #i2: #i3` : Set maximum l to #i3 for atomic number indicated by #i1 and ionisation stage indicated by #i2.
`ions old all` : Force the old calculation for all ions
`ions old iso #i1:` : Force the old calculation for the isoelectronic sequence indicated by #i1
`ions old z #i1:` : Force the old calculation for atomic number indicated by #i1
`ions old ion #i1 #i2:` : Force the old calculation for atomic number indicated by #i1 and ionisation stage indicated by #i2.
`ions new all` : Force the new calculation for all ions
`ions new iso #i1:` : Force the new calculation for the isoelectronic sequence indicated by #i1
`ions new z #i1:` : Force the new calculation for atomic number indicated by #i1
`ions new ion #i1 #i2:` : Force the new calculation for atomic number indicated by #i1 and ionisation stage indicated by #i2.
`ions mute line #i1 ion #i2 #i3` : Mute a line with id #i1 for element #i2 at ionisation stage #i3.
`ions unmute line #i1 ion #i2 #i3` : Unmute a line with id #i1 for element #i2 at ionisation stage #i3.

Examples

`ions ignore all` : Do not take any line calculation into account
`ions use iso 3` : Use ions from the $Z = 3$ (Li) iso-electronic sequence
`ions use iso 1:2` : Use ions from the H-like and He-like isoelectronic sequences
`ions ignore z 26` : Ignore all iron ($Z = 26$) ions
`ions use ion 6 5:6` : Use C V to C VI
`ions mute ion 8 7` : does eliminate the O VII continuum and lines from the displayed spectrum
`ions unmute iso 2` : shows the emission from all He-like ions (again).
`ions show` : Display the list of ions that are used
`ions ql` : Quicklook mode on
`ions old ion 6 6` : Use old calculation for C VI
`ions nmax ion 26 25 5` : Set maximum principal quantum number for Fe XXV to $n = 5$.
`ions lmax ion 26 25 3` : Set maximum angular momentum quantum number for Fe XXV to $\ell = 3$.
`ions mute line 1 ion 8 8` : Mute line id 1 for O VIII.
`ions unmute line 1 ion 8 8` : Unmute line id 1 for O VIII.

3.1.16 Log: Making and using command files

Overview

In many circumstances a user of SPEX wants to repeat his analysis for a different set of parameters. For example, after having analysed the spectrum of source A, a similar spectral model and analysis could be tried on source B. In order to facilitate such analysis, SPEX offers the opportunity to save the commands that were used in a session to an ascii-file. This ascii-file in turn can then be read by SPEX to execute the same list of commands again, or the file may be edited by hand.

The command files can be nested. Thus, at any line of the command file the user can invoke the execution of another command file, which is executed completely before execution with the current command file is resumed. Using nested command files may help to keep complex analyses manageable, and allow the user easy modification of the commands.

In order to facilitate the readability of the command files, the user can put comment lines in the command files. Comment lines are recognized by the first character, that must be #. Also blank lines are allowed in the command file, in order to enhance (human) readability.

Saving commands to file

After the command `log save #a` is given on the SPEX command line (`#a` is the file name), all the following commands entered by the user (NOT the ones before) are stored on the command file until the `log close out` command is given. Exceptions are the commands read from a nested command file (it is not necessary to save these commands, since they are already in the nested command file). Also, help calls and the command to open the file ("log save #a") are not stored.

All commands are expanded to the full keywords before they are saved in the file. However, for execution this is not important, since the interpreter can read both abbrevated and full keywords.

Saving the commands to the command file can be stopped by giving the command `log close save`. The file with the saved commands is closed and remains at the specified path. SPEX will automatically append the extension ".com" to the filename.

Saving output to file

It is also possible to store all the output that is printed on the screen to a file. This is usefull for long sessions or for computer systems with a limited screen buffer. The output saved this way could be inspected later by (other programs of) the user. It is also useful if SPEX is run in a kind of batch-mode. The command to save the output is `log out #a`, where `#a` should be the filename without extension. SPEX will automatically append the extension ".out" to the filename.

Executing commands from file

ASCII files with the .com extension containing SPEX commands can be executed in SPEX using the command `log execute #a`, where `#a` stands for the file name without the ".com" extension. When a command file is read and the end of file is reached, the text "Normal end of command file encountered" is printed on the screen, and execution of the calling command file is resumed, or if the command file was opened from the terminal, control is passed over to the terminal again.

For example, the user may have a command file named *run* which does his entire analysis. This command file might start with the line `log exe mydata` that will run the command file *mydata* that contains all information regarding to the data sets read, further data selection or binning etc. This could be followed by a second line in *run* like "log exe mymodel" that runs the command file *mymodel* which could contain the set-up for the spectral model and/or parameters. Also, often used plot settings (e.g. stacking of different plot types) could easily placed in separate command files.

Syntax

The following syntax rules apply for command files:

`log exe #a` : Execute the commands from the file #a. The suffix ".com" will be automatically appended to this filename.

`log save #a [overwrite] [append]` : Store all subsequent commands on the file #a. The suffix ".com" will be automatically appended to this filename. The optional argument "overwrite" will allow to overwrite an already existing file with the same name. The argument "append" indicates that if the file already exists, the new commands will be appended at the end of this file.

`log close save` : Close the current command file where commands are stored. No further commands will be written to this file.

`log out #a [overwrite] [append]` : Store all subsequent screen output on the file #a. The suffix ".out" will be automatically appended to this filename. The optional argument "overwrite" will allow to overwrite an already existing file with the same name. The argument "append" indicates that if the file already exists, the new output will be appended at the end of this file.

`log close output` : Close the current ascii file where screen output is stored. No further output will be written to this file.

Examples

`log save myrun` : writes all subsequent commands to a new file named "myrun.com". However, in case the file already exists, nothing is written but the user gets a warning instead.

`log save myrun append` : as above, but appends it to an existing file

`log save myrun overwrite` : as above, but now overwrites without warning any already existing file with the same name.

`log close save` : close the file where commands are stored.

`log exe myrun` : executes the commands in file myrun.com.

`log output myrun` : writes all subsequent output to file myrun.out.

`log close output` : closes the file above.

3.1.17 Menu: Menu settings

Overview

When command lines are typed, it frequently happens that often the first keywords are identical for several subsequent lines. This may happen for example when a plot is edited. SPEX offers a shortcut to this by using the menu command.

Syntax

The following syntax rules apply:

`menu none` : Quit the current menu text settings (i.e., return to the default spex prompt).

`menu text #a` : For all following commands, the text string #a will be appended automatically before the following commands.

Examples

`menu text plot` : All following commants will get the "plot" keyword put in front of them. So if the next commant would be "plot dev xs" it is sufficient to type "dev xs" instead.
`menu none` : Return to the normal SPEX prompt.
`menu text "par 1 2"` : All following commands wil get the "par 1 2" keywords put in front of them. The next command could be "t val 4.", which will be expanded to the full "par 1 2 t val 4." to set the temperature of sector 1, component 2 to 4 keV. Note that here the text has three keywords (par, 1, 2) and hence it has to be put between "", to indicate that it is a single text string. If there is only one keyword, these "" are not necessary.

3.1.18 Model: show the current spectral model

Overview

This commands prints the current spectral model, for each sector, to the screen. The model is the set of spectral components that is used, including all additive and multiplicative components. For all additive components, it shows in which order the multiplicative components are applied to the additive (emitted) components. See *Comp: create, delete and relate spectral components* (page 82) for more details.

Syntax

The following syntax rules apply:

`model show` : Prints the model for all sectors to the screen.
`model show #i` : Prints the model for sector #i to the screen.

Examples

`model show 2` : Prints the model for the second sector

3.1.19 Multiply: scaling of the response matrix

Overview

This command multiplies (a component of) the response matrix by a constant.

Warning: If this command is repeated for the same component then the original response matrix is changed by the multiplication of the constants. For example, after multiplying the response matrix by a factor of 2, the original matrix is recovered by multiplying the intermediate result by 0.5.

Warning: The instrument number must be given in this command even if you use only a single instrument.

Syntax

The following syntax rules apply:

`multiply #i1 [component #i2] #r` : Multiplies the response matrix of component #i2 of instrument #i1 by a constant #r.

Examples

`multiply 1 3.5` : Multiplies the response matrix from instrument 1 by a factor of 3.5.
`multiply 1 component 2 3.5` : Multiplies the second component of instrument 1 by the constant 3.5.

3.1.20 Obin: optimal rebinning of the data

Overview

This command rebins (a part of) the data (thus both the spectrum and the response) to the optimal bin size given the statistics of the source as well as the instrumental resolution. This is recommended to do in **all** cases, in order to avoid oversampling of the data. The theory and algorithms used for this rebinning are described in detail in *Optimal definition of respons matrices* (page 288). A simple cartoon of this is: binning to 1/3 of the FWHM, but the factor of 1/3 depends weakly upon the local count rate at the given energy and the number of resolution elements. The better the statistics, the smaller the bin size.

Note: Usually, the ignore command and the obin command are used around the same time. The safest order is to ignore parts of the spectrum first and then re-bin it. If the bin command is done first and then parts of the spectrum are ignored using a different energy range, then bins at the edges of the energy range could end up being reduced in width.

Syntax

The following syntax rules apply:

`obin #i1:` : Simplest command allowed. #i1: is the range in data channels over which the binning needs to take place.
`obin #r1: #i: unit #a` : The same command as above, except that now the ranges over which the data is to be binned (#r1:) are specified in units (#a) different from data channels. These units can be eV, keV, Å, as well as in units of Rydberg (ryd), Joules (j), Hertz (hz) and nanometers (nm).
`obin [instrument #i1:] [region #i2:] #i3:` : Here #i3: is the same as #i1: in the first command. However, here one can specify the instrument range #i1: and the region range #i2: as well, so that the binning is done only for one given data set.
`obin [instrument #i1:] [region #i2:] #r1: [unit #a]` : This command is the same as the above, except that here one can specify the range over which the binning should occur in the units specified by #a. These units can be eV, Å, keV, as well as in units of Rydberg (ryd), Joules (j), Hertz (hz) and nanometers (nm).

Examples

`obin 1:10000` : Optimally bins the data channels 1:10000.

`obin 1:4000 unit ev` : Does the same as the above, but now the data range to be binned is given in eV, from 1−4000 eV, instead of in data channels.

`obin instrument 1 region 1 1:19 unit a` : Bins the data from instrument 1 and region 1 between 1 and 19 Å.

3.1.21 Par: Input and output of model parameters

Overview

This command is used as an interface to set or display the parameters of the spectral model. Each model parameter (like temperature T, abundance, normalization etc.) has a set of attributes, namely its value, status, range and coupling. This is illustrated by the following example. Assume we have a spectral model consisting of a thermal plasma. Consider the silicon abundance (acronym in the model: 14). It can have a value (for example 2.0, meaning twice the solar abundance). Its status is a logical variable, true (thawn) if the parameter can be adjusted during the spectral fitting proces or false (frozen) if it should be kept fixed during the spectral fit. The range is the allowed range that the parameter can have during the spectral fit, for example 0–1000 (minimum – maximum values). This is useful to set to constrain a spectral search to a priori "reasonable" values (e.g. abundances should be non-negative) or "safe" values (SPEX could crash for negative temperatures for example). Finally the coupling allows you to couple parameters, for example if the Si abundance is coupled to the Fe abundance, a change in the Fe abundance (either defined by the user or in a spectral fitting process) will result automatically in an adjustment of the Si abundance. This is a useful property in practical cases where for example a spectrum has strong Fe lines and weaker lines from other species, but definitely non-solar abundances. In this case, one may fit the Fe abundance, but the statistics for the other elements may not be sufficient to determine their abundance accurately; however a priori insight might lead you to think that it is the global metallicity that is enhanced, and therefore you would expect the Si abundance to be enhanced in the same way as Fe.

With the par command, you can set for each parameter individually or for a range of parameters in the same command, their attributes (value, status, range and coupling). Also, you can display the parameters on the screen or write them on a SPEX command file, which allows you to start another run with the current set of parameters.

When setting parameters, you can also specify the sector (range) and component (range). For your first call, it is assumed that you refer to sector 1, component 1, first parameter. In all subsequent calls of the parameter command, it is assumed that you refer to the same sector(s) and component(s) as your last call, unless specified differently. This is illustrated by the following example. Suppose you have the following model: power law (component 1), blackbody (component 2) and RGS line broadening (lpro, component 3). If you type in the commands in that order, this is what happens:

- `par val 2` – sets the norm (= first parameter) of the power law to value 2
- `par gam val 3` – sets the photon index of the power law to value 3
- `par 2 t val 2.5` – sets the temperature of the blackbody to 2.5 keV
- `par norm val 10` – sets the norm of the blackbody to 10
- `par 1:2 norm v 5` – sets the norm of both the PL and BB to 5
- `par val 4.2` – sets the norm of both the PL and BB to 4.2
- `par 3 file avalue myprofile.dat` – sets for the LPRO component the file name with the broadening kernel to myprofile.dat. Note that the command 'value' changes here to 'avalue' to indicate that the parameter is an ascii string.

Instrument normalisations can be thawned by entering the instrument number as a negative sector number. For example, freeing the instrument normalisation of instrument 2 is done with the command `par -2 1 norm stat thawn`. The second value in the command (1) is the region number. Therefore, freeing the normalisation of the third region of the fourth instrument is done with the command: `par -4 3 norm stat thawn`.

Par show

The `par show` command shows an overview of the model parameters, coupled parameters, the fluxes and luminosities, the fit statistics, and, if necessary, the correlation matrix. For complicated models, this can result in a long list of parameters. In order for the user to make selections what they want to see, the `par show` command contains a couple of additional keywords:

- `free` : Show only free parameters.
- `couple` : Show an overview of the coupled parameters.
- `flux` : Show the fluxes and luminosities.
- `stat` : Show the best-fit statistics.
- `corr` : Show the correlation matrix.
- `all` : Show all of the above.

Or if specific sectors or components are desired, provide the sector and component numbers:

- `par show #i:` : Show a range of sectors
- `par show #i: #i:` : Show a range of sectors and components

For example, `par show 1 3:5` will show the parameters for components 3 to 5 for sector number 1.

Syntax

The following syntax rules apply:

`par [#i1:] [#i2:] [#a1:] avalue #a2` : Assign the value #a2 to the parameters specified by the (range of) name(s) #a1 of component range #i2: of sectors #i1:. The sector (range), component (range) and parameter name (range) are optional. If #a2 should be an empty text string, specify the value "none" (without quotes, and all four characters in lowercase). If they are not specified, the sector(s), component(s) and parameter(s) of the last call are used. This command containing the word "avalue" holds for input of text strings. For the input of real numbers "avalue" is replaced by "value" and #a2.

`par [#i1:] [#i2:] [#a:] value #r` : Assign the value #r to the parameters specified by the (range of) name(s) #a of component range #i2: of sectors #i1:. The sector (range), component (range) and parameter name (range) are optional. If not specified, the sector(s), component(s) and parameter(s) of the last call are used.

`par [#i1:] [#i2:] [#a:] status #l` : As above but for the status of a (range of) parameters; #l = T (true, thawn) means a parameter that can be adjusted during a spectral fit, #l = F (false, froze, fixed) means a parameter that will remain fixed during the spectral fitting.

`par [#i1:] [#i2:] [#a:] range #r1:#r2` : As above but fore the allowed range of the parameter. #r1 denotes the lower limit and #r2 denotes the upper limit. Both should be specified.

`par [#i1:] [#i2:] [#a1:] couple [#i3:] [#i4:] #a2:` : Here #i1:, #i2:, #a1: are the sector(s), component(s) and parameter(s) that are to be coupled. The parameter (s) to which they are coupled are specified by #a2: and the optional #i3:, #i4:. If ranges are used, take care that the number of sectors, components and parameters on the right and left match.

`par [#i1:] [#i2:] [#a1:] couple [#i3:] [#i4:] #a2: factor #r` : As above, but couple using a scaling factor #r.

`par [#i1:] [#i2:] #a: decouple` : Decouples the parameter(s) #a: of (optional) components #i2: of of (optional) sector(s) #i1:. Inverse operation of the couple command.

`par show [#a:]` : Shows on the display screen all parameters of all components. See *Par show* (page 105) for more options.

`par show [#i:] [#i:]` : Show the parameters for a range of components or sectors.

`par write #a [overwrite]` : Writes all parameters to a SPEX command file #a. #a should be specified without extension, SPEX automatically adds the .com extension. If the optional overwrite command is given, then the file #a will be overwritten with the new values.

Examples

`par val 10` : Sets the value of the current parameter to 10.

`par t val 4` : Sets the parameter named "t" to 4.

`par 06:28 value 0.3` : Sets the parameters named 06:28 (for example the abundances of C ($Z = 6$), N ($Z = 7$), ... Ni ($Z = 28$)) to the value 0.3.

`par 2 nh val 0.01` : Sets the parameter named "nh" of component 2 to the value 0.01

`par 1 1 norm value 1E8` : Sets parameter named "norm" of component 1 of sector 1 to the value 10^8.

`par file avalue myfile.with.data` : sets the ascii-type parameter named "file" to the value "myfile.with.data" (without quotes).

`par file avalue none` : sets the ascii-type parameter named "file" to the value "" (i.e. an empty string)

`par status frozen` : Sets the fit status of the current parameter to "frozen" (fixed).

`par 1 3 t stat thawn` : Specifies that parameter "t" of the third component of the model for sector 1 should be left free (thawn) during spectral fitting.

`par 2 3 gamm range 1.6:1.9` : Limit parameter "gamm" of component 3 of sector 2 to the range 1.6 – 1.9.

`par norm range -1E8 1E8` : Set the range for the parameter "norm" between -10^8 and $+10^8$. This command is necessary if for example the model is a delta line used to mimick an absorption line, which normally has a default minimum value of 0. (for an emission line).

`par 1 1 norm couple 2 1 norm` : Couples the norm of the first component for the first sector to the norm of the first component of the model for the second sector.

`par 1 1 norm couple 2 1 norm factor 3` : The same command as the above, but now the norm of the first component in the model for sector 1 is 3 times the norm of the first component of the model for sector 2. For example, if the norm of the 1st component of sector 2 gets the value 40, then the norm of the 1st component of sector 1 will automatically be updated to a value of $3 \times 40 = 120$.

`par 3:5 02:30 couple 1 02:30` : Couples the abundances of He–Zn of components 3, 4 and 5 to the He–Zn abundances of component 1.

`par norm decouple` : Decouples the norm of the current component from whatever it was coupled to.

`par -2 1 norm stat thawn` : Free the instrument normalisation of instrument 2 and region 1.

`par show` : Shows all the parameters of the current model for all sectors and how they are coupled (if applicable). For each parameter it shows the value, status, range and coupling information, as well as info on its units etc. It also shows the fluxes and restframe luminosities of the additive components, photon flux (phot/m**2/s) and energy flux (W/m**2) are the values observed at Earth (including any transmission effects like Galactic absorption or intrinsic absoprtion that have been taken into account), and the nr. of photons (photons/s) and luminosity (W) are all as emitted by the source, without any attenuation.

`par show free` : As above, but only displays the parameters that have the status thawn.

`par show couple` : Displays a list with coupled parameters.

`par write mypar overwrite` : SPEX writes all parameters for the model to a file named mypar.com. Any previously existing file mypar.com is overwritten.

`par write mypar` : Same command as above, but now a new file is created. If there already exists a file with the same filename SPEX will give an error message.

3.1.22 Plot: Plotting data and models

Overview

The plot command cause the plot to be (re)drawn on the graphics device. Multiple graphics devices can be defined in one SPEX session. For example, a plot can be sent to both a postscript and a xs device.

A user can also set the number of plot frames in the currently active device(s), e.g. the data and model can be displayed in one frame while the residuals can be displayed in a frame below. The user has to specify what is to be plotted in in each frame.

In each plot frame, there can be different sets that can be specified by the "set" keyword in the plot command. For example, for the plot type data, each set corresponds to the spectrum of an instrument or region; with this option

it is therefore possible to give different instruments different colours or plot symbols, etc., or just to omit one of the instruments from the plot.

The plot command is also used to select the axis plot units and scales as well as character commands like font style, line weights, plot colours, etc. Finally, the plot command can also be used to dump the data to an ascii file.

Warning: To make a plot, always start with a `plot device` command to open any plot device you wish. Next select a plot type using `plot type`. After this you can give any plot commands to modify or display the plot.

Warning: Some of the more fancy plotting options, like adding labels to a plot, are broken. To make more sophisticated plots, it is advisable to save the plot to a QDP file with `plot adum` and adapt them through QDP (Ftools). We intent to update the plotting system in a future version of SPEX.

Syntax

The following syntax rules apply for plot:

`plot` : (Re)draws the plot on the graphics device. Take care to open at least one device first (with a "plot dev" command)

`plot frame new` : Creates a new additional plot frame, where an other plot type can be displayed (or the same plot type but with different units). Take care to position the viewport of the new and old frames to the desired locations.

`plot frame #i` : Sets frame number #i to be the currently active frame. After this command has been issued, all plot commands will only affect the currently active frame.

`plot frame delete #i` : Deletes frame number #i.

`plot type #a` : Specifies what is to be plotted in the selected frame. #a can be any of the plot types specified in *Plot types* (page 123).

`plot device #a1 [#a2]` : Selects a graphics device #a1 and optional file name #a2 (in the case of a postscript or gif device). File extensions (for example .ps) should be specified. For valid devices, see *Plot devices* (page 123).

`plot close #i` : Close graphics device number #i. Note, always close a postscript device before quiting /spex, otherwise the contents may be corrupted.

`plot hlan` : Make a hardcopy in landscape orientation and send it to the standard printer. Use is made of the unix command `lp -c filename`.

`plot hpor` : As above, but for portrait orientation

`plot x lin` : Plot the x-axis on a linear scale

`plot x log` : Plot the x-axis on a log scale

`plot y lin` : Plot the y-axis on a linear scale

`plot y log` : Plot the y-axis on a log scale

`plot y mix #r1 #r2` : Plot the y-axis on a mixed linear/logarithmic scale; for y-values below #r1 the data will be plotted linear, and for y-values above #r1 the data will be plotted logarithmically; the linear part occupies the fraction of the total frame height given by #r2. See also *Plot axis scales* (page 133).

`plot z lin` : Plot the z-axis on a linear scale. The z-axis is defined for two-dimensional plots like contour plots.

`plot z log` : As above, but using a log scale.

`plot ux #a` : Set the plot units on the x-axis, #a can be Å, keV, eV, or whatever unit that is allowed. The allowed values depend upon the plot type. See *Plot axis units and scales* (page 129) for allowed units for each plot type.

`plot ux vel #r #a` : Plot the x-axis in velocity units ($\mathrm{km\ s^{-1}}$), with reference energy provided as #r if #a is "keV" or reference wavelength provided as #r if #a is "Ang". Note that the same zero scale will be set for all

plot windows.

`plot uy #a` : Same as above, but for the y-axis

`plot uz #a` : Same as above, but for the z-axis

`plot rx #r1:#r2` : set the plot range on the x-axis from #r1 to #r2

`plot ry #r1:#r2` : Same as above, but for the y-axis

`plot rz #r1:#r2` : Same as above, but for the z-axis

`plot view default #l` : For #l is true, the default viewport is used. For #l is false, the default viewport is overruled and you can specify your own viewport limits with the `set view x` and `set view y` commands. Useful for stacking different plot frames.

`plot view x #r1:#r2` : Set the x viewport range from #r1 to #r2, where #r1 and #r2 must be in the range 0–1.

`plot view y #r1:#r2` : Same as above, but for the y-range.

`plot view back #i` : Set the viewport background colour to value #i. See Section *Plot colours* (page 124) for the allowed plot colours.

`plot view transp #l` : If true, set the viewport background to to transparent, the plot frame will be shown with the background colour selected by the `plot view back` command. Default is false.

`plot box disp #l` : Display a box around the viewport, true/false

`plot box edge #a #l` : If true (default), display the edges of the plot box, #a should be one of the following keywords: top, bottom, left, right. Note that whether you see the edges or not also depends on the settings defined by the `plot box disp` command.

`plot box axis x #l` : Plots an x-axis with tick marks at the line $y = 0$. You only see this of course if your y-range includes the value 0.

`plot box axis y #l` : As above, but for the y-axis

`plot box grid x #l` : Plot a grid for the x-axis

`plot box grid y #l` : Plot a grid for the y-axis

`plot box numlab bottom #l` : Show the tick number labels at the bottom of the plot, true/false

`plot box numlab top #l` : Same as above, but for the top

`plot box numlab left #l` : Same as above, but for the left

`plot box numlab right #l` : Same as above, but for the right

`plot box numlab vert #l` : Plot the tick number labels on the y-axis vertically or horizontally, set #l to true for vertical tnumbers and false for horizontal numbers.

`plot box numlab xscal #i` : Way to denote the numerical numbers along the x-axis. Three values are allowed: 0=automatic, 1=forced decimal labeling, 2=forced exponential labeling. Default is 0.

`plot box numlab yscal #i` : As above, but for the y-axis

`plot box tick invert x #l` : Draw the tick marks on the x-axis on the inside or outside the box, set #l to true for outside and false for inside (default).

`plot box tick invert y #l` : Same as above, but for the y-axis.

`plot box tick minor x #l` : Draw minor tick marks on the x-axis, true/false

`plot box tick minor y #l` : Same as above, but for the y-axis

`plot box tick major x #l` : Draw major tick marks on the x-axis, true/false

`plot box tick major y #l` : Same as above, but for the y-axis

`plot box tick distance x #r` : Set the distance between the major/labelled tick marks on the x-axis to #r

`plot box tick distance y #r` : Same as above, but for the y-axis

`plot box tick subdiv x #i` : Draw #i minor tick marks between each major tick mark on the x-axis

`plot box tick subdiv y #i` : Same as above, but for the y-axis

`plot box col #i` : Set the box colour to colour number #i. See *Plot colours* (page 124) for the allowed plot colours.

`plot box lt #i` : Set the box line type to #i. See *Plot line types* (page 125) for allowed line types.

`plot box lw #i` : Set the box line weight to number #i. See *Plot line types* (page 125) for more about line weights.

`plot box fh #r` : Set the box font height to number #i.

`plot box font #i` : Set the box font to number #i. See *Plot text* (page 125) for more details about text fonts.

`plot cap #a disp #l` : If #l is true, display the caption (default). For more about captions see *Plot captions* (page 128). Here and below, #a can be x, y, z, id, lt or ut.

`plot cap #a col #i` : Plot caption #a in colour number #i. See *Plot colours* (page 124) for valid colour indices.

`plot cap #a back #i` : Set the background colour for caption #a to #i.

`plot cap #a lw #i` : Set the font line weight for caption #a to #i.

`plot cap #a fh #r` : Set the font height for caption #a to value #r.

`plot cap #a font #i` : Set the font type for caption #a to #i.

`plot cap #a1 text #a2` : Set the text for caption #a1 to #a2. Note that the caption text should be put between quotation marks, like "best fit results" if you want to see the text: best fit results.

`plot cap #a1 side #a2` : Plot caption #a1 (which can be x, y, z, id, lt, ut) at the side of the frame specified by #a2, which may stand for t (top), b (bottom), lh (left, horizontal), rh (right, horizontal), lv (left, vertical) and rv (right, vertical).

`plot cap #a off #r` : Offset caption #a by #r from the edge of the viewport, where #r is in units of the character height. Enter negative values to write inside the viewport.

`plot cap #a coord #r` : Plot caption #a along the specified edge of the viewport, in units of the viewport length, where $0.0 \leq \#r \leq 1.0$.

`plot cap #a fjust #r` : Controls justification of the caption #a parallel to the specified edge of the viewport. If #r = 0.0, the left hand of #a will be placed at the position specified by "coord" above; if #r = 0.5, the center of the string will be placed at "coord", if #r = 1.0 the right hand of #a will be placed at "coord". Other values can be used but are less useful.

`plot string new #r1 #r2 #a` : Plot a new string with text as specified in #a at x=#r1 and y = #r2. See *Plot text* (page 125) for more details about text strings. Also do not forget to put #a between "" if it consists of more than one word (i.e., if it contains spaces).

`plot string del #i:` : Delete string numbers specified by the range #i from the plot.

`plot string #i: disp #l` : If true (default), display the strings specified by the range #i:.

`plot string #i: text #a` : Change the text of strings #i: to #a

`plot string #i1: col #i2` : Set the colours of strings #i1: to #i2

`plot string #i1: back #i2` : Set the background colour for the strings #i1: to the value #i2.

`plot string #i1: lw #i2` : Set the line weight of strings #i1: to #i2.

`plot string #i: fh #r` : Set the font height of strings #i to #r.

`plot string #i1: font #i2` : Set the font style of strings #i1 to #i2.

`plot string #i: x #r` : Set the x position of strings #i: to #r.

`plot string #i: y #r` : Set the y position of string #i: to #r.

`plot string #i: angle #r` : Set the angle of strings #i: to #r.

`plot string #i: fjust #r` : Controls justification of the strings #i: parallel to the specified edge of the viewport. If #r = 0.0, the left hand of the strings will be placed at the position specified by "x y" above; if #r = 0.5, the center of the strings will be placed at "x y", if #r = 1.0 the right hand of #i: will be placed at "x y". Other values can be used but are less useful.

`plot string #i: box #l` : If #l is true, plot a box around the text strings #i:. The default value is false (no box).

`plot string #i1: box lt #i2` : Set the line style of the box around the strings #i1: to the value #i2.

`plot string #i1: box lw #i2` : As above, but for the line weight specified by #i2.

`plot string #i1: box col #i2` : As above, but for the colour index for the box specified by #i2.

`plot set #i:` : Selects data set numbers as specified by #i:. Afterwards most plot commands will only affect data sets #i:

`plot set all` : Selects all data sets that are present. All subsequent plot commands will be executed for all data sets.

`plot line disp #l` : If #l is true, plots a connecting line through the data points, (default is false).

`plot line col #i` : Set the colour of the connecting line to #i.

`plot line lt #i` : Set the line style of the connecting line to #i.

`plot line lw #i` : Set the line weight of the connecting line to #i.

`plot line his #l` : If #l is true, plot the connecting line in histogram format (default is true).

`plot elin disp #l` : If #l is true, plots a connecting line through the end points of the error bars, (default depends upon the plot type).

`plot elin col #i` : Set the colour of the connecting line through the end points of the error bars to #i.

`plot elin lt #i` : Set the line style of the connecting line through the end points of the error bars to #i.

`plot elin lw #i` : Set the line weight of the connecting line through the end points of the error bars to #i.

`plot elin his #l` : If #l is true, plot the connecting line through the end points of the error bars in histogram format (default is true).

`plot model disp #l` : If #l is true, plot the current model corresponding to the relevant data set (default is true).

`plot model col #i` : Set the colour of the model to number #i.

`plot model lt #i` : Set the line style of the model to number #i.

`plot model lw #i` : Set the line weight of the model to number #i.

`plot model his #l` : If #l is true, plot the model in histogram format (default is true).

`plot back disp #l` : If #l is true, plot the subtracted background (default is true).

`plot back col #i` : Set the colour of the subtracted background to number #i.

`plot back lt #i` : Set the line style of the subtracted background to number #i.

`plot back lw #i` : Set the line weight of the subtracted background to number #i.

`plot back his #l` : If true, plot the subtracted background in histogram format (default is true).

`plot fill disp #l` : If #l is true, fill the curve below the model with the colour specified by the next command or the default colour.

`plot fill col #i` : Change the filling colour to #i.

`plot fill lt #i` : Change the line type of the filling lines to #i.

`plot fill lw #i` : Change the line weight of the filling lines to #i.

`plot fill style #i` : Change the style of the filling lines to the value #i. Here #i has values between 1–4, with the following meaning: 1 = solid filling (default), 2 = outline, 3 = hatched, 4 = cross-hatched.

`plot fill angle #r` : Set the angle for the filling lines for hatched filling. Default is 45 degrees.

`plot fill sep #r` : Set the distance between the filling lines for hatched filling. The unit spacing is 1 % of the smaller of the height or width of the viewing surface. This should not be zero.

`plot fill phase #r` : The phase between the hatch lines that fill the area.

`plot data disp #l` : If #l is true, display the data.

`plot data errx #l` : If #l is true, display the error bars in the x-direction.

`plot data erry #l` : If #l is true, display the error bars in the y-direction.

`plot data col #i` : Give the data colour index #i.

`plot data lt #i` : Give the data line style #i.

`plot data lw #i` : Give the data line weight #i.

`plot data fh #r` : Give the symbols for the data font height #r.

`plot data symbol #i` : Plot the data with symbol number #i. For symbol numbers, see *Plot symbols* (page 129).

`plot adum #a [overwrite] [append]` : Dump the data and model in the plot in an ascii file with filename #a. The extension ".qdp" will automatically be appended. Note that the data will be written as they are, i.e. if you have a logarithmic x-axis or y-axis, the logs of the plotted quantities will be written. If you want to replot your data later with for example the qdp package, take care that you plot the data in SPEX on a lin-lin frame before you execute the "plot adum" command. Also note that the data will be written in the units that were specified in the plot (energy, wavelength or whatever is applicable). The output format is described in *Plot asciidump file format* (page 134). If the optional "append" keyword is present, the data will be appended to any existing file with the name #a; if the optional "overwrite" keyword is present, any pre-existing file with the name #a will be overwritten by the new data.

Examples

`plot device xs` : Open the graphic device xs (xserver)
`plot device ps myplot.ps` : Select a postscript device connected to the file name myplot.ps
`plot type data` : Plot the data on the selected graphics device(s)
`plot ux angstrom` : Set the x-axis plot units to Å
`plot uy angstrom` : Set the y-axis plot units to Counts/s/Å
`plot frame new` : Open a new frame in the selected graphics device(s)
`plot frame 2` : Go to the 2nd frame, all plot commands will now only affect frame 2
`plot type chi` : Plot the residuals in frame 2
`plot uy rel` : Set the y-axis plot units in frame 2 to (Observed - Model)/Model
`plot view default f` : Set the default viewport keyword to false so that new user viewport values can be specified for frame 2
`plot view y 0.2:0.8` : Set the y viewport limits of frame 2 from 0.2 to 0.8 of the full device window
`plot cap id disp f` : Do not display the id caption of frame 2
`plot cap ut disp f` : Do not display the upper top caption of frame 2
`plot cap lt disp f` : Do not display the lower top caption of frame 2
`plot ux a` : Set the x-axis plot units of frame 2 to Å
`plot ux 21.602 ang` : Plot the x-axis as velocity in km^{-1} relative to a wavelength of 21.602 Å.
`plot ry -1:1` : Set the y-axis plot range of frame 2 to between a lower limit of -1 and an upper limit of 1
`plot frame 1` : Go to frame 1
`plot view default f` : Set the default viewport keyword to false so that new user viewport values can be specified for frame 1
`plot view x 0.25:0.75` : Set the x viewport limits of frame 1 from 0.25 to 0.75 of the full device window
`plot de cps filename.ps` : Open a colour postscript graphics device and write the output file to filename.ps
`plot` : Redraw the plot on all frames and devices
`plot close 2` : Close device number 2, which is the postscript device in this case

3.1.23 Quit: finish the program

The quit option exits the execution of SPEX, closes the open plot-devices and scratch files (if any) and, if requested outputs the cpu-time statistics.

Syntax

The following syntax rule applies:

`quit` : quit the program as described above.

3.1.24 Sector: creating, copying and deleting of a sector

Overview

This allows one to create, delete, copy, and show the number of sectors, used for the analysis of the data. Sectors are designed to allow modeling of different sources of radiation with its own spectral model. This way, common components can be fit simultaneously, while the observed spectra are of a different origin. One can create a sector for another (unresolved) source in the spectrum, for modeling particle background, or for spectra extracted in a different time interval for a time variable source. See for more details about sectors and regions in the Section *Sectors and regions* (page 285).

For doing spectral fitting of data sets, the sectors need to be specified in the response matrix of the data: the response file should tell which sector number corresponds to a given part of the matrix.

The sector command features also a spectral dump mode (adump) that writes the model spectrum to an ascii file, formatted such that it is suited for the SPEX file model. The first line of the output file is an integer showing the number of bins, and the following lines show the energy bin centroid and the luminosity in 10^{44} ph/s/keV.

Syntax

The following syntax rules apply:

`sector new` : Creates a new sector, which can have its own model.
`sector show` : Gives the number of sectors that are currently used.
`sector copy #i` : Copies the model for sector #i to a new sector.
`sector delete #i` : Deletes sector #i.
`sector adump #i #a overwrite` : Writes the model spectrum of sector number #i to an ASCII file with name #a.

Examples

`sector new` : Creates a new sector.
`sector copy 2` : Creates a new sector, with the same spectral model as used in sector 2. This can be useful if the spectra of the different sectors are very similar in composition.
`sector delete 3` : Deletes sector number 3.
`sector adump 1 model.txt` : Dumps the spectrum in sector 1 to model.txt

3.1.25 Shiftplot: shift the plotted spectrum for display purposes

Overview

This command shifts the observed spectrum as well as the model spectrum by adding a constant or by multiplying a constant, as far as plots are concerned. The true data files do not change, so the shift is only for representational purposes.

There are basically two options, indicated by the mode variable *plotshift*: For *plotshift*=1, a constant *shift* is added to the plotted spectrum of a given part of the data, for *plotshift*=2 the plotted spectrum is multiplied by the constant *shift*.

The multiplicative constant shift (*plotshift*=2) is generally preferred for log-log plots, while for linear plots a constant additive shift (*plotshift*=1) is preferred.

Warning: In the case of addition (*plotshift*=1), the addition constant is given in counts/s. This thus leads to a different (energy-dependent) constant being added to the plotted spectrum if the units are not in counts/s. For the multiplicative case this is of course not the case.

Syntax

The following syntax rules apply:

`shiftplot #i #r` : #i indicates whether a constant should be added (#i=1) or multiplied (#i=2) to the spectrum and model. #r is then the constant to be added or multiplied by.
`shiftplot [instrument #i1:] [region #i2:] #i3 #r` : Shift the plot using a factor #r. Here #i3 (= *plotshift*) determines whether the constant is added (*plotshift=1*) or multiplied (*plotshift=2*). The optional instrument range #i1: can be used to select a single instrument or range of instruments for which the shift needs to be applied, and the optional region range #i2: the region for these instruments.

Examples

`shiftplot 1 2.0` : Adds 2 counts/s to the plotted spectrum. Since no instrument or region is specified, this applies for all spectra.
`shiftplot 2 10.0` : Multiplies the plotted spectrum by a factor of 10.0
`shiftplot instrument 1 region 2 1 2` : Adds 2 counts/s to the plotted spectrum for the data from instrument 1 and region 2.
`shiftplot region 2:3 1 2` : Adds 2 counts/s to the plotted spectrum for the data for all instruments and regions 2−3.

3.1.26 Simulate: Simulation of data

Overview

This command is used for spectral simulations. The user should start with a spectral model and a spectral data set (both matrix and spectrum). After giving the "simulate" command, the observed spectrum will be replaced by the simulated spectrum in SPEX. Note that the original spectrum file (with the .spo extension) is not overwritten by this command, so no fear to destroy your input data!

Different options exist and several parameters can be set:

- Instrument, region: the instrument(s) and region(s) for which the simulation should be done (i.e., if you have more instruments you can do the simulation only for one or a few instruments).
- time: set the exposure time t (s) of the source spectrum as well as the background error scale factor f_b. This last option allows you to see what happens if for example the background would have a ten times higher accuracy (for $f_b = 0.1$).
- syserr: add a systematic error to both the source and background spectrum. An alternative way to introduce systematic errors is of course to use the syserr command (*Syserr: systematic errors* (page 116)). Take care not to set the systematic errors twice, and remember that rebinning your spectrum later will reduce the systematic errors, as these will be added in quadrature to the statistical errors. So first rebin and then add systematics!
- noise: either randomize your data or just calculate the expected values.
- bnoise: randomize your background model (generally not recommended to do).
- seed: set the random seed either to a specific number or generate it from the system clock. By default, SPEX initializes the random number generator based on the system clock, but through this command a specific seed can be set. The command will show the seeds used for the maximum number of expected threads.

Warning: A response matrix and spectrum of the region and the instrument you want to simulate are necessary, because SPEX needs the response matrix as well as the background to be subtracted for the simulations.

Warning: When you include systematic errors in the simulation (by putting the "syserr" to non-zero values), you cannot use anymore Poissonian statistics hence the C-stat for fitting, but you have to use the "fit meth chi" to use Gaussian errors and χ^2-fitting, with all thr disadvantages of that.

Warning: When you use bnoise=true, your subtracted background (the scaled background from the background region) will be randomized, and the pure C-stat cannot be used; the W-stat can be used as alternative but has serious drawbacks and is not recommended to be used).

Warning: (obsolete) If your background is taken from the same observation as your source, and you multiply the original exposure time with a factor of S, you should put f_b to $S^{-0.5}$, reflecting the fact that with increasing source statistics the background statistics also improves. This is the case for an imaging observation where a part of the image is used to determine the background. If instead you use a deep field to subtract the background, then the exposure time of your background will probably not change and you can safely put $f_b = 1$ for any exposure time t.

Warning: (obsolete) If your subtracted background (one of the columns in the .spo file) is derived from a low statistics Poissonian variable (for example a measured count rate with few counts per channel), then scaling the background is slightly inaccurate as it takes the observed instead of expected number of background counts as the starting point for the simulation.

Syntax

Note that only the "simulate" keyword followed by a number (the exposure time will do the actual simulation. All other syntax rules just set some parameters. The following syntax rules apply:

`simulate #r` : Does the simulation, with #r the exposure time t in seconds.

`simulate instrument #i1` : Specify the instrument (range) to be used in the simulation. Default values are 1 (just the first instrument).

`simulate region #i1` : Specify the region (range) to be used in the simulation. Default values are 1 (just the first region). For simulating everything you have, you can put this range to a large value: the simulation will simply ignore non-existent regions. If you use complex settings, like only region 3 for instrument 1 and region 2 for instrument 2, you may have to run the simulation separately for each entity.

`simulate syserr #r1 #r2` : Specify the systematic errors as a fraction of the source and background spectrum, respectively; both should be specified together. Default values are 0.

`simulate noise #l` : If #l is true, Poissonian noise will be added (this is the default).

`simulate bnoise #l` : If #l is false, no Poissonian noise will be added to the model background (this is the default).

`simulate seed #i` : Set random seed to a particular number.

`simulate seed random` : Set random seed randomly based on system clock.

Examples

`simulate 10000.` : This simulates a new spectrum/dataset with 10 000 s exposure time.
`simulate noise f` : Set simulation flag to simulate without Poissonian noise. The nominal error bars will still plotted.
`simulate syserr 0.1 0.2` : Set simulation for a systematic error of 10 % of the source spectrum and 20 % of the subtracted background spectrum added in quadrature.
`simulate instrument 2:4` : Set simulation for a spectrum for instruments 2–4 only
`simulate region 2` : Set simulation for only region 2 (of every instrument involved
`simulate seed 2` : Set random number seed to 2.

3.1.27 Step: Grid search for spectral fits

Overview

A grid search can be performed of χ^2 versus 1, 2, 3 or 4 parameters. The minimum, maximum and number of steps for each parameter may be adjusted. Steps may be linear or logarithmic. For each set of parameters, a spectral fit is made, using the last spectral fit executed before this command as the starting point. For each step, the parameters and χ^2 are displayed. This option is useful in case of doubt about the position of the best fit in the parameter space, or in cases where the usual error search is complicated. Further it can be useful incases of complicated correlations between parameters.

Warning: Take care to do a spectral fit first!

Warning: Beware of the cpu-time if you have a fine grid or many dimensions!

Syntax

The following syntax rules apply:

`step dimension #i` : Define the number of axes for the search (between 1–4).
`step axis #i1 parameter [[#i2] #i3] #a range #r1:#r2 n #i4` : Set for axis #i1, optional sector #i2, optional component #i3 and parameter with name #a the range to the number specified by #r1 and #r2, with a number of steps n given by #i4. If $n > 0$, a linear mesh between #r1 and #r2 will be taken, for $n < 0$, a logarithmic grid will be taken.
`step` : Do the step search. Take care to do first the "step dimension" command, followed by as many "step axis" commands as you entered the number of dimensions.
`step file example` : Do the step search and also write the results to an .stp file used by the stepcontour program (see *Stepcontour* (page 186)).

Warning: The step file command will overwrite existing .stp files with the same name by default.

Examples

Below we give a worked out example for a CIE model where we took as free parameters the normalization "norm", temperature "t", Si abundance "14" and Fe abundance "26". To do a 3-dimensional grid search on the temperature (logarithmic between 0.1 and 10 keV with 21 steps), Fe abundace (linear between 0.0 and 1.0 with 11 steps) and Si abundance (linear between 0.0 and 2.0 with 5 steps, i.e. values 0.0, 0.4, 0.8, 1.2, 1.6 and 2.0) the following commands should be issued:

`fit` : Do not forget to do first a fit.
`step dimension 3` : We will do a three-dimensional grid search
`step axis 1 parameter 1 1 t range 0.1:10.0 n -21` : The logarithmic grid for the first axis, temperature; note the − before the 21!
`step axis 2 par 26 range 0:1 n 11` : The grid for axis 2, Fe abundance.
`step axis 3 par 14 range 0:2 n 5` : Idem for the 3rd axis, Silicon.
`step` : Now do the grid search. Output is in the form of ascii data on your screen (and/or output file if you opened one).
`step file example` : Grid search with output to your screen and .stp output file for use with the stepcontour task.

3.1.28 Syserr: systematic errors

Overview

This command calculates a new error, adding the systematic error of the source and the background to the Poissonian error in quadrature. One must specify both the systematic error as a fraction of the source spectrum as well as of the subtracted background spectrum. The total of these fractions can either be less or greater than 1.

Warning: This command mixes two fundamentally different types of errors: statistical (random fluctuations) and systematic (offset). The resulting uncertainties are unjustly treated as being statistical, which can lead to wrong results when the systematic offsets are substantial. Syserr should therefore be used with extreme caution.

Warning: One should first rebin the data, before running syserr. Run syserr however before fitting the data or finding errors on the fit.

Warning: Running syserr multiple times will increase the error every time. If the input to syserr is wrong one should restart SPEX and rerun syserr with the correct values to calculate the total error correctly.

Syntax

The following syntax rules apply:

`syserr #i: #r1 #r2` : The shortest version of this command. #i: is the range in data channels for which the systematic error is to be calculated and added (in quadrature) to the Poissonian error. #r1 is then the the relative systematic error due to the source and #r2 the relative systematic error due to the background.
`syserr [instrument #i1:] [region #i2:] #i3: #r1 #r2` : In this syntax one can also specify the instrument and the region one wants to calculate the combined error for. Both can be ranges as well. #i3: has the same role as #i: in the above command, and #r1 and #r2 are the same as above.

`syserr [instrument #i1:] [region #i2:] #i3: #r1 #r2 [unit #a]` : Exact same command as above, except that now the data range (#i3:) for which the errors are to be calculated are given in units different than data channels. These units can be Å (ang), eV (ev), keV (kev), Rydbergs (ryd), Joules (j), Hertz (hz) and nanometers (nm). This is the most general command.

Examples

`syserr 1:100000 0.3 0.5` : Calculates the combined Poissonian and systematic error for data channels 1:100000, where the fraction of the systematic error of the source is 0.3 and the background is 0.5.

`syserr 0:2000 0.3 0.5 unit ev` : The same as the above command, expect that now the error calculation is performed between 0 and 2000 eV instead of data channels.

`syserr instrument 2 region 1 0:2000 0.3 0.5 unit ev` : The same as the above command, but now the error calculation is only performed for the data set from the second instrument and the first region thereof.

3.1.29 System: call system executables

Overview

Sometimes it can be handy if SPEX interacts with the computer system, for example if you run it in command mode. You might want to check the existence of certain file, or run other programs to produce output for you, and depending on that output you want to continue SPEX.

Therefore there is an option to execute any shell type commands on your machine, using the fortran "call system" subroutine.

Another useful goody is the possibility to stop SPEX automatically if you find some condition to occur; this might be useful for example if you have a program running that calls SPEX, and depending on the outcome of SPEX you might want to terminate the execution. This is achieved in SPEX by testing for the existence of a file with a given filename; if the file exists, SPEX stops immediately execution and terminates; if the file does not exist, SPEX continues normally.

Syntax

The following syntax rules apply:

`system exe #a` : execute the command #a on your UNIX/linux shell.
`system stop #a` : stop SPEX if the file #a exists.

Examples

`system exe "ls -l"` : give a listing of all file names with length in the current directory.
`system exe "myfortranprogram"` : execute the fortran program with name "myfortranprogram".
`system stop testfile` : stop SPEX if the file with name testfile exists; if this file does not exist, continue with SPEX.

3.1.30 Use: reuse part of the spectrum

Overview

This command can only be used after the ignore command was used to block out part of the data set in the fitting. The command re-includes thus (part of) the data set for fitting. As it undoes what the ignore command did (see *Ignore: ignoring part of the spectrum* (page 95)) the syntax and use of both are very similar. Again one can/must specify the instrument and region for one wants the data to be re-included. Further can one specify the units chosen to give the range of data points to be included. The standard unit is that of data channels and does not need to be specified. The data points re-included with use will automatically also be plotted again.

The range to be reused can be specified either as a channel range (no units required) or in eiter any of the following units: keV, eV, Rydberg, Joule, Hertz, Å, nanometer, with the following abbrevations: kev, ev, ryd, j, hz, ang, nm. Note that spectra that were binned before using the bin command are not binned anymore after the spectrum is ignored and reused again. In this case, the bin command should be applied again to restore the desired spectral binning.

Syntax

The following syntax rules apply:

`use [instrument #i1] [region #i2] #r` : Use the range #r in fitting the data. The instrument #i1 and the region #i2 must be specified if either there are more than 1 instrument or more than 1 region used in the data sets.

`use [instrument #i1] [region #i2] #r unit #a` : Same as the above, but now the units #a in which the range #r of data points to be used is specified in any of the units mentioned above.

Examples

`use 1000:1500` : Re-include the data channels 1000:1500 for fitting the data.

`use instrument 1 region 1 1000:1500` : Re-include the data channels 1000:1500 of the first instrument and the first region in the data set for fitting the data and plotting.

`use instrument 1 region 1 1:1.5 unit kev` : Same as the above, but now the unites are specified as keV instead of in data channels.

`use 1000:1500 unit ev` : Re-include the data points between 1000 and 1500 eV for fitting and plotting. Note that in this case the input data should come from only one instrument and should contain no regions.

3.1.31 Var: various settings for the plasma models

Overview

For the plasma models, there are several quantities that have useful default values but can be adjusted manually for specific purposes. We list them below.

Free-bound emission

Usually the freebound emission takes most of the computing time for the plasma models. This is because it needs to be evaluated for all energies, ions and atomic sublevels that are relevant. In order to reduce the computational burden, there is a parameter gfbacc in SPEX that is approximately the accuracy level for the free-bound calculation. The default value is 10^{-3}. If this quantity is higher, less shells are taken into account and the computation proceeds faster (but less accurate). The users are advised not to change this parameter unless they are knowing what they do!

Line emission contributions

By default, all possible line emission processes are taken into account in the plasma models. For testing purposes, it is possible to include or exclude specific contributions. These are listed below in the table below.

Table 7: Possible line emission processes

Abbrevation	Process
ex	electron excitation
rr	radiative recombination
dr	dielectronic recombination
ds	dielectronic recombination satellites
ii	inner shell ionisation

Doppler broadening

By default, thermal Doppler broadening is taken into account. However, the user can put this off for testing purposes. The options for the broadening are:

1. No broadening at all
2. Only Doppler broadening (default)
3. Only natural broadening (works only for *var calc new*)
4. Doppler and natural broadening, Voigt profiles (best physical representation, works only for *var calc new*)

Atomic data

The user can choose between the "old" Mekal code (the current default, also referred to as SPEXACT v2) and updated calculations with the command `var calc new` (referred to as SPEXACT v3).

Calculation level occupations

In the new line calculations, the occupation of the different atomic energy levels is determined self-consistently. Consider an ion for which we want to determine these occupations. If the occupation of the different levels of all neighbouring ions is known, then given our atomic database we know all the rates we need to calculate in one step the level occupations of our ion. However, our initial guess of the occupation of the neighbours may have been not correct, therefore the rates are not correct, and we need to iterate a few times before all ions of an element are converged. This usually takes between 2 and about 25 iterations, depending on the precise physical conditions.

As starting values we assume that all ions are in the ground state, and then we iterate (this is the default mode). By setting `var occstart boltzmann`, we can change this to an initial Boltzmann distribution for the occupation. This may be more appropriate for very high density plasmas. To reduce the number of iterations, one may also start with `var occstart last`, in which case the results from the last calculation are used as starting values. This can be useful for spectral fitting and/or error searches, where for each call the parameters of the model are close to those for the previous call.

Warning: The `var occstart last` option will have limited advantage if the model during a fit is far off from the final values (large parameter changes during fitting), or with multiple components (in which case the last parameters stored in the newlin subroutine may be associated to a different spectral component compared to the one studied).

Mekal code

Over the years, we have made several minor improvements to the original Mekal plasma model. These improvements are included by default. However, it is possible to discard some of these improvements by going back to the old code. The improvements are classified as follows (with the appropriate syntax word added here).

`Wav`: wavelength corrections according to the work of Phillips et al. (1999), based upon Solar flare spectra; in addition, the 1s-np wavelengths of Ni XXVII and Ni XXVIII have been improved.

`Fe17`: The strongest Fe XVII lines are corrected in order to agree with the rates as calculated by Doron & Behar (2002).

`Update`: several minor corrections: the Si IX C7 line has been deleted; the Si VIII N6 line now only has the 319.83 Å line strength instead of the total triplet strength; the Ni XVIII and Fe XVI Na1A and NA1B lines plus their satellites have been deleted.

Cooling by collisional excitation

In the oldest version of the *pion* model, for the cooling by collisional excitation the Mekal code was used, with all processes except for collisonal excitation on. This has now improved by using a mixture of more modern data for collisional excitation taken from our new SPEX calculations, or for ions for which no new data are yet available, from the Chianti database, and in a few rare cases we still use the old Mekal data (4 ions only). For a full description see Stofanova et al. (2020, subm.). This new calculation is now the default, but the user can switch between both by selecting the `var newcoolexc #l` option.

Cooling by dielectronic recombination

In the oldest version of the *pion* model, cooling by di-electronic recombination was not included. This has now been corrected. Wherever available, cooling rates are calculated directly from the di-electronic recombination model from the latest model, for those ions for which Auger rates and associated energies are available. For all other ions, we use the old calculations from the Mekal code (with only the *dr* and *ds* flags turned on).

The user has the option, however, to use only Mekal data (for comparison) by setting the command `var newcooldr false`.

Syntax

The following syntax rules apply:

`var gacc #r` : Set the accuracy gfbacc for free-bound emission. Default is 10^{-3}, maximum value 1 and minimum value 0. Do not change if you do not know what it does.

`var gacc reset` : Reset gfbacc to its default value.

`var line #a #l` : For process #a (where #a is one of the abbreviations in *the Table above* (page **??**)) the process is allowed (if #l is true) or disabled (if #l is false). By default, all processes are allowed.

`var line reset` : Enable all line emission processes

`var line show` : Show the status of the line emission processses

`var doppler #i` : Line broadening, see the four allowed values in the above description

`var calc old` : Use the old Mekal code

`var calc new` : Use the new updated atomic data (for SPEX version 3.0 and higher)
`var occstart ground` : Start new line calculation iteration with initial guess that all ions are in the ground state. This is the default
`var occstart boltzmann` : Start new line calculation iteration with initial guess that all levels have a Boltzmann distribution.
`var occstart boltzmann` : Start new line calculation iteration with initial guess that all levels have same occupation as in last call to this routine.
`var newmekal wav #l` : if true (the default), use the updated wavelengths for the Mekal code
`var newmekal fe17 #l` : if true (the default), use the updated Fe XVII calculations for the Mekal code
`var newmekal update #l` : if true (the default), use the updates for some lines for the Mekal code
`var newmekal all #l` : if true (default), use all the above three corrections for the Mekal code
`var ibalmaxw #l` : if true use multi-Maxwellians (if relevant) for both the ionisation balance and the spectrum (default); if false, only use it for the spectrum.
`var newcoolexc #l` : if true (default), use the latest cooling by collisional excitation calculations
`var newcooldr #l` : if true (default), use the latest cooling by dielectronic recombination calculations

Examples

`var gacc 0.01` : Set the accuracy gfbacc for free-bound emission.to 0.01
`var gacc reset` : Reset the accuracy gfbacc for free-bound emission.to its default value of 0.001
`var line ex f` : Exclude electron excitation
`var line ds t` : Include dielectronic satellites
`var line reset` : Include all line emission processes
`var line show` : Show status of all line emission proceses
`var doppler f` : Do not use thermal Doppler bvroadening
`var calc new` : Use the new atomic data (EXPERIMENTAL)
`var calc boltzmann` : Start level occupation calculations with Boltzmann distribution
`var newmekal wav f` : Use the original Mekal wavelengths instead
`var newmekal fe17 t` : Use the updated Fe XVII calculations
`var newmekal all f` : Go back to the full old Mekal code
`var newmekal all t` : Take the full updated Mekal code
`var ibalmaxw f` : Do not use Multi-Maxwellians for the ionisation balance
`var newcoolexc f` : Change to the old collisional excitation cooling calculations
`var newcooldr f` : Change to the old dielectronic recombination cooling calculations

3.1.32 Vbin: variable rebinning of the data

Overview

This command rebins (a part of) the data (thus both the spectrum and the response) such that the signal to noise ratio in that part of the spectrum is at least a given constant. Thus instead of taking a fixed number of data channels for binning, the number of data channels binned is here variable as is the bin width. On top of specifying the S/N ratio, you also specify the minimal bin width. This is useful for rebinning low count data, which have no strong or sharp features or lines in it.

Warning: For high resolution spectra with sharp and strong lines, this binning can lead to very wrong results. In this case either the emission lines or the continuum (in case of absorption lines) have a much higher signal to noise ratio than the other component. As a result this function will try to bin the line and continuum together, resulting in a smoothing out of the lines.

Warning: It is not always guaranteed that the minimum signal-to-noise ratio is obtained in all channels. This is an effect of the applied algorithm. Channels with the highest S/N ratio and neighboring bins are merged until sufficient S/N ratio is obtained. This process is continued for the remaining number of bins. At the end of the process a few bins with a low S/N ratio will remain. These are merged with their neighbors, resulting in a possibly lower S/N ratio for that bin.

Syntax

The following syntax rules apply:

`vbin #i1: #i2 #r` : Simplest command allowed. #i1: is the range in data channels over which the binning needs to take place. #i2 in the minimum amount of data channels that need to be binned, and #r is the minimal S/N ratio one wants the complete data set to have.

`vbin #r1: #i #r2 unit #a` : The same command as above, except that now the ranges over which the data is to be binned (#r1:) are specified in units (#a) different from data channels. These units can be eV, keV, Å, as well as in units of Rydberg (ryd), Joules (j), Hertz (hz) and nanometers (nm).

`vbin [instrument #i1:] [region #i2:] #i3: #i4 #r` : Here #i3: and #i4 are the same as #i1: and #i2 in the first command, also #r is the minimal S/N ratio required. However, here one can specify the instrument range #i1: and the region range #i2: as well, so that the binning only is performed for a certain data set.

`vbin [instrument #i1:] [region #i2:] #r1: #i2 #r2 [unit #a]` : This command is the same as the above, except that here one can specify the range over which the binning should occur in the units specified by #a. These units can be eV, Å, keV, as well as in units of Rydberg (ryd), Joules (j), Hertz (hz) and nanometers (nm).

Examples

`vbin 1:10000 3 10` : Bins the data channels 1:10000 with a minimal bin width of 3 data channels, and a minimum S/N ratio of 10.

`vbin 1:4000 3 10 unit ev` : Does the same as the above, but now the data range to be binned is given in eV, from 1−4000 eV, instead of in data channels.

`vbin instrument 1 region 1 1:19 3 10 unit a` : Bins the data from instrument 1 and region 1 between 1 and 19 Å with the same minimum bin width and S/N as above.

3.1.33 Watch: track time and subroutines

Overview

If you have any problems with the execution of SPEX, you may set the watch-options. For example, if the computation time needed for your model is very large, it could be wise to check which parts of the program consume most of the cpu-time. You might then hope to improve the performance by assuming other parameters or by re-structuring a part of the code. In this case, you set the "time" flag to true (see below), and at the end of the program you will see how much time was spent in the most important subroutines.

Another case occurs in the unlikely case that SPEX crashes. In that case it is recommended to re-execute the program, saving all commands onto a log-file and use the "sub" flag to report the entering and exiting of all major subroutines. This makes it more easy to find the source of the error.

Timing is done by the use of the stopwatch package by William F. Mitchell of the NIST, which is free available at the web. If the time flag is set to true, on exit SPEX will report for each subroutine the following execution times (in s):

- The user time, i.e. the cpu-time spent in this subroutine

- The system time, i.e. the overhead system time for this subroutine
- The wall time, i.e. the total time spent while executing this subroutine.

Also the totals for SPEX as a whole are given (this may be more then the sum of the subroutine components, since not all subroutines are listed separately; the wall time for SPEX as a whole also includes the time that the program was idle, waiting for input from the user).

Syntax

The following syntax rules apply:

`watch time #l` : set the "time" flag to true or false.
`watch sub #l` : set the flag that SPEX causes to report each major subroutine it enters or exits

Examples

`watch time t` : set the "time" flag to true
`watch sub f` : set the subroutine report flag to false

3.2 Plotting reference

3.2.1 Plot devices

The following devices are available (may depend somewhat on your local operating system, i.e. the allowed devices in your local pgplot installation):

1. `null` : Null device (no output)
2. `xwin` : Workstations running X Window System
3. `xser` : Persistent window on X Window System
4. `ps` : PostScript printers, monochrome, landscape
5. `vps` : Postscript printers, monochrome, portrait
6. `cps` : PostScript printers, color, landscape
7. `vcps` : PostScript printers, color, portrait
8. `gif` : GIF-format file, landscape
9. `vgif` : GIF-format file, portrait

3.2.2 Plot types

There are several plot types available in SPEX. Below we list them together with the acronym that should be used when defining this plot type (in a `plot type` command).

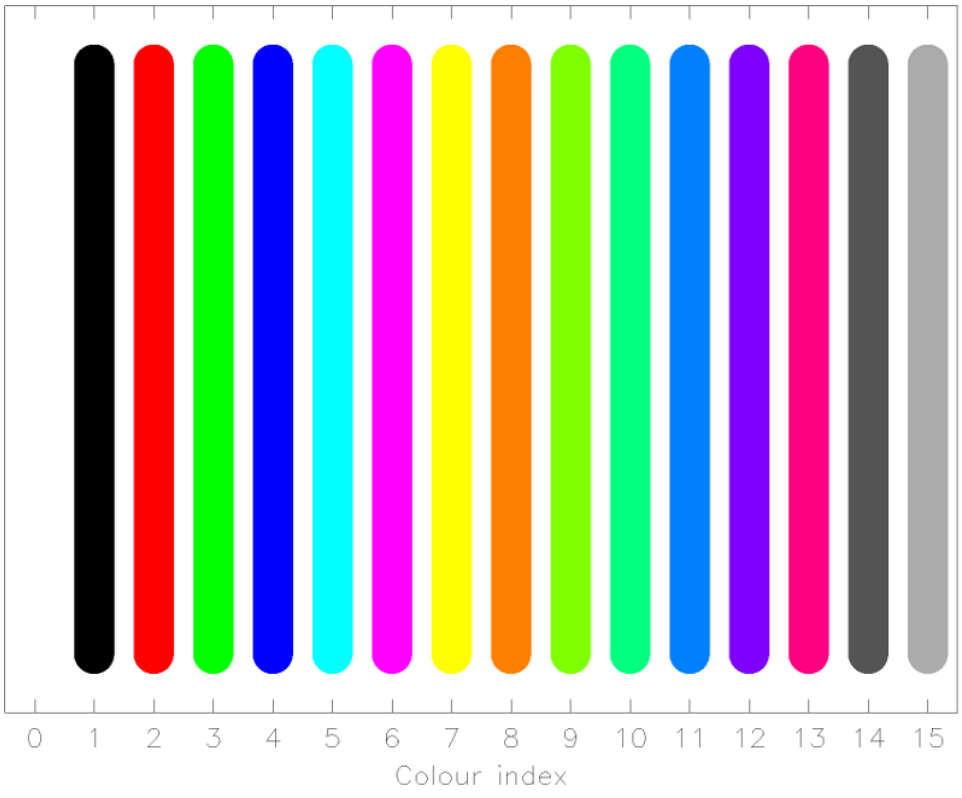
- `data` – a spectrum observed with an instrument plus optionally the folded (predicted) model spectrum and/or the subtracted background spectrum.
- `model` – the model spectrum, not convolved by any instrumental profile
- `area` – the effective area of the instrument. If the response matrix of the instrument contains more components, also the contribution of each component is plotted separately.

- `fwhm` – the peak energy of the response matrix (as the model) as well as the FWHM limits (as the data) are plotted. The FWHM limits are the energies at which the response reaches half of its maximum value. If for the y-axis the option "de/e" is chosen, all calculated energies are divided by the energy of the incoming photon.
- `chi` – the fit residuals, either expressed in units such as counts/s or the same units as the spectrum, or expressed in terms of number of standard deviations, or as a relative error.
- `dem` – a differential emission measure distribution.

3.2.3 Plot colours

The table below lists the plot colours that are available.

Value	Colour
00	Black (background)
01	White (default)
02	Red
03	Green
04	Blue
05	Cyan (Green + Blue)
06	Magenta (Red + Blue)
07	Yellow (Red + Green)
08	Orange (Red + Yellow)
09	Green + Yellow
10	Green + Cyan
11	Blue + Cyan
12	Blue + Magenta
13	Red + Magenta
14	Dark Grey
15	Light Gray



3.2.4 Plot line types

When drawing lines, the following line types are possible:

```
1. ----------  (Continuous line)
2. - - - - - -  (Dashed line)
3. .-.-.-.-.-.  (Dot-dashed line)
4. ..........  (Dotted line)
5. -..-..-..-.  (Dash-dotted line)
```

The first is the default value.

Further, the user can specify the thickness of the lines. Default value is 1, larger values give thicker lines. Maximum value is 200. Recommended value for papers: use line weight 3. For an example of different line weights, see also the plot below.

3.2.5 Plot text

In it is possible to draw text strings on the plot surface or near the frame of the plot. Several properties of the text can be adjusted, like the fontheight, font type, colour, orientation and location of the text. See the figure below for an example of possible fonts.

Font types (font)

The following fonts types are allowed (values between 1–4):

1. normal font (default)
2. roman font
3. *italic font*
4. script font

Font heights (fh)

The font height can be entered as a real number, the default value is 1. A useful value for getting results that are still readable for publications is 1.3. Note that if you make the font height for the captions of the figure too large, you may loose some text off the paper.

Special characters

The texts strings that are plotted may be modified using the pgplot escape sequences. These are character–sequences that are not plotted, but are interpreted as instructions to change the font, draw superscripts or subscripts, draw non-ASCII characters, Greek letters, etc. All escape sequences start with a backslash character (\). A list of the defined escape sequences is given in the first table below. A lookup table for Greek letters is presented in the table after that. Some useful non-ASCII characters are listed as well. The last table shows some examples of the use of pgplot escape sequences in character strings.

A list of available escape sequences.

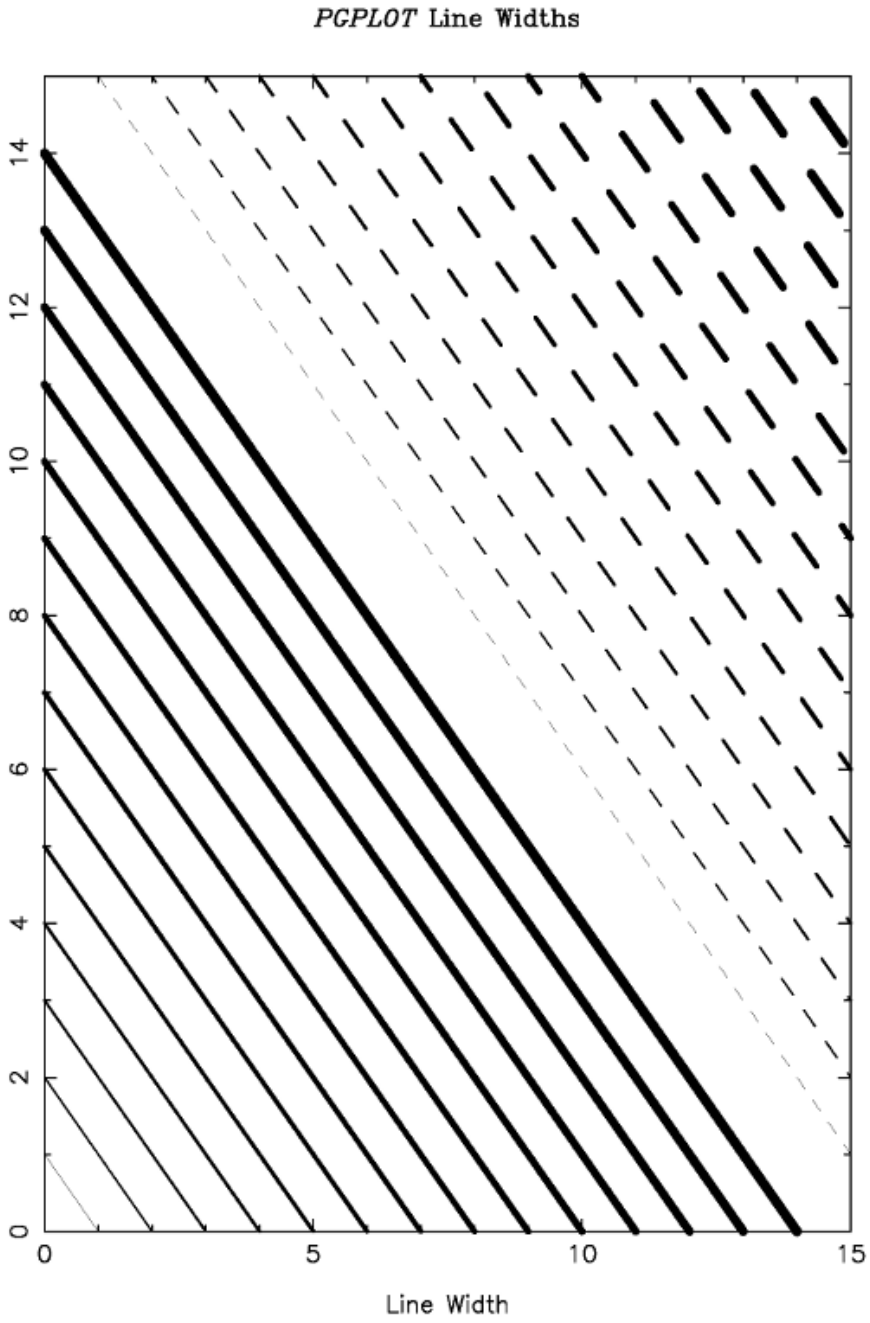


Fig. 2: Example of different line weights.

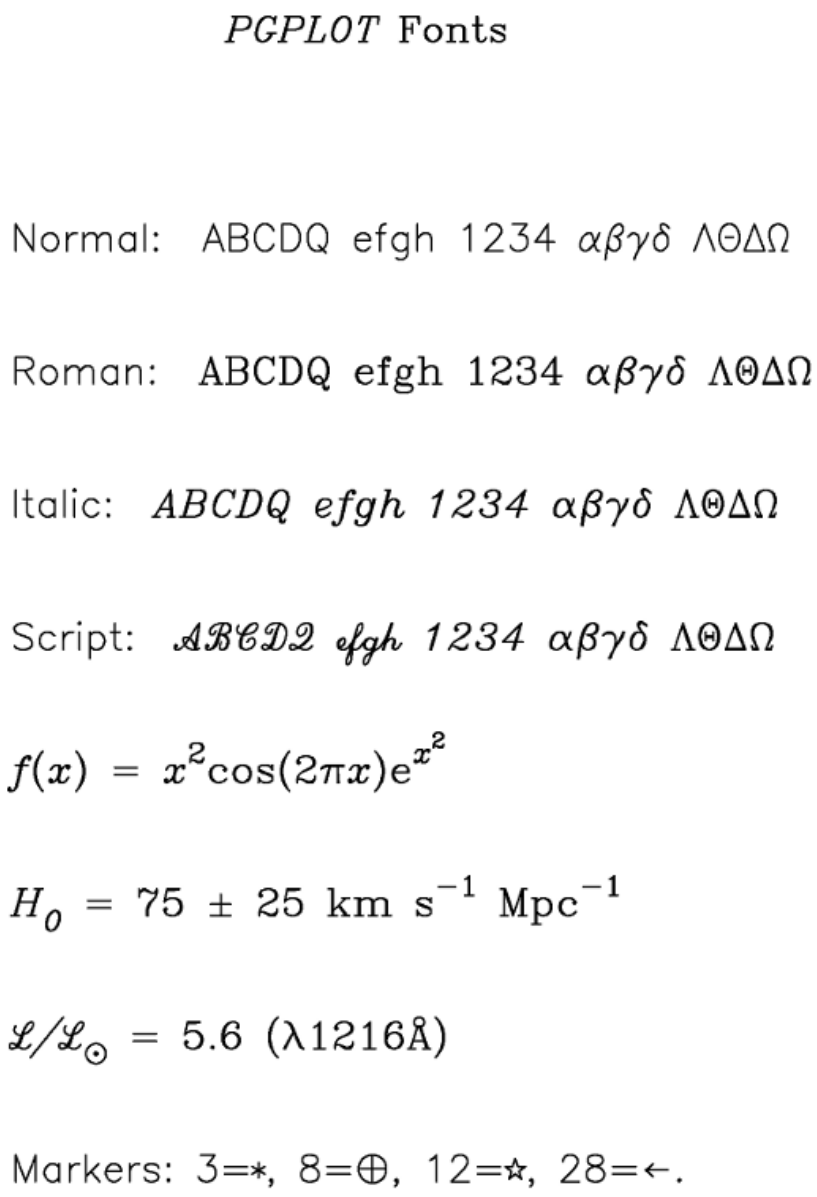


Fig. 3: Example of different fonts.

Seq.	Description
\u	Start a superscript or end a subscript. A \u must be ended by a \d!
\d	Start a subscript or end a superscript. A \d must be ended by a \u!
\\	Backspace (i.e. do not advance textpointer after plotting the previous character)
\A	Ångstrom symbol (Å)
\gx	Greek letter corresponding to roman letter x
\fn	Switch to Normal font
\fr	Switch to Roman font
\fi	Switch to Italic font
\fs	Switch to Script font
\(n)	Character number n, see *pgplot manual appendix B, tab. 1*

** List of upper- and lower case Greek letters (G) and their corresponding Roman letters (R).**

R :	A	B	G	D	E	Z	Y	H	I	K	L	M	N	C	O	P	R	S	T	U	F	X	Q	W
G :	A	B	Γ	Δ	E	Z	H	Θ	I	K	Λ	M	N	Ξ	O	Π	P	Σ	T	Υ	Φ	X	Ψ	Ω
R :	a	b	g	d	e	z	y	h	i	k	l	m	n	c	o	p	r	s	t	u	f	x	q	w
G :	α	β	γ	δ	ϵ	ζ	η	θ	ι	κ	λ	μ	ν	ξ	o	π	ρ	σ	τ	v	ϕ	χ	ψ	ω

$\sim$	2248	$\pm$	2233	∇	0583
$\approx$	0248	$\mp$	2234	$\surd$	2267
$\cong$	0250	$\times$	2235	$\int$	2268
$\propto$	2245	$\div$	2237	$\oint$	2269
$\neq$	2239	$\equiv$	2240	∞	2270
$\leq$	2243	†	2277	∂	2286
$\geq$	2244	‡	2278	$\odot$	2281

Displayed	**PGPLOT escape sequence**
$f(x) = x^2 \cos(2\pi x)$	\fif(x) = x\u2\d \frcos\fi(\fr2\fi\gpx)
$H_0 = 75 \pm 25$ km s^{-1} Mpc^{-1}	\fiH\d0\u\fr = 75 \(2233) 25 km s\u-1\d Mpc\u-1\d

3.2.6 Plot captions

SPEX has a set of captions defined for each plot. The properties of these captions as well as the text can be modified by the user. For the properties (font types etc.) see *Plot text* (page 125).

The following captions exist:

- x : the x-axis, plotted below the frame
- y : the y-axis, plotted left from the frame
- z : the z-axis (for contour plots and images), plotted at the upper left edge of the frame
- ut : the upper title text, plotted in the middle above the frame
- lt : the lower title text, plotted between the upper title text and the frame
- id : the plot identification, plotted to the upper right corner of the frame (usually contains the version and time and date).

With the "plot cap" series of commands, the text, appearance and position of all these captions can be modified.

3.2.7 Plot symbols

When plotting data, the data can be labeled with different symbols as indicated in the figure below.

3.2.8 Plot axis units and scales

Different units can be chosen for the quantities that are plotted. Which units are available depends upon the plot type that is used, for example an observed spectrum (data), a Differential Emission Measure distribution (DEM) etc. For the available plot types see Section *Plot types* (page 123).

Note also that depending upon the plot type, there are two or three axes available, the x-axis, y-axis and z-axis. The last axis is only used for contour maps or images, and corresponds to the quantity that is being plotted in the contours or image.

X-axis units

X-axis units for plot type model, data, chi, area, fwhm and spec

bin	Bin number
kev	keV (kilo-electronvolt), default value
ev	eV (electronvolt)
ryd	Rydberg units
hz	Hertz
ang	Å (Ångstrom)
nm	nm (nanometer)
vel	$\mathrm{km\ s^{-1}}$

Note: when using the velocity scale, negative values correspond to a blueshift, positive values to a redshift. Also note that in this case one needs to provide the reference energy or wavelength.

X-axis units for plot type dem

bin	Bin number
kev	keV (kilo-electronvolt), default value
ev	eV (electronvolt)
ryd	Rydberg units
k	K (Kelvin)
mk	MK (Mega-Kelvin)

Y-axis units

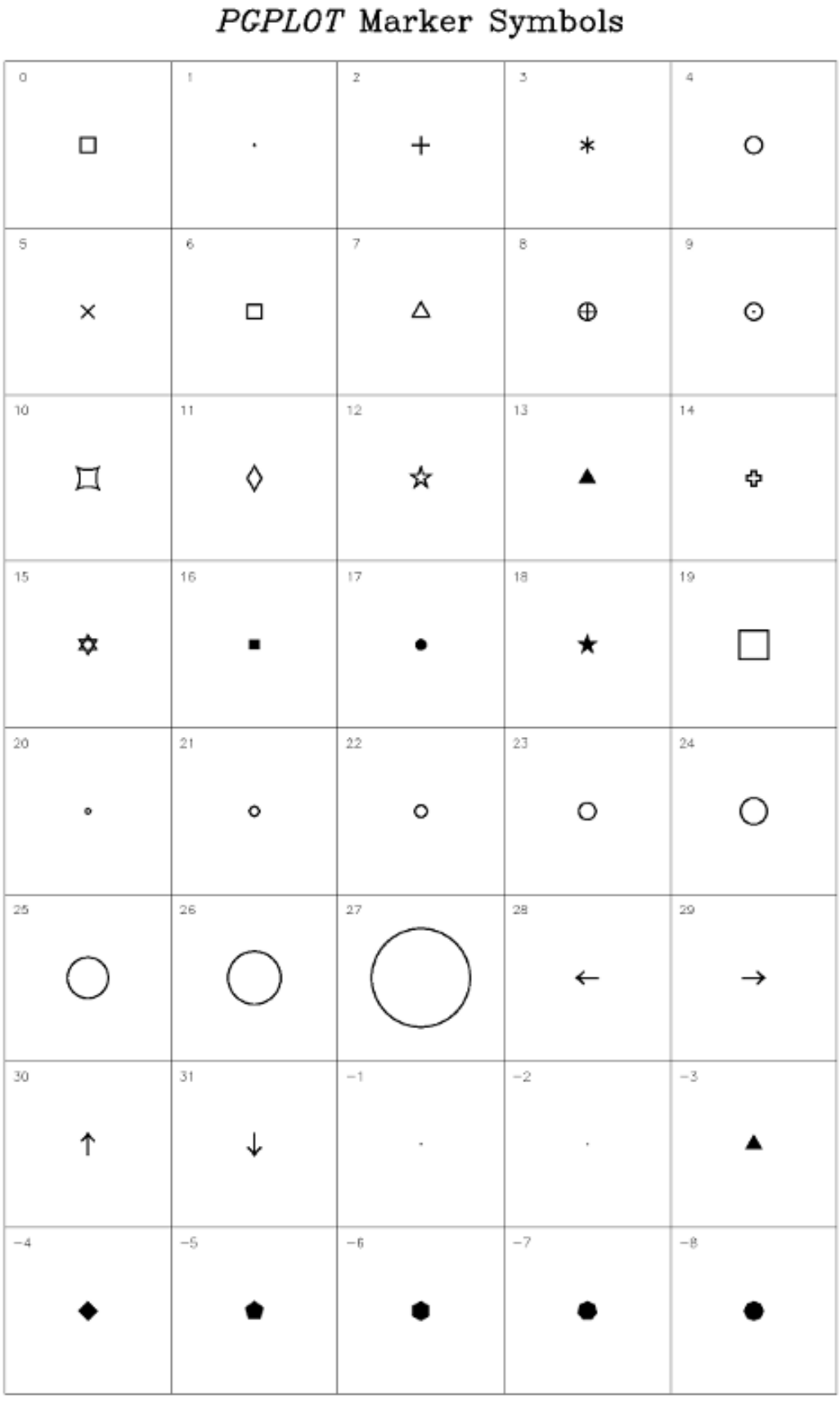


Fig. 4: Plot symbols that are available.

Y-axis units for plot type model, spec

	Photon numbers:
cou	Photons $\mathrm{m^{-2}s^{-1}bin^{-1}}$
kev	Photons $\mathrm{m^{-2}s^{-1}keV^{-1}}$ (default value)
ev	Photons $\mathrm{m^{-2}s^{-1}eV^{-1}}$
ryd	Photons $\mathrm{m^{-2}s^{-1}Ryd^{-1}}$
hz	Photons $\mathrm{m^{-2}s^{-1}Hz^{-1}}$
ang	Photons $\mathrm{m^{-2}s^{-1-1}}$
nm	Photons $\mathrm{m^{-2}s^{-1}nm^{-1}}$
	Energy units:
wkev	$\mathrm{W\ m^{-2}\ keV^{-1}}$
wev	$\mathrm{W\ m^{-2}\ eV^{-1}}$
wryd	$\mathrm{W\ m^{-2}\ Ryd^{-1}}$
whz	$\mathrm{W\ m^{-2}\ Hz^{-1}}$
wang	$\mathrm{W\ m^{-2\ -1}}$
wnm	$\mathrm{W\ m^{-2}\ nm^{-1}}$
jans	Jansky (10^{-26} $\mathrm{W\ m^{-2}\ Hz^{-1}}$)
	νF_ν units:
iw	$\mathrm{W\ m^{-2}}$
ij	Jansky Hz (10^{-26} $\mathrm{W\ m^{-2}}$)
	Various:
bin	Bin number

Y-axis units for plot type data

bin	Bin number
cou	Counts
cs	$\mathrm{Counts\ s^{-1}}$
kev	$\mathrm{Counts\ s^{-1}\ keV^{-1}}$ (default value)
ev	$\mathrm{Counts\ s^{-1}\ eV^{-1}}$
ryd	$\mathrm{Counts\ s^{-1}\ Ryd^{-1}}$
hz	$\mathrm{Counts\ s^{-1}\ Hz^{-1}}$
ang	$\mathrm{Counts\ s^{-1\ -1}}$
nm	$\mathrm{Counts\ s^{-1}\ nm^{-1}}$
fkev	$\mathrm{Counts\ m^{-2}\ s^{-1}\ keV^{-1}}$
fev	$\mathrm{Counts\ m^{-2}\ s^{-1}\ eV^{-1}}$
fryd	$\mathrm{Counts\ m^{-2}\ s^{-1}\ Ryd^{-1}}$
fhz	$\mathrm{Counts\ m^{-2}\ s^{-1}\ Hz^{-1}}$
fang	$\mathrm{Counts\ m^{-2}\ s^{-1\ -1}}$
fnm	$\mathrm{Counts\ m^{-2}\ s^{-1}\ nm^{-1}}$

Y-axis units for plot type chi

bin	Bin number
cou	Counts
cs	Counts s^{-1}
kev	Counts s^{-1} keV^{-1} (default value)
ev	Counts s^{-1} eV^{-1}
ryd	Counts s^{-1} Ryd^{-1}
hz	Counts s^{-1} Hz^{-1}
ang	Counts s^{-1} $^{-1}$
nm	Counts s^{-1} nm^{-1}
fkev	Counts m^{-2} s^{-1} keV^{-1}
fev	Counts m^{-2} s^{-1} eV^{-1}
fryd	Counts m^{-2} s^{-1} Ryd^{-1}
fhz	Counts m^{-2} s^{-1} Hz^{-1}
fang	Counts m^{-2} s^{-1} $^{-1}$
fnm	Counts m^{-2} s^{-1} nm^{-1}
dchi	(Observed - Model) / Error (default)
rel	(Observed - Model) / Model

Y-axis units for plot type area

bin	Bin number
m2	m^2 (default)
cm2	cm^2

Y-axis units for plot type fwhm

bin	Bin number
kev	keV (kilo-electronvolt), default value
ev	eV (electronvolt)
ryd	Rydberg units
hz	Hertz
ang	Å (Ångstrom)
nm	nm (nanometer)
de/e	$\Delta E/E$

Y-axis units for plot type dem

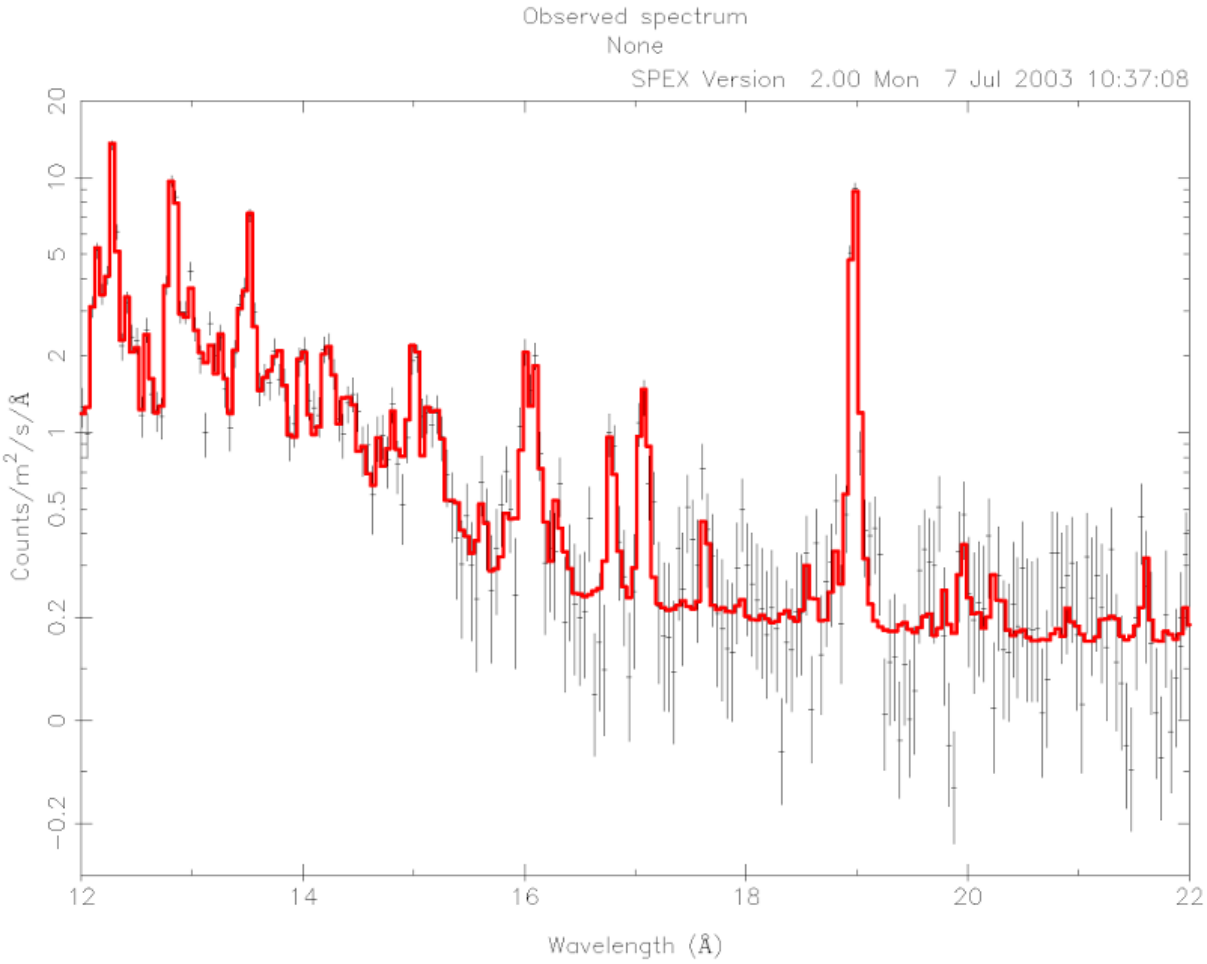
The emission measure Y is defined as $Y \equiv n_\mathrm{e} n_\mathrm{H} V$ and is expressed in units of 10^{64} m^{-3}. Here n_e is the electron density, n_e is the Hydrogen density and V the emitting volume.

bin	Bin number
em	Y (default)
demk	$\mathrm{d}Y/\mathrm{d}T$, per keV
demd	$\mathrm{d}Y/\mathrm{d}T$, per K

Plot axis scales

The axes in a plot can be plot in different ways, either linear or logarithmic.

For the y-axis we also have made an option for a mixed linear and logarithmic scale. The lower part of the plot will be on a linear scale, the upper part on a logarithmic scale. This option is useful for example for plotting a background-subtracted spectrum. Often there are bins with a low count rate which scatter around zero after background subtraction; for those channels a linear plot is most suited as a logarithmic scaling cannot deal with negative numbers. On the other hand, other parts of the spectrum may have a high count rate, on various intensity scales, and in that case a logarithmic representation might be optimal. The mixed scaling allows the user to choose below which y-value the plot will be linear, and also which fraction of the plotting surface the linear part will occupy. For an example see the figure below.



3.2.9 Plot asciidump file format

If one decides to save the plot as a file, then the `plot adum` command (*Plot: Plotting data and models* (page 106)) is used. This will save the spectrum in the QDP file format. Depending on the plot type, the number of columns and their content in the output QDP file changes. Note that the units of the output are the same as the ones that you set for the plot.

Plot type data

The `plot adum` output for the `data` plot type is:

Column	Value
1	Energy/wavelength center
2	Energy/wavelength negative distance until bin boundary
3	Energy/wavelength positive distance until bin boundary
4	Count rate center
5	Count rate negative error
6	Count rate positive error
7	Model value
8	Background value

Plot type chi

The `plot adum` output for the `chi` plot type is:

Column	Value
1	Energy/wavelength center
2	Energy/wavelength negative distance until bin boundary
3	Energy/wavelength positive distance until bin boundary
4	Difference
5	Difference negative error
6	Difference positive error
7	Zero level

Plot type model

The `plot adum` output for the `model` plot type is:

Column	Value
1	Energy/wavelength center
2	Energy/wavelength negative distance until bin boundary
3	Energy/wavelength positive distance until bin boundary
4	Model value

Plot type area

The `plot adum` output for the `area` plot type is:

Column	Value
1	Energy/wavelength center
2	Energy/wavelength negative distance until bin boundary
3	Energy/wavelength positive distance until bin boundary
4	Effective area

Plot type fwhm

The `plot adum` output for the `fwhm` plot type is:

Column	Value
1	Energy/wavelength center
2	Energy/wavelength negative distance until bin boundary
3	Energy/wavelength positive distance until bin boundary
4	Center value of FWHM limits
5	Negative half maximum
6	Positive half maximum
7	Peak energy of the response

CHAPTER

FOUR

SPECTRAL MODELS

4.1 Overview of spectral components

For more information on the definition of spectral components and the different types, see *Different types of spectral components* (page 285). Currently the following models are implemented in SPEX:

4.1.1 Absm: Morrison & McCammon absorption model

This model calculates the transmission of neutral gas with cosmic abundances as published first by Morrison & McCammon (1983). It is a widely used model. The following is useful to know when this model is applied:

1. The location of the absorption edges is not always fully correct; this can be seen with high resolution grating spectra
2. The model fails in the optical/UV band (i.e., it does not become transparent in the optical)
3. No absorption lines are taken into account
4. The abundances cannot be adjusted

If all the above is of no concern (as is the case in many situations), then the Morrison & McCammon model is very useful. In case higher precision or more detail is needed, the user is advised to use the *Hot: collisional ionisation equilibrium absorption model* (page 153) model with low temperature in SPEX, which gives the transmission of a slab in collisional ionisation equilibrium.

The parameters of the model are:

`nh` : Hydrogen column density in 10^{28} m^{-2}. Default value: 10^{-4} (corresponding to 10^{24} m^{-2}, a typical value at low Galactic latitudes).
`f` : The covering factor of the absorber. Default value: 1 (full covering)

Recommended citation: Morrison & McCammon (1983).

4.1.2 Amol: interstellar dust absorption model

This model calculates the transmission of various molecules considering both absorption and scattering. The extinction models are evaluated for grains with a standard MRN size distribution (Mathis & Rumpl & Nordsieck, 1977): $n(a) \propto a^{-3.5}$ with a the grain size and $a_{\mathrm{min}} = 0.005\ \mu m$ and $a_{\mathrm{max}} = 0.25\ \mu m$. All the extinction profiles are based on laboratory measurements. For further details on the lab processing see Zeegers et al. (2017), Rogantini et al. (2018), Costantini et al. (2019), Zeegers et al. (2019) and Rogantini et al. (2019).

The following compounds are presently taken into account (see Table *Compounds list* (page 138)).

Table 1: Compounds list

index	Name	formula	Form	Edge	Source
115	c-silicon	Si	crystalline	Si 1s	[1]
127	metallic iron	Fe	crystalline	Fe 1s	[2]
129	metallic nickel	Ni	crystalline	Ni 1s	[3]
130	a-carbon	C	amorphous	C 1s	[4]
131	diamond	C	crystalline	C 1s	[4]
132	graphite	C	crystalline	C 1s	[4]
2111	c-silicon carbide	SiC	crystalline	Si 1s	[1]
2230	troilite	FeS	crystalline	S 1s; Fe 1s	[5], [6]
2231	pyrrhotite	Fe_7S_8	crystalline	S 1s; Fe 1s	[5], [6]
2232	a-quartz	SiO_2	amorphous	Si 1s	[7]
2233	c-quartz	SiO_2	crystalline	Si 1s	[7]
2234	a-quartz	SiO_2	disorder	Si 1s	[7]
2235	c-silicon nitride	Si_3N_4	crystalline	Si 1s	[1]
2236	magnesia	MgO	crystalline	Si 1s	[8]
2237	aluminium oxide	Al_2O_3	crystalline	Al 1s	[9]
2238	alabandite	MnS	crystalline	S 1s	[5]
2239	pyrite	FeS_2	crystalline	S 1s	[5]
2240	titanium dioxide	TiO_2	crystalline	Ti 1s	[10]
2241	a-hydrocarbon	CH	amorhpous	C 1s	[11]
3230	c-forsterite	Mg_2SiO_4	crystalline	Mg 1s; Si 1s	[12], [7]
3231	a-enstatite	$MgSiO_3$	amorphous	Mg 1s; Si 1s	[12], [7]
3232	c-enstatite	$MgSiO_3$	crystalline	Mg 1s; Si 1s	[12], [7]
3233	c-spinel	$MgAl_2O_4$	crystalline	Mg 1s; Al 1s	[12], [9]
3270	calcium aluminate	$CaAl_2O_4$	crystalline	Ca 1s	[13]
3271	tri-Ca aluminate	$Ca_3Al_2O_6$	crystalline	Ca 1s	[13]
3302	c-fayalite	Fe_2SiO_4	crystalline	Si 1s; Fe 1s; Fe 2p	[7], [14], [15]
4230	a-olivine	$MgFeSiO_4$	amorphous	Mg 1s; Si 1s	[12], [7]
4231	c-olivine	$Mg_{1.56}Fe_{0.4}Si_{0.91}O_4$	crystalline	Mg 1s; Si 1s; Fe 1s	[12], [7], [6]
4232	c-En60Fe40	$Mg_{0.6}Fe_{0.4}SiO_3$	crystalline	Mg 1s; Si 1s; Fe 1s	[12], [7], [6]
4233	a-En60Fe40	$Mg_{0.6}Fe_{0.4}SiO_3$	amorphous	Mg 1s; Si 1s; Fe 1s	[12], [7], [6]
4234	a-En75Fe25	$Mg_{0.75}Fe_{0.25}SiO_3$	amorphous	Mg 1s; Si 1s	[12], [7]
4235	a-En90Fe10	$Mg_{0.9}Fe_{0.1}SiO_3$	amorphous	Mg 1s; Si 1s; Fe 1s	[12], [7], [6]
4236	c-En90Fe10	$Mg_{0.9}Fe_{0.1}SiO_3$	crystalline	Mg 1s; Si 1s; Fe 1s	[12], [7], [6]
4237	c-hypersthene	$Mg_{1.502}Fe_{0.498}Si_2O_6$	crystalline	Mg 1s; Si 1s; Fe 1s	[12], [7], [6]
4270	c-diopside	$MgCaSi_2O_6$	crystalline	Ca 1s	[13]
4271	a-diopside	$MgCaSi_2O_6$	amorphous	Ca 1s	[13]
4272	c-anorthite	$CaAl_2Si_2O_8$	crystalline	Ca 1s	[13]

[1] Chang et al. (1999), [2] exafsmaterials.com, [3] Van Loon et al. (2015), [4] Albella et al. (1998), [5] esrf.eu, [6] Rogantini et al. (2018), [7] Zeegers et al. (2019), [8] Fukushi et al. (2017), [9] Costantini et al. (2019), [10] Shin et al. (2013), [11] Bonnin-Mosbah et al. (2002), [12] Rogantini et al. (2019), [13] Neuville et al. (2007), [14] Lee et al. (2005), [15] Lee et al. (2009).

Additional molecules are listed in Table *Additional compounds list* (page 138). These models do not include scattering and were not integrated over a size distribution. They will be updated in future versions.

Table 2: Additional compounds list

108	molecular oxygen	O_2	O 1s	[16]
126	metallic iron	Fe	Fe 2p	[17]
2001	water	H_2O	O 1s	[18]
2002	crystalline ice	H_2O	O 1s	[19]
2003	amorphous ice	H_2O	O 1s	[19]
2010	carbon monoxide	CO	O 1s	[16]

continues on next page

Table 2 – continued from previous page

2011	carbon dioxide	CO_2	O 1s	[16]
2020	laughing gas	N_2O	O 1s	[16], [21]
2102	silicon monoxide	SiO	Si 1s	[20]
2200	eskolaite	Cr_2O_3	O 1s	[22]
2300	iron monoxide	FeO	Fe 1s	[23]
2301	iron oxide	$Fe_{1-x}O$	O 1s	[22]
2302	magnetite	Fe_3O_4	O, Fe 1s	[22], [23]
2303	hematite	Fe_2O_3	O, Fe 1s; Fe 2p	[22], [23], [17]
2304	iron sulfite	FeS_2	Fe 1s	[23]
2400	nickel monoxide	NiO	O 1s	[22]
2500	cupric oxide	CuO	O 1s	[22]
3001	adenine	$C_5H_5N_5$	O 1s	[24]
3103	pyroxene	$MgSiO_3$	O 1s	[25]
3200	calcite	$CaCO_3$	Ca 1s	[26]
3201	aragonite	$CaCO_3$	Ca 1s	[26]
3202	vaterite	$CaCO_3$	Ca 1s	[26]
3203	perovskite	$CaTiO_3$	O 1s	[22]
3300	hercynite	$FeAl_2O_4$	O 1s	[22]
3301	lepidocrocite	$FeO(OH)$	Fe 2p	[17]
3303	iron sulfate	$FeSO_4$	Fe 2p	[17]
3304	ilmenite	$FeTiO_3$	O 1s	[22]
3305	chromite	$FeCr_2O_4$	O 1s	[22]
4001	guanine	$C_5H_5N_5O$	O,N 1s	[24]
4002	cytosine	$C_4H_5N_3O$	O,N 1s	[24]
4003	thymine	$C_5H_6N_2O_2$	O,N 1s	[24]
4004	uracil	$C_4H_4N_2O_2$	O,N 1s	[24]
4100	andradite	$Ca_3Fe_2Si_3O_{12}$	O 1s	[22]
4101	acmite	$NaFeSi_2O_6$	O 1s	[22]
4102	franklinite	$Zn_{0.6}Mn_{0.8}Fe_{1.6}O_4$	O 1s	[22]
4103	olivine	$Mg_{1.6}Fe_{0.4}SiO_4$	O 1s	[22]
4104	almandine	$Fe_3Al_2(SiO_4)_3$	O 1s	[22]
4105	hedenbergite	$CaFeSi_2O_6$	O 1s	[22]
5001	dna (herring sperm)	$C_{39}H_{61}N_{15}O_{36}P_4$	O,N 1s	[24]
6001	montmorillonite	$Na_{0.2}Ca_{0.1}Al_2Si_4O_{10}(OH_2)(H_2O)_{10}$	Si 1s	[20]
6002	nontronite	$Na_{0.3}Fe_2^{3+}Si_3AlO_{10}(OH)_2 \bullet (H_2O)$	Si 1s	[20]
7001	enstatite_paulite	$Ca_2Mg_4Al_{0.75}Fe_{0.25}Si_7AlO_{22}(OH)_2$	Si 1s	[20]

[16] Barrus et al. (1979), [17] Lee et al. (2009), [18] Hiraya et al. (2001), [19] Parent et al. (2002), [20] Lee et al. (2010), [21] Wight et al. (1974), [22] Van Aken et al. (1998), [23] Lee et al. (2005), [24] Fujii et al. (2003), [25] Lee et al. (2008), [26] Hayakawa et al. (2008).

The chemical composition of these minerals was mainly taken from the Mineralogy Database of David Barthelmy. For DNA we assume equal contributions of adenine, cytosine, guanine and thymine, plus for each of these on average one phosphate and one 2-deoxyribose molecule. We take the cross-sections from the references as listed in *Additional compounds list* (page 138) in the energy interval where these are given, and use the cross section for free atoms Verner & Yakovlev (1995) outside this range.

Van Aken et al. (1998) do not list the precise composition of iron oxide. We assume here that $x = 0.5$.

Some remarks about the data from Barrus et al. (1979): not all lines are given in their tables, because they suffered from instrumental effects (finite thickness absorber combined with finite spectral resolution). However, Barrus et al. (1979) have estimated the peak intensities of the lines based on measurements with different column densities, and they also list the FWHM of these transitions. We have included these lines in the table of cross sections and joined smoothly with the tabulated values.

For N_2O, the fine structure lines are not well resolved by Barrus et al. (1979). Instead we take here the relative peaks from Wight et al. (1974), that have a relative ratio of 1.00 : 0.23 : 0.38 : 0.15 for peaks 1, 2, 3, and 4,

respectively. We adopted equal FWHMs of 1.2 eV for these lines, as measured typically for line 1 from the plot of Wight. We scale the intensities to the peak listed by Barrus et al. (1979).

Further, we subtract the C and N parts of the cross section as well as the oxygen 2s/2p part, using the cross sections of Verner & Yakovlev (1995). At low energy, a very small residual remains, that we corrected for by subtracting a constant fitted to the 510–520 eV range of the residuals. The remaining cross section at 600 eV is about 10 % above the Verner cross section; it rapidly decreases; we approximate the high-E behaviour by extrapolating linearly the average slope of the ratio between 580 and 600 eV to the point where it becomes 1. The remaining cross section at 600 eV is about 10% above the Verner & Yakovlev (1995) cross section; it rapidly decreases; we approximate the high-E behaviour therefore by extrapolating linearly the average slope of the ratio between 580 and 600 eV to the point where it becomes 1.

Warning: The normalisation is the total *molecular* column density. Thus, a value of 10^{-7} for CO_2 means 10^{21} CO_2 molecules m^{-2}, but of course 2×10^{21} O atoms m^{-2}, because each CO_2 molecule contains 2 oxygen atoms.

Warning: The Tables above shows for which edges and atoms the XAFS are taken into account. For all other edges and atoms not listed there, we simply use the pure atomic cross-section (without absorption lines). Note that for almost all constituents this may give completely wrong cross sections in the optical/UV band, as at these low energies the effects of chemical binding, crystal structure etc. are very important for the optical transmission constants. This is contrary to the SPEX models for pure atomic or ionised gas, where our models can be used in the optical band.

Warning: It is possible to change the values of the output atomic column densities of H–Zn, that are shown when you issue the "show par" command of SPEX. However, SPEX completely ignores this and when you issue the `calc` or `fit` commands, they will be reset to the proper values. Morale: just read of those parameters, don't touch them!

The parameters of the model are:

`n1--n4` : Molecular column density in 10^{28} m^{-2} for molecules 1–4. Default value: 10^{-6} for molecule 1, and zero for the others.
`i1--i4` : the molecule numbers for molecules 1–4 in the list (*Compounds list* (page 138) and *Additional compounds list* (page 138)). Default value: 108 (O_2) for molecule 1, zero for the others. A value of zero indicates that for that number no molecule will be taken into account. Thus, for only 1 molecule, keep *i2–i4* $= 0$.

The following parameters are common to all our absorption models:

- `f` : The covering factor of the absorber. Default value: 1 (full covering)
- `zv` : Average systematic velocity v of the absorber

The following parameters are *only* output parameters:

- `h--zn` : The column densities in 10^{28} m^{-2} for all *atoms* added together for the all molecules that are present in this component.

Recommended citation: Pinto et al. (2010).

4.1.3 Bb: blackbody model

The surface *energy* flux of a blackbody emitter is given by

$$F_\nu = \pi B_\nu = \frac{2\pi h\nu^3/c^2}{e^{h\nu/kT}-1}$$

(Chapter 1 of Rybicki & Lightman 1986). We transform this into a spectrum with energy units (conversion from Hz to keV) and obtain for the total *photon* flux:

$$S(E)\mathrm{d}E = 2\pi c[10^3 e/hc]^3 \frac{E^2}{e^{E/T}-1}\mathrm{d}E$$

where now E is the photon energy in keV, T the temperature in keV and e is the elementary charge in Coulomb. Inserting numerical values and multiplying by the emitting area A, we get

$$N(E) = 9.883280 \times 10^7\, E^2 A/(e^{E/T}-1)$$

where N(E) is the photon spectrum in units of 10^{44} photons/s/keV and A the emitting area in 10^{16} m^2.

The parameters of the model are:

`norm` : Normalisation A (the emitting area, in units of 10^{16} m^2. Default value: 1.
`t` : The temperature T in keV. Default value: 1 keV.

Recommended citation: Kirchhoff (1860).

4.1.4 Cf: isobaric cooling flow differential emission measure model

This model calculates the spectrum of a standard isobaric cooling flow. The differential emission measure distribution $\mathrm{d}Y(T)/\mathrm{d}T$ for the isobaric cooling flow model can be written as

$$D(T) \equiv \mathrm{d}Y(T)/\mathrm{d}T = \frac{5\dot{M}k}{2\mu m_\mathrm{H}\Lambda(T)},$$

where $\dot{M}$ is the mass deposition rate, k is Boltzmann's constant, μ the mean molecular weight (0.618 for a plasma with 0.5 times solar abundances), m_H is the mass of a hydrogen atom, and $\Lambda(T)$ is the cooling function. We have calculated the cooling function Λ using our own SPEX code for a grid of temperatures and for 0.5 times solar abundances. The spectrum is evaluated by integrating the above differential emission measure distribution between a lower temperature boundary T_1 and a high temperature boundary T_n. We do this by creating a temperature grid with n bins and calculating the spectrum for each temperature.

Warning: Take care that n is not too small in case the relevant temperature is large; on the other hand if n is large, the computational time increases. Usually a spacing with temperature differences between adjacent bins of 0.1 (in $\log$) is sufficient and optimal.

Warning: The physical relevance of this model is a matter of debate.

The parameters of the model are:

`norm` : The mass deposition rate $\dot{M}$ in $\mathrm{M}_\odot$ yr^{-1}.
`t1` : Lower temperature cut-off temperature T_1. Default: 0.1 keV.
`tn` : Upper temperature cut-off temperature T_n. Default: 1 keV.

`nr` : Number of temperature bins n used in the integration. Default value: 16
`p` : Slope $p = 1/\alpha$. Default: 0.25 ($\alpha = 4$).
`cut` : Lower temperature cut-off, in units of $T_{\rm max}$. Default value: 0.1.
The following parameters are the same as for the cie-model:
`ed` : Electron density in 10^{20} $\rm m^{-3}$
`it` : Ion temperature in keV
`vrms` : RMS Velocity broadening in km/s (see *Definition of the micro-turbulent velocity in SPEX* (page 293))
`ref` : Reference element
`01...30` : Abundances of H to Zn
`file` : Filename for the nonthermal electron distribution

Recommended citation: Kaastra et al. (2004) and Fabian et al. (1984).

4.1.5 Cie: collisional ionisation equilibrium model

This model calculates the spectrum of a plasma in collisional ionisation equilibrium (CIE). It consists essentially of two steps, first a calculation of the ionisation balance and then the calculation of the X-ray spectrum. The basis for this model is formed by the mekal model, but several updates have been included (see *Plasma model in SPEX 3.0* (page 275)).

Temperatures

Some remarks should be made about the temperatures. SPEX knows three temperatures that are used for this model.

First there is the electron temperature $T_{\rm e}$. This is usually referred to as "the" temperature of the plasma. It determines the continuum shape and line emissivities and the resulting spectrum is most sensitive to this.

Secondly, there is the ion temperature $T_{\rm i}$. This is only important for determining the line broadening, since this depends upon the thermal velocity of the ions (which is determined both by $T_{\rm i}$ and the atomic mass of the ion). Only in high resolution spectra the effects of the ion temperature can be seen.

Finally, we have introduced here the ionization balance temperature $T_{\rm b}$ that is used in the determination of the ionization equilibrium. It is the temperature that goes into the calculation of ionization and recombination coefficients. In equilibrium plasmas, the ratio $R_{\rm b} \equiv T_{\rm b}/T_{\rm e}$ is 1 by definition. It is unphysical to have $R_{\rm b}$ not equal to 1. Nevertheless we allow for the possibility of different values of $R_{\rm b}$, in order to mimick out of equilibrium plasmas. For $R_{\rm b} < 1$, we have an ionizing plasma, for $R_{\rm b} > 1$ a recombining plasma. Note that for ionizing plasmas SPEX has also the `nei` model (*Neij: non-equilibrium ionisation jump model* (page 160)), which takes into account explicitly the effects of transient (time dependent) ionization.

It is also possible to mimic the effects of non-isothermality in a simple way. SPEX allows for a Gaussian emission measure distribution:

$$Y(x) = \frac{Y_0}{\sqrt{2\pi}\sigma_T} e^{-(x-x0)^2/2\sigma_T^2}$$

where Y_0 is the total, integrated emission measure. By default $x \equiv \log T$ and $x_0 \equiv \log T_0$ with T_0 the average temperature of the plasma (this is entered at the "T" value in SPEX). However, this can be changed to $x \equiv T$ and $x_0 \equiv T_0$ by setting logt to 0. If the parameter `sup` is set $> 10^{-5}$, then the Gaussian emission measure distribution model becomes asymmetric. The `sig` parameter determines the slope of the low-temperature part of the Gaussian and sup determines the high-temperature side. Usually (default) $\sigma_T = 0$ and in that case the normal isothermal spectrum is chosen. Note that for larger values of σ_T the cpu time may become larger due to the fact that the code has to evaluate many isothermal spectra.

Line broadening

Apart from line broadening due to the thermal velocity of the ions (caused by $T_\mathrm{i} > 0$) it is also possible to get line broadening due to (micro)turbulence. Since SPEX version 3.06, we use the RMS velocity broadening in km/s in our models. For more information about this change see: *Definition of the micro-turbulent velocity in SPEX* (page 293).

Density effects

It is also possible to vary the electron density n_e of the plasma. This does not affect the overall shape of the spectrum (i.e., by changing n_e only SPEX still uses the previous value of the emission measure $Y \equiv n_\mathrm{H} n_\mathrm{e} V$), but spectral lines that are sensitive to the electron density will get different intensities. Usually this occurs for higher densities, for example under typical ISM conditions (n_e = 1 cm^{-3}) this is not an important effect.

Non-thermal electron distributions

The effects of non-thermal electron distribution on the spectrum can be included. See *Non-thermal electron distributions* (page 282) for more details.

Abundances

The abundances are given in Solar units. Which set of solar units is being used can be set using the `abun` command (*Abundance: standard abundances* (page 77)). For spectral fitting purposes it is important to distinguish two situations.

In the first case there is a strong thermal continuum. Since in most circumstances the continuum is determined mainly by hydrogen and helium, and the X-ray lines are due to the other elements, the line to continuum ratio is a direct measurement of the metal abundances compared to H/He. In this situation, it is most natural to have the hydrogen abundance fixed and allow only for fitting of the other abundances (as well as the emission measure).

In the other case the thermal continuum is weak, but there are strong spectral lines. Measuring for example the Fe abundance will give large formal error bars, not because the iron lines are weak but because the continuum is weak. Therefore, all abundances will get rather large error bars, and despite the fact of strong O and Fe lines, the formal error bars on the O/Fe ratio becomes large. This can be avoided by choosing another reference element, preferentially the one with the strongest lines (for example Fe). Then the O/Fe ratio will be well constrained, and it is now only the H abundance relative to Fe that is poorly constrained. In this case it is important to keep the nominal abundance of the reference element to unity. Also keep in mind that in general we write for the normalisation $n_\mathrm{e} n_\mathrm{X} V$ in this case; when the reference element is H, you mays substitute X=H; however when it is another element, like O, the normalisation is still the product of $n_\mathrm{e} n_\mathrm{X} V$ where X stands for the fictitious hydrogen density derived from the solar O/H ratio.

Example

Suppose the Solar O abundance is 1E-3 (i.e. there is 0.001 oxygen atom per hydrogen atom in the Sun). Take the reference atom oxygen ($Z = 8$). Fix the oxygen abundance to 1. Fit your spectrum with a free hydrogen abundance. Suppose the outcome of this fit is a hydrogen abundance of 0.2 and an emission measure 3 (in SPEX units). This means $n_\mathrm{e} n_\mathrm{X} V = 3 \times 10^{64}$ m^{-3}. Then the "true" emission measure $n_\mathrm{e} n_\mathrm{H} V = 0.2 \times 3 \times 10^{64}$ m^{-3}. The nominal oxygen emission measure is $n_\mathrm{e} n_\mathrm{O} V = 0.001 \times 3 \times 10^{64}$ m^{-3}, and nominally oxygen would have 5 times overabundance as compared to hydrogen, in terms of solar ratios.

Parameter description

Warning: When you make the temperature too low such that the plasma becomes completely neutral, the model will crash. This is because in that case the electron density becomes zero, and the emission measure is undefined. The nominal temperature limits that are implemented in SPEX usually avoid that condition, but the results may depend somewhat upon the metallicity because of the charge exchange processes in the ionisation balance.

Warning: In high resolution spectra, do not forget to couple the ion temperature to the electron temperature, as otherwise the ion temperature might keep its default value of 1 keV during spectral fitting and the line widths may be wrong.

Warning: Some people use instead of the emission measure $Y \equiv n_{\rm H} n_{\rm e} V$, the quantity $Y' = n_{\rm e}^2 V$ as normalisation. This use should be avoided as the emission is proportional to the product of electron and ion densities, and therefore use of Y' makes the spectrum to depend nonlinear on the elemental abundances (since an increase in abundances also affects the $n_{\rm e}/n_{\rm H}$ ratio).

Warning: The default line broadening is just Doppler broadening. This is fine and self-consistent for the 'old' line calculation. To incorporate the natural line broadening for the 'new' calculations, the user must use the `var dopp 4` option to get Voigt profiles. This is physically better but takes more computation time.

The parameters of the model are:

`norm` : the normalisation, which is the emission measure $Y \equiv n_{\rm H} n_{\rm e} V$ in units of 10^{64} m^{-3}, where $n_{\rm e}$ and $n_{\rm H}$ are the electron and Hydrogen densities and V the volume of the source. Default value: 1.

`t` : the electron temperature $T_{\rm e}$ in keV. Default value: 1.

`sig` : the width σ_T of the gaussian emission measure profile. Default value: 0. (no temperature distribution i.e. isothermal)

`sup` : the width σ_T of the high-temperature part of the gaussian emission measure profile. If larger than 10^{-5} keV, the sig parameter becomes the sigma value for the low-temperature end. Default value: 0

`logt` : Switch between linear and logarithmic temperature scale for the gaussian emission measure profile. Default value: 1 (logarithmic)

`ed` : the electron density $n_{\rm e}$ in units of 10^{20} m^{-3} (or 10^{14} cm^{-3}). Default value: 10^{-14}, i.e. typical ISM conditions, or the low density limit.

`it` : the ion temperature $T_{\rm i}$ in keV. Default value: 1

`rt` : the ratio of ionization balance to electron temperature, $R_{\rm b} = T_{\rm b}/T_{\rm e}$ in keV. Default value: 1.

`vrms` : RMS Velocity broadening in km/s (see *Definition of the micro-turbulent velocity in SPEX* (page 293))

`ref` : reference element. Default value 1 (hydrogen). See above for more details. The value corresponds to the atomic number of the reference element.

`01` : Abundance of hydrogen (H, Z=1) in Solar units. Default 1.

`02` : Abundance of helium (He, Z=2) in Solar units. Default 1.

`...` `30` : Abundance of zinc (Zn, Z=30) in Solar units. Default 1.

`file` : Filename for the non-thermal electron distribution. If not present, non-thermal effects are not taken into account (default).

Recommended citation: Kaastra et al. (1996).

4.1.6 Comt: comptonisation model

This is the model for Comptonisation of soft photons in a hot plasma, developed by Titarchuk (1994). See the XSPEC manual for more details. The code was substantially improved and brought to the SPEX standards.

Some modifications

Titarchuk (1994) gives an analytical approximation for $\beta(\tau)$ to the exact calculations as provided by Sunyaev & Titarchuk (1985) for $0.1 < \tau < 10$. Their approximation has the proper asymptotic behaviour for small and large values of τ, but unfortunately has deviations up to 7 % over the $0.1 - 10$ range for τ. The approximation by Titarchuk is given in their equations (27) and (28) by

$$\text{disk:}\beta = \frac{\pi^2}{12(\tau+2/3)^2}(1-\mathrm{e}^{-1.35\tau}) + 0.45\mathrm{e}^{-3.7\tau}\ln\frac{10}{3\tau},$$
$$\text{sphere:}\beta = \frac{\pi^2}{3(\tau+2/3)^2}(1-\mathrm{e}^{-0.7\tau}) + \mathrm{e}^{-1.4\tau}\ln\frac{4}{3\tau}.$$

We found an approximation that is better than 1 % using a slight modification of the above equations. We use these new approximations in our calculations:

$$\text{disk:}\beta = \frac{0.8351}{(\tau+0.867)^2}(1-\mathrm{e}^{-3.876\tau}) + 0.572\mathrm{e}^{-3.75\tau}\ln\frac{1.06}{\tau},$$
$$\text{sphere:}\beta = \frac{3.504}{(\tau+1.2)^2}(1-\mathrm{e}^{-2.09\tau}) + 1.013\mathrm{e}^{-2.2\tau}\ln\frac{1.6}{\tau}.$$

Warning: For the physically valid range of parameters, consult the XSPEC manual; see also Hua & Titarchuk (1995), their Fig. 7.

The parameters of the model are:

`norm` : Normalisation A of the power law, in units of 10^{44} ph s^{-1} keV^{-1}. Due to the way the model is calculated, this is not the flux at 1 keV! Default value: 1.
`t0` : The temperature T of the seed photons in keV. Default value: 1 keV.
`t1` : The plasma temperature T in keV. Default value: 100 keV.
`tau` : The optical depth. Default value: 3.
`type` : The type of geometry. Allowed values are 0 for a disk (default) or 1 for a sphere. This is a frozen parameter (cannot be fitted).

Recommended citation: Titarchuk (1994)

4.1.7 CX: model for charge exchange plasmas

This model calculates the spectrum emitted from a hot plasma when it recombines with cold neutral materials. This model is based on three key assumptions: (1) it considers only single electron capture in a ion-neutral collision; (2) all cross section data are obtained only with a atomic hydrogen target, (3) electronic collisional excitation and recombination are ignored in the spectral calculation. More information can be found in Gu et al. (2016).

Charge exchange cross sections

The CX cross section data used in the model are partly taken from literature, including quantum molecular-orbital close-coupling calculations for C^{5+} and O^{6+} by Wu et al. (2012) and Nolte et al. (2012), multi-channel Laudau-Zener results for Fe^{25+} and Fe^{26+} by Mullen et al. (2016), other data compilations for C^{6+} and O^{8+} by Janev et al. (1993), and the NIFS Charge Transfer Database (CHART)[1] for Be^{4+}, B^{5+}, N^{7+}, and Ne^{10+}. For CHART database, we extracted all the data, from both theoretical calculations and experiments (see a full list in Table 1 of Gu et al. (2016)), and fitted them with Eq.2 of Gu et al. (2016) in the energy range of interests. In typical astrophysical velocity range ($\sim 100 - 5000$ km s^{-1}), the useful CHART data are usually from molecular-orbital and atomic-orbital close-coupling methods, and a few classical trajectory Monte Carlo calculations. All the above data are dependent on collision energy, and resolved to levels described by quantum number n and l.

For ions not available in public sources, we developed a new method to interpolate by analyzing the known ions. First we used a scaling law to determine total cross section for each ion, and applied another scaling law to represent the $n-$ selectivity. The $l-$ dependence is approximated by one of the five empirical weighting functions presented in Eqs.$4 - 8$ of Gu et al. (2016).

Warning: The CX model only works with the updated atomic database set through the command `var calc new`.

Warning: All Beryllium-like sequence ions are not included in the current version; will be available later.

Warning: We will keep updating the CX model when new data (especially for molecular targets) from theoretical calculations and experiments become available.

Parameter description

The parameters of the CX model are:

`norm` : the normalisation, which is the emission measure $Y \equiv n_{\rm H} n_{\rm nh} V$ in units of 10^{64} m^{-3}, where $n_{\rm H}$ and $n_{\rm nh}$ are the Hydrogen densities of the ionized and neutral materials, respectively, and V is the effective interaction volume. Default value: 1.

`hden` : Hydrogen density of the neutral materials in units of 10^{20} m^{-3} (or 10^{14} m^{-3}). Default value: 10^{-14}.

`mode` : Switch between a hot-cold interaction driven by thermal motion of hot plasma, and the one dominated by flow velocity. Default value: 2 (kinematic).

`t` : the ionization temperature of hot matter in keV. It is also used to approximate the thermal motion when mode is set to 1. Default value: 1.

`sig` : the width σ_T of the gaussian emission measure profile. Default value: 0. (no temperature distribution i.e. isothermal)

`sup` : the width σ_T of the high-temperature part of the gaussian emission measure profile. If larger than 10^{-5} keV, the sig parameter becomes the sigma value for the low-temperature end. Default value: 0

`logt` : Switch between linear and logarithmic temperature scale for the gaussian emission measure profile. Default value: 1 (logarithmic)

`zv` : Collision velocity in unit of km s^{-1}, used when mode is set to 2. Default value: 100

`op` : Switch between single and multiple collisions for each ion. In multiple collision case, one ion would continuously undergo CX and produce various emission lines, until it becomes neutral. Default: 1 (single)

`wt` : Weighting functions for subshell $l-$ population. When wt is set to 1, the $l-$ population is approximated by a series of empirical functions that switchs from one to another as a function of collision velocity. See Gu et al.

[1] http://dbshino.nifs.ac.jp/

(2016) for details. These empirical functions are defined in Eqs. $4-8$ of Gu et al. (2016), and will be selected when wt is set to $2-6$, respectively. Default: 1

`vrms` : RMS Velocity broadening in km/s (see *Definition of the micro-turbulent velocity in SPEX* (page 293))

`ref` : reference element. Default value 1 (hydrogen). See above for more details. The value corresponds to the atomic number of the reference element.

`01` : Abundance of hydrogen (H, Z=1) in Solar units. Default 1.

`02` : Abundance of helium (He, Z=2) in Solar units. Default 1.

`...30` : Abundance of zinc (Zn, Z=30) in Solar units. Default 1.

`file` : Filename for the nonthermal distribution. If not present, nonthermal effects are not taken into account (default).

Recommended citation: Gu et al. (2016).

4.1.8 Dabs: dust absorption model

This model allows the user to make a zeroth order approximation to the effects of dust on an X-ray absorption spectrum. Basically, compared to absorption by atomic gas, the effects of dust are two-fold:

- Modifications to the fine structure near the edge
- Reduced opacity due to self-shielding in grains

The *dabs* model only accounts for the second effect, reduced opacity due to self-shielding. The formalism of Wilms et al. (2000) is followed, and we refer to that paper for a full explanation.

In the set of abundances, we use the depletion factors of Wilms et al. (2000).

H	He	Li	Be	B	C	N	O	F	Ne	Na	Mg	Al	Si	P
0	0	1	0	0	0.5	0	0.4	0	0	0.75	0.8	0.98	0.9	0.4
S	Cl	Ar	K	Ca	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Cu	Zn
0.4	0.5	0	0	0.997	0	0.998	0	0.97	0.93	0.7	0.95	0.96	0	0

The numbers represent the fraction of the atoms that are bound in dust grains. Noble gases like Ne and Ar are chemically inert hence are generally not bound in dust grains, but other elements like Ca exist predominantly in dust grains.

Warning: For a realistic model, it is advised to combine the *dabs* model with a *hot* model, where (for Solar abundances), the hot model should contain the elements *not* bound in the dust.

Note: Finally, we note that the *amol* model can deal with the modified structure near the oxygen edge, but that model does not take self-shielding into account.

The parameters of the model are:

`nh` : Nominal hydrogen column density in 10^{28} m^{-2}. Default value: 1.

`amin` : Minimum grain size in μm. Default value: 0.025.

`amax` : Minimum grain size in μm. Default value: 0.25.

`p` : Power index grain size distribution. Default value: 3.5.

`rho` : Mean density of the grains in units of 1000 kg m^{-3}. Default value: 3.

`ref` : Reference atom. Default value: 3 (Lithium). because dust, according to the depletion factors used, does not contain hydrogen, we substitute here formally Lithium. The user is advisednot to change this.

`01...30` : Abundances of H to Zn. See the remark on depletion factors above.

The following parameters are common to all our absorption models:
`fcov` : The covering factor of the absorber. Default value: 1 (full covering)
`zv` : Average systematic velocity v of the absorber

Recommended citation: Wilms et al. (2000) and Kaastra et al. (2009).

4.1.9 Dbb: disk blackbody model

We take the model for a standard Shakura-Sunyaev accretion disk. The radiative losses from such a disk are given by

$$Q = \frac{3GM\dot{M}(1-\sqrt{r_i/r})}{8\pi r^3},$$

where Q is the loss term in W m^{-2} at radius r, M the mass of the central object, $\dot{M}$ the accretion rate through the disk and r_i the inner radius. If this energy loss is radiated as a black body, we have

$$Q = \sigma T^4$$

with σ the constant of Stefan-Boltzmann and $T(r)$ the local temperature of the black body. The total spectrum of such a disk is then obtained by integration over all radii. We do this integration numerically. Note that for large r, $T \sim r^{-3/4}$.

Warning: A popular disk model (diskbb in XSPEC) assumes this temperature dependence over the full disk. However, we correct it using the factor $(1-\sqrt{r_i/r})$ in Q which corresponds to the torque-free condition at the inner boundary of the disk.

The photon spectrum of the disk is now given by

$$N(E) = \frac{8\pi^2 E^2 r_i^2 \cos i}{h^3 c^2} f_d(E/kT_i, r_o/r_i),$$

where i is the inclination of the disk (0 degrees for face-on disk, 90 degrees for an edge-on disk), E the photon energy, r_o the outer edge of the disk, and T_i is defined by

$$T_i^4 = 3GM\dot{M}/8\pi r_i^3\sigma \tag{4.1}$$

and further the function $f_d(y,r)$ is defined by

$$f_d(y,r) = \int_1^r \frac{x\mathrm{d}x}{e^{y/\tau}-1}$$

where $\tau(x)$ is defined by $\tau^4(x) = (1-1/\sqrt{x})/x^3$.

In addition to calculating the spectrum, the model also allows to calculate the average radius of emission R_e at a specified energy E_r. This is sometimes useful for time variability studies (softer photons emerging from the outer parts of the disk).

Given the fit parameters T_i and r_i, using (4.1) it is straightforward to calculate the product $M\dot{M}$. Further note that if $r_i = 6GM/c^2$, it is possible to find both M and $\dot{M}$, provided the inclination angle i is known.

The parameters of the model are:

`norm` : Normalisation A ($= r_i^2 \cos i$), in units of 10^{16} m^2. Default value: 1.
`t` : The nominal temperature T_i in keV. Default value: 1 keV.
`ro` : The ratio of outer to inner disk radius, r_o/r_i
`ener` : Energy E_r at which the average radius of emission will be calculated

`rav` : Average radius R_e of all emission at energy E_r specified by the parameter above. Note that this is not a free parameter, it is calculated each time the model is evaluated.

Recommended citation: Shakura & Sunyaev (1973).

4.1.10 Delt: delta line model

The delta line model is a simple model for an infinitely narrow emission line.

The spectrum is given by:

$$F(E) = A\delta(E - E_0),$$

where E is the photon energy in keV, F the photon flux in units of 10^{44} ph s^{-1} keV^{-1}, E_0 is the line energy of the spectral line (in keV) and A is the line normalisation (in units of 10^{44} ph s^{-1}). The total line flux is simply given by A.

To ease the use for dispersive spectrometers (gratings) there is an option to define the wavelength instead of the energy as the basic parameter. The parameter `type` determines which case applies: type=0 (default) corresponding to energy, type=1 corresponding to wavelength units.

Warning: When changing from energy to wavelength units, take care about the frozen/thawn status of the line centroid and FWHM.

Warning: You need to do a "calc" or "fit" command to get an update of the wavelength (for type=0) or energy (type=1).

The parameters of the model are:

`norm` : Normalisation A, in units of 10^{44} ph s^{-1}. Default value: 1.
`e` : The line energy E_0 in keV. Default value: 6.4 keV.
`type` : The type: 0 for energy units, 1 for wavelength units.
`w` : The line wavelength λ in Å. Default value: 20 Å.

4.1.11 Dem: differential emission measure model

This model calculates a set of isothermal CIE spectra, that can be used later in one of the DEM analysis tools (see the "dem" commands of the syntax). The CIE spectra are evaluated for a logarithmic grid of temperatures between T_1 and T_2, with n bins.

Warning: For the DEM methods to work, the dem model is the only allowed additive component that can be present. No other additive components are allowed. But of course the spectrum of the dem model may be modified by applying any combination of multiplicative models (redshifts, absorptions, line broadening, etc.).

Warning: Because of the above, do not use the `fit` commands when you have a dem model. If you really need to fit, use the `pdem` model (*Pdem: DEM models* (page 162)) instead.

Warning: In general, the spacing between the different temperature components should not be smaller than 0.10 in $\log T$. Smaller step sizes will produce unstable solutions.

The parameters of the model are:

`t1` : Lower temperature T_1 in keV. Default value: 0.001 keV.
`t2` : Upper temperature T_2 in keV. Default value: 100 keV.
`nr` : Number of temperature bins. Default value: 64.
The following parameters are the same as for the cie-model:
`ed` : Electron density in 10^{20} m^{-3}.
`it` : Ion temperature in keV.
`vrms` : RMS Velocity broadening in km/s (see *Definition of the micro-turbulent velocity in SPEX* (page 293)).
`ref` : Reference element.
`01...30` : Abundances of H to Zn.
`file` : Filename for the non-thermal electron distribution.

Recommended citation: Mewe et al. (1995)

4.1.12 Dust: dust scattering model

This model calculates the effective transmission of dust that scatters photons out of the line of sight. No re-emission is taken into account, i.e. it is assumed that all scattered photons are lost from the line of sight. This makes the model useful for instruments with a high spatial resolution (i.e, the spatial resolution is much better than the typical size of the dust scattering halo).

Dust scattering is described for example by Draine et al. (2003) In that paper, a link is given to the website of Draine. This website contains slightly updated cross sections for three cases as listed below. The scattering is computed for a Carbonaceous - Silicate Model for Interstellar Dust. The cases are:

1. set=1: for $R_V = 3.1$;
2. set=2: for $R_V = 4.0$;
3. set=3: for $R_V = 5.5$.

Warning: For any instrument where the extraction region has a size comparable to the size of the dust scattering halo, this model should not be used, as the scattered X-rays will fall within the extraction region. Take care when fitting data from different instruments simultaneously.

Warning: This model only calculates the dust scattering, not the dust absorption.

The parameters of the model are:

`nh` : Hydrogen column density in 10^{28} m^{-2}. Default value: 10^{-4} (corresponding to 10^{24} m^{-2}, a typical value at low Galactic latitudes).
`f` : The covering factor of the absorber. Default value: 1 (full covering)
`set` : The set of cross sections being used. See table above.

Recommended citation: Draine et al. (2003).

4.1.13 Ebv: Galactic interstellar extinction model

This model can be used to correct fluxes of optical/UV data for interstellar reddening in our Galaxy. The reddening is caused by dust absorption and scattering (extinction) in the interstellar medium. The model uses the extinction curve of Cardelli et al. (1989). including the update for near-UV given by O'Donnell et al. (1994).

The extinction law can be expressed as:

$$(A_\lambda/A_V) = a_\lambda + (b_\lambda/R_V)$$

where, A_λ and A_V are the total extinctions (measured in magnitude) at wavelength λ and the V-band, respectively. The wavelength- dependent parameters a_λ and b_λ are provided by the aforementioned papers. The scalar R_V is defined as the ratio of total extinction A_V to selective extinction $A_B - A_V$, where A_B is the total extinction at the B-band. So the selective extinction $A_B - A_V$ represents a colour excess, which is commonly denoted as $E(B-V)$. For Milky Way, the typical value for R_V is reported to be 3.1. Thus, by specifying $E(B-V)$ and R_V, the extinction A_λ can be derived, which in turn gives us the unreddened flux (F_λ) from the observed reddened flux (F^*_λ) using $F_\lambda = (10^{\,0.4A_\lambda})\,F^*_\lambda$. Note that at energies above the Lyman limit, the transmission of the model is set to 1 in SPEX, thus it can be used alongside the Galactic interstellar X-ray absorption models in SPEX.

This extinction model is currently the best model for the Milky Way. However, the extinction curve of the Milky Way has a famous broad bump at 2175 . The origin of this bump is still not fully understood. Interestingly, this feature (due to dust) is seen in the Milky Way and not in other galaxies like in AGN. If one would use the `ebv` model to correct for extinction in the host galaxy of an AGN, it is best to remove this bump from the model as it can have an effect on the fit to the data. The bump can be removed by setting the `mway` parameter to 0.

The parameters of the model are:
`ebv` : The colour excess $E(B-V)$. The value is set by the user.
`rv` : The scalar R_V. Default (recommended) value: 3.1
`fcov` : The covering factor of the absorber. Default value: 1 (full covering)
`mway` : Include the Milky Way broad bump (1 = Yes, 0 = No).

Recommended citation: Cardelli et al. (1989) and O'Donnell et al. (1994).

4.1.14 Etau: simple transmission model

This model calculates the transmission $T(E)$ between energies $E1$ and E_2 for a simple (often unphysical!) case:

$$T(E) = e^{-\tau(E)},$$

with the optical depth $\tau(E)$ given by:

$$\tau(E) = \tau_0 E^a.$$

In addition, we put here $T \equiv 1$ for $E < E_1$ and $E > E_2$, where E_1 and E_2 are adjustable parameters. This allows the user for example to mimick an edge. Note however, that in most circumstances there are more physical models present in SPEX that contain realistic transmissions of edges! If you do not want or need edges, simply keep E_1 and E_2 at their default values.

Note that τ_0 should be non-negative. For $a > 0$ the spectrum has a high-energy cut-off, for $a < 0$ it has a low-energy cut-off, and for $a = 0$ the transmission is flat. The larger the value of a, the sharper the cut-off is.

The model can be used as a basis for more complicated continuum absorption models. For example, if the optical depth is given as a polynomial in the photon energy E, say for example $\tau = 2 + 3E + 4E^2$, one may define three *etau* components, with τ_0 values of 2, 3, and 4, and indices a of 0, 1 and 2. This is because of the mathematical identity $e^{-(\tau_1+\tau_2)} = e^{-\tau 1} \times e^{-\tau 2}$.

The parameters of the model are:

`tau0` : Optical depth τ_0 at $E = 1$ keV. Default value: 1.
`a` : The index a defined above. Default value: 1.
`e1` : Lower energy E_1 (keV). Default value: 10^{-20}.
`e2` : Upper energy E_2 (keV). Default value: 10^{20}.
`f` : The covering factor of the absorber. Default value: 1 (full covering)

4.1.15 Euve: EUVE absorption model

This model calculates the transmission of gas with cosmic abundances as published first by Rumph et al. (1994). It is a widely used model for the transmission at wavelenghts covered by the EUVE satellite ($\lambda > 70$ Å). As it these wavelengths metals are relatively unimportant, it only takes into account hydrogen and helium, but for these elements it takes into account resonances. However it is not recommended to use the model for harder X-rays, due to the neglect of metals such as oxygen etc.

Otherwise the user is advised to use the `absm` model (*Absm: Morrison & McCammon absorption model* (page 137)) or the `hot` model (*Hot: collisional ionisation equilibrium absorption model* (page 153)) with low temperature in SPEX, which gives the transmission of a slab in collisional ionisation equilibrium.

The parameters of the model are:

`nh` : Hydrogen column density in 10^{28} m^{-2}. Default value: 10^{-4} (corresponding to 10^{24} m^{-2}, a typical value at low Galactic latitudes).
`he1` : The He / H ratio. Default value: 0.1.
`he2` : The He / H ratio. Default value: 0.01.
`f` : The covering factor of the absorber. Default value: 1 (full covering)

Recommended citation: Rumph et al. (1994).

4.1.16 File: model read from a file

This model reads a spectrum from a file. The user inputs a spectrum at n energy bins ($n > 1$). The file containing the spectrum should have the follwing format:

- first line: n
- second line: E_1, S_1
- third line: E_2, S_2
- ...
- last line: E_n, S_n

Here E_i is the energy of a photon in keV, and S_i is the spectrum in units of 10^{44} photons s^{-1} keV^{-1}. All energies E_i must be positive, and the grid must be monotonically increasing (two subsequent energies may not have the same value). Also, the spectrum S_i must be strictly positive (i.e. $S_i = 0$ is not allowed).

SPEX then calculates the spectrum by linear interpolation in the $\log E - \log S$ plane (this is why fluxes must be positive). For $E < E_1$ and $E > E_n$ however, the spectrum is put to zero. Finally, the spectrum is multiplied by the scale factor N prescribed by the user.

The parameters of the model are:

`norm` : Normalisation factor N. Default value: 1.
`file` : The file name of the file containing the spectrum

4.1.17 Gaus: gaussian line model

The Gaussian line model is the simplest model for a broadened emission line.

The spectrum is given by:

$$F(E) = Ae^{(E-E_0)^2/2\sigma^2},$$

where E is the photon energy in keV, F the photon flux in units of 10^{44} ph s^{-1} keV^{-1}, E_0 is the average line energy of the spectral line (in keV) and A is the line normalisation (in units of 10^{44} ph s^{-1}). The total line flux is simply given by A. Further σ is the Gaussian width, which is related to the full width at half maximum $FWHM$ by $FWHM = \sigma\sqrt{\ln 256}$ or approximately $FWHM = 2.3548\sigma$.

To ease the use for dispersive spectrometers (gratings) there is an option to define the wavelength instead of the energy as the basic parameter. The parameter *type* determines which case applies: type=0 (default) corresponding to energy, type=1 corresponding to wavelength units.

Warning: Do not confuse σ with $FWHM$ when interpreting your fit results with other data.

Warning: When changing from energy to wavelength units, take care about the frozen/thawn status of the line centroid and FWHM.

Warning: You need to do a `calc` or `fit` command to get an update of the wavelength (for type=0) or energy (type=1).

The parameters of the model are:

`norm` : Normalisation A, in units of 10^{44} ph s^{-1}. Default value: 1.
`e` : The line energy E_0 in keV. Default value: 6.4 keV.
`fwhm` : The line $FWHM$, in keV.
`type` : The type: 0 for energy units, 1 for wavelength units.
`w` : The line wavelength λ in Å. Default value: 20 Å.
`awid` : The line $FWHM$, in Å.

Recommended citation: Gauss (1809).

4.1.18 Hot: collisional ionisation equilibrium absorption model

This model calculates the transmission of a plasma in collisional ionisation equilibrium with cosmic abundances.

For a given temperature and set of abundances, the model calculates the ionisation balance and then determines all ionic column densities by scaling to the prescribed total hydrogen column density. Using this set of column densities, the transmission of the plasma is calculated by multiplying the transmission of the individual ions.

The transmission includes both continuum and line opacity. For a description of what is currently in the absorption line database, we refer to *Absorption model theory* (page 277). By default, the model mimics the transmission of a neutral plasma by setting the default temperature to 8E-3 eV (8 10^{-6} keV).

The parameters of the model are:

`nh` : Hydrogen column density in 10^{28} m^{-2}. Default value: 10^{-4} (corresponding to 10^{24} m^{-2}, a typical value at low Galactic latitudes).

`t` : the electron temperature $T_{\rm e}$ in keV. Default value: 1.
`rt` : the ratio of ionization balance to electron temperature, $R_{\rm b} = T_{\rm b}/T_{\rm e}$ in keV. Default value: 1.
`hden` : Hydrogen density in 10^{20} $\rm m^{-3}$. Default value: 10^{-14} (corresponding to 10^{6} $\rm m^{-3}$, a typical value for the ISM).

The following parameters are common to all our absorption models:

`f` : The covering factor of the absorber. Default value: 1 (full covering)
`v` : Root mean square velocity $\sigma_{\rm v}$
`rms` : Rms velocity $\sigma_{\rm b}$ of line blend components
`dv` : Velocity distance Δv between different blend components
`zv` : Average systematic velocity v of the absorber

The following parameters are the same as for the cie-model (*Cie: collisional ionisation equilibrium model* (page 142)):

`ref` : Reference element
`01...30` : Abundances of H to Zn
`file` : Filename for the nonthermal electron distribution

Recommended citation: de Plaa et al. (2004) and Steenbrugge et al. (2005).

4.1.19 Hyd: model with user-own hydrodynamical simulation

This model calculates the spectrum of a plasma with a given set of ion concentrations. It allows users to calculate the X-ray spectrum corresponding to his own hydrodynamical simulation results. The basic usage is that the user first runs own calculations to obtain the ion concentrations, and stores them in an ascii file. The name should be either of `spexicon_abs.dat` or `spexicon_rel.dat`. Then the user runs SPEX and loads the Hyd model to calculate the spectrum.

For users who are familiar with Fortran, we offer the supporting fortran90 subroutine, *hydro_driver*, to make the ion concentration file conveniently. The usage of the *hydro_driver* is described in *Hydro driver* (page 188).

For more general cases users can directly load the Hyd model and just calculate the spectrum. The model has two modes to specify the format for the ion concentrations: absolute (mode 1) or relative (mode 2). For mode 1, the input file must have the name `spexicon_abs.dat`, and the values are treated as absolute ion concentrations (relative to hydrogen). In this mode the parameters of the elemental abundances do not affect the ion concentration nor the spectrum at all (they are ignored). For the case that users wish to treat elemental abundances as fitting parameters, mode=2 can be used. In this mode the input file must have the name `spexicon_rel.dat`, which contains the ion concentrations relative to the concentration of the relevant chemical element.

The parameters of the model are:

`hyd` : Hydrogen density in 10^{20} $\rm m^{-3}$
`mode` : Mode of the model. Mode=1: read absolute ion concentration from spexicon_abs.dat. mode=2: read relative ion concentration from spexicon_rel.dat.

The following parameters are the same as for the cie-model:

`norm` : the normalisation, which is the emission measure $Y \equiv n_{\rm e} n_{\rm H} V$ in units of 10^{64} m^{-3}, where $n_{\rm e}$ and $n_{\rm H}$ are the electron and Hydrogen densities and V the volume of the source. Default value: 1.
`t` : the electron temperature $T_{\rm e}$ in keV. Default value: 1.
`it` : Ion temperature in keV.
`vrms` : RMS Velocity broadening in km/s (see *Definition of the micro-turbulent velocity in SPEX* (page 293)).
`ref` : Reference element.
`01...30` : Abundances of H to Zn.
`file` : Filename for the non-thermal electron distribution.

Warning: By default, SPEX starts with `var calc old` (see the var menu for explanation). If you want to use this model with the latest atomic database, you should set the ionisation balance to U17. Note that this is only used to calculate inner-shell ionisation rates for the spectral evaluation, it will not affect the ion concentrations that the user provides.

Recommended citation (first use): Kosenko et al. (2015).

4.1.20 Knak: segmented power law transmission model

The knak model is used to model the transmission of any spectrum using a set of contiguous segments with a power-law transmission at each segment.

This component can be useful for new instruments, in order to test large scale calibration errors (effective area errors for example), but other applications can also be made, of course. For example, if the spectrum of the source has an unknown continuum shape with a superimposed absorption model, it is a good trick to model the continuum by a power law, modify that by a knak model with adjustable or fixed transmission at some relevant energies, and then apply the absorption. An example of this last application can be found in Porquet et al. (2004).

The Transmission is given by:

$$T(E) = c_i E^{a_i} \quad \text{if} \quad E_i < E < E_{i+1}$$

for each value of i between 0 and n, the number of grid points. The transmission is 1 for $E < E_1$ and $E > E_2$. Further, instead of using the constants c_i and a_i, we use instead the values of the transmission at E_i, i.e. $T_i \equiv T(E_i) = c_i E_i^{a_i}$. This allows for a continuous connection between neighbouring segments.

Finally, for historical reasons we use here a wavelength grid instead of an energy grid; the corresponding nodes λ_i should therefore be in strictly increasing order.

Warning: When applying this model, take care that at least one of the n transmission values is kept fixed (otherwise you may run the risk that your model is unconstrained, for example if the normalisation of the continuum is also a free parameter).

The parameters of the model are:

`n` : The number of grid points. Maximum value is 9.
`w1` : Wavelength λ_1 (Å) of the first grid point
`f1` : Transmission $T(\lambda_1)$ at λ_1.
`w2` : Wavelength λ_2 (Å) of the second grid point
`f1` : Transmission $T(\lambda_2)$ at λ_2.
...
`w9` : Wavelength λ_9 (Å) of the last grid point
`f9` : Transmission $T(\lambda_9)$ at λ_9.

Note that if $n < 9$, the values of T_i and λ_i will be ignored for $i > n$.

Recommended citation: Porquet et al. (2004).

4.1.21 Laor: relativistic line broadening model

This multiplicative model broadens an arbitrary additive component with a relativistic line profile. The relativistic line profile is interpolated from tables produced by Laor (1991). In his model, the line transfer function is determined for emission from the equatorial plane around a Kerr black hole, assuming an emissivity law $I(\cos\theta) \sim 1. + 2.06\cos\theta$. The transfer function is calculated at a grid of 35 radii ($r_n = 1600/(n+1)^2$ for $n = 1, \ldots, 35$, in units of GM/c^2), 31 inclinations uniformly spaced in $\cos i$, and 300 frequencies, logarithmically spaced between 0.1 and 5 times the emission frequency, respectively.

Using these numbers, a radial integration is done using an emissivity law

$$I(r) \sim 1/(r^2 + h^2)^{q/2},$$

where h is a characteristic scale height and q an asymptotic power law index (for large r, $I(r) \sim r^{-q}$). The integration is done between an inner radius r_1 and an outer radius r_2. Given the radial grid that is provided in Laor's data, the outer radius extends out to at most $400GM/c^2$.

Warning: Of course, any line or continuum emission component can be convolved with the this broadening model; for continuum components the computational time may be very large, however, due to the convolution involved.

Warning: The outer radius cannot be larger than $400GM/c^2$.

The parameters of the model are:

`r1` : Inner radius of the disk, in units of GM/c^2. The minimum allowed value is 1.234 (for a maximally rotating Kerr black hole). For a Schwarzschild black hole, one should take $r_i = 6$. Default value: 1.234.
`r2` : Outer radius of the disk, in units of GM/c^2. Keep this radius less than 400 (default value)
`q` : Emissivity slope q as described above. Default value: 2.
`h` : Emissivity scale height. Default value: 0.
`i` : Inclination angle (in degrees) of the disk (angle between line of sight and the rotation axis of the disk). Default value: 45 degrees.

Recommended citation: Laor (1991).

4.1.22 Line: transmission model for a single spectral line

This model calculates the transmission $T(E)$ for a single spectral emission or absorption model. It is used as a multiplicative component, and can be used both for absorption and emission lines. The line profile is a Voigt profile (the convolution between a Gaussian and a Lorentzian profile):

$$T(E) = e^{-\tau(E)},$$

with the optical depth $\tau(E)$ given by:

$$\tau(E) = \tau_0 K(x, y),$$

where $K(x, y)$ is the Voigt profile given by

$$K(x,y) = \frac{y}{\pi}\int_{-\infty}^{\infty}\frac{e^{-t^2}}{y^2+(x-t)^2}\mathrm{d}t,$$

with

$$x = \frac{2\sqrt{\ln 2}}{F_\mathrm{g}}(E - E_0)$$

and

$$y = \sqrt{\ln 2}\frac{F_\mathrm{l}}{F_\mathrm{d}}$$

where F_g and F_l are the Full Width at Half Maximum of the Gaussian and Lorentzian components, respectively. Further, E_0 is the line centroid.

The program also calculates the equivalent width of the line.

The model can be used both in energy (keV) mode or wavelength (Å) mode. In each case, the quantities for the other scaling (centroid, equivalent width, and FWHMs of the Gaussian and Lorentzian components) are calculated. In addition, like all SPEX absorption models, also the covering factor can be varied.

Warning: The equivalent width is calculated at the energy grid in use for SPEX. Line flux outside this range is not taken into account. That may be important in the case of strong Lorentzian wings.

Warning: The parameter τ_0 only represents the optical depth at the line centre for a pure Gaussian case. When the Lorentzian component becomes important, the "true" depth at line center is smaller.

Warning: When changing from energy to wavelength units, take care about the frozen/thawn status of the line centroid and FWHM.

Warning: You need to do a `calc` or `fit` command to get an update of the wavelength (for `type=0`) or energy (`type=1`).

The parameters of the model are:

`tau0` : Optical depth τ_0 at line center. Default value: 1. Positive values correspond to absorption lines, negative values to emission lines.
`el` : Line center energy E_0 (keV). Default value: 6.4 keV.
`fwhg` : FWHM of the Gaussian component F_g in keV. Default value: 0.1 keV.
`fwhl` : FWHM of the Lorentzian component F_l in keV. Default value: 0.1 keV.
`type` : Type of calculation. Type=0: energy units; type=1: wavelength units.
`wl` : Line center wavelength E_0 (Å). Default value: 20 Å.
`awhg` : FWHM of the Gaussian component F_g in Å. Default value: 0.1 Å.
`awhl` : FWHM of the Lorentzian component F_l in Å. Default value: 0.1 Å.
`ewk` : Equivalent width of the line in keV. Calculated by the program, not an input variable.
`ewa` : Equivalent width of the line in Å. Calculated by the program, not an input variable.
`fcov` : The covering factor of the absorption/emission line. Default value: 1 (full covering).

4.1.23 Lpro: spatial broadening model

This multiplicative model broadens an arbitrary additive component with an arbitrarily shaped spatial profile, in the case of dispersive spectrometers such as the RGS of XMM-Newton. In many instances, the effects of a spatially extended source can be approximated by making use of the fact that for small off-axis angles θ the expression $\mathrm{d}\lambda/\mathrm{d}\theta$ is almost independent of wavelength λ. This holds for example for the RGS of XMM-Newton (for which $\mathrm{d}\lambda/\mathrm{d}\theta = 0.138/m$ Å/arcmin, with m the spectral order).

We can utilize this for a grating spectrum as follows. Make an image $I(\Delta\theta)$ of your source projected onto the dispersion axis, as a function of the off-axis angle $\Delta\theta$. From the properties of your instrument, this can be transformed into an intensity $I(\Delta\lambda)$ as function of wavelength using $\Delta\lambda = \Delta\theta \mathrm{d}\lambda/\mathrm{d}\theta$. Assume that the spatial profile $I(\theta)$ is only non-zero within a given angular range (i.e. the source has a finite extent). Then we can transform $I(\Delta\lambda)$ into a probability distribution $f(x)$ with $f = 0$ for very small or large values of x (here and further we put $x = \Delta\lambda$). The auxilliary task rgsvprof (see *Rgsvprof* (page 189)) is able to create an input file for the lpro component from a MOS1 image.

The resulting spatially convolved spectrum $S_c(\lambda)$ is calculated from the original spectrum $S(\lambda)$ as

$$S_c(\lambda) = \int f(\lambda - \lambda_0) S(\lambda_0) \mathrm{d}\lambda_0.$$

The function $f(x)$ must correspond to a probability function, i.e. for all values of x we have

$$f(x) \geq 0$$

and furthermore

$$\int_{-\infty}^{\infty} f(x) \mathrm{d}x = 1.$$

In our implementation, we do not use $f(x)$ but instead the cumulative probability density function $F(x)$, which is related to $f(x)$ by

$$F(x) \equiv \int_{-\infty}^{x} f(y) \mathrm{d}y,$$

where obviously $F(-\infty) = 0$ and $F(\infty) = 1$. The reason for using the cumulative distribution is that this allows easier interpolation and conservation of photons in the numerical integrations.

If this component is used, you must have a file available which we call here `vprof.dat` (but any name is allowed). This is a simple ascii file, with n lines, and at each line two numbers: a value for x and the corresponding $F(x)$. The lines must be sorted in ascending order in x, and for $F(x)$ to be a proper probability distribution, it must be a non-decreasing function i.e. if $F(x_i) \leq F(x_{i+1})$ for all values of i between 1 and $n-1$. Furthermore, we demand that $F(x_1) \equiv 0$ and $F(x_n) \equiv 1$.

Note that $F(x)$ is dimensionless. In addition, we allow for two other parameters: a scale factor s and an offset λ_o. Usually, $s = 1$, but if s is varied the resulting broadening scales proportional to s. This is useful if for example one has an idea of the shape of the spatial profile, but wants to measure its width directly from the observed grating spectrum. In addition, the parameter λ_o can be varied if the absolute position of the source is unknown and a small linear shift in wavelength is necessary.

Warning: This model can be applied to grating spectra (like RGS), but if you include in your fit also other data (for example EPIC), the same broadening will also be applied to that other data SET. This can be avoided by using a separate sector for each detector type.

Warning: The above approximation of spatially extended sources assumes that there are no intrinsic spectral variations over the surface area of the X-ray source. Only total intensity variations over the surface area are taken into account. Whenever there are spatial variations in spectral shape (not in intensity) our method is strictly speaking not valid, but still gives more accurate results than a point-source approximation. In principle in those cases a more complicated analysis is needed.

The parameters of the model are:

`s` : Scale parameter s, dimensionless. Default value: 1.
`dlam` : Offset parameter λ_o, in Å. Default value: 0 Å.
`file` : Ascii character string, containing the actual name of the vprof.dat file

Recommended citation: Tamura et al. (2004)

4.1.24 Mbb: modified blackbody model

This model describes the spectrum of a black body modified by coherent Compton scattering. This is in several instances a much better description than a simple black body (for example accretion disk spectra of AGN). The physical background is described for example by Rybicki & Lightman (1986), pages 218–219. The formulae that we use here with a derivation are given by Kaastra & Barr (1989). From that work we derive the spectrum (10^{44} photons/s/keV):

$$N(E) = 1358. \frac{AE^{0.25}}{e^{E/T}(e^{E/T}-1)}$$

where E is the photon energy in keV, T the temperature in keV and A the normalisation in units of 10^{26} $\mathrm{m}^{0.5}$, defined by

$$A = n_{\mathrm{e}}^{0.5} O$$

with n_{e} the electron density (units: 10^{20} m^{-3}) and O the emitting source area (units: 10^{16} m).

The parameters of the model are:

`norm` : Normalisation A (in units of 10^{26} $\mathrm{m}^{0.5}$). Default value: 1.
`t` : The temperature T in keV. Default value: 1 keV.

Recommended citation: Kaastra & Barr (1989) and Rybicki & Lightman (1986).

4.1.25 Musr: User defined multiplicative model

The user model provides an interface to create an multiplicative model component calculated by a user program. The interface is rather simple and therefore very flexible. The interface uses a set of two ASCII files: `input-?-?.prm` and `output-?-?.spc`, where the '?' stands for the sector and component number of the musr model component.

The `input-?-?.prm` is generated by the musr model and contains the parameter values and the energy grid that SPEX uses to calculate the model. The file has a fixed structure:

- The first line contains one integer number containing the number of parameters that the model needs to calculate (current maximum is 40).
- On the second line the array of parameter values is listed (with the length indicated on the previous line.
- On the third line, an integer value is shown which indicates the number of model bins SPEX has in its model grid.
- On the following lines, the model grid points are shown in three columns. The first column contains the bin upper boundary (egb), the second column is the bin center (eg) and the third column is the full bin width (deg). The energy values are in keV.

The user program is supposed to read the input file and calculate the multiplicative factors for each energy bin on the grid. The spectrum array is called `sener` and has the same length as the energy grid. In addition, it is also possible to calculate the model for the emission weighted center of the bin. In that case the spectrum calculation is a bit more complicated, but it leads to better results. The user has the option of also calculating the weighted spectrum, which is the average photon energy $E_{\rm aver}$ minus the bin centroid $E_{\rm centroid}$ times the flux (F):

$$W = (E_{\rm aver} - E_{\rm centroid}) * F$$

The W values end up in the `wener` array. If you do not use weighing by average photon energy, `wener` values can be 0.

The user program should create an output ASCII file named `output-?-?.spc` with the following structure:

- On the first row, put the number of model bins. This should be the same number as written in the `input-?-?.prm` file.
- Write the `sener` and `wener` values in two columns. The first column is the `sener` value that represents the multiplication factor and the second column is the `wener` value.

In *How to use the SPEX user model* (page 73) some example user model programs can be found. The source code can be adapted to accommodate the user's wishes.

The equivalent of `musr` for additive models is *User: User defined model* (page 173).

The parameters of the model are:

`exec` : User executable, for example './model.py' for a python script in the same directory where you run SPEX.
`npar` : Number of free parameters in the model. Current maximum is 40.
`p01...p40` : Model parameters of the user model.

4.1.26 Neij: non-equilibrium ionisation jump model

This model calculates the spectrum of a plasma in non-equilibrium ionisation (NEI). For more details about NEI calculations, see *Non-equilibrium ionisation (NEI) calculations* (page 281).

The present model calculates the spectrum of a collisional ionisation equilibrium (CIE) plasma with uniform electron density $n_{\rm e}$ and temperature T_1, that is instantaneously heated or cooled to a temperature T_2. It then calculates the ionisation state and spectrum after a time t. Obviously, if t becomes large the plasma tends to an equilibrium plasma again at temperature T_2.

The ionisation history can be traced by defining an ionisation parameter,

$$u \equiv \int n_{\rm e} {\rm d}t$$

with $u = 0$ defined at the start of the shock.

By default the model describes a classical NEI condition with a flat temperature profile:

$$\begin{aligned} u < 0: &\quad T = T_1, \\ u > 0: &\quad T = T_2. \end{aligned}$$

For the case the user wands to calculate more complex situations, SPEX offers two modes to treat a temperature profile $T(u)$: analytic expression (mode 1) or ascii-file input (mode 2).

The temperature profile in mode=1 (analytic case) is given by

$$\begin{aligned} u < 0: &\quad T = T_1, \\ 0 < u < U: &\quad T = T_2, \\ U < u < U + dU: &\quad T = T(u). \end{aligned}$$

By setting a non-zero value for dU, this model offers the opportunity to calculate more complex evolution in the last epoch ($U < u < U + dU$); e.g. with secondary heating/cooling process and/or change in density. We

introduce parameters α and β, which describe a power-law like evolution respectively for temperature and density of the plasma after the "break" of constancy at time $t_{\rm br}$:

$$\begin{aligned} T(t) = &\quad T_2\ (t/t_{\rm br})^{\alpha} \\ n_{\rm e}(t) = &\quad n_{\rm e}\ (t/t_{\rm br})^{\beta}, \end{aligned} \tag{4.2}$$

An immediate application of this break feature would be a recombining plasma due to rarefaction (adiabatic expansion). Such a condition can be realised with $\alpha = -2$ and $\beta = -3$. Note that we include the effect of the density change here in the NEI calculation for the ion concentration, but of course the line emission is calculated at the density prescribed by the parameter *ed* of the model, which represents the true density at the epoch of emission of the spectrum.

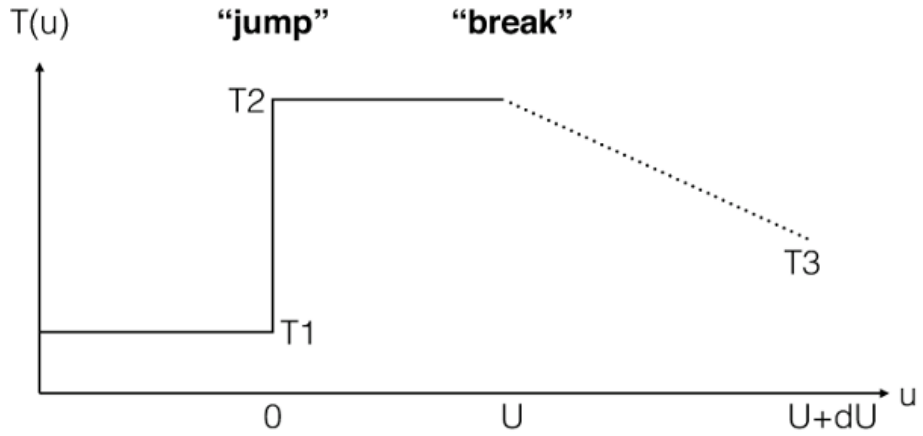


Fig. 1: The temperature profile with mode=1.

To get the expression for $T(u)$, we first calculate the increase of the ionisation parameter after $t = t_{\rm br}$ as follows:

$$\begin{aligned} u - U = &\quad \int_{t_{\rm br}} n(t)dt = \int_{t_{\rm br}} n_{\rm e}(t/t_{\rm br})^{\beta} dt \\ = &\quad n_{\rm e} t_{\rm br} \int_1 (t/t_{\rm br})^{\beta}\ d(t/t_{\rm br}) \\ = &\quad U/(\beta+1) \cdot [(t/t_{\rm br})^{\beta+1} - 1], \end{aligned} \tag{4.3}$$

Then, by combining equations (4.2) and (4.3), we obtain:

$$T(u) = T_2 \cdot [1 + (\beta+1) \cdot (u-U)/U]^{\alpha/(\beta+1)},$$

and we get the final temperature at $u = U + dU$ to be

$$T_3 = T_2 \cdot [1 + (\beta+1) \cdot dU/U]^{\alpha/(\beta+1)}.$$

It should be noted that, for fixed values of α and β, the temperature change after the break is determined by the ratio dU/U rather than dU itself. The user can check T_3 with the `ascdump plas` command (see *Ascdump: ascii output of plasma properties* (page 79)) and also the histories of u and $T(u)$ with the `ascdump nei` command (see *Ascdump: ascii output of plasma properties* (page 79)).

In some rare cases with a large negative β, T_3 can get an unphysical value ($T_3 < 0$). In such a case the calculation will automatically be stopped at a lower-limit of $T(u) = 10^{-4}$ keV.

For mode 2, the user may enter an ascii-file with u- and T-values. The format of this file is as follows: the first line contains the number of data pairs (u, T). The next lines contain the values of u (in the SPEX units of 10^{20} s m^{-3}) and T (in keV). Note that $u_1 = 0$ is a requirement, all T_i should be positive, and the array of u-values should be in ascending order. The pairs (u, T) determine the ionisation history, starting from $T = T_1$ (the pre-shock temperature), and the final (radiation) temperature is the temperature of the last bin.

The parameters of the model are:

`t1` : Temperature T_1 before the sudden change in temperature, in keV. Default: 0.002 keV.
`t2` : Temperature T_2 after the sudden change in temperature, in keV. Default: 1 keV.
`u` : Ionization parameter $U = n_e t$ before the "break", in 10^{20} m^{-3} s. Default: 10^{-4}.
`du` : Ionization parameter dU after the "break" in 10^{20} m^{-3} s. Default value is 0 (no break).
`alfa` : Slope α of the $T(t)$ curve after the "break". Default value is 0 (constant T).
`beta` : Slope β of the $n(t)$ curve after the "break". Default value is 0 (constant n).
`mode` : Mode of the model. Mode=1: analytical case; mode=2: $T(u)$ read from a file. In the latter case, also the parameter `hisu` needs to be specified.
`hisu` : Filename with the $T(u)$ values. Only used when mode=2.

The following parameters are the same as for the cie-model (*Cie: collisional ionisation equilibrium model* (page 142)):

`ed` : Electron density in 10^{20} m^{-3}
`it` : Ion temperature in keV
`vrms` : RMS Velocity broadening in km/s (see *Definition of the micro-turbulent velocity in SPEX* (page 293))
`ref` : Reference element
`01...30` : Abundances of H to Zn
`file` : Filename for the nonthermal electron distribution

Recommended citation: Kaastra & Jansen (1993).

4.1.27 Pdem: DEM models

The pdem model is intended to be used for differential emission measure analysis, simultaneous with fitting of abundances etc. of an optically thin plasma.

It works as follows. The user gives a a number of temperature grid points n, a minimum temperature T_1, a maximum temperature T_n, a total emission measure Y and relative contributions $y_1 \ldots y_n$. SPEX assumes that the grid points between T_1 and T_n are distributed logarithmically.

The relative contributions $y_1 \ldots y_n$ represents the values of $\mathrm{d}Y/\mathrm{d}\ln T$ (note the logarithm!) at the grid points. SPEX then interpolates in the $\log T_i - \log y_i$ space on a finer grid using splines. That temperature grid on which the data are being evaluated has a fine mesh: step size is about 0.10 in $\log T$ (finer is not useful because uniqueness is no longer guaranteed), with the additional constraint that the number of mesh points is at least n and not larger than 64 (practical limit in order to avoid excessive cpu-time). The emission measure distribution is then simply scaled in such a way that its sum over the fine mesh equals the total emission measure Y that went into the model.

Warning: At least one of the y_i values should be kept frozen during fitting, when Y is a free parameter, otherwise no well defined solution can be obtained! If Y is fixed, then all y_i can be free.

The parameters of the model are:

`norm` : Normalisation, i.e. total integrated emission measure Y in units of 10^{64} m^{-3}.
`t1` : Lower temperature T_1 in keV.
`tn` : Upper temperature T_n in keV.
`npol` : Number of temperature grid points n; minimum value 2, maximum value 8.
`y1` : Relative contribution y_1 at T_1.
`y2` : Relative contribution y_2 at T_2
...

`y8` : Relative contribution y_8 at T_8; note that the higher contributions y_i are neglected if $i > n$.

The following parameters are the same as for the cie-model:

`ed` : Electron density in 10^{20} m^{-3}.
`it` : Ion temperature in keV.
`vrms` : RMS Velocity broadening in km/s (see *Definition of the micro-turbulent velocity in SPEX* (page 293))
`ref` : Reference element.
`01...30` : Abundances of H to Zn.
`file` : Filename for the nonthermal electron distribution

Note that the true emission measure on the finer mesh can be displayed by using the `ascdump term # # dem` command; you will get a list of temperature (keV) versus emission measure.

4.1.28 Pion: SPEX photoionised plasma model

The *pion* model calculates the transmission and emission of a slab of photo-ionized plasma, where all ionic column densities are linked through a photoionisation model. The relevant parameter is the ionization parameter $\xi = L/nr^2$, with L the source luminosity, n the hydrogen density and r the distance from the ionizing source. The major difference is that while for the *xabs* model (*Xabs: photoionised absorption model* (page 177)) the photoionisation equilibrium is pre-calculated for a grid of ξ-values by an external code, for instance Cloudy or XSTAR (or even with the present *pion* model ...), in the present *pion* model the photoionisation equilibrium is calculated self-consistently using the available plasma routines of SPEX.

Warning: The default energy grid in SPEX has 8192 bins between 0.001 and 100 keV. This may not be sufficient for the pion model when you use it without data. Recommended is a logarithmic grid between $10^{-6} - 10^{6}$ keV with a step size of 0.005. You can get this by issuing the following command: "Egrid log 1E-6 : 1E6 step 0.005 keV". Note that if you have read in data, SPEX automatically uses the resolution of the response matrix within its energy range and expands the total energy grid to $10^{-6} - 10^{6}$ keV.

Warning: This model is still under development and not all atomic data is fully updated. For instance, no cooling by collisional excitation for ions of the K- to Zn-isoelectronic sequences is taken into account yet. So use with care!

Warning: When setting up the model, be aware that the pion model is both additive and multiplicative (even if you put the emission to zero). Therefore, the pion model needs the same com rel sequence as you use for your absorption component. Example: `com pow` — `com reds` — `com pion` — `com rel 1 3,2` (a powerlaw that powers a pion model and is then redshifted) needs as next command `com rel 3 2`, telling SPEX that the pion emission model is also redshifted.

Warning: Please note that all PION components must be multiplied by at least one continuum component (using the usual comp rel command). Otherwise, photoionisation cannot be calculated and SPEX may crash. A typical AGN SED spanning optical to X-ray energies can be set up with three continuum components in SPEX: pow (primary hard X-ray power-law emission), refl (reflected power-law emission), comt (thermal optical/UV disk continuum + the soft X-ray excess). See the SED of NGC 5548 derived in Mehdipour et al. (2015).

Warning: Starting from SPEX Version 3.06, we have updated the cooling due to collisional excitation (work performed by Stofanova et al., 2021, submitted). In addition, starting from SPEX Version 3.06.01, we have improved the collisional excitation rates of neutral hydrogen; the latter affects both emitted H I spectra, and the total cooling rate at the lower temperatures (few eV). That cooling rate may differ at some temperatures by a factor of 3-4. Therefore results obtained with the most recent version will differ from those of version 3.05 and earlier.

The main advantage, however, is that the user can define his own ionising continuum by combining any additive components of SPEX, and that ionising spectrum is fed through the *pion* component to get the transmitted spectrum. In addition, multiple *pion* components can be used, with the light leaking from one layer to the next. And it allows for spectral fitting, where the parameters of the ionising continuum are determined simultaneously with the parameters of the absorbing layer. The number of parameters is therefore unlimited (well, the memory and cpu speed of the computer will be the true limiting factors).

Prior to fitting a PION component, it is best to first calculate the model spectrum with your initial parameter values (see *Calculate: evaluate the spectrum* (page 81)). This helps to see if the initial values are reasonable numbers and the model is not too far off the data.

Emission from the *pion* model

We have now incorporated a first version of emission from the same layer. We cannot give you any guarantee at the moment that it is bug-free. We know at the moment that the source has problems when the density gets too high; this is different for each ion; so unless you limit the density in fitting, SPEX may encounter a situation where you surpass the critical density and you may get a warning message. Here is an example. For a photoionised case, with $\log \xi = 3$ (resulting $kT = 0.64$ keV) the nominal occupation of the ground state of H becomes negative for a density $> 0.15 \times 10^{20}$ m^{-3}. This can be traced down to incomplete atomic data. For H, we include collisional excitation and de-excitation up to principal quantum number $n = 5$ but not above. As a result, in this example the 1s–5s levels are mainly populated/depopulated by collisions, while 6s–16s are mainly populated by radiative recombination and depopulated by radiative cascades downward or photon absorption. The nominal occupation of 1s–5s then decreases from 0.035 (1s) to 0.0024 (5s), while for 6s–16s they increase slowly from 0.020 to 0.027. This is of course physically unacceptable. It causes the lower levels to leak to the higher levels, with eventually the catastrophe of negative occupation for the ground state. We mitigate this by replacing the level populations by the LTE populations and issueing a warning. Without this mitigation, SPEX would crash.

Warning: You can get the emission by putting the covering factor (omeg) to a non-zero value; it will slow down the calculations compared to absorption-only calculations, so be aware.

Normally, to calculate the emission from a full thin shell surrounding an ionising source, you should set the parameter *omeg* to unity (a full shell of 4π steradians). Smaller factors could be associated e.g. to ionisation cones; values larger than unity make physically no sense but you can formally play around with it (for very large values, the emitted spectrum would start dominating the absorbed primary continuum, but if you want to suppress the primary continuum in the observed spectrum, it is better to define your model like the example below as:

```
SPEX> com pow
SPEX> com pion
SPEX> com etau
SPEX> com rel 1 2,3
SPEX> com rel 2 0
SPEX> par 3 tau v 1e10
SPEX> par 3 a v 0
```

In this example, the powerlaw goes through the *pion* component and is killed afterwards by the *etau* component (*Etau: simple transmission model* (page 151)), while the emission from the *pion* component is not attenuated by *etau*.

You can vary the new parameter *mix* to get a different ratio of forwards to backwards emission. Putting it to 1 (default) means you get the forward emission, putting it to 0 the backwards emission, and intermediate values give

you a mix.

Warning: The emission model uses currently only one layer. When the continuum optical depth of the absorbed continuum, weighted with the incoming flux, becomes of order unity, the layer will become inhomogeneous in terms of temperature structure, and our single-layer approximation will break down.

In order to make a *pion* component produce emission only, fix the covering fraction (cf) parameter to zero so that no absorption is produced. Then fit the omega parameter. Note that any *pion* component with a non-zero omega acts as an additive component in SPEX. Therefore, multiply these components with your multiplicative components (like the Galactic absorption) using the `comp rel` command.

For more information on this model, the atomic data and parameters we refer to *Different types of absorption models* (page 277).

More options

No energy balance solution needed

The default option (*tmod=0*) for the *pion* model is to solve simultaneously for the ionisation balance and the energy balance equations. This option is useful for e.g. photoionised winds of AGN or X-ray binaries.

However, there are situations where photo-ionisation or photo-excitation play a role but do not determine the thermal structure. Examples are winds of hot stars, where shocks heat the wind but UV radiation from the star can affect He-like triplet line emission ratios. Another example are the most teneous parts of the WHIM, where photoionisation by the cosmic background can be important compared to collisional ionisations.

For such cases, the user can set the parameter *tmod=1*; in that case, the user should also provide the temperature *tinp* of the plasma. In this case, only the ionisation balance equation is solved, and there is in general no energy balance (this can be checked by using the ascii-output option *heat*, *Ascdump: ascii output of plasma properties* (page 79)). Do not forget to set the parameter *omeg* to a finite value (the default is zero), otherwise the emitted spectrum is zero.

External heat sources

In some cases there may be an other external heat or cooling source, like shock heating, magnetic reconnection, adiabatic expansion etc. If one wishes to solve for the photoionisation equilibrium, then this additional heat source can be used by putting the parameter *exth* to the proper value (units: W m^{-3}. A negative value would mean a cooling contribution.

Multiple solutions

There are situations where there is not a unique solution to the energy balance equations. A simple example can be obtained as follows: take a logarithmic energy grid between $10^{-6} - 10^{6}$ keV, use a powerlaw with photon index 1.5, apply the pion model to it and put *exth* to 5×10^{-25} W m^{-3}. In this case there are 3 solutions. SPEX chooses by default the hottest solution. You can see all solutions by putting the parameter *fmod=1* and using the *heat* ascii output option. Or check the behaviour of the heating balance by issuing the *ebal* ascii output option (*Ascdump: ascii output of plasma properties* (page 79)). You can select which solution you want to use in SPEX by setting the *soln* parameter. Default is 0 (hottest solution), and for the above case of 3 solutions values of 1, 2 and 3 renders you the coldest, second ant hottest solution. Test this with the *heat* or *plas* output options.

Warning: When you set *soln* to a non-zero value, use fmod=1, otherwise SPEX may crash.

No equilibrium solution

There are also situations where there is no equilibrium solution to the energy balance equations. This may happen for instance if you put so much heat in the plasma that it cannot be balanced anymore by cooling. Another example is a too hard powerlaw without high energy cut-off, where Compton-heating might be very strong. In this case SPEX renders an error message, and you cannot trust the result of the calculation anymore. The only remedy is to adjust your model parameters or the allowed range for them in case of spectral fitting or error searches.

Adiabatic cooling

The effects of adiabatic cooling can be taken into account by setting the parameter *tadi*. This represents the adiabatic cooling time t_{adi}. The associated cooling rate is calculated as $R_{adi} = \frac{3}{2}nkT/t_{adi}$, where n is the total particle density (electrons and ions). The default setting is such that this process can be neglected. If the user takes this process into account, it should be verified afterwards that the physical conditions for adiabatic cooling are met, i.e. energy losses by radiation or heat conduction must be small compared with those by the adiabatic expansion. Check this for example by running the `asc ter ... heat` output.

Radiative acceleration

The radiative acceleration caused by the absorption or scattering of the incoming radiation on the layer is calculated, and given as output parameter *acc*. Physically, it is given by the following equation, which can be easily derived:

$$a = F_{abs}/cfm_{\rm p}N_{\rm H},$$

where $F_{abs} = \int F(E)(1-T(E))\mathrm{d}E$ is the absorbed flux ($F(E)$ is the incoming flux in W m^{-2} keV and $T(E)$ the transmission of the layer), c the speed of light, $m_{\rm p}$ the proton mass, $N_{\rm H}$ the hydrogen column density and f is a dimension less quantity determined from $\rho = fn_{\rm H}m_{\rm p}$ with $n_{\rm H}$ the hydrogen density and ρ the mass density (kg m^{-3}) of the plasma, for example $f = 1.4287$ for the present default abundances of SPEX (you can check this number from the `asc ter ... plas` ascii output option).

Warning: It is important to note that the acceleration is proportional to the hydrogen density, so take care to choose the appropriate hydrogen density, even in the low density limit where the spectral shape does not depend on the density. This counter-intuitive behaviour is caused by the use of ξ as main parameter. Given the absolute ionising luminosity L of the ionising source, and the value of ξ and $n_{\rm H}$ provided by the user, the pion model then calculates the distance r from the equation $\xi = L/nr^2$. Thus, higher density yields smaller distance, hence larger absorbed flux by the gas layer, hence stronger acceleration.

Model parameters

The parameters of the model are:

`nh` : Hydrogen column density in 10^{28} m^{-2}. Default value: 10^{-4} (corresponding to 10^{24} m^{-2}, a typical value at low Galactic latitudes).
`xi` : the $^{10}\log$ of the ionisation parameter ξ in units of 10^{-9} W m. Default value: 1.
`u` : the Davidson (Cloudy) ionisation parameter U (dimensionless). This is calculated from the SED and the value of ξ. Not fittable, just output.
`hden` : the hydrogen density in units of 10^{20} m^{-3}. Default value: 10^{-14} (corresponding to 10^{6} m^{-3}, to be consistent with the order of magnitude of the electron density (which is used in the cie and neij models; do NOT confuse both quantities!).

The following parameters are common to all our absorption models:

`fcov` : The covering factor of the absorber. Default value: 1 (full covering)
`v` : Root mean square velocity $\sigma_{\rm v}$
`rms` : Rms velocity $\sigma_{\rm b}$ of line blend components
`dv` : Velocity distance Δv between different blend components
`zv` : Average systematic velocity v of the absorber
The following parameters are the same as for the cie-model (see there for a description):
`ref` : Reference element
`01...28` : Abundances of H to Ni; only here we take H, He, C, N, O, Ne, Na, Mg, Al, Si, S, Ar, Ca, Fe, Ni.
`file` : File name for the electron distribution (in case of a sum of Maxwellians distribution)
The following parameters are unique for the *pion* model: `type` : If type equals 0 (default value), it uses ξ as its main parameter; if type equals 1, it uses lixi (see next parameter) as its main parameter
`lixi` : Optional alternative ionisation parameter, defines as $L_{\rm ion}/\xi$ in units of 10^{39} $\rm m^{-1}$. This is useful for time-variable spectra where ξ has been determined from one spectrum and where one wants to calculated the transmitted spectrum for fixed nr^2 for a different ionising spectrum; in that case lixi can be kept constant.
`omeg` : Covering factor $\Omega/4\pi$, needed for emission. At this stage, keep it to zero, please.
`mix` : Fraction of emitted spectrum to the forward direction relative to the total. default value: 1 (all emission forward). A value of 0 means SPEX gives all backwards emission.
`exth` : External heating in W $\rm m^{-3}$. default value: 0.
`fmod` : Show all solutions in ascii output of the heating (fmod=1). Default is fmod=0. Set fmod=1 also when you set soln>0.
`soln` : The temperature solution to be used, from low to high values. Default value is 0 (hottest solution). If this parameter is larger than the hootest solution, it adopts the hottest solution instead. Should be used with fmod=1 in case soln>0.
`tmod` : Temperature mode. Default value: 0 (solve for the temperature that provides energy balance). If tmod=1, use *tinp* instead as temperature and do not solve for energy balance.
`tinp` : Temperature of the plasma in keV. Default: 1 keV. Only relevant if *tmod=1*.
`tadi` : Adiabatic cooling time scale (s). See description above. Default value: 10^{30} s.
`acc` : Radiative acceleration. See description above. Note: only output.

Recommended citation: Miller et al. (2015) and Mehdipour et al. (2016)

4.1.29 Pow: power law model

The power law spectrum as given here is a generalization of a simple power law with the possibility of a break, such that the resultant spectrum in the $\log F - \log E$ plane is a hyperbola.

The spectrum is given by:

$$F(E) = AE^{-\Gamma}e^{\eta(E)},$$

where E is the photon energy in keV, F the photon flux in units of 10^{44} ph $\rm s^{-1}keV^{-1}$, and the function $\eta(E)$ is given by

$$\eta(E) = \frac{r\xi + \sqrt{r^2\xi^2 + b^2(1-r^2)}}{1-r^2},$$

with $\xi \equiv \ln(E/E_0)$, and E_0, r and b adjustable parameters. For high energies, ξ becomes large and then η approaches $2r\xi/(1-r^2)$, while for low energies ξ approaches $-\infty$ and as a consequence η goes to zero. Therefore the break $\Delta\Gamma$ in the spectrum is $\Delta\Gamma = 2r\xi/(1-r^2)$. Inverting this we have

$$r = \frac{\sqrt{1+(\Delta\Gamma)^2}-1}{|\Delta\Gamma|}.$$

The parameter b gives the distance (in logarithmic units) from the interception point of the asymptotes of the hyperbola to the hyperbola. A value of $b = 0$ therefore means a sharp break, while for larger values of b the break gets smoother.

The simple power law model is obtained by having $\Delta\Gamma = 0$, or the break energy E_0 put to a very large value.

Warning: By default, the allowed range for the photon index Γ is (-10,10). If you manually increase the limits, you may run the risk that SPEX crashes due to overflow for very large photon indices.

Warning: Note the sign of Γ: positive values correspond to spectra decreasing with energy. A spectrum with $\Delta\Gamma > 0$ therefore steepens/softens at high energies, for $\Delta\Gamma < 0$ it hardens.

As an extension, we allow for a different normalisation, namely the integrated luminosity L in a given energy band E_1–E_2. If you choose this option, the parameter "type" should be set to 1. The reason for introducing this option is that in several cases you may have a spectrum that does not include energies around 1 keV. In that case the energy at which the normalisation A is determined is outside your fit range, and the nominal error bars on A can be much larger than the actual flux uncertainty over the fitted range. Note that the parameters E_1 and E_2 act independently from whatever range you specify using the "elim" command. Also, the luminosity is purely the luminosity of the power law, not corrected for any transmission effects that you may have specified in other spectral components.

Warning: When you do spectral fitting, you **must** keep either A or L a fixed parameter! The other parameter will then be calculated automatically whenever you give the calculate or fit command. SPEX does not check this for you! If you do not do this, you may get unexpected results / crashes.

Warning: The conversion factor between L and A is calculated numerically and not analytically (because of the possible break). In the power law model, photon fluxes above the nominal limit (currently $e^{34} = 5.8 \times 10^{14}$ in unscaled units) are put to the maximum value in order to prevent numerical overflow. This implies that you get inaccurate results for low energies, for example for a simple power law with $\Gamma = 2$ the results (including conversion factors) for $E < 10^{-7}$ keV become inaccurate.

Warning: Note that when you include a break, the value of Γ is the photon index at energies below the break. Also, the normalisation A is the nominal normalisation of this low-energy part. In such a case of a break, the true flux at 1 keV may be different from the value of A! Of course, you can always calculate the flux in a given band separately.

The parameters of the model are:

`norm` : Normalisation A of the power law, in units of 10^{44} ph s^{-1} keV^{-1} at 1 keV. Default value: 1. When $\Delta\Gamma$ is not equal to 0, it is the asymptotic value at 1 keV of the low-energy branch.

`gamm` : The photon index Γ of the spectrum. Default value: 2. When $\Delta\Gamma$ is not equal to 0, it is the slope of the low-energy branch.

`dgam` : The photon index break $\Delta\Gamma$ of the spectrum. Default value: 0. and frozen. If no break is desired, keep this parameter 0 (and frozen!).

`e0` : The break energy E_0 (keV) of the spectrum. Default value: 10^{10} and frozen.

`b` : Smoothness of the break b. Default: 0.

`type` : Type of normalisation. Type$= 0$ (default): use A; type$= 1$: use L.

`elow` : E_1 in keV, the lower limit for the luminosity calculation. Default value: 2 keV.

`eupp` : E_2 in keV, the upper limit for the luminosity calculation. Default value: 10 keV. Take care that $E_2 > E_1$.

`lum` : Luminosity L between E_1 and E_2, in units of 10^{30} W.

4.1.30 Reds: redshift model

This multiplicative model applies a redshift z to an arbitrary additive component. If a photon is emitted at energy E, the redshift model will put it at energy $(1+z)E$. In addition, a time dilatation correction is applied such that the spectrum $S(E)$ (expressed as photons/s per bin) is divided by $1+z$.

However, it is necessary to distinguish two different cases:

Case 1 (flag=0): In this case, the time dilatation correction described above is applied to the spectrum, which makes this the (default) option to be used for cosmological redshifts.

Case 2 (flag=1): By setting this flag, the time dilatation correction to the flux is not applied. This is used for a redshift caused by motion away from us. It should be used for any velocity fields other than the Hubble flow.

Warning: Note that this component should be used in tandem with the distance command (*Distance: set the source distance* (page 86)) to take into account the cosmological redshift and its influence on the flux completely.

The parameters of the model are:

`z` : Redshift z. Default value: 0. Dimensionless. Redshifts are positive, blueshifts negative.
`flag` : Flag: 0 (cosmological redshift) or 1 (velocity redshift). Default value: 0.

4.1.31 Refl: reflection model

This model was kindly provided by Piotr Zycki. The related programs in XSPEC are `felipl` and `ferfschw`. The first one gives the reflected continuum (i.e. pexrav/pexriv) plus the line with correct energy and intensity, the second one is the first one with the relativistic smearing added. Both are combined into the single refl model in SPEX.

It is a model of reflection from a constant density X-ray illuminated atmosphere. It computes the Compton-reflected continuum (Magdziarz & Zdziarski, 1995); and the iron K alpha line (Zycki & Czerny, 1994), as described in Zicky et al. (1999). In addition, it can be convolved with a relativistic diskline model (for Schwarzschild geometry).

Chemical elements taken into account in this model are H, He, C, N, O, Ne, Mg, Si, S and Fe. The standard abundances are taken from Morrison & McCammon (1983).

The incoming spectrum is characterized by:

$$N_i(E) = AE^{-\Gamma} \exp\left[-E/E_c\right],$$

where E is the photon energy in keV, $N_i(E)$ the number of photons per per second per keV, Γ is the photon index and E_c a cut-off energy. The normalisation A is in units of 10^{44} photons s^{-1} keV^{-1} at 1 keV, just the same as in the standard power law model. The total spectrum $N(E)$ is then given by

$$N(E) = N_i(E) + sR(E),$$

where $R(E)$ is the reflected spectrum and s is a scaling factor.

The parameters of the model are:

`norm` : Normalisation A of the power law.
`gamm` : The photon index Γ of the ionising spectrum.
`ecut` : The cut-off energy (keV) of the ionising spectrum. If no cut-off is desired, take this parameter zero (and keep it frozen!).

`pow` : If pow=1, the incoming power law is added to the spectrum (default); if pow=0, only the reflected spectrum is given.
`disk` : If disk=1, the spectrum will be convolved with an accretion disk profile (default); if disk=0, this is not done.
`fgr` : Full general relativity used (default, for fgr=1).
`t` : Temperature of the reflector (disk) in keV.
`xi` : Ionisation parameter $\xi = L/nr^2$ in the usual (c.g.s. based) units of 10^{-9} W m.
`abun` : The abundance of all metals excluding H and He, in solar units
`feab` : The iron abundance with respect to the other metals
`cosi` : The cosine of the inclination angle of the disk. $\cos i = 0$ ($i = \pi/2$) corresponds to edge-on
`scal` : Scale s for reflection. For an isotropic source above the disk $s = 1$. This value corresponds to seeing equal contributions from the reflected and direct spectra.
`q` : Emissivity index for the accretion disk; default value -3 (the emissivity scales with r^{+q} at large radii, so $q = -3$ means r^-3. Note the sign difference with the Laor model.
`r1` : Inner radius of the disk in units of GM/c^2. Default: 10.
`r2` : Outer radius of the disk in units of GM/c^2. Default: 10^4.

Recommended citation: Magdziarz & Zdziarski (1995) and Zicky et al. (1999)

4.1.32 Rrc: radiative recombination continuum model

This is a simplified model, aimed to calculate radiative recombination continua for photoionized plasmas. It is a simple, limited shortcut to a more fully complete model for the emission from a recombining plasma.

The user essentially prescribes the emission measure for each ion as well as the radiation temperature, and then this model calculates the *continuum* emission corresponding to this temperature and set of ionic emission measures. Line radiation is *not* taken into account. However, for physical self-consistency, we take account of all three continuum emission components: Bremsstrahlung, two photon emission and free-bound radiation (= the RRC).

The reason for having no physical model to couple the ionic emission measures (contrary to for example the CIE model), is that this allows the user to fit these emission measures without making a priori assumptions about the ionization equilibrium. The user might then combine later the set of derived emission measures with any of his relevant models.

Warning: Take care that for too high temperatures, two photon emission might be stronger than the free-bound (RRC) emission!

Warning: Take care that the fit parameters are emission measures of a given ion, while the radiation occurs in the next ion. For example radiative recombination of O XI to O VIII is proportional to the emission measure of O IX ($n_\mathrm{e} n_{\mathrm{O\,IX}} V$), but produces an emission edge in O VIII at 14.22 Å.

Warning: No recombination is possible to neutrals, so therefore there is no H , O or Fe in this model.

The parameters of the model are:

`t` : The temperature T in keV. Default value: 1 keV.
`h2` : The H emission measure $n_\mathrm{e} n_{\mathrm{H\,II}} V$ in units of 10^{64} m^{-3}. Default value: 0.
`he2` : The He emission measure $n_\mathrm{e} n_{\mathrm{He\,II}} V$ in units of 10^{64} m^{-3}. Default value: 0.
`he3` : The He emission measure $n_\mathrm{e} n_{\mathrm{He\,III}} V$ in units of 10^{64} m^{-3}. Default value: 0.

...
ni29 : The Ni emission measure $n_{\rm e} n_{\rm Ni\,XXIX} V$ in units of 10^{64} $\rm m^{-3}$. Default value: 0.

4.1.33 Slab: thin slab absorption model

The *slab* model calculates the transmission of a slab of material, where all ionic column densities can be chosen independently. This has the advantage that a spectrum can be fit without any knowledge of the ionisation balance of the slab. After a spectral fit has been made, one may try to explain the observed column densities by comparing the with predictions from any model (as calculated by SPEX, Cloudy, XSTAR, ION or any other existing (photo)ionisation code).

For more information on this model, the atomic data and parameters we refer to *Different types of absorption models* (page 277). Below we do not list all the parameters of the model, but for each ion of H, He, C, N, O, Ne, Na, Mg, Al, Si, S, Ar, Ca, Fe and Ni there is a parameter, namely the logarithm (base 10) of the column density in SI units ($\rm m^{-2}$). So, a H column of 10^{28} $\rm m^{-2}$ (10^{24} $\rm cm^{-2}$ should be entered as 28.0. The default values of 1 therefore correspond to an almost negligible column.

Warning: We do include here fully stripped ions, as their free electrons do produce Thomson scattering. However, the user is advised not to take more than one column density of a bare nucleus as a free parameter, as the shape of the Thomson scattering contribution is the same for all electrons, regardless from which ion they came from. In the spectral fitting, there would be 100% correlations between the column densities of these bare ions, which is undesirable.

The parameters of the model are:

h1 : log column density ($\rm m^{-2}$) of H . Default value: 1.
h2 : log column density ($\rm m^{-2}$) of H . Default value: 1.
he1 : log column density ($\rm m^{-2}$) of He . Default value: 1.
he2 : log column density ($\rm m^{-2}$) of He . Default value: 1.
he2 : log column density ($\rm m^{-2}$) of He . Default value: 1.
c1 : log column density ($\rm m^{-2}$) of H . Default value: 1.
...
ni27 : log column density ($\rm m^{-2}$) of Ni . Default value: 1.
ni28 : log column density ($\rm m^{-2}$) of Ni . Default value: 1.

The following parameters are common to all our absorption models:

fcov : The covering factor of the absorber. Default value: 1 (full covering)
v : Root mean square velocity $\sigma_{\rm v}$
rms : Rms velocity $\sigma_{\rm b}$ of line blend components
dv : Velocity distance Δv between different blend components
zv : Average systematic velocity v of the absorber

Recommended citation: Kaastra et al. (2002).

4.1.34 Spln: spline continuum model

Sometimes the continuum of an X-ray source may be too complex to model with known physical components. A situation like that may be found in AGN continua, which are a complex superposition of hard power law, soft continuum excess, relativistically broadened and "normal" broad lines with a priori unknown line shape, etc., while in addition a superimposed warm absorber may have well defined narrow absorption lines. In that case it might be useful to fit the continuum with an arbitrary profile in order to get first an accurate description of the absorber, and then after having "removed" the absorber try to understand the underlying continuum spectrum.

For these situations the *spln* model introduced here is useful. It allows the user to model the continuum within two boundaries b_1 and b_2 with a cubic spline.

The algorithm works as follows. The user selects the limits b_1 and b_2 as well as the number of grid points n. SPEX then creates a grid x_1, x_2, ..., x_n with uniform spacing in b (see below for details). The spectrum at these grid points is contained in the corresponding array y_1, y_2, ..., y_n. These have the usual units of 10^{44} photons s^{-1} keV^{-1} used throughout SPEX, and is the spectrum emitted at the source. The parameters y_i can be adjusted during the spectral fitting, but b_1, b_2 and n and thereby x_i remain fixed. At intermediate points between the x_i, the photon spectrum is determined by cubic spline interpolation on the (x_i, y_i) data pairs. We take a natural spline, i.e. at x_1 and x_n the second derivative of the spline is zero.

Outside of the range b_1–b_2 however, the photon spectrum is put to zero, i.e. no extrapolation is made!

Finally note that we have chosen to define the spline in logarithmic space of y, i.e. the $\log$ of the photon spectrum is fit by a spline. This is done in order to guarantee that the spectrum remains non-negative everywhere. However, the y-values listed is the (linear) photon spectrum itself.

There are four different options for the energy grid x_i, indicated by the parameter *type*:

- *type*=1: b_1 is the lower energy in keV, b_2 is the upper energy in keV, and the grid is linear in energy in between.
- *type*=2: b_1 is the lower energy in keV, b_2 is the upper energy in keV, and the grid is logarithmic in energy in between.
- *type*=3: b_1 is the lower wavelength in Å, b_2 is the upper wavelength in Å, and the grid is linear in wavelength in between.
- *type*=4: b_1 is the lower wavelength in Å, b_2 is the upper wavelength in Å, and the grid is logarithmic in wavelength in between.

Note that the logarithmic grids can also be used if one wants to keep a fixed velocity resolution (for broadened line features for example). Further, each time that the model is being evaluated, the relevant values of the x_i grid points are evaluated.

Warning: Be aware that if you just set b_1, b_2 and n but do not issue the "calc" command or the "fit" command, the x_i values have not yet been calculated and any listed values that you get with the `par show` command will be wrong. After the first calculation, they are right.

Warning: If at any time you change one of the parameters $type$, b_1, b_2 or n, the y_i values will not be appropriate anymore as they correspond to the previous set of x_i values.

The maximum number n of grid points that is allowed is 999, for practical reasons. Should you wish to have a larger number, then you must define multiple *spln* components, each spanning its own (disjunct) b_1–b_2 range.

It should be noted, however, that if you take n very large, spectral fitting may become slow, in particular if you take your initial guesses of the y_i parameters not too close to the true values. The reason for the slowness is two-fold; first, the computational time for each fitting step is proportional to the number of free parameters (if the number of free parameters is large). The second reason is unavoidable due to our spectral fitting algorithm: our splines are defined in $\log$ photon spectrum space; if you start for example with the same value for each y_i, the fitting algorithm will start to vary each parameter in turn; if it changes for example parameter x_j by 1, this means a

factor of 10; since the neighbouring points (like x_{j-1} and x_{j+1} however are not adjusted in third step, the photon spectrum has to be drawn as a cubic spline through this very sharp function, and it will show the well-known over- and undershooting at the intermediate x-values between x_{j-1} and x_j and between x_j and x_{j+1}; as the data do not show this strong oscillation, χ^2 will be poor and the fitting algorithm will decide to adjust the parameter y_j only with a tiny amount; the big improvement in χ^2 would only come if *all* values of y_i were adjusted simultaneously.

The parameters of the model are:

`type` : The parameter *type* defined above; allowed values are 1–4. Default value: 1.
`n` : The number of grid points n. Should be at least 2.
`low` : Lower x-value b_1.
`upp` : Upper x-value b_2. Take care to take $b_2 > b_1$.
`x001` : First x-value, by definition equal to b_1. x-values are not allowed to vary (i.e. you may not fit them).
`x002` : Second x-value
`x003` : Third x-value
`...` : Other x-values
`x999` : last x-value, by definition equal to b_n. If $n < 999$, replace the 999 by the relevant value (for example, if $n = 237$, then the last x-value is x237).
`y001` : First y-value. This is a fittable parameter.
`y002` : Second y-value
`y003` : Third y-value
`...` : Other y-values
`y999` : last y-value. If $n < 999$, replace the 999 by the relevant value (for example, if $n = 237$, then the last y-value is 237).

4.1.35 User: User defined model

The user model provides an interface to create an additive model component calculated by a user program. The interface is rather simple and therefore very flexible. The interface uses a set of two ASCII files: `input-?-?.prm` and `output-?-?.spc`, where the '?' stands for the sector and component number of the user model component.

The `input-?-?.prm` is generated by the user model and contains the parameter values and the energy grid that SPEX uses to calculate the model. The file has a fixed structure:

- The first line contains one integer number containing the number of parameters that the model needs to calculate (current maximum is 40).
- On the second line the array of parameter values is listed (with the length indicated on the previous line.
- On the third line, an integer value is shown which indicates the number of model bins SPEX has in its model grid.
- On the following lines, the model grid points are shown in three columns. The first column contains the bin upper boundary (egb), the second column is the bin center (eg) and the third column is the full bin width (deg). The energy values are in keV.

The user program is supposed to read the input file and calculate the model spectrum for each energy bin on the grid. The spectrum array is called `sener` and has the same length as the energy grid and unit photons/s/bin. In addition, it is also possible to calculate the model for the emission weighted center of the bin. In that case the spectrum calculation is a bit more complicated, but it leads to better results. The user has the option of also calculating the weighted spectrum, which is the average photon energy $E_{\rm aver}$ minus the bin centroid $E_{\rm centroid}$ times the flux (F):

$$W = (E_{\rm aver} - E_{\rm centroid})F$$

The W values end up in the `wener` array. If you do not use weighing by average photon energy, `wener` values can be 0.

The user program should create an output ASCII file named `output-?-?.spc` with the following structure:

- On the first row, put the number of model bins. This should be the same number as written in the `input-?-?.prm` file.
- Write the `sener` and `wener` values in two columns. The first column is the `sener` value in photons/s/bin and the second column is the `wener` value.

In *How to use the SPEX user model* (page 73) some example user model programs can be found. The source code can be adapted to accommodate the user's wishes.

The equivalent of `user` for multiplicative models is *Musr: User defined multiplicative model* (page 159).

The parameters of the model are:

`exec` : User executable, for example './model.py' for a python script in the same directory where you run SPEX
`npar` : Number of free parameters in the model. Current maximum is 40.
`p01...p40` : Model parameters of the user model.

4.1.36 Vblo: rectangular velocity broadening model

This multiplicative model broadens an arbitrary additive component with a rectangular Doppler profile, characterized by the half-width v. Therefore if a delta-line at energy E is convolved with this component, its full energy width will be $2Ev/c$, and line photons get a rectangular distribution between $E - Ev/c$ and $E + Ev/c$. Of course, any line or continuum emission component can be convolved with the this broadening model.

The parameters of the model are:

`vblo` : Velocity broadening half-width v, in km/s. Default value: 3000 km/s.

4.1.37 Vgau: gaussian velocity broadening model

This multiplicative model broadens an arbitrary additive component with a Gaussian Doppler profile, characterized by the Gaussian σ. The broadening kernel is therefore proportional to $e^{-(c^2/2\sigma^2)(E-E_0)^2/E_0^2}$.

The parameters of the model are:

`sig` : Gaussian velocity broadening σ, in km/s. Default value: 1 km/s

4.1.38 Vpro: velocity profile broadening model

This multiplicative model broadens an arbitrary additive component with an arbitrarily shaped Doppler profile, characterized by the half-width v and a profile shape $f(x)$. The resulting spectrum $S_c(E)$ is calculated from the original spectrum $S(E)$ as

$$S_c(E) = \int f\big(\frac{E-E_0}{E_0}\frac{v}{c}\big) S(E_0) \mathrm{d}E_0.$$

The function $f(x)$ must correspond to a probability function, i.e. for all values of x we have

$$f(x) \geq 0$$

and furthermore

$$\int_{-\infty}^{\infty} f(x) \mathrm{d}x = 1.$$

In our implementation, we do not use $f(x)$ but instead the cumulative probability density function $F(x)$, which is related to $f(x)$ by

$$F(x) \equiv \int_{-\infty}^{x} f(y)\mathrm{d}y,$$

where obviously $F(-\infty) = 0$ and $F(\infty) = 1$. The reason for using the cumulative distribution is that this allows easier interpolation and conservation of photons in the numerical integrations.

If this component is used, you must have a file available which we call here `vprof.dat` (but any name is allowed). This is a simple ascii file, with n lines, and at each line two numbers: a value for x and the corresponding $F(x)$. The lines must be sorted in ascending order in x, and for $F(x)$ to be a proper probability distribution, it must be a non-decreasing function i.e. if $F(x_i) \leq F(x_{i+1})$ for all values of i between 1 and $n-1$. Furthermore, we demand that $F(x_1) \equiv 0$ and $F(x_n) \equiv 1$.

Note that both x and $F(x)$ are dimensionless. The parameter v serves as a scaling parameter for the total amount of broadening. Of course for a given profile there is freedom for the choice of both the x-scale as well as the value of v, as long as e.g. $x_n v$ remains constant. In practice it is better to make a logical choice. For example, for a rectangular velocity broadening (equivalent to the *vblo* broadening model) one would choose $n = 2$ with $x_1 = -1$, $x_2 = 1$, $F(x_1) = 0$ and $F(x_2) = 1$, and then let v do the scaling (this also allows you to have v as a free parameter in spectral fits). If one would instead want to describe a Gaussian profile (for which of course also the vgau model exists, *Vgau: gaussian velocity broadening model* (page 174)), one could for example approximate the profile by taking the x-scale in units of the standard deviation; an example with a resolution of 0.1 standard deviation and a cut-off approximation at 5 standard deviations would be x =-5, -4.9, -4.8, . . ., 4.8, 4.9, 5.0 with corresponding values for F given by F = 0, 0.00000048, 0.00000079, . . ., 0.99999921, 0.99999952, 1.

The parameters of the model are:

`v` : Velocity broadening parameter v, in km/s. Default value: 1 km/s.
`file` : Ascii character string, containing the actual name of the vprof.dat file

4.1.39 Warm: continuous photoionised absorption model

The *warm* model is a multi-component version of the *xabs* model. In the *warm* model, we construct a model for a continuous distribution of column density N_H as a function of ξ. It is in some sense comparable to the differential emission measure models used to model the emission from multi-temperature gas. Here we have absorption from multi-ionization gas. Depending upon the physics of the source, this may be a better approximation than just the sum of a few *xabs* components. A disadvantage of the model may be (but this also depends upon the physics of the source), that all dynamical parameters for each value of ξ are the same, like the outflow velocity and turbulent broadening. If this appears to be the case in a given source, one may of course avoid this problem by taking multiple, non-overlapping *warm* components.

The model assumes a logarithmic grid of n equidistant values of $\log\xi$, between a lower limit ξ_1 and an upper limit ξ_2. For each of the grid points ξ_i, a value of f_i can be adjusted; here f_i is given as $f_i = \mathrm{d}N_\mathrm{H}/\mathrm{d}\ln\xi$ evaluated at ξ_i, where the differential column density is assumed to be a continuous function of ξ. At each intermediate point, the value of $\mathrm{d}N_\mathrm{H}/\mathrm{d}\ln\xi$ is then determined by doing cubic spline interpolation in the $\ln f - \ln\xi$ space. In order not to consume too much computer time, the step size for numerical integration $\Delta\log\xi$ can be set. A typical, recommended value for this (the default) is 0.2.

For more information on this model, the atomic data and parameters we refer to *Different types of absorption models* (page 277).

The parameters of the model are:

`xil1` : $\log\xi_1$ of the lower limit of the ionisation parameter range in units of 10^{-9} W m. Default value: -4.
`xil2` : $\log\xi_2$ of the upper limit of the ionisation parameter range in units of 10^{-9} W m. Default value: 5.
`npol` : The number of grid points for the $\log\xi$ grid, including the end points for ξ_1. Default value: 19; lower values are generally recommended; minimum value is 2.

`dxi` : step size for numerical integration $\Delta \log \xi$. Default value: 0.2.
`f01...f19` : Values of $f_i = \mathrm{d}N_\mathrm{H}/\mathrm{d}\ln\xi$ at the grid points. Default values 10^{-6}. When npol<19, the remaining values of f_i are simply ignored.

The following parameters are common to all our absorption models:

`fcov` : The covering factor of the absorber. Default value: 1 (full covering)
`v` : Root mean square velocity σ_v
`rms` : Rms velocity σ_b of line blend components
`dv` : Velocity distance Δv between different blend components
`zv` : Average systematic velocity v of the absorber
The following parameter is the same as for the xabs-model (see there for a description):
`col` : File name for the photoionisation balance parameters

Recommended citation: Steenbrugge et al. (2005)

4.1.40 Wdem: power law differential emission measure model

This model calculates the spectrum of a power law distribution of the differential emission measure distribution. It appears to be a good empirical approximation for the spectrum in cooling cores of clusters of galaxies. It was first introduced by Kaastra et al. (2004) and is defined as follows:

$$\frac{\mathrm{d}Y}{\mathrm{d}T} = \begin{cases} 0 & \text{if} \quad T \le \beta T_\mathrm{max}; \\ cT^\alpha & \text{if} \quad \beta T_\mathrm{max} < T < T_\mathrm{max}; \\ 0 & \text{if} \quad T \ge T_\mathrm{max}. \end{cases} \tag{4.4}$$

Here Y is the emission measure $Y \equiv n_\mathrm{H} n_\mathrm{e} V$ in units of 10^{64} m^{-3}, where n_e and n_H are the electron and Hydrogen densities and V the volume of the source.

For $\alpha \to \infty$, we obtain the isothermal model, for large α a steep temperature decline is recovered while for $\alpha = 0$ the emission measure distribution is flat. Note that Peterson et al. (2003) use a similar parameterisation but then for the differential luminosity distribution). In practice, we have implemented the model (4.4) by using the integrated emission measure Y_tot instead of c for the normalisation, and instead of α its inverse $p = 1/\alpha$, so that we can test isothermality by taking $p = 0$. The emission measure distribution for the model is binned to bins with logarithmic steps of 0.10 in $\log T$, and for each bin the spectrum is evaluated at the emission measure averaged temperature and with the integrated emission measure for the relevant bin (this is needed since for large α the emission measure weighted temperature is very close to the upper temperature limit of the bin, and not to the bin centroid). Instead of using T_min directly as the lower temperature cut-off, we use a scaled cut-off β such that $T_\mathrm{min} = \beta T_\mathrm{max}$.

From the parameters of the wdem model, the emission weighted mean temperature kT_mean can be calculated de Plaa et al. (2006):

$$T_\mathrm{mean} = \frac{(1+\alpha)}{(2+\alpha)} \frac{(1-\beta^{2+\alpha})}{(1-\beta^{1+\alpha})} T_\mathrm{max}$$

Warning: Take care that $\beta < 1$. For $\beta = 1$, the model becomes isothermal, regardless the value of α. The model also becomes isothermal for p=0, regardless of the value of β.

Warning: For low resolution spectra, the α and β parameters are not entirely independent, which could lead to degeneracies in the fit.

The parameters of the model are:

`norm` : Integrated emission measure between $T_{\rm min}$ and $T_{\rm max}$
`t0` : Maximum temperature $T_{\rm max}$, in keV. Default: 1 keV.
`p` : Slope $p = 1/\alpha$. Default: 0.25 ($\alpha = 4$).
`cut` : Lower temperature cut-off β , in units of $T_{\rm max}$. Default value: 0.1.

The following parameters are the same as for the cie-model:

`ed` : Electron density in 10^{20} $\rm m^{-3}$
`it` : Ion temperature in keV
`vrms` : RMS Velocity broadening in km/s (see *Definition of the micro-turbulent velocity in SPEX* (page 293))
`ref` : Reference element
`01...30` : Abundances of H to Zn
`file` : Filename for the nonthermal electron distribution

Recommended citation: Kaastra et al. (2004).

4.1.41 Xabs: photoionised absorption model

The *xabs* model calculates the transmission of a slab of material, where all ionic column densities are linked through a photoionization model. The relevant parameter is the ionization parameter $\xi = L/nr^2$, with L the source luminosity, n the hydrogen density and r the distance from the ionizing source. The advantage of the *xabs* model over the *slab* model is that all relevant ions are taken into account, also those which would be detected only with marginal significance using the *slab* model. In some circumstances, the combined effect of many weak absorption features still can be significant. A disadvantage of the *xabs* model happens of course when the ionization balance of the source is different from the ionization balance that was used to produce the set of runs with the photo ionization code. In that case the *xabs* model may fail to give an acceptable fit, while the *slab* model may perform better.

The *xabs* model needs an ascii-file as input. The user can provide such a file (see parameter "col" in the parameter list), but there is also a default file in SPEX that is read if the user does not provide a separate file. The default is based on a run with Cloudy, using the spectral energy distribution of NGC 5548 as used in Steenbrugge et al. (2005). Such an ascii-files contains a pre-calculated list of ionic column densities versus ionisation parameter, as well as electron temperatures versus ionisation parameter (needed for the thermal line broadening). If you want to produce such a file, you can use the auxiliary program *xabsinput*, that will run Cloudy for you and make the file to be used in SPEX. See *Xabsinput* (page 187) for more details how to use that program.

For more information on this model, the atomic data and parameters we refer to *Different types of absorption models* (page 277).

The parameters of the model are:

`nh` : Hydrogen column density in 10^{28} $\rm m^{-2}$. Default value: 10^{-4} (corresponding to 10^{24} $\rm m^{-2}$, a typical value at low Galactic latitudes).
`xi` : the $^{10}\log$ of the ionisation parameter $\log \xi$ in units of 10^{-9} Wm. Default value: 1.

The following parameters are common to all our absorption models:

`fcov` : The covering factor of the absorber. Default value: 1 (full covering)

`v` : Root mean square velocity $\sigma_{\rm v}$
`rms` : Rms velocity $\sigma_{\rm b}$ of line blend components
`dv` : Velocity distance Δv between different blend components
`zv` : Average systematic velocity v of the absorber

The following parameters are the same as for the cie-model (see there for a description):

`ref` : Reference element
`01...28` : Abundances of H to Ni; only here we take H, He, C, N, O, Ne, Na, Mg, Al, Si, S, Ar, Ca, Fe, Ni.

The following parameter is unique for the xabs-model and the warm model:

`col` : File name for the photoionisation balance parameters

Recommended citation: Steenbrugge et al. (2003).

4.2 Optimizing model performance

Since the release of SPEX version 3, which contain major atomic data updates, the models that depend on this new atomic data run slower then before. Especially for cases which need a lot of model evaluations, this can be a nuisance.

In this section, we provide a number of suggestions that may help you speed the calculation up. Of course, speeding up the calculation usually means you also pay a (small) price in terms of accuracy. Therefore, we will also provide an estimate of the loss of accuracy where possible to help you to find the right balance between calculation speed and accuracy for your project.

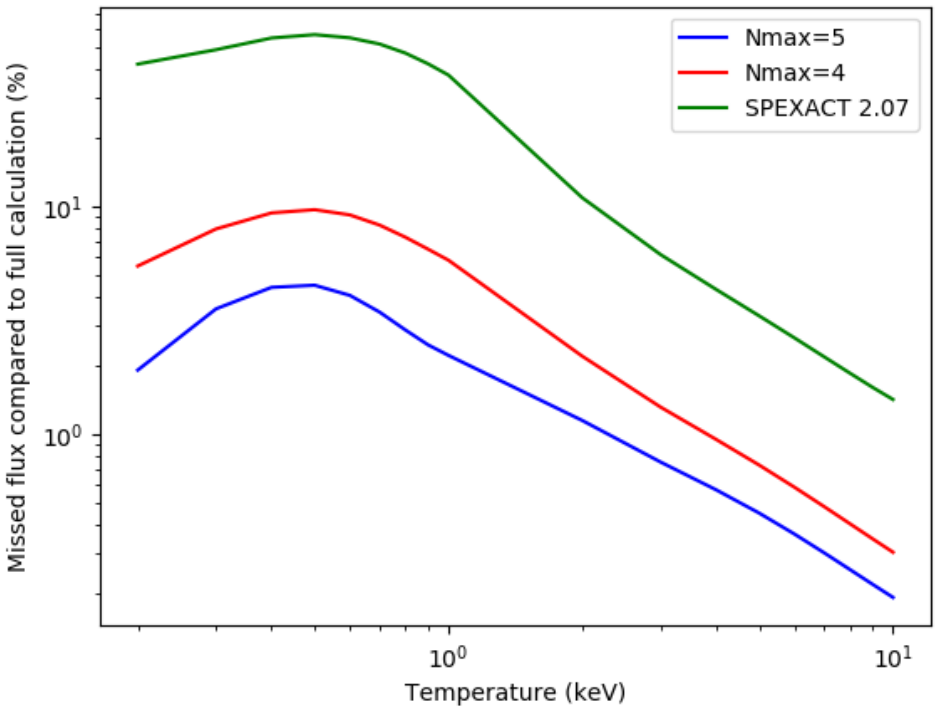
4.2.1 Setting a maximum N

In many cases, the most important spectral lines are formed by transitions between the lower principle quantum numbers *n*. In the past couple of years, we added atomic levels with a higher *n* to improve the accuracy of the models. Although it takes more time for SPEX to calculate the spectrum including these higher levels, the line flux produced by these levels is rather small. Especially above temperatures of 2 keV.

To make the calculation faster, it is a good option to skip the calculation of the higher levels. In SPEX, this can be done quite easily by setting `nmax` using the `ions` command (*Ion: select ions for the plasma models* (page 96)):

```
SPEX> ions nmax all 5
```

The command above tells SPEX to only calculate levels for which *n* <= *5*. Of course we miss out on some of the line flux by setting this parameter. We have run some tests what fraction of the flux in the 0.2-10 keV band we lose due to this setting with respect to a full model. The tests described below were done for a CIE model.



The figure above shows that the accuracy loss is highest for CIE models with temperatures below 2 keV. This is logical, because CIE models below 2 keV are dominated by line emission, rather than Brehmsstralung which dominates at higher temperatures. For *Nmax <= 4* the missing flux is 10% of the total at maximum and for *Nmax <= 5* it is 5% at maximum. For temperatures well above 1 keV, however, the error in the flux is less than 5%.

The corresponding time gain is significant, although not spectacular. On our test machine, a full CIE calculation took about 6 seconds, with *Nmax = 5* it took about 4 seconds and with *Nmax = 4* about 3 seconds were needed. The old *Mekal* calculation finishes in a fraction of a second, although at a much higher cost of accuracy.

In the Table below, we show an overview of the results. The maximum error is given as the fraction of the (line)flux that is missing in the model.

Setting	CPU Time gain	Max. Err (kT<2 keV)	Max. Err (kT>2 keV)
Full	0%	0%	0%
Nmax = 5	~35%	~5%	~1%
Nmax = 4	~50%	~10%	~3%
ACT 2	95-98%	~50%	~10%

Conclusion: If you need to reduce the processing time, then setting a maximum *n* is generally a good idea. You can even do the error calculations with the Nmax setting for most of the parameters, as long as you do the final fit with the full model to update the best-fit values. The errors should not be affected significantly within their precision, unless the parameter is strongly correlated to another one.

4.2.2 Replacing secondary components by the file model

To deal with astrophysical complexity, models can become rather large. Especially when there are many atomic model components to calculate, this can cost a significant amount of time. In some cases, it is not really necessary to calculate a component every iteration, because its parameters are rather stable or even fixed. In these cases, this component can be replaced by a spectrum read from file.

Let us consider the case where we model the Galactic foreground when analysing a cluster of galaxy spectrum. The local hot bubble and the Galactic halo are usually modeled with two CIE components with temperatures around 0.1-0.2 keV. Since these components contain many lines, they also take quite some time to calculate. Many of these calculations can be avoided if we save the best-fit foreground model in a text file.

To do this, the `sector` command (*Sector: creating, copying and deleting of a sector* (page 111)) can be of help. Once you have the best-fit model for the Galactic foreground, either from an offset pointing or an outer shell, then you can save the model with the `sector adum` command. Before you do this, you may want to save your best fit parameters with `par write`, just to be sure. Then, follow these steps:

1. Since the `sector adum` command saves the total model spectrum for the sector, please make sure that the normalisations of the models that you do not want to save are set to 0. In this particular case, the normalisation of the cluster component should be 0. Also choose whether you want to apply the absorption already in the saved spectrum or during your future fits. In the latter case, put the absorption to 0 as well.
2. If the model components to save are set, then we can execute the `sector adum` command:

```
SPEX> sector adum 1 foreground.model
```

3. Now, you can delete the model components that you saved to file from the model. For example, the cie components that model the local hot bubble and Galactic foreground. In their place, you can now add a file model:

```
SPEX> com 1 file
SPEX> par 1 4 file av foreground.model
# In this example the file model is added as component number 4.
```

4. With the file model set, you can continue fitting. Remember to restore the normalisations of the other components to their previous values. These can be found in the output file of the `par write` command that you did before. The `norm` parameter of the file model allows you to scale the spectrum. Please also check whether the absorption should be applied to the file model component or not!

The fitting should now be faster, because there are now two CIE components that do not need to be calculated anymore.

4.2.3 Optimizing the number of threads

By default, SPEX tries to use as many CPU cores as the system has available. However, dividing the calculations over multiple cores also causes overhead. The more cores, the more overhead! It has been observed that calculations performed on 6 cores were faster than using 12 cores on the same machine! If you are running on a machine with a large number of CPU cores, it may be better to limit SPEX to about 4 to 8 cores (depending on the model configuration). You can limit the number of cores by setting the `OMP_NUM_THREADS` environment variable before you start SPEX.

In bash shells, this is the command to run:

```
linux:~> export OMP_NUM_THREADS=4
```

In csh shells, it is done like this:

```
linux:~> setenv OMP_NUM_THREADS 4
```

If you need to do a large number of calculations, it may be good to test a few settings first to see which one is optimal for your model and machine.

CHAPTER

FIVE

ADDITIONAL TOOLS

5.1 Trafo

Program *trafo* allows you to transform spectra and response matrices from standard OGIP format (or from SPEX version 1 format) to the format used by SPEX. It also gives you some options to combine data sets for simultaneous spectral fitting, and to optimises the spectral data set for speed, We discuss here first the different file formats, and then explain the use of *trafo*.

Some theoretical background and technical details on the SPEX file format are given in *Optimal definition of respons matrices* (page 288) and we refer the interested reader to that page for full details. Here we summarise some of the basics.

The basic philosophy behind the SPEX file format is:

Note: Keep things as simple as possible unless there is absolutely no alternative.

Therefore, SPEX uses one file containing all the information on the spectrum, including subtracted background, that must have the extension *.spo* and has FITS format with well-defined extensions and columns. Also, there is only a single file containing all the information on the response matrix, including effective area information, that must have the extension *.res* and also has FITS format with well-defined extensions and columns.

Note: *Trafo* always produces the .spo and the .res files at the same time, as both are linked together tightly.

Examples of the usage of *trafo* can be found in *Analysis threads* (page 21) and *How to convert spectra to SPEX format* (page 9).

5.1.1 File formats spectra

There are a number of differences between OGIP and SPEX spectral file formats. See the OGIP Spectral file format specification.

Table 1: Spectral format comparison

Property	OGIP	SPEX
Files	Separate files for source & optionally background and correction file	One file for all
Channel	2-byte or 4-byte column in spectral file	Not used (should be by definition the row number of the table, starting to count at 1 without gaps)
Bin boundaries	Real, but not in spectral file but in *ebounds* extension matrix	Double precision (for high-res spectra), and in spectral file
Data	Counts (2-byte or 4-byte integer) or count rate (real, in counts s^{-1})	Only count rate (real, in counts s^{-1})
Errors	Either explicitly given, or calculated using Poissonian statistics if *poiserr* keyword is given	Always explicitly given
Exposure (s)	One value for full spectrum	Can differ for each bin
Background	Unscaled value in separate file; needs *backscal* keyword or column in source & background file to determine scaling for source region	Background subtracted value in one column; subtracted background (and its statistical error) in other columns
Quality flag	2-byte integer with 4 different values (0:math:=good, 1=bad by s/w, 2=dubious by s/w, 5=set bad by user	"no-nonsense" logical, bin is either to be used (true) or not (false)
Grouping	2-byte integer, $+1$ for start channel, -1 for continuation bin, 0 if no grouping defined	Two logicals: true/false is bin is first/last of a group (i.e., for no grouping, both are true)
Area scaling	*Areascal* keyword or column	Not allowed (can be handled with exposure time or response matrix)
Background scaling	*Backscal* keyword or column	Worked out through explicit background subtraction in the spectrum
Systematic error	Keyword or column *sys_err* in both source & background files	Two columns: one for fraction of source, one for fraction of background

Table: Differences between OGIP and SPEX in spectral file structure

In Table *Spectral format comparison* (page 182) we show the difference in structure between the OGIP spectral file format and the format used by SPEX. In addition to that, we make the following remarks.

For the energy grids, only energy (keV) units are allowed (not wavelength units), as there is a one-to-one relation between both. This number should be in double precision: for high-resolution spectra the difference between upper- and lower energy can be small, and close enough to the machine precision for single precision to cause some annoying numerical problems.

In the error column, it is essential to give real Poissonian errors (in case the spectrum is based on counts), and the use of e.g. Gehrels statistics must be avoided; in those cases, it is better to calculate the formal errors just from the square root of the number of raw counts. Chandra grating spectra are sometimes delivered with Gehrels errors, but this gives problems when the data need to be re-binned, and this is usually the case as the Chandra-data are delivered over-sampled. Also, never use 90 % errors, but always r.m.s. ("1σ") errors.

For the same reasons, the use of systematic errors should be avoided in the spectral file, because after rebinning they would become smaller. It is better to use no systematic errors in the spectral file, but in case you really need them, within SPEX you can set them after you have done the proper binning.

Also the use of quality flags should be avoided. It is better to provide the user either with only "good" data, or to make the user aware of any systematic limitations of the dats. When needed (either for instrumental or astrophysical reasons), within SPEX it is possible to ignore data ranges.

The usefulness of the *areascal* keywords is not very clear; the XMM-Newton RGS software uses the *arescal* for some dead-time corrections, but SPEX solves this in a more elegant way by allowing the exposure time to be different for each data bin. Whenever *trafo* encounters the arescal, it uses it to adjust the exposure time per bin. If you give the "show data" command in , you see for each data set some statistics, including mean, minimum and maximum exposure time per bin.

Finally, OGIP uses three columns (the background count rate, and the backscal for the source and background region) to determine the background that needs to be subtracted. In SPEX, this reduces to two columns containing essentially the same information, namely the scaled background count rate and its statistical error. Actually, what is important in this is only the ratio of the backscal columns, not their individual values.

In summary, whenever possible we recommend to use only the first seven columns (bin boundaries, exposure time, source and background count rates with their errors), and leave the other columns empty / default (first/last channel flags, used, systematic errors).

5.1.2 File formats responses

The OGIP standard is described on this page.

Table 2: Response format comparison

Property	OGIP	SPEX
Response:		
Files	Separate files for response (.rmf) & ancillary response file (.arf)	One file for all
Rmf extension:		
Components	1 component only	Matrix may have multiple components
Energy grid	One grid for the matrix, single precision	Each component can have its own grid, double precision
Response groups (≡ contiguous row of non-zero matrix elements for the same energy)	Multiple groups per row; specify number of groups (2-byte), first channel & number of channels (2-byte or 4-byte) and the matrix elements for each group	In extension "spex_resp_group" specify bin lower and upper energy, first channel, last channel and number of channels for the group; in extension "spex_resp_resp give the matrix elements Optimalisation & No & Matrix may also contain derivatives of the matrix with respect to photon energy **Ebounds extension:** & & Channel energies & Single precision & Not here, but in spectral file and double precision **Arf extension:** & & Columns & Contains lower, upper energy and area (in cm^2) & N/A (included in matrix; but note units are SI, i.e. m^2)

Table: Differences between OGIP and SPEX in response matrix structure

In Table *Response format comparison* (page 183) we show the difference in structure between the OGIP response file format and the format used by SPEX.

5.1.3 Multiple spectra

With *trafo*, you can combine different datasets into one combined spectrum and response file. There can be various reasons to do so:

1. You may want to combine different, similar instruments from the same observation (e.g. RGS1 and RGS2) or different spectral orders (e.g. RGS1 -1 and RGS1 -2 spectral order), or even combine pn with RGS. However, in these cases you may prefer to have the spectra in separate files, as that allows you easier to use only part of the data.
2. You may have time-resolved spectra of the same source taken with the same instrument
3. You may have multiple spatial regions of the same source, observed with the same instruments.

For more info, we refer to *Sectors and regions* (page 285). *Trafo* allows you to achieve this.

5.1.4 How to use *trafo*

Trafo is an interactive program. It asks a few questions, which are usually self-explanatory. However, here we give a brief overview.

1. The first question *trafo* asks if you want to transform data from OGIP format (option 1), the now abandoned SPEX version 1 binary format (option 2) or the new SPEX format (option 3).
2. The next question is how many spectra you want to transform. Usually you will enter here the number 1, but if you want to create concantinated spectra (see previous section) you should enter here the number of spectra. In that case, the next questions will be repeated for each spectrum.
3. Now you are asked to enter the maximum number of response groups per energy per spectrum. This is needed in order to allocate scratch memory; for almost any "sane" spectrum that we have seen sofar, a large number like 10000 is always sufficent here. Anyway, *trafo* will warn you and quit whenever you will reach this limit.
4. Optional: for multiple spectra, you are asked how many sectors you want to create. See the description in Sect. [sec:sectorsandregions] for more info on sectors and regions.
5. Optional: if you have more than 1 spectrum, *trafo* asks you to enter the sector and region number that you want to assign to the spectrum that you will enter next. If The sector or region number are out of the allowed range as specified before, *trafo* will keep repeating this question until you have entered valid numbers.
6. Next a question is asked about partitioning of the matrix. You have here basically three options. Option 1: keep the structure of the matrix essentially as provided by the software package that created the OGIP files. The SPEX matrix will have 1 component, with no re-arrangements. Option 2: rearrange the matrix into contiguous groups. Your matrix may have been splitted into multiple regions, and for one photon energy you might have multiple regions (for instance, higher spectral orders for grating spectrometers without energy-sensitive detectors like the LETG/HRC-S or EUVE spectrometers); or a main diagonal and a fluorescence component, etc. *Trafo* will attempt to sort your response matrix accrding to these physically distinct components, by checking if for a given energy a response group has overlap in energy with an already existing group. Option 3: split into N roughly equal-sized components. This option is recommended for large matrices of high-resolution instruments such as RGS. It allows for the optimal usage of multiple processors during spectral fitting, provided your machine has of course more than one processor.

   Note that if you combine spectra that are already in SPEX version 2.0 or higher format, you do not have this option because we assume you have already optimised your matrix when creating the original version 2.0 files.
7. Optional: if you have selected option 3 above, *trafo* ask you the number of components. This can be any number, but experience has shown that a power of 2, typically between 8 and 32 works best (even on dual core processors).
8. Now you must enter the file name of the OGIP-type spectrum. Alternatively, if you combine SPEX version 2.0 files, you are prompted for the .spo file and you do not need to answer some of the questions below.

9. If the OGIP spectral file does not provide you the name of a background spectrum, you are prompted if you want to subtract a background spectrum. Be aware that sometimes background subtraction has already been taken into account in OGIP-spectra. Check carefully. If you answer "yes" to this question, then *trafo* will ask you for the filename of the background file.

10. In a few rare cases (e.g. Einstein SSS data), there is a second background file needed, the so-called "correction file". If such a file is to be used, *tarfo* will read it but it *must* be indicated then as the corfil extension in the spectral file.

11. *Trafo* now makes a few sanity checks. If there is some problem, *trafo* will report it and stop. It checks for the same number of data channels in source and background (or correction) file. Further, and this is important to know, data bins that are being qualified as "bad" in the background or correction files, but "good" in the source file, will end up as "bad" bins in the final, background subtracted spectrum.

12. *Trafo* reports the total number of bad channels. You now can decide if you want to ignore the bad channels. Default is no (keep data; why would you otherwise have taken the burden to keep them in your datasets), but if you answer "yes", the bad channels will be ignored by *trafo* (well, flaged as not to be used in the .spo file).

13. Optional: if the OGIP-spectrum contains grouping information, *trafo* asks if that grouping should be used or ignored in the final spectrum.

14. Optional: if there is no response matrix file specified in the spectral pha-file, *trafo* asks for the name of the matrix.

15. Optional: while reading the response matrix, *trafo* makes some sanity checks. For instance, if the lower bin boundary of an energy bin is not smaller than the upper bin boundary, the user can correct this manually (some matrices are provided erroneously with zero width bins). But be sure that you understand here whjat you are doing!

16. Optional: also, some instruments assign an energy 0 to the lower energy boundary of the first bin. SPEX does not like this (as it gives trouble if you convert to a wavelength scale, for instance), so you can change the lower boundary manually to a small, non-zero number here.

17. Optional: in a few rare cases, matrices/data are poorly designed, such that the spectrum starts with a channel 0, but the matrix starts with channel 1. It is then not always clear which spectral element corresponds to which response element. *Trafo* tests for occurrences where the "flchan" keyword in the matrix equals 1, but the first channel in the data is 0. In this case it is possible to shift the response array by 1 channel, although this should be done as a last resort, and needs careefull checking if no mistakes are made! *Trafo* also tests for occurrences where the "flchan" keyword in the matrix does not equal 1 (usually 0), but the first channel in the data is 0. In this case it is advisable and possible to shift the response array by 1 channel, but again care should be taken!

18. Optional: if there is no ancillary (effective area) file specified in the spectrum, *trafo* reports this and asks if the user wants to use such an arf-file nevertheless. Default is "no", but if "yes" is entered, the name of the arf-file is prompted for.

19. As a next step, model energy bins with zero effective area are deleted from the file. Such bins usually occur at the low and high-energy side of the matrix. Deleting them saves computing time. Further, any necessary rebinning (if indicated by the grouping falgs) is done.

20. *Trafo* will tell you how many components it has found or created. It now asks you if you want to apply a shift to your spectra. Usually you should enter here 0. Useful cases to enter a non-zero shift s are situations where for instance your energy or wavelength scale is not yet sufficiently well calibrated. *trafo* then in that case puts the data of bin nr $k + s$ into bin s.

21. *Trafo* reports if the spectrum and matrix will be swapped (in case the original OGIP data were in wavelength order). Remember that SPEX *always* uses energy order.

22. The pre-last question is the filename for the spectrum that has been created (without the .spo extension, that *trafo* will add automatically).

23. Finally the filename for the response matrix that has been created is asked (without the .res extension, that *trafo* will add automatically).

5.1.5 Alternatives for trafo

Since we provide a package with Python tools for SPEX, there are a few scripts available that perform conversions from OGIP to SPEX format. ogip2spex is the program closest to the trafo program and can convert one spectrum from OGIP to SPEX format using only command line arguments.

5.2 Stepcontour

The stepcontour task produces 1D and contour plots of step searches produced using the `step` commands. At the end of the step command definitions (see *Step: Grid search for spectral fits* (page 115)), add the text `file` and a file name to the step command to write out the step results to a .stp file.

For example:

```
SPEX> step dimension 2
SPEX> step axis 1 parameter 1 1 norm range 0.5 1.5 n 10
SPEX> step axis 2 parameter 1 1 t range 0.75 1.25 n 10
SPEX> step file norm-kt
```

These commands will produce a norm-kt.stp file. Subsequently, the `stepcontour` command can be run from the command line or the SPEX> prompt:

```
linux:~> stepcontour

or

SPEX> sys exe "spexcontour"
```

The program will start and ask the user for the name of the .stp file:

```
Program stepcontour version 1.0.
Use this program to plot contours from SPEX steppar (.stp) files.

Enter filename of SPEX steppar file (.stp): norm-kt.stp
```

Then provide a PGPLOT device to show the plot. For example '/xw' for X output or 'file.ps/ps' for Postscript plots:

```
Number of axes in your steppar file:            2
Graphics device/type (? to see list, default /NULL): /xw
PGPLOT device type: XWINDOW
Plotting contours for: 68.3%, 90%, 95.4%, 99%, 99.99%.
```

If you would like to write the plot data to QDP format, say yes to the following question:

```
Would you like to write this plot to a QDP file (y/n)? y
```

The program asks for an output QDP filename and writes the data to that file.

Warning: The stepcontour program does not support logarithmic axes on 2D plots. Use another program to plot these or perform the step command on a linear grid.

5.3 Xabsinput

For the *xabs* and *warm* models, an input file is needed that contains the temperature and ionic column densities as a function of the ionisation parameter ξ. The standard input file provided by SPEX is based on a run with an older version of Cloudy, using the spectral energy distribution of NGC 5548 as used in Steenbrugge et al. (2005).

The present program *xabsinput* allows the user to create such a file. The user basically needs to provide the following information, discussed in more detail below: spectral energy distribution (SED), abundances, root directory, and how to call Cloudy.

The SED should be provided in an ascii-file with two columns. The first column gives the energy in Rydberg units (note these units!), and the second column gives the spectral energy distribution in terms of $E\,\mathrm{d}N/\mathrm{d}E$ (for instance, in units of keV $\mathrm{m}^{-2}\ \mathrm{s}^{-1}\ \mathrm{keV}^{-1}$ or Ryd $\mathrm{m}^{-2}\ \mathrm{s}^{-1}\ \mathrm{Ryd}^{-1}$, or Jansky, etc. Note that the absolute normalisation of the SED does not matter here, only the shape is used, so therefore all these units can be used as long as it corresponds to energy per area per time per unit energy.

Warning: In order to have a sufficiently broad energy band, *xabsinput* adds a value of 10^{-10} at energies of 10^{-8} and 10^{9} Ryd. Take care that the SED scaling would essentially imply 10^{-10} to be a very small value. Also, the energies in the file should be in increasing order, and within the 10^{-8} and 10^{9} Ryd interval.

Default abundances are presently the Lodders et al. (2009) proto-Solar values, presently the default of SPEX.

Warning: These default abundances are not equal to the default abundances of Cloudy.

The user is prompted for a file that contains the abundances relative to these standard abundances. This ascii files should have one line for each element with non-standard abundances. Each such line has two numbers: the atomic number of the element and the abundance relative to standard (in linear units, i.e. if you have iron two times solar, enter "26 2.0", without the quotes, of course). If all abundances are standard, simply use an empty but existing file.

The user also should provide the name of a directory where the files that are calculated will be stored. Basically, the following files will appear in this directory:

- a file "run.in", which contains the input file for Cloudy; at the end of the program, it contains the input of the last Cloudy run (with the highest ξ).
- a file "run.out", which contains the output of Cloudy; at the end of the program, it contains the output of the last Cloudy run (with the highest ξ).
- a file "xabs_inputfile", which is the input file that you will need for the *xabs* and *warm* models of ; you can rename or move that file after finishing this program. The format of that file is as follows: first part two columns ($\log\xi$ (in cgs, bah) and temperature (in eV); this is followed by blocks for each ion from H to Zn, with as first entry the nuclear charge and ionisation stage, and then simply the logs of the column densities, scaled to 28 for hydrogen.
- ascii files named "column01.dat", "column02.dat", ... "column30.dat" containing for each element with nuclear charge $Z = 1 - 30$ the ion concentrations as a function of ξ. does not need those files, but it can be educative to look at them (or to plot using qdp). The format of those files is $\log\xi$, followed by the logs of the column densities for each ion (1 is neutral, 2 is singly ionised etc). The column densities take account of the abundances and have, for practical reasons, been scaled in such a way that Hydrogen has log column 10; values below 0 are cut-off to 0.

Finally, the user should provide the name of a script that runs Cloudy. The script is given as:

```
#!/bin/csh -f
#
setenv CLOUDY_DATA_PATH /path/to/cloudy/c13.01/data
setenv LD_LIBRARY_PATH
/opt/local/HEA/cloudy/c13.01/source/cloudy.exe < ${1}
```

Finally, our runs with Cloudy are done using the following default settings (apart from the SED and abundances that are entered by the user):

- Hydrogen density small, i.e. 10^{14} m^{-3} (10^{8} cm^{-3})
- Column density small, i.e. 10^{20} m^{-2} (10^{16} cm^{-2})
- Use the "iterate to convergence" command of Cloudy
- Values of $\log \xi$ between -8.5 and $+6.5$ with steps of 0.1

Note that, depending on the computer used, this program may run for several hours. During execution it should display the present value of $\log \xi$ and the electron temperature in eV for each step of $\log \xi$ between -8.5 and $+6.5$. This number should update regularly. If it does not, then it is possible that the calculation is stuck. This usually happens if the used model cannot calculate the lowest values for ξ. If so, you may want to change the $\log \xi$ range. This can be done by providing the flag `-r` at the command line when xabsinput is started.

Warning: We note that up to and including version 13.03 of Cloudy, L_{ion} in the definition of ξ was actually the total bolometric ionising luminosity. However, from the upcoming version 13.04 of Cloudy this is corrected in the Cloudy code to be consistent with the commonly used definition, where L_{ion} ranges between 1 to 1000 Ryd. Thus, we recommend using the xabsinput program with Cloudy version 13.04.

5.4 Hydro driver

Some users run complex and computationally expensive hydrodynamical simulations, and want to be able to calculate the corresponding X-ray spectrum. Without having to implement these hydro-codes into it is now possible to do this.

For that purpose we offer the fortran90 subroutine *hydro_driver*. The source code of that routine is available in the SPEX distribution. Basically, it does the following:

1. It reads the needed parameters as subroutine arguments.
2. It does a few sanity checks (ion concentrations should be non-negative, and the total H+H column density must be 1.
3. It then creates a file called "spexicon.dat" containing the ion concentrations.
4. It also creates a file called "spexgrid.egr" containing the energy grid.
5. It then creates a file called "spexdriver.com" that contains the commands to run in batch mode.
6. This file is executed, and creates a file "spexoutput.asc" containing the emitted spectrum.
7. SPEX terminates.
8. The subroutine reads the file "spexoutput.asc" and puts the spectrum in the appropriate output array of the subroutine.

The input arguments of the subroutine are as follows:

- `hden` : The Hydrogen density in units of 10^{20} m^{-3}. Real number.
- `t` : The electron temperature in keV. Real number.
- `it` : The ion temperature in keV, used for line broadening. Real number.
- `vmic` : The microturbulence velocity, in $\mathrm{km\ s}^{-1}$, used for line broadening. Real number.
- `volume` : The emitting volume, in units of 10^{24} m^{3}. Real number.
- `con(31,30)` : Array with ion concentrations relative to hydrogen (number densities). The abundances of the metals should be taken into account in this; thus, for example, for cosmic abundances, oxygen has a number density of about 0.001 per hydrogen atom, hence the sum of all oxygen ion concentrations should then be 0.001. The array con(jz,iz) contains for each element (labeled with atomic number iz, H=1, He=2

etc.) the concentration; jz indicates the ionisation stage (1=I=neutral, 2=II=singly ionized, etc.; do not forget to include the bare nucleus as the last one. Array elements with $jz > iz + 1$ will be ignored. Note that as things are normalised to hydrogen, $con(1,1) + con(2,1) = 1$ is obligatory! Real array.

- `neg` : Number of energy bins. Integer number.
- `eg(0:neg)` : Energy bin boundaries in keV; the boundaries of the first bin are stored in eg(0) and eg(1); the second bin in eg(1) and eg(2); etc. Real array.

The output arguments of the subroutine are as follows:

- `spec(neg)` : The emitted spectrum in units of 10^{44} photons/s/keV. Divide this number by $4\pi d^2$ with d the source distance to get the flux at Earth. Real array.

5.5 Rgsvprof

In SPEX, the *lpro* component (see *Lpro: spatial broadening model* (page 158)) can be used to fold the spectrum with a user defined broadening profile. This is particularly useful for the analysis of extended sources with grating spectrometers, like RGS aboard XMM-Newton. The rgsvprof program creates an input file (usually called vprof.dat) for the *lpro* component from a MOS1 detector image.

The program will ask for the following input:

- MOS1 detector image. In order to obtain a profile along the dispersion direction of RGS within the same field of view, the program asks for a MOS1 image of the source in detector coordinates (DETX,DETY) created by, for example, the XMM-Newton SAS task *evselect*. Rgsvprof does not require a particular resolution for the image, but a resolution of about 1/3 of the XMM-Newton PSF, i.e. $4''$, is recommended. It is recommended to extract an image using events with energies falling in the RGS band: $\sim$0.3-2.0 keV.
- Cross-dispersion selection region. If the RGS spectrum is extracted from a certain area in the cross-dispersion direction, then provide the lower and upper boundary here in arcminutes with respect to the centre of the field of view. The full RGS strip corresponds to a range between -2.5 and 2.5 arcminutes.
- Source width in the dispersion direction. This parameter determines the boundaries of the resulting profile in wavelength space. Rgsvprof asks for the width of the source in arcmin centred on the peak of the emission. Choose this width to be somewhat larger than the actual width of the source to be sure that the brightest parts are included in the profile. A width of $>$10 arcmin is typically enough, but a larger width will increase the size of the vprof.dat file and increase processing times.
- Output file name. The name of the output file (usually vprof.dat). Note that rgsvprof does not overwrite files that already exist.

Rgsvprof creates an ASCII file with two columns. The left column consists of the grid of wavelengths that the profile is based upon and the right column gives the normalised cumulative profile over this range starting with 0 and ending at 1. The profile will be centred on the peak of the emission.

Note: The cross-dispersion axis in RGS is parallel to the DETX coordinate axis in MOS1 and has the same direction. This is particularly helpful when one extracts spatial regions in the cross-dispersion direction of RGS.

Warning: This program works best for bright extended sources, where the background contribution is negligible. For sources with a low surface brightness, in principle a background subtracted image should be used. This program, however, cannot deal with negative values in the MOS1 image.

Warning: This program does not take into account vignetting and chip gaps. For offset observations or very extended sources, this may start to play a role.

5.5.1 Theory

The wavelength scale corresponding to the angular width of the source in RGS can be calculated using the dispersion equation defined in e.g. Peterson et al. (2004):

$$m\lambda = d(\cos\beta - \cos\alpha),$$

where m is the spectral order, λ the wavelength, d the grating spacing, α the incidence angle of a photon, and β the diffracted angle. The wavelength shift $\Delta\lambda$ due to an offset angle $\Delta\theta$ relative to the telescope boresight can be derived by differentiating the equation above with respect to $\theta = \frac{F}{L}(\alpha - \alpha_0)$, where F is the distance between the gratings (RGA) and the camera (RFC), L the focal length of the telescope, and α_0 the incidence angle corresponding to the telescope boresight. The resulting relation for $\Delta\lambda$ is:

$$\Delta\lambda = \frac{dL}{mF}\sin(\alpha_0)\Delta\theta.$$

We then use the following numbers from den Herder et al. (2001): d is 1/645.6 mm, L is 7.5 m, F is 6.7 m, and α_0 is 1.5762° to calculate the typical conversion factor from $\Delta\theta$ in arcmin:

$$\Delta\lambda = 0.1387 \;\; \Delta\theta$$

Note that the value in Peterson et al. (2004) is not correct, because the factor $\frac{L}{F}$ was not applied. The value in the above equation is used in rgsvprof to convert the spatial profile from arcminute scales to wavelength scales.

5.6 RGS_fluxcombine

Program *RGS_fluxcombine* combines fluxed spectra, created by the XMM-Newton SAS task *rgsfluxer*. It has two options:

1. Option 1: combine RGS spectra of the same detector and spectral order.
2. Option 2: combine spectra of RGS1, RGS2 and first and second order of both (4 spectra) into one spectrum.

This is not a trivial task, due to the presence of bad pixels, CCD gaps etc, that for each individual spectrum can be slightly different, for instance because of the use of the multi-pointing mode (MPM) of the RGS, so that the same bad pixel falls at a different wavelength for each individual spectrum, or due to the transient nature of some bad pixels/columns.

We discuss both options separately. In general, if you have RGS data for multiple epochs of a source, you may have to run this program first four times with option 1 (RGS1 and 2, orders 1 and 2), and then run it ones with option 2 to get a single stacked RGS spectrum.

Warning: You must run the SAS task rgsfluxer with 3600 wavelength bins between 4–40 Å. Thus, do not use the former β-bins! Also, do not change these wavelength limits or number of bins.

The program needs an input file, basically telling it how many spectra should be combined, followed by one line for each spectrum (fixed format, in fortran (F9.2,1X,A) for option 1), containing the exposure time in s (unfortunately not contained in the output of *rgsfluxer*, but you can get it from the spectral files) and the name of the relevant output file of *rgsfluxer*. For option 2 there are two additional numbers, namely the RGS (1 or 2 for RGS1 and RGS2), and the spectral order (1 or 2 for first and second spectral order), in such a way that the format is (I1,1X,I1,1X,F11.2,1X,A).

5.6.1 Option 1: combining RGS spectra of the same detector and spectral order

It is important to realise that when data of a time-variable source are combined, artifacts due to missing data (bad pixels) will arise. This can be simply illustrated by an example. Suppose that source X is observed twice with RGS, with exactly the same exposure time, the first time with 100 counts/bin, the second time with 20 counts/bin. Suppose that at wavelength bin j the first observation has a bad pixel, hence no counts, while the second observation has no bad pixel at bin j. The averaged spectrum will have (100+20)/2=60 counts per bin, except for bin j where the average value is (0+20)/2=10. Now since both exposure times are the same, the model will predict 60 counts for all bins (as required), except for bin j where it will predict 30 counts (the pixel is dead for half of the exposure). Thus, the model has 30 counts while the data have only 10 counts. An innocent observeer might therefore think there is an absorption line at bin j taking awat 20 counts, but it is obvious that that is not true.

For this reason, when we combine spectra, we will weigh with the exposure time for all "normal" bins, but when in some of the datasets there is a bad pixel j, we will use a different procedure. In that case, we look to the neighbouring pixels, and *assume that the spectral shape in this local neighbourhoud remains the same*. From the neighbours with complete data we can estimate how much flux the spectra with good data contribute to the total flux, and we use this fraction to estimate with what factor we should scale the flux measured at pixel j in these good data sets, to obtain the best estimate of the flux at pixel j for the total spectrum.

This procedure gives reliable results as long as the spectral shape does not change locally; as there is less exposure at pixel j, only the error bar on the flux will be larger.

When there is reason to suspect that the bad pixel is at the location of a spectral line that changes in equivalent width, this procedure cannot be applied!

To alleviate this problem, the user has to enter a minimum exposure fraction. If this value is zero, for spectra with constant shape the best result is obtained. On the other hand, if this value is 1, only bins will be included that have no bad pixel in any of the observations that are combined. Advantage in that case is that there is no bias in the stacked spectrum, but a disadvantage may be that a significant part of the spectrum may be lost near important diagnostic lines (in particular for the multi-pointing mode).

Due to the binning and randomisation procedures within the SAS task *rgsfluxer*, it is still possible that despite careful screening on bad pixels, a few bad pixels remain in the fluxed spectrum, often adjacent or close to discarded pixels. For this reason, the present program checks for any bins that are more than 3σ below their right or left neighbouring bin (taking the statistical errors in both into account). Typically, the algorithm finds a few additional bad pixels in an individual observation that is discarded for combination.

The program further allows to apply two other corrections (both are done in the same step). The first correction is an effective area correction. In very high signal to noise data, there are broadband flux differences between RGS1 and RGS2 and first and second order spectra, in some parts of the spectrum, at the few percent level. A simpel model has been produced to correct for this, in the sense that when RGS1 and RGS2 are combined, the true spectrum is assumed to be the average of both. The correction is based on the monitoring campaign on Mrk 509, taken in 2009, and is found to be consistent with a large data set of Mrk 421 spectra that are present in the archive.

Local continuum fits to spectral bands that are poor of lines yield in general χ^2 values that are too low. This is an artifact of the SAS procedures related to the rebinning of the data. Data have to be binned from the detector pixel grid to the fixed wavelength grid that we use in our analysis. The bin boundaries of both grids do not match. As a consequence of this process, the natural Poissonian fluctuations on the spectrum as a function of detector pixel are distributed over the wavelength bin coinciding most with the detector pixel and its neighbours. In addition, there is a small smoothing effect caused by pointing fluctuations of the satellite. Due to this tempering of the Poissonian fluctuations, chi^2 values will be lower for a "perfect" spectral fit.

We have quantified the effect by fitting a linear model $F_\lambda = a + b\lambda$ to fluxed spectra in relatively line-poor spectral regions, in 1 Å wide bins in the 7–10, 17–18 and 35–38 Å ranges. The median reduced χ^2 is 0.72, with a 67 % confidence range between 0.65 and 0.79.

The rebinning process conserves the number of counts, hence the nominal error bars (square root of the number of counts) are properly determined. The lower reduced χ^2 is caused by the smoothing effect on the data. For correct inferences about the spectrum, such a bias in χ^2 is not appropriate. As we cannot change the observed flux values, we opt to multiply the nominal errors on the fluxes by $\sqrt{(}0.72) = 0.85$, in order to get get acceptable fits for ideal data and models.

5.6.2 Option 2: combine spectra of RGS1, RGS2 and first and second order of both (4 spectra) into one spectrum

This procedure is similar to option 1, but with some differences.

No corrections for effective area or binning bias are allowed here, as this should be taken into account in the steps that use option 1. In case there is only one observation, the present program can also be run with only one spectrum for option 1, to get the efective area and binning corrections in.

Further, the spectra are not combined according to exposure time (that should be in general almost the same for RGS1 and RGS2), but according to signal to noise ratio at each bin. Reason is that ofteh the second order spectra have poorer statistics than the first order spectra, hence they should be weighted less in the combined spectra.

Spectra created with rgsfluxcombine can only be fitted using χ^2 statistics. C-statistics will produce wrong values.

5.7 RGS_fmat

Program *RGS_fmat* produces a response matrix for a fluxed spectrum created by the XMM-Newton SAS task *rgsfluxer*.

The user should provide an FITS file that is the output of the *rgsfluxer* program. That program *must* be run with 3600 equidistant wavelength bins between 4 and 40 Å. The program takes this fluxed spectrum, and creates a response matrix with effective area 1 m^2 for each wavelength, and a redistribution function normalised to unity for each wavelength.

This redistribution function is slightly different for RGS1 and RGS2, and also different for the second order spectra. Hence, the user must provide *RGS_fmat* the information about which RGS and spectral order is used.

In addition, it is also possible to make a single response matrix for combined RGS1 and RGS2 data, first and second order. In this case, first the program *RGS_fluxcombine* must have been run with option=2. That program then creates the *rgsfluxer*-compatible file, with the addition of four extra columns, containing for each wavelength the relative weight of the different instrument/order to the total spectrum. These weights are used to weigh the redistribution functions in the combined spectra.

The redistribution function has been parametrised as the sum of three (first order) or 4 (second order) Gaussians, with wavelength-dependent width and relative normalisations. These parametrisations have been obtained by fitting the "true" redistribution function as it is in SAS in 2014 after having re-investigated the RGS line-spread-function by the RGS consortium.

The model energy grid used for the matrix has a uniform spacing of 1/9th of the instrumental FWHM, approximated here simply by $0.06/|m|$, where m is the spectral order. The matrix is fully -compatible, in the sense that it also uses the derivatives of the matrix with respect to energy to get the most accurate results. There is no way back from this to OGIP-typs spectra!

Warning: The redistribution function is limited to ± 1 Å around the central wavelength. For most continuum and absorption line spectra this is sufficient. In a few rare cases of very bright emission line sources, the user is advised to compare the resulting fit with a direct fit using the original SAS response matrices and spectral files.

The matrix is much smaller: the combined RGS matrix has a size of 8.2 Mbyte. This should be compared to the total size of the four matrices in OGIP format, which is over 200 Mbyte.

5.8 Uvtospex

Many optical/UV spectra are delivered without a response matrix. Hubble COS spectra, for example, are delivered as data/error points on a wavelength grid. To include these spectra in a SPEX fit, a response matrix with a given spectral resolution should be created. `Uvtospex` is a tool to transform such optical spectra into SPEX format.

Note: Some UV/optical instruments do provide response matrices, for example XMM-Newton OM and Swift UVOT. Please see this thread to import *Import UV/Optical data* (page 43).

To create a spectrum in SPEX format from UV/Optical grating data, the spectrum needs to be stored in a text file with three columns. The `uvtospex` program shows the format when opening:

```
Your input spectral file should have 3 columns:
Column 1: wavelength in Angstrom
Column 2: flux in ergs/cm^2/s/Angstrom
Column 3: flux error in same units

Enter input file name spectrum: cos.txt
```

The program first asks for the ascii file name containing the spectrum. Then the output file name of the spo file needs to be provided:

```
Enter file name output spectrum (with .spo): cos.spo
```

To be able to create a response matrix for the spectrum, the spectral resolution of the instrument needs to be known. Uvtospex asks for the resolution in units of km/s (FWHM):

```
Enter spectral resolution (km/s):
```

It is assumed that this resolution in km/s is constant over the entire spectral range. Finally, the name of the output response file should be entered:

```
Enter file name output response (with .res):
```

The result should be a .res and .spo file containing the UV spectrum that can be read into SPEX.

5.9 Calling SPEX from Fortran

5.9.1 Goal

Sometimes you may want to combine SPEX with some of your own code. For instance, you may have a hydrodynamical code calculating the evolution of a cluster of galaxies or a supernova remnant, and you want SPEX to calculate the corresponding spectrum. Or you need to make realistic simulations for a large sample of sources or a broad range of parameters. In that case it can be useful to call SPEX from a fortran program.

5.9.2 Solution

The basic flow of the solution is as follows:

- Start your fortran program and calculate whatever is needed
- create a command file for SPEX
- call SPEX through a system call
- read the output of SPEX
- continue with your program

To explain how you can do this, we give below a sample Fortran 90 program. The program first creates a command file called spex.com that contains the input for the run with SPEX. In your case, you can replace the "/opt/spex/bin/spex" to whatever path is appropriate for you to call SPEX. The "<<STOP" is essential for linux to tell when the input stream is finished.

In the command file, it tells how to read the pn spectrum of a cluster of galaxies with roughly metallicity 0.3, temperature 2 keV, redshift 0.01 and Galactic foreground absorption of 10^{24} m^{-2}. It ignores data at the lowest and highest energies, sets the initial model and parameters for SPEX. It will then perform a fit with SPEX. Next we tell SPEX to open an output file named "spex.out", that will contain the results of the next part of the SPEX run, an error search on the temperature. Note also that we have set-up the plotting device to "null", to speed up the calculation (but implying that you do not get to see the fit itself, so use with care). After the error run, we stop SPEX, first by writing the "quit" command for SPEX and then the end of input signal "STOP" for linux. Next we simply close the command file spex.com.

The fortran program then continues by making spex.com executable, and then by executing that file, thereby doing the run with SPEX. When SPEX is finished, fortran takes over control again and reads the spex.out file created by SPEX. The program grabs any info it needs from that file (here the errors on the temperature) and then closes the file.

As a last step, the intermediate outputfile "spex.out" is deleted, as we do not need it longer.

```
   real nh,z,t
   character*128 s

   z = 0.01    !redshift 0.01
   nh = 1e-4   !1E20 cm**-2 absorption column density (SPEX uses units of 1E28 m**-
↪2)
   t = 2       !temperature 2 keV

   open(unit=12,file='spex.com',status='replace') !open a spex command file called␣
↪spex.com
   write (12,'("/usr/local/bin/spex <<STOP")')    !call spex
   write (12,'("abu lodders")')                   !set abunances to Lodders scale
   write (s,'("dis ",f6.4," z")') z
   write (12,'(a)') trim(s)                       !write the distance
   write (12,'("data pn pn)')                     !read some spex data set: pn.res␣
↪& pn.spo
   write (12,'("ign in 1 r 1 1:60")')             !ignore first 60 channels
   write (12,'("ign in 1 12:10000 u kev")')       !ignore E>12 keV region
   write (12,'("com cie")')                       !define a CIE component
   write (12,'("com reds")')                      !define a redshift component
   write (12,'("com abs")')                       !define galactic absorption
   write (12,'("com rel 1 2,3")')                 !couple the components
   write (s,'("par 1 2 z v ",f6.4)') z
   write (12,'(a)') trim(s)                       !set the redshift
   write (s,'("par 1 3 nh v ",1pe12.3)') nh
   write (12,'(a)') trim(s)                       !set the galactic absorption
   write (12,'("par 1 3 nh s f")')                !freeze NH
   write (12,'("par 1 1 06:28 v 0.3")')           !set abundances of C to Ni to 0.
↪3 time solar
```

(continues on next page)

(continued from previous page)

```
   write (s,'("par 1 1 t v ",f5.2)') t
   write (12,'(a)') trim(s)                          !set the temperature
   write (12,'("fit meth cstat")')                   !use C-statistics for fitting
   write (12,'("plot dev null")')                    !no visible plots
   write (12,'("plot type data")')                   !plot type data
   write (12,'("fit prin 1")')                       !show each fitting step
   write (12,'("fit")')                              !do a fit
   write (12,'("err dchi 1.")')                      !for error search, do Delta␣
↪chi**2 = 1
   write (12,'("log out spex o")')                   !open a file for the SPEX output
                                                     !file will be called "spex.out"
                                                     !the option "o" means overwrite
   write (12,'("err 1 1 t")')                        !do an error search on T
   write (12,'("quit")')                             !done with the spex run, so quit
   write (12,'("STOP")')                             !last command needed to close␣
↪the command file
   close (12)                                        !close the file spex.com and save

   call system("chmod +x spex.com")                  !make file spex.com executable
   call system("spex.com")                           !execute the file spex.com

   open (unit=12,file='spex.out',status='old')       !open the outputfile spex.out
   do k = 1,100000                                   !loop over all the lines in this␣
↪file
     read (12,'(a)',end=20) s                        !read the text of the line; put␣
↪it in string s
     if (s(2:11).eq."Parameter:") then               !search for the line with text␣
↪Parameter:
       read (s(34:),*) e1,e2                         !read from this line the errors␣
↪on T
       err = 0.5 * (-e1 + e2)                        !calculate average pos and neg␣
↪error
       !..... do further all you wish
     endif
   enddo                                             !close loop over lines file spex.
↪out
20 close(12)                                         !close spex.out
   call system("rm spex.out")                        !remove the file spex.out
   end
```

CHAPTER

SIX

PYTHON INTERFACE

Welcome to the documentation of PYSPEX, the Python interface to SPEX. Since version 3.06.00 we provide a Python interface layer to the SPEX program. This documentation section explains how to use this interface, which is an alternative for the standard command-line interface of SPEX.

Warning: Since this is the first public release of pyspex, the amount of testing in the field has been limited. Be critical and expect errors. Please report issues at our Github page.

Note: As an alternative to some auxilliary programs of SPEX (e.g. trafo), we offer a separate Python package called pyspextools. Please see the Pyspextools documentation for more information.

6.1 Basic commands

6.1.1 Install PYSPEX

PYSPEX is currently available in a special branch of the SPEX source code (pyspex), but not necessarily installed by default. PYSPEX can be enabled by compiling the source again while indicating the Python major version number. The version number can be set in the cmake command prior to compilation:

```
linux:~/spex> cmake . -DPYTHON=2
```

for python version 2 series and:

```
linux:~/spex> cmake . -DPYTHON=3
```

for python version 3. CMake should automatically find the dependencies of PYSPEX, which are `numpy`, `future` and `matplotlib`. Make sure that these python modules are installed for the right python version (see below in the Section "Dependencies").

The CMake installation should create a `spexdist.sh` and `spexdist.csh` script to set the necessary environment variables. These are the `LD_LIBRARY_PATH`, which needs to be set to `$SPEX90/lib` and the `PYTHONPATH`, which is set to `$SPEX90/python`.

Dependencies

PYSPEX depends on the `numpy`, `future` and `matplotlib` python modules. The current numpy versions should include the f2py program that is necessary to create the Fortran to python interface.

Please take care that both modules are often distributed in a python 2 and python 3 version. If you compile SPEX for python 3, then also the python3 versions of the modules need to be installed. In Linux distributions, packages often have the python version in the name. In Debian, for example, the python 2 version of `numpy` is called `python-numpy` and the python 3 version `python3-numpy`.

6.1.2 Basic usage of PYSPEX

The PYSPEX python module offers a python interface to the SPEX program. This means that you can use SPEX directly either in interactive python mode and in python scripts. In this short tutorial, we assume you are using PYSPEX in interactive mode.

The structure of PYSPEX is chosen such that for many SPEX commands, there is an equivalent class in PYSPEX. These classes contain the functions that copy the functionality of the SPEX command, and if applicable, the classes contain also the output parameters of the SPEX command. This makes it easy to incorporate the SPEX results inside your own program.

In addition to the classes, you can also simply send a SPEX command-line command using the `command` function (see Section below).

Start a PYSPEX session

For this example PYSPEX session in python, we use the interactive mode of python. We simply start python from the Linux command line (making sure that the environment is set using a spexdist script located in the SPEX install directory):

```
linux:~> source spex/spexdist.sh
linux:~> python
Python 2.7.13
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If the PYTHONPATH variable is correctly set, we should now be able to import spex from pyspex:

```
>>> from pyspex import spex
```

From here, we can start a SPEX session easily:

```
>>> s=spex.Session()
 Welcome user to SPEX version 3.05.00

 NEW in this version of SPEX:
11-06-2018 Added Ext_Rate column to new spo files
18-12-2018 SPEX is now using the GPL license

Currently using SPEXACT version 2.07.00. Type `help var calc` for details.
>>>
```

The `s` object now contains all the functions and commands that are available to control the SPEX session. For example, we can send SPEX the following command-line commands:

```
>>> s.command('var calc new')
 Now using SPEXACT version 3.05.00
0
>>> s.com('cie')
 You have defined    1 component.
```

(continues on next page)

(continued from previous page)

```
0
>>> s.calc()
0
```

With these commands SPEX calculates a CIE spectrum for a temperature of 1 keV.

Although it is possible to call `s.command('quit')`, the session will be terminated automatically once python is closed or `s` is reassigned.

Using the `command` function only does not provide an advantage over the regular command line of SPEX. To really benefit from PYSPEX, you need to use the additional classes which are described in the next sections.

The command function

The PYSPEX module contains a general function to send SPEX commands to the SPEX session. This function is especially helpful for SPEX commands that have not been implemented (yet) in PYSPEX. The function returns 0 when the command was successfully executed. If not a non-zero value is returned.

`Session.`**`command`**(*comm*)

Send a command to SPEX. The command string uses the same syntax as the SPEX commands on the command line.

Parameters **comm** (*str*) – SPEX command.

6.1.3 Basic PYSPEX commands

The main PYSPEX commands that can be used in Python are designed to follow the SPEX command names. The syntax, however, deviates slightly because it needs to follow the Python syntax. This page contains the full list of the basic PYSPEX commands that are available.

This manual intents to show the basic usage of the commands intended for users already familiar with the SPEX syntax. If you need more information what the command does, please check the SPEX reference manual. For each command, the equivalent SPEX command will be mentioned for reference.

In the examples below, we assume PYSPEX has been started in python as follows:

```
>>> from pyspex import spex
>>> s = spex.Session()
 Welcome user to SPEX version 3.05.00

 NEW in this version of SPEX:
11-06-2018 Added Ext_Rate column to new spo files
18-12-2018 SPEX is now using the GPL license

Currently using SPEXACT version 2.07.00. Type `help var calc` for details.
>>>
```

Note: Note that only one SPEX session can be used within one Python run or program.

Data related commands

The commands in this section are all related to the data structures within SPEX. They deal with spectral data and responses, simulation of data, etc.

Data

Loading spectral data and responses in SPEX is done with the `data` command. Data can be added by loading a .res and .spo file that contain the response and spectral information for an instrument. The PYSPEX command is:

`Session.`**`data`**(*resfile*, *spofile*)
Load instrument data into SPEX using a .res and a .spo file.

Parameters

- **`resfile`** (*`str`*) – SPEX response file name (including .res extension).
- **`spofile`** (*`str`*) – SPEX spectrum file name (including .spo extension).

where resfile is the response file and spofile the spectrum file. Note that this command needs the full filename (and if necessary the path to the file) including the extension. This is the only difference with the SPEX syntax. Example:

```
>>> s.data('xifu.res','xifu.spo')
```

Data delete

Deleting a loaded spectrum and response is done in SPEX with the `data delete instrument #i` command. In PYSPEX, this command is:

`Session.`**`data_del`**(*ins*)
Delete an instrument from the dataset.

Parameters **`ins`** (*`int`*) – Instrument number to delete.

where `ins` is the instrument number to be deleted. For example:

```
>>> s.data_del(1)
```

Data save

Simulated spectra can be saved as a .spo file. In SPEX, this is done using the `data save #i <spofile>` command. In PYSPEX, the command is:

`Session.`**`data_save`**(*ins*, *spofile*, *overwrite=False*)
Save the (simulated) spectrum to a .spo file.

Parameters

- **`ins`** (*`int`*) – Instrument number to save.
- **`spofile`** (*`str`*) – Output filename for SPEX output file (including .spo extension).
- **`overwrite`** (*`bool`*) – (Optional) Overwrite an existing .spo file with the same name.

where `ins` is the instrument number of the data to be saved, `spofile` is the name of the output .spo file. Again note that you should give the full file name, including the .spo extension. Optionally, you can tell the command that existing files may be overwritten. For example:

```
>>> s.data_save(1,'sim.spo',overwrite=True)
```

or simply:

```
>>> s.data_save(1,'sim.spo')
```

if the file does not exist yet.

Binning and data selection

Unlike some other fitting packages, SPEX rebins the spectra within the program. The easiest binning method is to just rebin with a certain integer factor. This is the `bin` command in SPEX. In addition, SPEX contains a binning method based on the bin statistics (`vbin`) and an optimal binning algorithm (`obin`) which takes the instrument resolution into account.

Binning with factor

Binning with an integer factor is done with the bin command:

`Session.`**`bin`**(*inst*, *reg*, *elow*, *ehigh*, *factor*, *unit=None*)
Bin the spectrum using a fixed factor.

Parameters

- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **elow** (*float*) – Start point of energy/channel interval.
- **ehigh** (*float*) – End point of energy/channel interval.
- **factor** (*int*) – Binning factor
- **unit** (*str*) – Unit of the energy/channel range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'

In this example, we bin the spectrum with a factor of 5 between 0.3 and 10 keV:

```
>>> s.bin(1,1,0.3,10.,5,'kev')
```

Variable binning

Variable binning is done with the vbin command:

`Session.`**`vbin`**(*inst*, *reg*, *elow*, *ehigh*, *factor*, *snr*, *unit=None*)
Bin the spectrum using a variable bin size, given a minimum bin factor and a minimum signal to noise ratio.

Parameters

- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **elow** (*float*) – Start point of energy/channel interval.
- **ehigh** (*float*) – End point of energy/channel interval.
- **factor** (*int*) – Minimal binning factor
- **snr** (*float*) – Minimal signal to noise for a data point after binning.
- **unit** (*str*) – Unit of the energy/channel range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'

In this example, we bin the spectrum with a minimum factor of 3 and a minimum signal-to-noise ratio of 25 between 0.3 and 10 keV:

```
>>> s.vbin(1,1,0.3,10.,3,25.,'kev')
```

Optimal binning

The optimal binning algorithm bins based on the instrument resolution and statistics:

Session.**obin**(*inst*, *reg*, *elow*, *ehigh*, *unit=None*)
Bin the spectrum optimally given the instrument resolution and statistics.

Parameters

- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **elow** (*float*) – Start point of energy/channel interval.
- **ehigh** (*float*) – End point of energy/channel interval.
- **unit** (*str*) – Unit of the energy/channel range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'

In this example, we optimally bin the spectrum between 0.3 and 10 keV:

```
>>> s.obin(1,1,0.3,10.,'kev')
```

Data selection

Selecting data is done using the `use` and `ignore` commands. By default, the bin selection in the `.spo` file is loaded, which should use all the good bins.

Use function

Session.**use**(*inst*, *reg*, *elow*, *ehigh*, *unit=None*)
Use the bins given by the energy/channel range.

Parameters

- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **elow** (*float*) – Start point of energy/channel interval.
- **ehigh** (*float*) – End point of energy/channel interval.
- **unit** (*str*) – Unit of the energy/channel range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'

Example:

```
>>> s.use(1,1,0.3,10.,'kev')
```

Ignore function

Session.**ignore**(*inst*, *reg*, *elow*, *ehigh*, *unit=None*)
Ignore the bins given by the energy/channel range.

Parameters

- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **elow** (*float*) – Start point of energy/channel interval.
- **ehigh** (*float*) – End point of energy/channel interval.

- **unit** (*str*) – Unit of the energy/channel range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'

Example:

```
>>> s.ignore(1,1,0.0,0.3,'kev')
>>> s.ignore(1,1,10.,100.,'kev')
```

Systematic errors

If needed, the error bars on the source and background spectrum can be enlarged by the `syserr` command:

`Session.`**`syserr`**(*inst*, *reg*, *elow*, *ehigh*, *src*, *bkg*, *unit=None*)

Add an additional error to the source and background spectrum.

Parameters

- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **elow** (*float*) – Start point of energy/channel interval.
- **ehigh** (*float*) – End point of energy/channel interval.
- **src** (*float*) – Error value to be quadratically added to the current error bar of the source spectrum.
- **bkg** (*float*) – Error value to be quadratically added to the current error bar of the background spectrum.
- **unit** (*str*) – Unit of the energy/channel range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'

Warning: To use this function, you need to know what you are doing statistically. In many cases, this function would produce wrong results.

Simulate

Based on an instrument response and a model spectrum, SPEX can simulate an observed spectrum using the `simulate` commands. In SPEX, the options can be set by giving multiple commands, but in PYSPEX this is done in a single command with optional parameters:

`Session.`**`simulate`**(*extime*, *inst=None*, *reg=None*, *ssys=None*, *bsys=None*, *noise=None*, *bnoise=None*, *seed=None*)

Simulate a spectrum using a loaded response file.

Parameters

- **extime** (*float*) – Exposure time to simulate.
- **inst** (*str*) – (Optional) Instrument range to simulate (default all)
- **reg** (*str*) – (Optional) Region range to simulate (default all)
- **ssys** (*float*) – (Optional) Add a systematic error to the source spectrum (Default 0).
- **bsys** (*float*) – (Optional) Add a systematic error to the background spectrum (Default 0).
- **noise** (*bool*) – (Optional) Add Poisson noise to the simulated source spectrum (Default True).

- **bnoise** (*bool*) – (Optional) Add Poisson noise to the simulated background spectrum (Default False).
- **seed** (*int*) – (Optional) Set the random seed for the simulation (Default: system clock).

The only required parameter is `extime`, which is the desired exposure time (float) for the simulation.

The optional parameters are:

- `inst` The instrument number range to simulate (string). Just like in SPEX, the range can be provided by giving two numbers with a `:` in between, like `1:3`.
- `reg` The region number range to simulate (string). This has the same behaviour as `inst`.
- `ssys` Adds a systematic error to the source spectrum (float, default is 0.).
- `bsys` Adds a systematic error to the background spectrum (float, default is 0.).
- `noise` Adds Poisson noise to the simulated spectrum (default is True).
- `bnoise` Adds Poisson noise the the background spectrum (default is False).
- `seed` Provide a random seed for the simulation (by default the seed is randomly generated using the system clock).

For example:

```
>>> s.simulate(1E+5, inst='1:3', bnoise=True, seed=42)
```

Model related commands

A SPEX model is organised in sectors and components. Sectors are groups of components that model a particular source of radiation. Below, the basic commands for building the model and change its parameters are listed.

Abundance

The solar abundance table used is set in SPEX using the `abundance` command. In PYSPEX, this is abbreviated to `abun`. The desired abundance table should be loaded by providing the abbreviation for the table (`abunset`):

`Session.`**abun**(*abunset*)

Set the abundance table in SPEX. The abunset parameter is one of 'reset', 'ag', 'allen', 'asplund', 'ra', 'grevesse', 'gs', 'lodders', and 'solar'.

Parameters **abunset** (*str*) – String representing the abundance set used in SPEX.

Returns Object containing the abundance class.

Return type *pyspex.model.Abundance* (page 244)

The possible abundance tables are:

- `reset`: Lodders et al. (2009)
- `allen`: Allen (1973)
- `ra`: Ross & Aller (1976)
- `grevesse`: Grevesse et al. (1992)
- `gs`: Grevesse & Sauval (1998)
- `lodders`: Lodders proto-Solar (2003)
- `solar`: Lodders Solar Photospheric (2003)
- `ag`: Anders & Grevesse (1989)
- `asplund`: Asplund et al. (2009)

The `abun` command returns an object with the abundance information, which can optionally be assigned to a parameter and be accessed through Python. See the advanced class descriptions for details.

For example:

```
>>> a = s.abun('asplund')
>>> dir(a)
['__doc__', '__init__', '__module__', 'get', 'index', 'list', 'ref', 'set', 'update
↪']
```

Abundance show

To show the current abundance table, use the `abun_show` method:

`Session.`**`abun_show`**`()`
Show the current ionisation balance.

Returns Reference to the used ionisation balance.

Return type str

The method returns the reference as a text string:

```
>>> ab = s.abun_show()
Lodders et al. (2009)
>>> print(ab)
Lodders et al. (2009)
```

Calculate

Once the model sectors and components are set-up and the parameters are set, the model spectrum can be calculated using the SPEX `calculate` command. For convenience, this command has been abbreviated to `calc` in PYSPEX.

`Session.`**`calc`**`()`
Calculate the current model.

Example:

```
>>> s.calc()
```

Components

Spectral components, like power laws, thermal and absorption models are loaded using the SPEX `comp` command. Since the command is often typed simply as `com` in practice, the PYSPEX command is also `com`:

`Session.`**`com`**`(`*name*, *isect=1*`)`
Add model component.

Parameters

- **`name`** (*`str`*) – Name of the model component, for example 'reds', 'hot', 'cie', etc.
- **`isect`** (*`int`*) – Sector number to add component to (default is sector 1).

The command adds a component by default to sector 1. If the component should be added to a different sector, then please use the optional `isect` parameter to specify the target sector.

For example:

```
>>> s.com('cie')
>>> s.com('po', isect=2)
```

See the SPEX reference manual for a list of spectral components.

Component delete

Deleting a component from the model is done using the sector and component number of the component.

Session.**com_del**(*isect*, *icomp*)
Delete component from model.

Parameters

- **isect** (*int*) – Sector number of the component to delete.
- **icomp** (*int*) – Component number to delete.

For example:

```
>>> s.com_del(1,1)
```

The command above deletes the first component in sector number 1.

Component relate

The relation between the additive and multiplicative components is set with a `com rel` command in SPEX. In PYSPEX this is:

Session.**com_rel**(*isect*, *icomp*, *rel*)
Relate model components.

Parameters

- **isect** (*int*) – Sector number of the component to relate.
- **icomp** (*int*) – Component number to relate.
- **rel** (*numpy.ndarray*) – Array containing the multiplicative component numbers counted from the source to the observer.

The relations are set per component (so no ranges, unfortunately) and the related multiplicative models should be entered (in the right order) using a numpy array. For example:

```
>>> s.com('reds')
>>> s.com('hot')
>>> s.com('cie')
>>> s.com_rel(1, 3, numpy.array([1,2]))
```

Distance

To calculate fluxes and luminosities, SPEX needs an assumed distance of the source. In SPEX this is done with the `distance` command. In PYSPEX this is abbreviated to `dist` for convenience.

The distance can be set with the dist command:

Session.**dist**(*isect*, *dist*, *unit*)
Set the distance in SPEX.

Parameters

- **isect** (*int*) – Sector number.

- **dist** (*float*) – Distance value.
- **unit** (*str*) – The unit of the distance value, for example: 'm', 'au', 'ly', 'pc', 'kpc', 'mpc', 'z', 'cz'

where `isect` is the sector number, `dist` the distance (float) and `unit` the unit of the distance that is put in. The function returns an object containing the distance in all available units.

Examples:

```
>>> d = s.dist(1,0.5,'z')      # Redshift of z=0.5
>>> d = s.dist(1,2.0,'kpc')    # Distance of 2 kiloparsec
>>> dir(d)
['__doc__', '__init__', '__module__', 'age', 'au', 'cz', 'get', 'h0', 'kpc', 'ly',
↪'m', 'mpc', 'omega_l', 'omega_m', 'omega_r', 'pc', 'set', 'set_cosmo', 'z']
```

If you do not want to set the distance, but just get the current parameters, the `dist_get` command can be used:

Session.**dist_get**(*isect*)
Get the current distances in SPEX.

Parameters **isect** (*int*) – Sector number.

Like the dist command, this method returns an object with the distances in all available units.

Cosmology

Next to the distance, the cosmology used by SPEX can also be specified. In SPEX all parameters should be provided through seperate lines, but in PYSPEX this has been combined in one command:

Session.**dist_cosmo**(*h0*, *omega_m*, *omega_l*, *omega_r*)
Set the cosmology for the distance calculation.

Parameters

- **h0** (*float*) – Hubble constant (km/s/Mpc).
- **omega_m** (*float*) – Omega matter.
- **omega_l** (*float*) – Omega lambda.
- **omega_r** (*float*) – Omega R.

The commands needs values for the Hubble constant `h0` (70 km/s/Mpc), Omega Matter `omega_m` (0.3), Omega Lambda `omega_l` (0.7) and Omega R `omega_r` (0.0). For example:

```
>>> s.dist_cosmo(75,0.33,0.67,0.0)
```

(The command will write the distances 4 times to the terminal since in the background all SPEX commands are executed separately. . .)

Energy grid

The model energy grid can be manipulated with the SPEX `egrid` command. In PYSPEX, this command has been splitted into two varieties:

Session.**egrid**(*elow*, *ehigh*, *nbins*, *unit*, *log*)
Egrid command when the number of desired bins is known.

Parameters

- **elow** (*float*) – Lowest energy/wavelength for energy grid.
- **ehigh** (*float*) – Highest energy/wavelength for energy grid.
- **nbins** (*int*) – Number of bins for energy grid.

- **unit** (*str*) – Unit of the energy/wavelength range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'
- **log** (*bool*) – Make the energy grid logarithmic (True or False)

Session.**egrid_step**(*elow*, *ehigh*, *step*, *unit*, *log*)
Egrid command when the stepsize for the grid is known.

Parameters

- **elow** (*float*) – Lowest energy/wavelength for energy grid.
- **ehigh** (*float*) – Highest energy/wavelength for energy grid.
- **step** (*float*) – Step size for the energy grid.
- **unit** (*str*) – Unit of the energy/wavelength range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'
- **log** (*bool*) – Make the energy grid logarithmic (True or False)

For the first method, `egrid`, the number of spectral bins `nbins` is known, while for `egrid_step` the step size (`step`) is an input value. The lowest and highest energy of the grid needs to be provided using the `elow` and `ehigh` input values. The unit is a text string and the grid can be logarithmic if the `log` parameter is set to `True`.

Examples:

```
>>> s.egrid(0.1,10.,9990,'kev',True)
>>> s.egrid_step(0.1,10.,0.01,'kev',False)
```

Reading & saving grids

Grids can also be save and read from a text file. The two methods below save and read a `.egr` file, respectively:

Session.**egrid_save**(*savefile*)
Save energy grid to file.

Parameters **savefile** (*str*) – Filename to save the energy grid to (including .egr extension)

Session.**egrid_read**(*readfile*)
Read energy grid from file.

Parameters **readfile** (*str*) – Filename to read the energy grid from (including .egr extension)

The `savefile` or `readfile` parameter should provide the method with the filename to save or read, including the `.egr` extension! If necessary, the full path to the file can be included.

Examples:

```
>>> s.egrid_save('mygrid.egr')
>>> s.egrid_read('mygrid.egr')
```

Get & set custom grids

If the grid needs to be transfered from or to Python memory, then the `get` and `set` methods can be used:

`Session.`**`egrid_get`**`()`
Get the energy grid and return the grid as an object.

Returns An Egrid object from pyspex.

Return type *pyspex.model.Egrid* (page 245)

`Session.`**`egrid_set`**`(`*ebounds*`)`
Provide a grid to SPEX by providing a numpy array with the bin boundaries. Please note that the length of this array is the number of bins + 1!

Parameters **ebounds** (*numpy.ndarray*) – Array containing the energy boundaries of the new energy grid.

The `get` routine returns a Python object with the egrid arrays. The `set` routine requires an `ebounds` numpy array containing the energies of the bin boundaries. Note that the number of elements of this array would be of length n + 1, where n is the number of bins in the array.

Examples:

```
>>> grid = s.egrid_get()
>>> ebounds = 0.1 + 0.01 * numpy.arange(9991, dtype=float)
>>> s.egrid_set(ebounds)
```

Flux & Luminosity

For each component, the fluxes and luminosities are calculated using the set distance and energy boundaries. These energy limits for the flux and luminosity can be set using the `elim` command:

`Session.`**`elim`**`(`*elow*, *ehigh*, *unit*`)`
Set the energy limits for flux output.

Parameters

- **elow** (*float*) – Lowest energy/wavelength for energy interval.
- **ehigh** (*float*) – Highest energy/wavelength for energy interval.
- **unit** (*str*) – Unit of the energy/wavelength interval, for example: ‘kev’, ‘ev’, ‘ryd’, ‘j’, ‘hz’, ‘ang’, ‘nm’.

where `elow` is the lower boundary of the flux and `ehigh` the higher boundary. The `unit` determines the units of the input values, for example ‘kev’ for keV.

Examples:

```
>>> s.elim(13.6E-3,13.6,'kev')
```

Get flux

The fluxes and luminosities calculated in SPEX can be extracted using the `flux_get` method.

`Session.`**`flux_get`**`(`*isect*, *icomp*`)`
Get the fluxes and luminosities for component icomp in sector isect.

Parameters

- **isect** (*int*) – Sector number of the component to obtain the flux from.
- **icomp** (*int*) – Component number to obtain the flux from.

Returns A Flux object from pyspex.

Return type pyspex.model.Flux

The values are returned in a python object so that they can be accessed easily:

```
>>> flx = s.flux_get(1,1)
>>> print flx.enerflux
1.51011622912e-18
```

For the details about the contents of the object, see the advanced class description of the Fluxes class.

Ionisation balance

There are several ionisation balances available in SPEX. The Urdampilleta ionisation balance is the current default set.

The ionisation balance can be set using the `ibal` method:

`Session.`**`ibal`**`(`*`ref`*`)`
Set the ionisation balance.

Parameters **`ref`** (*`str`*) – Abbreviation of the reference to the ionisation balance.

The `ref` is the short text string describing the paper reference for the ionisation balance:

- `ar92`: Arnaud & Raymond (1992) for Fe, Arnaud & Rothenflug (1985) for other elements.
- `ar85`: Arnaud & Rothenflug (1985).
- `oldbryans`: Old Bryans et al. data (NOT recommended).
- `bryans09`: Bryans et al. (2009).
- `u17`: Urdampilleta et al. (2017).

Examples:

```
>>> s.ibal('u17')
```

Show

To show the current ionisation balance, the `ibal_show` method can be used:

`Session.`**`ibal_show`**`()`
Show the current ionisation balance.

Returns Reference to the used ionisation balance.

Return type str

This method returns the reference of the ionisation balance as a string.

Example:

```
>>> ib = s.ibal_show()
Urdampilleta et al. (2017)
```

Ion selection

In original SPEX models that use the SPEX atomic data, ions can be turned on or off, or can be calculated using the old SPEX version 2 or the new SPEX version 3. In addition, the maximum principle quantum number (nmax) and the maximum angular momentum (lmax) can be set.

The functions have been created such that each function selects the ions either by atomic number, iso-electronic sequence or ion.

`Session.`**`ions_all`**(*par*, *value*)
Set parameter for all values.

Parameters

- **par** (*str*) – Parameter name to set, e.g.: 'use', 'new', 'nmax', 'lmax'.
- **value** (*bool or int*) – Value to set.

`Session.`**`ions_iso`**(*iso*, *par*, *value*)
Set parameter for an iso-electronic sequence.

Parameters

- **iso** (*int*) – Iso-electronic sequence number
- **par** (*str*) – Parameter name to set, e.g.: 'use', 'new', 'nmax', 'lmax'.
- **value** (*bool or int*) – Value to set.

`Session.`**`ions_z`**(*z*, *par*, *value*)
Set parameter for an atom.

Parameters

- **z** (*int*) – Atomic number of the element.
- **par** (*str*) – Parameter name to set, e.g.: 'use', 'new', 'nmax', 'lmax'.
- **value** (*bool or int*) – Value to set.

`Session.`**`ions_ion`**(*z*, *i*, *par*, *value*)
Set parameter for a specific ion.

Parameters

- **z** (*int*) – Atomic number of the element.
- **i** (*int*) – Ionisation stage (in astrophysical notation, e.g. 1, 2, 3, etc.)
- **par** (*str*) – Parameter name to set, e.g.: 'use', 'new', 'nmax', 'lmax'.
- **value** (*bool or int*) – Value to set.

Show

The ion selections can be shown by calling the `ions_show` function below:

`Session.`**`ions_show`**()
Show the ions in use and with which parameters.

Line selection

Up to 10 specific lines can be 'muted' using the `ions_line` function below:

`Session.`**`ions_line`**(*z*, *i*, *lid*, *add*)
Mutes or unmutes a spectral line from an emission spectrum.

Parameters

- **z** (*int*) – Atomic number of the element.
- **i** (*int*) – Ionisation stage (in astrophysical notation, e.g. 1, 2, 3, etc.)
- **lid** – Line ID number (see ascdump_line for the line ID number).
- **add** – If False, the line is not added to the spectrum. True will add the line to the spectrum again.

Setting parameters

Model parameters in SPEX are set using the `par` command. Since this command has subcommands, there are a number of methods to cover most of the functionality in PYSPEX. The most basic function is to set a parameter value and determine whether it should be free in the fit or thawn. These functions have been combined into one:

`Session.`**`par`**(*isect*, *icomp*, *name*, *value*, *thawn=False*)
Set parameter value and status.

Parameters

- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **name** (*str*) – Parameter name.
- **value** (*float*) – New value of the parameter.
- **thawn** (*bool*) – (Optional) Should the parameter be free (True) or frozen (False).

`Session.`**`par_text`**(*isect*, *icomp*, *name*, *value*)
Set text type parameter value.

Parameters

- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **name** (*str*) – Parameter name.
- **value** (*str*) – New value of the parameter.

The `par` method is used for setting numerical values. It needs the sector number (isect), component number (icomp) and the name of the parameter (name) to set. Optionally, the parameter can be set free by setting `thawn` to True.

For text values, like filenames of model input files, the `par_text` method is used. The usage is very similar to the `par` method, but just with the difference a text string is passed instead of a value. Text parameters cannot be free parameters as well.

Examples:

```
>>> s.par(1, 1, 'norm', 1E+8, thawn=True)
>>> s.par_text(1, 1, file, 'dist.dat')
```

Fix & Free parameters

Many times, we want to fix and free parameters without changing the values. For this purpose, two convenience functions have been created:

Session.**par_fix**(*isect*, *icomp*, *name*)
Fix a model parameter during the fit.

Parameters

- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **name** (*str*) – Parameter name.

Session.**par_free**(*isect*, *icomp*, *name*)
Free a model parameter during the fit.

Parameters

- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **name** (*str*) – Parameter name.

`par_fix` and `par_free` fix and free the parameter with name (`name`) in sector (`isect`) and component (`icomp`).

Examples:

```
>>> s.par_free(1,1,'26')
>>> s.par_fix(1,1,'t')
```

Parameter range

Parameters have ranges in which they can be safely varied without causing undesired errors or unphysical results. These ranges can be set using the `par_range` method:

Session.**par_range**(*isect*, *icomp*, *name*, *rlow*, *rupp*)
Set parameter range.

Parameters

- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **name** (*str*) – Parameter name.
- **rlow** (*float*) – Lower limit of the parameter range.
- **rupp** (*float*) – Upper limit of the parameter range.

In addition to the sector number (`isect`), component number (`icomp`), and the parameter name (`name`), this function needs the lower (`rlow`) and upper range (`rupp`) limits of the parameter.

Example:

```
>>> s.par_range(1,1,'t',0.1,10.)
```

Couple parameters

Parameters can be coupled to each other such they have the same values in the fit. Or, optionally, remain coupled with a given multiplication factor. The PYSPEX method for this is `par_couple`:

`Session.`**`par_couple`**(*isect*, *icomp*, *iname*, *csect*, *ccomp*, *cname*, *factor*)
Couple one parameter to another with a given multiplication factor.

Parameters

- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **iname** (*str*) – Parameter name.
- **csect** (*int*) – Sector number of the parameter to couple to.
- **ccomp** (*int*) – Component number of the parameter to couple to.
- **cname** (*str*) – Parameter name of the parameter to couple to.
- **factor** (*float*) – Multiplication factor (value(name) = factor * value(cname)).

The parameter located in `isect`, `icomp` and with `iname` will be coupled to the parameter in `csect`, `ccomp`, and `cname`. The `factor` sets the multiplication factor for the coupling.

To decouple a parameter again, simply use:

`Session.`**`par_decouple`**(*isect*, *icomp*, *name*)
Decouple a parameter.

Parameters

- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **name** (*str*) – Parameter name.

Examples:

```
>>> s.par_couple(1, 2, 't', 1, 1, 't', 0.5)  # Couple the temperature in component␣
↪2 to 0.5 times the temperature in component 1
>>> s.par_decouple(1, 2, 't')
```

Set instrument normalisation

With SPEX the instrument normalisations can be set with the par command, but then with negative sector numbers. Since that can be confusing, there is a separate command to set the instrument normalisation, which has a similar syntax as the `par` method:

`Session.`**`par_norm`**(*ins*, *reg*, *value*, *thawn*)
Set an instrument normalisation.

Parameters

- **ins** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **value** (*float*) – New value of the instrument normalisation.
- **thawn** (*bool*) – (Optional) Should the parameter be free (True) or frozen (False).

This methods sets the instrument normalisation to `value` for the instrument with number `ins` and region number `reg`. The `status` parameter is a logical/boolean where `True` means thawn or free and `False` frozen.

Example:

```
>>> s.par_norm(1,2, 0.95, True)
```

Show parameters

Show the model parameters in the terminal (or Jupyter Notebook). One can specify a couple of options to show more or less information:

Session.**par_show**(*option=''*)
Display parameter overview in terminal. Options include:

- all: Show all information.
- free: Show only free parameters.
- couple: Show coupled parameters.
- flux: Show summary of the fluxes of each emission component.

Parameters **option** (*str*) – Select which information should be shown (free/couple/flux/all)

Example:

```
>>> s.par_show('free')
```

The models can also be shown through the Fortran backend, but then the output will be shown in the terminal only (not in the Jupyter notebook).

Session.**par_show_classic**(*option=''*)
Display parameter overview in terminal through the Fortran backend.

Parameters **option** (*str*) – Select which information should be shown (free/couple/flux/stat/corr/all)

Example:

```
>>> s.par_show_classic('flux')
```

See also the SPEX documentation for par_show.

Write parameters to .com file

The current parameter settings can be saved to a command file (.com) and be loaded later by the `log_exe` command. The `par_write` method in pyspex is called like this:

Session.**par_write**(*comfile*, *overwrite=False*)
Write parameters to a .com file.

Parameters

- **comfile** (*str*) – Output file name (with .com extension!)
- **overwrite** (*bool*) – (Optional) Overwrite existing files if necessary (True/False).

Example:

```
>>> s.par_write('myparam.com', overwrite=True)
```

Sectors

Sectors group spectral components to form the model for a particular source or phenomenon. If the sectors need a different response, the sectors should also be defined in the .spo and .res files. When starting SPEX, the number of sectors is 1 by default, even if loaded data files contain more sectors. Sectors can be added to SPEX with the `sector` command.

In PYSPEX a new sector is created easily with the `sector` method:

`Session.`**`sector`**`()`
Create a new sector.

This creates an empty sector. Sometimes, the new sector needs to have the same components as a previous one. In this case, the sector can be copied:

`Session.`**`sector_copy`**`(`*isect*`)`
Copy sector with number isect.

Parameters **`isect`** (*int*) – Sector number.

If a sector is no longer needed, it can be deleted:

`Session.`**`sector_del`**`(`*isect*`)`
Delete sector with number isect.

Parameters **`isect`** (*int*) – Sector number.

Examples:

```
>>> s.sector()
There are 2 sectors
>>> s.sector_copy(1)
 You have defined    1 component.
There are 3 sectors
>>> s.sector_del(2)
```

Plasma model parameters

There are a number of settings for the SPEX plasma models that can be changed by the user. In SPEX these are done using the `var` command. The `var` commands have been implemented in pyspex through the methods below. The current settings can be obtained using the *Var class* (page 249) which is referenced in the SPEX session as `s.mod_var`.

Free-bound accuracy

`Session.`**`var_gacc`**`(`*value*`)`
Set the free-bound emission accuracy.

Parameters **`value`** (*float*) – Free-bound emission accuracy.

Example:

```
>>> s.var_gacc(1.0E-2)
```

Line emission contributions

Session.**var_line**(*ltype*, *status*)
Set a line emission contribution on or off.

Parameters

- **ltype** (*str*) – Line emission contribution ('ex', 'px', 'rr', 'dr', 'ds', 'ii', 'reset')
- **status** (*bool*) – Boolean indicator whether contribution is on (True) or off (False).

Example:

```
>>> s.var_line('ex', False)
```

Doppler broadening

Session.**var_doppler**(*value*)
Doppler broadening.

Parameters **value** (*int*) – Doppler broadening type.

Example:

```
>>> s.var_doppler(1)
```

SPEXACT version 3 calculations

Session.**var_calc**(*status*)
Perform SPEXACT 3 line calculations.

Parameters **status** (*bool*) – Use SPEXACT 3 (True) or SPEXACT 2 (False)

Example:

```
>>> s.var_calc(True)
```

Occupation numbers starting values

Session.**var_occstart**(*otype*)
At which occupation level to start.

Parameters **otype** (*str*) – Occupation level ('ground', 'boltz', 'last')

Example:

```
>>> s.var_occstart('ground')
```

SPEXACT version 2 settings (MEKAL)

Session.**var_mekal**(*utype*, *status*)
Switch old Mekal updates on/off.

Parameters

- **utype** (*str*) – Update type ('wav', 'fe17', 'update', 'all')
- **status** (*bool*) – On (True) or off (False)

Examples:

```
>>> s.var_mekal('wav', False)
>>> s.var_mekal('fe17', False)
```

Multi-Maxwellians for the ionisation balance

Session.**var_ibalmaxw**(*status*)
Switch the Multi-Maxwellians for the ionisation balance on/off (True/False).

Parameters **status** (*bool*) – On (True) or off (False)

Example:

```
>>> s.var_ibalmaxw(False)
```

SPEXACT version 3 cooling

Session.**var_newcoolexc**(*status*)
Cooling by collisional excitation by Stofanova (SPEXACT 3) on/off (True/False).

Parameters **status** (*bool*) – On (True) or off (False)

Example:

```
>>> s.var_newcoolexc(False)
```

And for the cooling by di-electronic recombination:

Session.**var_newcooldr**(*status*)
Cooling by dielectronic recombination (SPEXACT 3) on/off (True/False).

Parameters **status** (*bool*) – On (True) or off (False)

Example:

```
>>> s.var_newcooldr(False)
```

Ascdump commands

The SPEX ascdump command is designed to return internal model parameters in SPEX through terminal output or an ASCII file. With pyspex we can return these numbers in a python object, making them directly available in your script. The outputs below have currently been implemented. Each command returns a specific object with the parameters. A link to the object structure is provided with every command (see return type).

Basic plasma parameters

Session.**ascdump_plas**(*isect*, *icomp*)
Basic plasma properties like temperature, electron density etc.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump plasma object

Return type *pyspex.ascdump.Plas* (page 256)

Example:

```
>>> s.com('cie')
>>> asc = s.ascdump_plas(1, 1)
 number of layer lines :         4156           0
>>> print(asc.cs)
507.26305074069296      # Sound speed in km/s
```

Abundances

Session.**ascdump_abun**(*isect*, *icomp*)
Elemental abundances and average charge per element.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump abundance object

Return type *pyspex.ascdump.Abun* (page 256)

Example:

```
>>> s.com('cie')
>>> asc = s.ascdump_abun(1,1)
 number of layer lines :         4156           0
>>> print(asc.name)
['H' 'He' 'C' 'N' 'O' 'Ne' 'Na' 'Mg' 'Al' 'Si' 'S' 'Ar' 'Ca' 'Fe' 'Ni']
```

Note: For some versions of numpy, returning a character array leads to a ValueError.

Ion concentrations

Session.**ascdump_icon**(*isect*, *icomp*)
Ion concentrations, both with respect to Hydrogen and the relevant elemental abundance.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump ion concentrations object

Return type *pyspex.ascdump.Icon* (page 257)

Example:

```
>>> s.com('cie')
>>> asc = s.ascdump_icon(1,1)
 number of layer lines :        4156           0
>>> print(asc.conrel[0])
1.70394098831e-08
```

Note: For some versions of numpy, returning a character array leads to a ValueError.

Rates per ion

Session.**ascdump_rate**(*isect*, *icomp*)

Total ionization, recombination and charge-transfer rates specified per ion.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump rate object

Return type *pyspex.ascdump.Rate* (page 257)

Example:

```
>>> s.com('cie')
>>> asc = s.ascdump_rate(1,1)
 number of layer lines :        4156           0
>>> print(asc.irate[0])
2.12289918977e-08
```

Rates per process

Session.**ascdump_rion**(*isect*, *icomp*)

Ionization rates per atomic subshell, specified according to the different contributing processes.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump rion object

Return type *pyspex.ascdump.Rion* (page 258)

Example:

```
>>> s.com('cie')
>>> asc = s.ascdump_rion(1,1)
 number of layer lines :        4156           0
>>> print(asc.elion[0])
2.12289912301e-08
```

Level populations

Session.**ascdump_pop**(*isect*, *icomp*)

The occupation numbers as well as upwards/downwards loss and gain rates to all quantum levels included.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump pop object

Return type *pyspex.ascdump.Pop* (page 263)

Example:

```
>>> s.com('cie')
>>> s.var_calc(True)
>>> asc = s.ascdump_pop(1,1)
 number of layer lines :     3511799           0
>>> print(asc.nlev)
111262
```

Electron collision excitation and de-excitation rates

Session.**ascdump_elex**(*isect*, *icomp*)

The electron collision excitation and de-excitation rates.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump Elex object

Return type *pyspex.ascdump.Elex* (page 264)

Example:

```
>>> s.com('cie')
>>> s.var_calc(True)
>>> asc = s.ascdump_elex(1,1)
 number of layer lines :     3511799           0
>>> print(asc.exrate)
[8.35325837e-09 1.08863779e-09 1.68747116e-08 ... 4.41492557e-10
 2.50513539e-10 6.16963868e-12]
```

Proton collision excitation and de-excitation rates

Session.**ascdump_prex**(*isect*, *icomp*)

The proton collision excitation and de-excitation rates.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump Prex object

Return type *pyspex.ascdump.Prex* (page 264)

Example:

```
>>> s.com('cie')
>>> s.var_calc(True)
>>> asc = s.ascdump_prex(1,1)
 number of layer lines :     3511799           0
>>> print(asc.exrate)
[1.01738931e-10 4.81612313e-12 2.07581551e-10 1.92587482e-11
 ...
 7.49964952e-17 7.82087768e-17 1.67573062e-16 2.56251319e-17]
```

Radiative transition rates

Session.**ascdump_rad**(*isect*, *icomp*)

The radiative transition rates from each level.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump Rad object

Return type *pyspex.ascdump.Rad* (page 265)

Example:

```
>>> s.com('cie')
>>> s.var_calc(True)
>>> asc = s.ascdump_rad(1,1)
 number of layer lines :     3511799           0
```

Two-photon emission

Session.**ascdump_two**(*isect*, *icomp*)

The two-photon emission transition rates from each level.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump Two object

Return type *pyspex.ascdump.Two* (page 265)

Example:

```
>>> s.com('cie')
>>> s.var_calc(True)
>>> asc = s.ascdump_two(1,1)
 number of layer lines :     3511799           0
>>> print(asc.prob)
[8.22910023e+00 5.09000015e+01 5.26599976e+02 1.94000000e+03
 ...
 4.76030003e+09 9.88000051e+09 5.82309990e+09]
```

Recombination timescale

Session.**ascdump_time**(*isect*, *icomp*)

Recombination time scale per ion according to the Bottorf et al. (2000) definition, and relative ion concentrations. Note that the recombination time scale depends upon the hydrogen density, so do not forget to set the relevant density in the model.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump Two object

Return type *pyspex.ascdump.Time* (page 266)

Example:

```
>>> s.com('cie')
>>> s.var_calc(True)
>>> asc = s.ascdump_time(1,1)
>>> print(asc.texp)
[39107415.99632238 -2295115632800309.5 -196243868063781.56
 ...
 -155719028701.42026]
```

Recombination and inner-shell ionisation

Session.**ascdump_rec**(*isect*, *icomp*)

Outputs for each atomic level the populating contributions from radiative, dielectronic and charge exchange recombination, as well as inner-shell ionisation.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump Rec object

Return type *pyspex.ascdump.Rec* (page 266)

Example:

```
>>> s.com('cie')
>>> s.var_calc(True)
>>> asc = s.ascdump_rec(1,1)
>>> print(asc.arr)
[1.88556293e-08 3.16058547e-10 2.52684149e-09 ... 0.00000000e+00
 0.00000000e+00 0.00000000e+00]
```

Energy and wavelength grid

Session.**ascdump_grid**(*isect*, *icomp*)

The energy and wavelength grid used in the last evaluation of the spectrum.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump grid object

Return type *pyspex.ascdump.Grid* (page 258)

Example:

```
>>> s.com('cie')
>>> asc = s.ascdump_grid(1,1)
 number of layer lines :        4156           0
>>> print(asc.emean[0])
0.00100070331246
```

Continuum and line spectra

Session.**ascdump_clin**(*isect*, *icomp*)

The continuum, line and total spectrum for each energy bin for the last plasma layer of the model.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump clin object

Return type *pyspex.ascdump.Clin* (page 259)

Example:

```
>>> s.com('cie')
>>> asc = s.ascdump_clin(1,1)
 number of layer lines :        4156           0
>>> dir(asc)
['__doc__', '__init__', '__module__', 'emean', 'flux', 'fluxcon', 'fluxlin', 'get',
↪ 'ibin', 'nbin']
```

And the total for all the plasma layers:

Session.**ascdump_tcl**(*isect*, *icomp*)

The continuum, line and total spectrum for each energy bin added for all plasma layers of the model.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump tcl object

Return type *pyspex.ascdump.Tcl* (page 262)

Example:

```
>>> s.com('cie')
>>> asc = s.ascdump_tcl(1,1)
 number of layer lines :        4156           0
>>> dir(asc)
['__doc__', '__init__', '__module__', 'emean', 'flux', 'fluxcon', 'fluxlin', 'get',
↪ 'ibin', 'nbin']
```

Line emission

Session.**ascdump_line**(*isect*, *icomp*, *sort='ener'*, *erange=0.1, 10.0*, *fluxlim=1e+35*)

The line energy and wavelength, as well as the total line emission (photons/s) for each line contributing to the spectrum, for the last plasma layer of the model. Also given is the natural line width and the Doppler broadening (including thermal and turbulent broadening), expressed as a FWHM in keV.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.
- **sort** (*str*) – Column to sort.
- **erange** (*Tuple*) – Energy range for output.
- **fluxlim** (*float*) – Minimum line emissivity to show (in photons/s).

Returns Ascdump Line object

Return type *pyspex.ascdump.Line* (page 259)

Example:

```
>>> s.com('cie')
>>> asc = s.ascdump_line(1,1)
>>> asc = s.ascdump_line(1,1)
 number of layer lines :     3511799           0
-------------------------------------------------------------------
The following selection criteria were used to generate this output:
Energy range between   0.100000     and    10.0000     keV.
Minimum flux level:   1.000000E+35
See the ascdump set command to change the criteria.
>>> print(asc.flux)
[3.40979220e+35 1.99786227e+35 1.76156817e+37 ... 1.33953384e+35
 1.85409623e+35 1.08065805e+35]
```

Ion contribution to the continuum

Session.**ascdump_con**(*isect*, *icomp*)

List of the ions that contribute to the free-free, free-bound and two-photon continuum emission, followed by the free-free, free-bound, two-photon and total continuum spectrum for the last plasma layer of the model.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump con object

Return type *pyspex.ascdump.Con* (page 261)

Example:

```
>>> s.com('cie')
>>> asc = s.ascdump_con(1,1)
```

The total contribution for all plasma layers:

Session.**ascdump_tcon**(*isect*, *icomp*)

List of the ions that contribute to the free-free, free-bound and two-photon continuum emission,

followed by the free-free, free-bound, two-photon and total continuum spectrum added for all plasma layers of the model.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump con object

Return type *pyspex.ascdump.Tcon* (page 262)

Example:

```
>>> s.com('cie')
>>> asc = s.ascdump_tcon(1,1)
```

History of ionisation and temperature in NEI

Session.**ascdump_nei**(*isect*, *icomp*)

The history of ionisation parameter and temperature in NEI calculations.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump nei object

Return type *pyspex.ascdump.Nei* (page 267)

Example:

```
>>> s.com('nei')
>>> asc = s.ascdump_nei(1,1)
 number of layer lines :        4269           0
>>> dir(asc)
['__doc__', '__init__', '__module__', 'get', 'kt', 'nbin', 'u']
```

Plasma heating rates (photo-ionized)

Session.**ascdump_heat**(*isect*, *icomp*)

Plasma heating rates (only for photoionized plasmas).

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump heat object

Return type *pyspex.ascdump.Heat* (page 267)

Example:

```
>>> s.com('po')
 You have defined    1 component.
>>> s.com('pion')
 You have defined    2 components.
 ** Pion model: take care about proper COM REL use: check manual!
>>> s.com_rel(1, 1, numpy.array([2]))
>>> asc = s.ascdump_heat(1,2)
```

(continues on next page)

(continued from previous page)

```
>>> dir(asc)
['__doc__', '__init__', '__module__', 'cool', 'cooladi', 'coolcom', 'cooldr',
↪'coolexc', 'coolffe', 'coolion', 'coolrec', 'get', 'heat', 'heataug', 'heatcio',
↪'heatcom', 'heatdex', 'heatext', 'heatffa', 'heatphi']
```

Energy balance

Session.**ascdump_ebal**(*isect*, *icomp*)

Obtain the cooling and heating balance from the model for each model iteration (photoionization models only).

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump ebal object

Return type *pyspex.ascdump.Ebal* (page 260)

Example:

```
>>> s.com('po')
 You have defined    1 component.
>>> s.com('pion')
 You have defined    2 components.
 ** Pion model: take care about proper COM REL use: check manual!
>>> s.com_rel(1, 1, numpy.array([2]))
>>> asc = s.ascdump_ebal(1,2)
>>> print(asc.cooladi)
[5.51856103e-44 9.20874021e-44 1.53654277e-43 2.56369787e-43
 ...
 2.17232976e-42 2.17207890e-42]
```

Ionic column densities in absorption

Session.**ascdump_col**(*isect*, *icomp*)

The ionic column densities for the hot, pion, slab, xabs and warm models.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump col object

Return type *pyspex.ascdump.Col* (page 268)

Example:

```
>>> asc = s.ascdump_col(1,2)
>>> dir(asc)
['__doc__', '__init__', '__module__', 'atom', 'column', 'get', 'ion', 'logcol',
↪'name', 'nline', 'roman']
```

Transmission of absorption lines and edges

For both lines and edges:

Session.**ascdump_tran**(*isect*, *icomp*, *sortn='ener'*)

In two subsequent objects, the transmission and equivalent width of absorption lines (asc.line) and absorption edges (asc.edge) are listed for the hot, pion, slab, xabs and warm models. Optionally, the lines can be sorted according to one of the output parameters. See the sortn option for details.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.
- **sortn** (*str*) – Sort the data based on column: energy (ener), wavelength (wav), ion (ion), line power (powe), natural line width (wid).

Returns Ascdump tran object

Return type *pyspex.ascdump.Tran* (page 268)

For lines only:

Session.**ascdump_tranline**(*isect*, *icomp*, *sortn='ener'*)

The transmission and equivalent width of absorption lines for the hot, pion, slab, xabs, and warm models. Optionally, the lines can be sorted according to one of the output parameters. See the sortn option for details.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.
- **sortn** (*str*) – Sort the data based on column: energy (ener), wavelength (wav), ion (ion), line power (powe), natural line width (wid).

Returns Ascdump tran object

Return type *pyspex.ascdump.Tranline* (page 268)

For edges only:

Session.**ascdump_tranedge**(*isect*, *icomp*)

The transmission and equivalent width of absorption edges for the hot, pion, slab, xabs, and warm models.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump tran object

Return type *pyspex.ascdump.Tranedge* (page 269)

Example:

```
>>> s.com('po')
>>> s.com('pion')
>>> s.com_rel(1,1,numpy.array([2]))
>>> asc = s.ascdump_tranedge(1,2)
>>> print(asc.ew)
[7.63924345e-06 2.29914061e-04 7.08947133e-04 1.54622412e-05
 ...
 4.60384717e-06 7.68042719e-06]
```

Properties of the warm model

Session.**ascdump_warm**(*isect*, *icomp*)
The column densities, effective ionization parameters and temperatures of the warm model.

Parameters

- **isect** (*int*) – Sector number.
- **icomp** (*int*) – Component number.

Returns Ascdump warm object

Return type *pyspex.ascdump.Warm* (page 269)

Example:

```
>>> asc = s.ascdump_warm(1,2)
>>> dir(asc)
['__doc__', '__init__', '__module__', 'atom', 'col', 'dndlnxi', 'get', 'ion', 'name
↪', 'nline', 'nxil', 'roman', 't', 'xi', 'xilgrid']
```

Optimization commands

Once the data have been read in and the model has been set, the model parameters can be optimized to fit the data. The methods in this section are used for the optimization and determination of the uncertainties in the fit.

Fit

The SPEX command to fit is simply `fit` and this has been implemented as `fit` in PYSPEX as well:

Session.**fit**(*niter=100*)
Fit command.

Parameters **niter** (*int*) – (Optional) Maximum number of iterations.

This method can be called without any arguments. Optionally, the number of iteration steps can be given, but this is only useful in some particular cases. When done fitting, the values of the fit statistics can be found in the object `s.opt_fit`. See the Fit() class for details.

Examples:

```
>>> s.fit()
>>> s.fit(niter=50)
```

Fit results

The resulting C-statistics/Chi-square value and degrees of freedom can be obtained with separate commands:

Session.**fit_cstat**()
Get the current C-statistics from the model.

Returns Tuple containing the C-stat value and the degrees of freedom.

Return type tuple

Session.**fit_chisq**()
Get the current Chi2-statistics from the model.

Returns Tuple containing the χ^2 value and the degrees of freedom.

Return type tuple

These commands return a tuple with the latest statistics value and the degrees of freedom.

Examples:

```
>>> cstat = s.fit_cstat()
>>> print(cstat)
(2979.7792968, 3000)
>>> chisq = s.fit_chisq()
>>> print(chisq)
(2997.3247823, 3000)
```

Fit statistics

Although C-statistics is recommended for Poisson distributed data like X-ray spectra, sometimes it may be better to switch to chi^2 statistics. This can be done using the `fit_stat` method:

`Session.`**`fit_stat`**(*stat*)
Set the fit statistics.

Parameters **`stat`** (*str*) – Fit statistics, for example: 'csta', 'chi2', 'wsta'

The statistics (`stat`) can be either `csta` (C-statistics), `chi2` (Chi-square statistics), or `wsta` (W-statistics). The W-statistics is added to SPEX for reference, but is not recommended for use in analysis.

Example:

```
>>> s.fit_stat('chi2')
```

Fit verbosity

The intermediate results from the fit iterations can be shown in the terminal (and in a pgplot window). In PYSPEX, this feature is on by default. If you want to turn it off, then call

`Session.`**`fit_print`**(*status*)
Set the fit output verbosity. Default is true.

Parameters **`status`** (*bool*) – Fit output verbosity setting.

with `False` as `status`. The intermediate steps will not be printed anymore.

Example:

```
>>> s.fit_print(False)
```

Error

The command for calculating errors on fitted parameters is `error` and has a direct equivalent in PYSPEX:

`Session.`**`error`**(*isect*, *icomp*, *name*, *dchi=None*)
Calculate the error for a parameter (name) in sector (isect) and component (icomp).

Parameters

- **`isect`** (*int*) – Sector number of the component to calculate the error for.
- **`icomp`** (*int*) – Component number to calculate the error for.
- **`name`** (*str*) – Parameter name to calculate the error for.
- **`dchi`** (*float*) – (Optional) $\Delta\chi^2$ value to optimize for (Default: 1.0, 68% errors)

Returns An Error object from pyspex.

Return type *pyspex.optimize.Error* (page 252)

The method calculates the lower and upper error value for the parameter with name `name` in sector `isect` and component `icomp`. Optionally, the target delta-c-stat value can be set. By default, `dchi` is set to 1.0, which results in 68% (1 sigma) errors.

The error command returns an object with the results of the error calculation. See the Error() class definition elsewhere in this manual.

Examples:

```
>>> err_si = s.error(1,1,'14')
>>> err_fe = s.error(1,1,'26',dchi=2.71)
>>> print(err_fe.value, err_fe.lerr, err_fe.uerr)
1.023902 -0.119324 0.109223
```

Plot commands

Next to the PGPLOT plotting system of SPEX, the python interface can also be used for plotting. In this section, we provide a list of commands for plotting spectral models, the effective area and spectral data (including residuals).

Plot spectral model

The SPEX plot type 'model' plots the model spectrum as a function of energy. In PYSPEX there is a function to obtain the same plot using matplotlib:

Session.**plot_model**(*xlog=False*, *ylog=False*, *title='SPEX Model Spectrum'*, *show=True*)

Plot the spectral model.

Parameters

- **xlog** (*bool*) – (Optional) Set the X-axis to be logarithmic.
- **ylog** (*bool*) – (Optional) Set the Y-axis to be logarithmic.
- **title** (*str*) – (Optional) Set the title of the plot.
- **show** (*bool*) – (Optional) Show the plot (True) or return the plot object (False).

Returns Plot object and optionally the matplotlib plt object.

Return type *pyspex.plot.PlotModel* (page 252), matplotlib.pyplot

This function plots the model spectrum as a function of energy (keV). The axis can be made logarithmic by setting `xlog=True` and `ylog=True` for the x and y axis, respectively.

Examples:

```
>>> s.plot_model()
>>> s.plot_model(xlog=True, ylog=True)
```

Plot data

The SPEX data plot type plots the observed spectrum, the folded model and background spectrum. In PYSPEX, this is done with one command called `plot_data`:

`Session.`**`plot_data`**(*xlog=False*, *ylog=False*, *title='SPEX'*, *show=True*)
Plot the observed spectrum with the convolved model for all instruments.

Parameters

- **`xlog`** (*bool*) – (Optional) Set the X-axis to be logarithmic.
- **`ylog`** (*bool*) – (Optional) Set the Y-axis to be logarithmic.
- **`title`** (*str*) – (Optional) Set the title of the plot.
- **`show`** (*bool*) – (Optional) Show the plot (True) or return the plot object (False).

Returns Plot object and optionally the matplotlib plt object.

Return type *pyspex.plot.PlotData* (page 253), matplotlib.pyplot

This function plots the observed spectra in black, the convolved model in red, and the background spectrum as a dashed blue line. The axis can be made logarithmic by setting `xlog=True` and `ylog=True` for the x and y axis, respectively.

Examples:

```
>>> s.plot_data()
>>> s.plot_data(xlog=True, ylog=True)
```

Plot data with residuals

It is also possible to attach a subplot with residuals to the main data plot. To do this, one uses the `plot_chi` command:

`Session.`**`plot_chi`**(*xlog=False*, *ylog=False*, *chi='dchi'*, *title='SPEX'*, *show=True*)
Plot the observed spectrum with the convolved model and their residuals for all instruments.

Parameters

- **`xlog`** (*bool*) – (Optional) Set the X-axis to be logarithmic.
- **`ylog`** (*bool*) – (Optional) Set the Y-axis to be logarithmic.
- **`chi`** (*str*) – (Optional) Type of residual, either 'dchi' or 'rel'.
- **`title`** (*str*) – (Optional) Set the title of the plot.
- **`show`** (*bool*) – (Optional) Show the plot (True) or return the plot object (False).

Returns Plot object and optionally the matplotlib plt object.

Return type *pyspex.plot.PlotData* (page 253), matplotlib.pyplot

In addition to the xlog and ylog parameter, this function can also set the residual type. Currently, `dchi` and `rel` are implemented.

Examples:

```
>>> s.plot_chi()
>>> s.plot_chi(chi='rel')
```

Plot data with model components

For models with multiple additive components, it is possible in PYSPEX to plot the total model and the contributions from the individual additive components with the `plot_comp` method.

`Session.`**`plot_comp`**(*xlog=False*, *ylog=False*, *title='SPEX'*, *show=True*)
Plot the individual model components.

Parameters

- **xlog** (*bool*) – (Optional) Set the X-axis to be logarithmic.
- **ylog** (*bool*) – (Optional) Set the Y-axis to be logarithmic.
- **title** (*str*) – (Optional) Set the title of the plot.
- **show** (*bool*) – (Optional) Show the plot (True) or return the plot object (False).

Returns Plot object and optionally the matplotlib plt object.

Return type *pyspex.plot.PlotData* (page 253), matplotlib.pyplot

Examples:

```
>>> s.plot_comp()
>>> s.plot_comp(ylog=True, title='Spectral components')
```

Plot effective area

The SPEX plot type 'area' plots the effective area as a function of energy. In PYSPEX there is a function to obtain the same plot using matplotlib:

`Session.`**`plot_area`**(*xlog=False*, *ylog=False*, *title='Effective Area'*, *show=True*)
Plot the effective area of all instruments.

Parameters

- **xlog** (*bool*) – (Optional) Set the X-axis to be logarithmic.
- **ylog** (*bool*) – (Optional) Set the Y-axis to be logarithmic.
- **title** (*str*) – (Optional) Set the title of the plot.
- **show** (*bool*) – (Optional) Show the plot (True) or return the plot object (False).

Returns Plot object and optionally the matplotlib plt object.

Return type *pyspex.plot.PlotArea* (page 255), matplotlib.pyplot

This function plots the effective area as a function of energy (keV). The axis can be made logarithmic by setting `xlog=True` and `ylog=True` for the x and y axis, respectively.

Examples:

```
>>> s.plot_area()
>>> s.plot_area(xlog=True, ylog=True)
```

Log commands

Being an interactive program, SPEX has facilities to save commands, execute lists of commands and save outputs to files. This is arranged through the `log` commands.

Log execute

Executing lists of SPEX commands can be done using the `log_exe` method, similar to the `log exe` command in SPEX:

`Session.`**`log_exe`**(*filename*)
Execute log file.

Parameters **`filename`** (*str*) – Filename of the ASCII file with the SPEX commands (including .com extension).

The `filename` should include a .com extension.

Example:

```
>>> s.log_exe('mysource.com')
```

Log save

To save the commands typed into SPEX (or passed through the `command` function in PYSPEX or the API), one can use the `log_save` command.

`Session.`**`log_save`**(*savefile*, *status=None*)
Save the commands for a session in an ascii file. The status is optional and either overwrite or append.

Parameters

- **`savefile`** (*str*) – Filename of the output command file (including .com extension).
- **`status`** (*str*) – (Optional) Overwrite or append an existing file name. String can be 'overwrite' or 'append'.

The `savefile` should have the .com extension included in the name. Optionally, the status `overwrite` or `append` can be added.

Example:

```
>>> s.log_save('mycommands.com')
>>> s.log_save('session_01.com', status='append')
```

Log output

The terminal output can be saved in another file with the .out extension using the method `log_out`:

`Session.`**`log_out`**(*outfile*, *status=None*)
Save SPEX terminal output to file.

Parameters

- **`outfile`** (*str*) – Filename of the output file (including .out extension).
- **`status`** (*str*) – (Optional) Overwrite or append an existing file name. String can be 'overwrite' or 'append'.

The `outfile` should have the .out extension included in the name. Optionally, the status `overwrite` or `append` can be added.

Examples:

```
>>> s.log_out('session.out')
>>> s.log_out('mylog.out', status='overwrite')
```

Log close

The logging done by the `log_save` or `log_out` methods can be stopped by calling the `log_close` method:

`Session.`**`log_close`**(*logtype*)
Close the save or output file.

Parameters **`logtype`** (*str*) – String indicating the type of log to close ('save' or 'out').

where `logtype` is either `save` or `out`.

Examples:

```
>>> s.log_close('out')
>>> s.log_close('save')
```

6.2 Analysis threads

6.2.1 Jupyter Notebooks

- A basic PYSPEX example

6.3 Advanced class descriptions

6.3.1 Session class structure

The PYSPEX session class is the top-level class for PYSPEX and the most important class managing user interaction. When the class is initialized, a SPEX session is started for the duration of the Python session. All subsequent commands communicate with this SPEX session in the background.

The start of a SPEX session is managed by the __init__ method. At the same time, objects are created to contain much of the internal SPEX data that pyspex can access. These objects are instances of the Data, Model, Optimization, Ascdump and Log classes that are explained later in this Chapter.

class `pyspex.spex.`**`Session`**(**args*, ***kwargs*)
This class contains all the classes and commands that are available in a pySPEX session.

`__init__`()
Function to initialize the PYSPEX session. It starts an instance of SPEX in the background. Furthermore, it initializes a set of objects containing information about the loaded data, model values, optimization and logs.

Variables

- **`version`** (*str*) – The SPEX version number
- **`dataset`** (`data.Data` (page 236)) – Class containing the data information
- **`mod_abundance`** (`model.Abundance` (page 244)) – Model class for the abundance setting
- **`mod_distance`** (`model.Distance` (page 244)) – Model class containing distance tools

- **mod_egrid** (model.Egrid (page 245)) – Model class for the definition of energy grids
- **mod_flux** (model.Fluxes (page 246)) – Model class containing flux and luminosity information
- **mod_ibal** (model.Ibal (page 247)) – Model class for the ionisation balance setting
- **mod_ions** (model.Ions (page 247)) – Model class for setting the used ions
- **mod_var** (model.Var (page 249)) – Model class for plasma model settings
- **mod_spectrum** (model.Spectrum (page 249)) – Model class containing commands to extract model results from SPEX
- **mod** (model.Model (page 240)) – Model class containing sectors, components and parameters.
- **opt_fit** (optimize.Fit (page 251)) – Initialize the Fit class for spectral fitting
- **asc** (*ascdump.Ascdump*) – Class for the Ascdump output
- **logs** (log.Log (page 270)) – Class for Log saving and execution

During the SPEX session, the object variables can be used to obtain values from the SPEX session in the background. Please note that for most of these there are existing PYSPEX commands to retrieve the information safely.

Descriptions of the classes behind the object variables in __init__ can be found by clicking the link behind the variable.

Warning: You can only start one PYSPEX session during a python session. Unfortunately, the Fortran to Python interface that we use cannot handle multiple sessions properly.

Session class commands

The Session class also contains the main PYSPEX commands listed in *Basic PYSPEX commands* (page 199).

6.3.2 Data class structure

Spectral & response data

class pyspex.data.**Data**

Main data class used to set and get observed data.

Variables

- **ninst** (*int*) – Number of instruments loaded.
- **inst** (*list of objects*) – Python list of instrument objects.

delete(*ins*)

Delete instrument with number *ins*.

Parameters **ins** (*int*) – Instrument number to delete.

load(*resfile*, *spofile*)

Add a new set of spectrum and response file to the dataset. Provide the full filename including extension!

Parameters

- **resfile** (*str*) – SPEX response file name (including .res extension).
- **spofile** (*str*) – SPEX spectrum file name (including .spo extension).

Return type int

save(*ins*, *spofile*, *overwrite=False*)
Save a spectrum to a .spo file.
Parameters
- **ins** (*int*) – Instrument number to save.
- **spofile** (*str*) – Output filename for SPEX output file (including .spo extension).
- **overwrite** (*bool*) – (Optional) Overwrite an existing .spo file with the same name.

update()
Update the number of instruments.

class pyspex.data.**Instrument**
Properties of an instrument.

Variables

- **nsector** (*int*) – Number of sectors in instrument.
- **nregion** – Number of data regions in response.
- **nreg** (*int*) – Number of regions in data.
- **ncomp** (*int*) – Number of response components.
- **index** (*int*) – Index number of this instrument.
- **sponame** (*str*) – Filename of .spo file.
- **resname** (*str*) – Filename of .res file
- **reg** (*list of objects*) – List of regions for this instrument.

update(*iins*)
Update the instrument information.
Parameters **iins** (*int*) – Instrument number to update.

class pyspex.data.**Region**
Properties of one region.

Variables

- **index** (*int*) – Region number.
- **emin** (*float*) – Min data energy range (keV).
- **emax** (*float*) – Max data energy range (keV).
- **srccount** (*float*) – Net source counts.
- **srccerr** (*float*) – Net source count error.
- **bkgcount** (*float*) – Subtracted background counts.
- **bkgcerr** (*float*) – Error subtracted background counts.
- **srcrate** (*float*) – Net source count rate (counts/s).
- **srcrerr** (*float*) – Net source count rate error (counts/s).
- **bkgrate** (*float*) – Background count rate subtracted.
- **bkgrerr** (*float*) – Error background count rate subtracted.
- **mbkgrate** (*float*) – Model background count rate.
- **tintmin** (*float*) – Minimum integration time per channel.
- **tintmax** (*float*) – Maximum integration time per channel.
- **tintaver** (*float*) – Average integration time per channel.

update(*iins*, *ireg*)
Update the region properties.
Parameters
- **iins** (*int*) – Instrument number.
- **ireg** (*int*) – Region number to update.

Spectral binning and data selection

class pyspex.data.**Bins**
Class containing the binning methods.

bin(*inst*, *reg*, *elow*, *ehigh*, *factor*, *unit=None*)
Bin the spectrum using a fixed factor.
Parameters
- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **elow** (*float*) – Start point of energy/channel interval.
- **ehigh** (*float*) – End point of energy/channel interval.
- **factor** (*int*) – Binning factor
- **unit** (*str*) – Unit of the energy/channel range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'

ignore(*inst*, *reg*, *elow*, *ehigh*, *unit=None*)
Ignore the bins given by the energy/channel range.
Parameters
- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **elow** (*float*) – Start point of energy/channel interval.
- **ehigh** (*float*) – End point of energy/channel interval.
- **unit** (*str*) – Unit of the energy/channel range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'

obin(*inst*, *reg*, *elow*, *ehigh*, *unit=None*)
Bin the spectrum optimally given the instrument resolution and statistics.
Parameters
- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **elow** (*float*) – Start point of energy/channel interval.
- **ehigh** (*float*) – End point of energy/channel interval.
- **unit** (*str*) – Unit of the energy/channel range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'

reset(*inst*, *reg*)
Reset the binning and use status to use all with the original binning.
Parameters
- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.

syserr(*inst*, *reg*, *elow*, *ehigh*, *src*, *bkg*, *unit=None*)
Add an additional error to the source and background spectrum.
Parameters
- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **elow** (*float*) – Start point of energy/channel interval.
- **ehigh** (*float*) – End point of energy/channel interval.
- **src** (*float*) – Error value to be quadratically added to the current error bar of the source spectrum.
- **bkg** (*float*) – Error value to be quadratically added to the current error bar of the background spectrum.

- **unit** (*str*) – Unit of the energy/channel range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'

use(*inst*, *reg*, *elow*, *ehigh*, *unit=None*)

Use the bins given by the energy/channel range.

Parameters

- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **elow** (*float*) – Start point of energy/channel interval.
- **ehigh** (*float*) – End point of energy/channel interval.
- **unit** (*str*) – Unit of the energy/channel range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'

vbin(*inst*, *reg*, *elow*, *ehigh*, *factor*, *snr*, *unit=None*)

Bin the spectrum using a variable bin size, given a minimum bin factor and a minimum signal to noise ratio.

Parameters

- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **elow** (*float*) – Start point of energy/channel interval.
- **ehigh** (*float*) – End point of energy/channel interval.
- **factor** (*int*) – Minimal binning factor
- **snr** (*float*) – Minimal signal to noise for a data point after binning.
- **unit** (*str*) – Unit of the energy/channel range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'

Simulate spectra

class pyspex.data.**Simulate**

Class to simulate spectra.

set_bnoise(*status*)

Add Poisson noise to the background spectrum (status is True or False).

Parameters **status** (*bool*) – Add Poisson noise to the simulated background spectrum.

set_instrument(*irange*)

Define the range of instruments to simulate.

Parameters **irange** (*str*) – Instrument range to simulate (default all)

set_noise(*status*)

Add Poisson noise to the source spectrum (status is True or False).

Parameters **status** (*bool*) – Add Poisson noise to the simulated source spectrum.

set_random()

Set the random seed to a random number (default).

set_random_seed(*seed*)

Set the random seed to an integer value.

Parameters **seed** (*int*) – Set the random seed for the simulation.

set_region(*rrange*)

Define the range of regions to simulate.

Parameters **rrange** (*str*) – Region range to simulate (default all)

set_syserr(*src*, *bkg*)

Add a systematic error to the source spectrum (src) and to the background spectrum (bkg).

Parameters

- **src** (*float*) – Add a systematic error to the source spectrum.
- **bkg** (*float*) – Add a systematic error to the background spectrum.

simulate(*extime*, *inst=None*, *reg=None*, *ssys=None*, *bsys=None*, *noise=None*, *bnoise=None*, *seed=None*)
Simulate a spectrum for exposure time extime and optionally with a number of options.

Parameters
- **extime** (*float*) – Exposure time to simulate.
- **inst** (*str*) – (Optional) Instrument range to simulate (default all)
- **reg** (*str*) – (Optional) Region range to simulate (default all)
- **ssys** (*float*) – (Optional) Add a systematic error to the source spectrum (Default 0).
- **bsys** (*float*) – (Optional) Add a systematic error to the background spectrum (Default 0).
- **noise** (*bool*) – (Optional) Add Poisson noise to the simulated source spectrum (Default True).
- **bnoise** (*bool*) – (Optional) Add Poisson noise to the simulated background spectrum (Default False).
- **seed** (*int*) – (Optional) Set the random seed for the simulation (Default: system clock).

simulate_exposure(*extime*)
Simulate the spectrum for the provided exposure time.

Parameters **extime** (*float*) – Exposure time to simulate.

6.3.3 Model class structures

Model

class pyspex.model.**Model**
Top class containing the entire model, containing all sectors and components.

Variables
- **nsector** (*int*) – Number of sectors in this model
- **sect** (*list*) – List of sector objects

comp_delete(*isect*, *icomp*)
Delete component from sector.

Parameters
- **isect** (*int*) – Sector number of the component to delete.
- **icomp** (*int*) – Component number to delete.

comp_new(*name*, *isect=1*)
Add new component to the model. By default in sector 1.

Parameters
- **name** (*str*) – Name of the model component, for example 'reds', 'hot', 'cie', etc.
- **isect** (*int*) – Sector number to add component to (default is sector 1).

comp_set_rel(*isect*, *icomp*, *rel*)
Set component relation.

Parameters
- **isect** (*int*) – Sector number of the component to relate.
- **icomp** (*int*) – Component number to relate.
- **rel** (*numpy.ndarray*) – Array containing the multiplicative component numbers counted from the source to the observer.

par_aset(*isect*, *icomp*, *name*, *value*)
Set a text type parameter value.

Parameters
- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.

- **name** (*str*) – Parameter name.
- **value** (*str*) – New value of the parameter.

par_couple(*isect*, *icomp*, *iname*, *csect*, *ccomp*, *cname*, *factor*)

Couple parameter (iname) to another parameter with cname, with a coupling factor.

Parameters

- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **iname** (*str*) – Parameter name.
- **csect** (*int*) – Sector number of the parameter to couple to.
- **ccomp** (*int*) – Component number of the parameter to couple to.
- **cname** (*str*) – Parameter name of the parameter to couple to.
- **factor** (*float*) – Multiplication factor (value(name) = factor * value(cname)).

par_decouple(*isect*, *icomp*, *iname*)

Decouple parameter.

Parameters

- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **iname** (*str*) – Parameter name.

par_fix(*isect*, *icomp*, *name*)

Fix a parameter in the fit.

Parameters

- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **name** (*str*) – Parameter name.

par_free(*isect*, *icomp*, *name*)

Free a parameter in the fit.

Parameters

- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **name** (*str*) – Parameter name.

par_get(*isect*, *icomp*)

Get the parameter object for sector isect, component icomp and parameter with name.

Parameters

- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.

Return type *model.Parameter* (page 243)

par_norm_get(*ins*, *reg*)

Get the instrument normalisation for instrument ins and region reg.

Parameters

- **ins** (*int*) – Instrument number.
- **reg** (*int*) – Region number.

par_norm_set(*ins*, *reg*, *value*, *status*)

Set the instrument normalisation for an instrument number and region number.

Parameters

- **ins** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **value** (*float*) – New value of the instrument normalisation.
- **status** (*bool*) – (Optional) Should the parameter be free (True) or frozen (False).

par_set(*isect*, *icomp*, *name*, *value*, *thawn=False*)

Set parameter to value and optionally indicate thawn or frozen status.

Parameters

- **isect** (*int*) – Sector number of the parameter.

- **icomp** (*int*) – Component number of the parameter.
- **name** (*str*) – Parameter name.
- **value** (*float*) – New value of the parameter.
- **thawn** (*bool*) – (Optional) Should the parameter be free (True) or frozen (False).

par_set_range(*isect*, *icomp*, *name*, *rlow*, *rupp*)
Set the fit range for a parameter.

Parameters
- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **name** (*str*) – Parameter name.
- **rlow** (*float*) – Lower limit of the parameter range.
- **rupp** (*float*) – Upper limit of the parameter range.

par_show(*option=''*)
Display parameter overview through the Python interface.

Parameters **option** (*str*) – Select which information should be shown (free/couple/flux/stat/corr/all)

par_show_classic(*option=''*)
Display parameter overview in terminal through the Fortran interface.

Parameters **option** (*str*) – Select which information should be shown (free/couple/flux/stat/corr/all)

par_show_couple()
Display the coupled parameters through the Python interface.

par_show_flux()
Display the flux information of the model components.

par_show_free()
Display the free parameters through the Python interface.

par_show_param()
Display the parameters through the Python interface.

par_write(*comfile*, *overwrite=False*)
Write parameters to a .com file.

Parameters
- **comfile** (*str*) – Output file name (with .com extension!)
- **overwrite** (*bool*) – (Optional) Overwrite existing files if necessary (True/False).

sector_copy(*isect*)
Copy an existing sector to a new one.

Parameters **isect** (*int*) – Sector number to copy.

sector_delete(*isect*)
Delete an existing sector.

Parameters **isect** (*int*) – Sector number to delete.

sector_new()
Create a new sector.

update()
Obtain the Sectors from SPEX.

Sector

class pyspex.model.**Sector**
Class describing a Sector.

Variables

- **index** (*int*) – Sector number.
- **ncomp** (*int*) – Number of components in sector.
- **comp** (*list*) – List of component objects.
- **distance** (*float*) – Distance for this sector.
- **spectrum** (pyspex.model.Spectrum (page 249)) – Model spectrum for this sector

update(*isect*)
Obtain the information from SPEX for sector isect.
Parameters **isect** (*int*) – Sector number.

Component

class pyspex.model.**Component**
Class containing a model component.

Variables

- **index** (*int*) – Component number.
- **umodel** (*int*) – Model ID number.
- **add** (*bool*) – Is this an additive component?
- **mul** (*bool*) – Is this a multiplicative component?
- **operator** (*bool*) – True for complex components.
- **npar** (*int*) – Number of parameters in components.
- **name** (*str*) – Name of model.
- **nmul** (*int*) – Number of elements in addmul (below).
- **addmul** (*numpy.ndarray*) – Array with the numbers of the multiplicative components to multiply with.
- ***par*** (page 212) (*list*) – List of parameter objects.

update(*isect*, *icomp*)
Obtain the information from SPEX for sector isect and component icomp.
Parameters
- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number of the component.

Parameter

class pyspex.model.**Parameter**
Class for model parameters.

Variables

- **index** (*int*) – Parameter number
- **name** (*str*) – Name of parameter
- **type** (*str*) – Type of parameter (norm, abun, fitp, cons or text)
- **desc** (*str*) – Description of parameter
- **value** (*float*) – Value
- **low** (*float*) – Lower boundary of parameter range
- **upp** (*float*) – Upper boundary of parameter range
- **err_low** (*float*) – Lower error boundary
- **err_upp** (*float*) – Upper error boundary
- **free** (*bool*) – True if parameter is free
- **linked** (*bool*) – True if parameter is linked
- **link_sector** (*int*) – Sector to which parameter is linked
- **link_comp** (*int*) – Component to which parameter is linked
- **link_par** (*int*) – Parameter to which parameter is linked
- **coupling** (*float*) – Coupling factor

update(*isect*, *icomp*, *ipar*)
Obtain the parameter values from SPEX memory.

Parameters
- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **ipar** (*int*) – Parameter number.

Abundance

class pyspex.model.**Abundance**
This class manages the SPEX abundance setting.

Variables

- **index** (*int*) – Abundance index in list
- **ref** (*str*) – Reference to abundance set (string)
- **list** (*tuple*) – The available abundance sets.

get()
Get the current Abundance setting (reference).

set(*abun*)
Set the abundance in SPEX to another set.

Parameters **abun** (*str*) – Abbreviation of the abundance set.

update()
Update the abundance setting in pyspex.

Distance

class pyspex.model.**Distance**
This class is used to set and get the distances in SPEX.

Variables

- **m** (*float*) – Distance in meter
- **au** (*float*) – Distance in Astronomical Units
- **ly** (*float*) – Distance in light year
- **pc** (*float*) – Distance in parsec
- **kpc** (*float*) – Distance in kiloparsec
- **mpc** (*float*) – Distance in megaparsec
- **z** (*float*) – Distance in redshift
- **cz** (*float*) – Distance in cz
- **age** (*float*) – Lookback time (yr)
- **h0** (*float*) – Hubble constant (km/s/Mpc)
- **omega_m** (*float*) – Omega Matter
- **omega_l** (*float*) – Omega Lambda
- **omega_r** (*float*) – Omega R

get(*isect*)
Get the distances for a sector from SPEX.
Parameters **isect** (*int*) – Sector number.

set(*isect*, *dist*, *unit*)
Set the distance in units for a particular sector (isect).
Parameters
- **isect** (*int*) – Sector number.
- **dist** (*float*) – Distance value.
- **unit** (*str*) – Unit of the distance, for example: 'm', 'au', 'ly', 'pc', 'kpc', 'mpc', 'z', 'cz'.

set_cosmo(*h0*, *omega_m*, *omega_l*, *omega_r*)
Set the cosmological constants for the distance calculation.
Parameters
- **h0** (*float*) – Hubble constant (km/s/Mpc).
- **omega_m** (*float*) – Omega matter.
- **omega_l** (*float*) – Omega lambda.
- **omega_r** (*float*) – Omega R.

Energy grids

class pyspex.model.**Egrid**
This class is used to specify the model energy grid.

Variables

- **nbins** (*int*) – Number of bins
- **energy** (*numpy.ndarray*) – Centroids of bins (Energy, keV)
- **energy_upper** (*numpy.ndarray*) – Upper boundaries of bins (Energy, keV)
- **energy_width** (*numpy.ndarray*) – Widths of bins (Energy, keV)

get()
Get the current energy grid from SPEX.

grid(*ebounds*)
Provide a grid to SPEX by providing a numpy array with the bin boundaries. Please note that the length of this array is the number of bins + 1!
Parameters **ebounds** (*numpy.ndarray*) – Array containing the energy boundaries of the new energy grid (keV).

read(*readfile*)
Read the energy grid from a file named readfile (extension: .egr).
Parameters **readfile** (*str*) – Filename to read the energy grid from (including .egr extension)

save(*savefile*)
Save the energy grid to a file named savefile (extension: .egr).
Parameters **savefile** (*str*) – Filename to save the energy grid to (including .egr extension)

set(*elow*, *ehigh*, *nbins*, *unit*, *log*)
Set egrid using limits and number of bins.
Parameters
- **elow** (*float*) – Lowest energy/wavelength for energy grid.
- **ehigh** (*float*) – Highest energy/wavelength for energy grid.
- **nbins** (*int*) – Number of bins for energy grid.
- **unit** (*str*) – Unit of the energy/wavelength range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'
- **log** (*bool*) – Make the energy grid logarithmic (True or False)

set_step(*elow*, *ehigh*, *step*, *unit*, *log*)
Set egrid using limits and step size.
Parameters
- **elow** (*float*) – Lowest energy/wavelength for energy grid.
- **ehigh** (*float*) – Highest energy/wavelength for energy grid.
- **step** (*float*) – Step size for the energy grid.
- **unit** (*str*) – Unit of the energy/wavelength range, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'
- **log** (*bool*) – Make the energy grid logarithmic (True or False)

Fluxes and luminosities

class pyspex.model.**Fluxes**
This class is used to calculate fluxes and luminosities of spectra in a spectral band.

Variables

- ***sector*** (page 216) (*int*) – Sector number of flux calculation
- **component** (*int*) – Component number of flux calculation
- **photflux** (*float*) – Photon flux (phot/m**2/s)
- **enerflux** (*float*) – Energy flux (W/m**2)
- **photlum** (*float*) – Photon luminosity (photons/s)
- **enerlum** (*float*) – Energy luminosity (W)
- ***elimflux*** (page 246) (*numpy.ndarray*) – Flux energy limits (keV)

calc(*isect*, *icomp*)
Calculate and get flux and luminosity from SPEX for a given sector and component number.
Parameters
- **isect** (*int*) – Sector number of the component to calculate.

- **icomp** (*int*) – Component number to calculate.

elim(*elow*, *ehigh*, *unit*)
Set the energy limits for the flux and luminosity calculation.
Parameters
- **elow** (*float*) – Lowest energy/wavelength for energy interval.
- **ehigh** (*float*) – Highest energy/wavelength for energy interval.
- **unit** (*str*) – Unit of the energy/wavelength interval, for example: 'kev', 'ev', 'ryd', 'j', 'hz', 'ang', 'nm'.

elimflux
Flux energy limits (keV)

get(*isect*, *icomp*)
Get the flux and luminosity from SPEX for a given sector and component number.
Parameters
- **isect** (*int*) – Sector number of the component to obtain the flux from.
- **icomp** (*int*) – Component number to obtain the flux from.

Ionisation balance

class pyspex.model.**Ibal**
This class manages the SPEX ionisation balance setting.

Variables

- **index** (*int*) – Index number for list of ionisation balance sets.
- **ref** (*str*) – Reference to the current ionisation balance.
- **list** (*tuple*) – List of available ionisation balance data.

get()
Get the current Abundance setting (reference).

set(*ibal*)
Set the abundance in SPEX to another set.
Parameters **ibal** (*str*) – Abbreviation of the reference to the ionisation balance.

update()
Update the abundance setting in pyspex.

Ion selection

class pyspex.model.**Ions**
Class to manage the ions taken into account in the model calculation.

Variables

- **nz** (*int*) – Total number of atoms considered.
- **atoms** (*list*) – List of atoms (with information about each ion).

ignore_all()
Ignore all ions.

ignore_ion(*z*, *i*)
Ignore this ion.
Parameters
- **z** (*int*) – Atomic number
- **i** (*int*) – Ion number

ignore_iso(*iso*)
Ignore this iso-electronic sequence.

Parameters **iso** (*int*) – Iso-electronic sequence.

ignore_z(*z*)
Ignore all ions of this element.
Parameters **z** (*int*) – Atomic number

lmax_all(*lmax*)
Set all ions to maximum angular momentum lmax.
Parameters **lmax** (*int*) – Maximum angular momentum to use.

lmax_ion(*z*, *i*, *lmax*)
Set this ion to maximum angular momentum lmax.
Parameters
- **z** (*int*) – Atomic number
- **i** (*int*) – Ion number
- **lmax** (*int*) – Maximum angular momentum to use.

lmax_iso(*iso*, *lmax*)
Set this iso-electronic sequence to maximum angular momentum lmax.
Parameters
- **iso** (*int*) – Iso-electronic sequence.
- **lmax** (*int*) – Maximum angular momentum to use.

lmax_z(*z*, *lmax*)
Set all ions of this element to maximum angular momentum lmax.
Parameters
- **z** (*int*) – Atomic number
- **lmax** (*int*) – Maximum angular momentum to use.

new_all()
Use new calculations for all ions.

new_ion(*z*, *i*)
Use new calculations for this ion.
Parameters
- **z** (*int*) – Atomic number
- **i** (*int*) – Ion number

new_iso(*iso*)
Use new calculations for this iso-electronic sequence.
Parameters **iso** (*int*) – Iso-electronic sequence.

new_z(*z*)
Use new calculations for all ions of this element.
Parameters **z** (*int*) – Atomic number

nmax_all(*nmax*)
Set all ions to maximum quantum number nmax.
Parameters **nmax** (*int*) – Maximum principle quantum number to use.

nmax_ion(*z*, *i*, *nmax*)
Set this ion to maximum quantum number nmax.
Parameters
- **z** (*int*) – Atomic number
- **i** (*int*) – Ion number
- **nmax** (*int*) – Maximum principle quantum number to use.

nmax_iso(*iso*, *nmax*)
Set this iso-electronic sequence to maximum quantum number nmax.
Parameters
- **iso** (*int*) – Iso-electronic sequence.
- **nmax** (*int*) – Maximum principle quantum number to use.

nmax_z(*z*, *nmax*)
Set all ions of this element to maximum quantum number nmax.

Parameters
- **z** (*int*) – Atomic number
- **nmax** (*int*) – Maximum principle quantum number to use.

old_all()
Use old calculations for all ions.

old_ion(*z*, *i*)
Use old calculations for this ion.
Parameters
- **z** (*int*) – Atomic number
- **i** (*int*) – Ion number

old_iso(*iso*)
Use old calculations for this iso-electronic sequence.
Parameters **iso** (*int*) – Iso-electronic sequence.

old_z(*z*)
Use old calculations for all ions of this element.
Parameters **z** (*int*) – Atomic number

show()
Show the settings for the ions.

update()
Update the properties of all atoms.

use_all()
Use all ions.

use_ion(*z*, *i*)
Use this ion.
Parameters
- **z** (*int*) – Atomic number
- **i** (*int*) – Ion number

use_iso(*iso*)
Use this iso-electronic sequence.
Parameters **iso** (*int*) – Iso-electronic sequence.

use_z(*z*)
Use all ions of this element.
Parameters **z** (*int*) – Atomic number

Model spectra

class pyspex.model.**Spectrum**
This class obtains and stores the model spectrum from SPEX.

Variables

- **nbins** (*int*) – Number of bins
- **energy** (*numpy.ndarray*) – Centroids of bins (Energy, keV)
- **energy_upper** (*numpy.ndarray*) – Upper boundaries of bins (Energy, keV)
- **energy_width** (*numpy.ndarray*) – Widths of bins (Energy, keV)
- **spectrum** (*numpy.ndarray*) – Spectrum of bins (in ph/s/m**2/bin at observatory)
- **luminosity** (*numpy.ndarray*) – Spectrum of bins (in 10^44 ph/s/keV at source distance)

get(*isect*)
Get the model spectra from SPEX for sector number isect.

Plasma parameters

class pyspex.model.**Var**
Various settings for the plasma models.

Variables

- **gacc** (*float*) – Free-bound accuracy
- **line_ex** (*bool*) – Electron excitation included
- **line_px** (*bool*) – Proton excitation included
- **line_rr** (*bool*) – Radiative recombination included
- **line_dr** (*bool*) – Di-electronic recombination included
- **line_ds** (*bool*) – Di-electronic satellites included
- **line_ii** (*bool*) – Inner shell ionisation included
- **doppler** (*int*) – Doppler broadening
- **newcalc** (*bool*) – SPEXACT 3 calculations (False: SPEXACT 2)
- **occstart** (*int*) – Occupation calculations
- **mekal_wav** (*bool*) – Wavelength corrections according to the work of Phillips et al. (1999)
- **mekal_fe17** (*bool*) – The strongest Fe XVII lines by Doron & Behar (2002).
- **mekal_update** (*book*) – Several minor corrections
- **ibalmaxw** (*bool*) – Multi-Maxwellians for the ionisation balance
- **newcoolexc** (*bool*) – Cooling by collisional excitation by Stofanova (SPEXACT 3)
- **newcooldr** (*bool*) – Cooling by dielectronic recombination (SPEXACT 3)

reset_gacc()
Reset the free-bound emission accuracy.

set_calc(*status*)
Perform SPEXACT 3 line calculations.
Parameters **status** (*bool*) – Use SPEXACT 3 (True) or SPEXACT 2 (False)

set_doppler(*value*)
Doppler broadening.
Parameters **value** (*int*) – Doppler broadening type.

set_gacc(*value*)
Set the free-bound emission accuracy.
Parameters **value** (*float*) – Free-bound emission accuracy.

set_ibalmaxw(*status*)
Switch the Multi-Maxwellians for the ionisation balance on/off (True/False).
Parameters **status** (*bool*) – On (True) or off (False)

set_line(*ltype*, *status*)
Set a line emission contribution on or off.
Parameters
- **ltype** (*str*) – Line emission contribution ('ex', 'px', 'rr', 'dr', 'ds', 'ii', 'reset')

- **status** (*bool*) – Boolean indicator whether contribution is on (True) or off (False).

set_mekal (*utype*, *status*)
Switch old Mekal updates on/off.
Parameters
- **utype** (*str*) – Update type ('wav', 'fe17', 'update', 'all')
- **status** (*bool*) – On (True) or off (False)

set_newcooldr (*status*)
Cooling by dielectronic recombination (SPEXACT 3) on/off (True/False).
Parameters **status** (*bool*) – On (True) or off (False)

set_newcoolexc (*status*)
Cooling by collisional excitation by Stofanova (SPEXACT 3) on/off (True/False).
Parameters **status** (*bool*) – On (True) or off (False)

set_occstart (*otype*)
At which occupation level to start.
Parameters **otype** (*str*) – Occupation level ('ground', 'boltz', 'last')

update ()
Obtain the current plasma model settings from SPEX.

6.3.4 Optimization functions

Fitting spectra

class pyspex.optimize.**Fit**
This is the parent class of all fit related commands.

Variables

- **stat** (*basestring*) – Fit statistics.
- **cstat** (*float*) – C-statistics value.
- **chisq** (*float*) – Chi-squared value.
- **wstat** (*float*) – W-statistics value (not recommended).
- **nfree** (*int*) – Degrees of freedom.
- **cstatexp** (*float*) – Expected C-statistics value (C-stat only).
- **cstatrms** (*float*) – RMS uncertainty on expected C-stat value.

fit (*niter=100*)
Execute the SPEX fit command. The maximum number of iterations (niter) can be optionally set.
Parameters **niter** (*int*) – Number of fit iterations.

get_method ()
Get the current type of statistics being used in the fit, for example chi2, cstat or wstat.

get_statistic ()
Get the current type of statistics being used in the fit, for example chi2, cstat or wstat.

print (*status*)
Print each fit iteration to the console (default is True). The status variable can be either True or False, which means printing is on or off, respectively.
Parameters **status** (*bool*) – Set fit to high verbosity (True is yes, False is no).

set_method (*meth*)
Set the desired fit statistics.

Parameters **stat** (*str*) – Abbreviation for the fit statistics to be used. For example: 'csta', 'chi2', 'wsta'.

set_statistic(*stat*)
Set the desired fit statistics.
Parameters **stat** (*str*) – Abbreviation for the fit statistics to be used. For example: 'csta', 'chi2', 'wsta'.

show()
Print the fit statistics to the terminal.

update()
Get the most recent statistics values from SPEX.

Error calculation

class pyspex.optimize.**Error**
Class to calculate errors for free fit parameters.

Variables

- ***sector*** (page 216) (*int*) – Sector number of parameter
- **component** (*int*) – Component number of parameter
- **parameter** (*str*) – Parameter name
- **value** (*float*) – Parameter value
- **lerr** (*float*) – Lower error boundary
- **uerr** (*float*) – Upper error boundary
- **lc** (*bool*) – Is there a lower C-stat or chi**2 value found?
- **cmin** (*float*) – Lowest C-stat or chi**2 value
- **pmin** (*float*) – Parameter value for which a better C-stat or Chi**2 was found
- **dchi** (*float*) – Delta C-stat or chi**2 to optimize for
- **calculated** (*bool*) – Is the error calculated?

error(*isect*, *icomp*, *name*, *dchi=None*)
Calculate the error value for a particular parameter.
Parameters
- **isect** (*int*) – Sector number of the parameter.
- **icomp** (*int*) – Component number of the parameter.
- **name** (*str*) – Parameter name.
- **dchi** (*float*) – (Optional) $\Delta\chi^2$ value to optimize for (Default: 1.0, 68% errors)

get_value()
Convenience function to return the parameter value and the errors.
Returns A tuple with the parameter value, lower error and upper error.
Return type tuple

set_dchi(*dchi*)
Set the delta c-stat or delta chi**2 value that the error search should optimize for. The default value is 1.0.
Parameters **dchi** (*float*) – $\Delta\chi^2$ value to optimize for (Default: 1.0, 68% errors)

6.3.5 Plot functions

Model spectrum plots

class pyspex.plot.**PlotModel**
Class containing the model spectra for plotting purposes.

Variables

- **nsector** (*int*) – Number of sectors available.
- ***sector*** (page 216) (*list of objects*) – List of plot objects for each sector.

plot(*xlog=False*, *ylog=False*, *title='SPEX Model Spectrum'*, *show=True*)
Plot the model spectrum for all sectors.

Parameters
- **xlog** (*bool*) – (Optional) Set the X-axis to be logarithmic.
- **ylog** (*bool*) – (Optional) Set the Y-axis to be logarithmic.
- **title** (*str*) – (Optional) Set the title of the plot.
- **show** (*bool*) – (Optional) Show the plot (True) or return the plot object (False).

Returns Matplotlib plt object.
Return type matplotlib.pyplot

update()
Update the model spectrum information for all sectors.

class pyspex.plot.**PlotModelSector**
Class containing the spectral model for a particular sector for plotting purposes.

Variables

- **xc** (*numpy.ndarray*) – Central bin energy.
- **xb** (*numpy.ndarray*) – Upper boundary of bin energy.
- **m** (*numpy.ndarray*) – Model spectrum.
- **n** (*int*) – Number of bins.

update(*isect*)
Update the values of the model spectrum for this sector.
Parameters **isect** (*int*) – Sector number.

Data plots

class pyspex.plot.**PlotData**
Class containing the observed spectrum and the convolved model spectrum for plotting.

Variables

- **ninst** (*int*) – Number of instruments
- **inst** (*list*) – List of instruments with data plot information
- **colors** (*tuple*) – List of colors.

chiplot(*xlog=False*, *ylog=False*, *chi='dchi'*, *title='SPEX'*, *show=True*)
Plot the observed data, model and background spectra for all instruments, and include a subwindow with the residuals. Chi options are 'dchi': (observed - model)/error and 'rel': (observed - model)/model.

Parameters
- **xlog** (*bool*) – (Optional) Set the X-axis to be logarithmic.
- **ylog** (*bool*) – (Optional) Set the Y-axis to be logarithmic.

- **chi** (*str*) – (Optional) Type of residual. Either 'dchi' (data-model)/error or 'rel' (data-model)/model.
- **title** (*str*) – (Optional) Set the title of the plot.
- **show** (*bool*) – (Optional) Show the plot (True) or return the plot object (False).

Returns Matplotlib plt object.
Return type matplotlib.pyplot

plot(*xlog=False*, *ylog=False*, *title='SPEX'*, *show=True*)
Plot the observed data, model and background spectra for all instruments.

Parameters
- **xlog** (*bool*) – (Optional) Set the X-axis to be logarithmic.
- **ylog** (*bool*) – (Optional) Set the Y-axis to be logarithmic.
- **title** (*str*) – (Optional) Set the title of the plot.
- **show** (*bool*) – (Optional) Show the plot (True) or return the plot object (False).

Returns Matplotlib plt object.
Return type matplotlib.pyplot

plot_chi(*ax*)
Generate plot data for a plot of the residuals.
Parameters **ax** (*matplotlib.axes.Axes*) – Axis object from Matplotlib.

plot_comp(*xlog=False*, *ylog=False*, *title='SPEX'*, *show=True*)
Plot the total spectrum and the individual additive model components.

Parameters
- **xlog** (*bool*) – (Optional) Set the X-axis to be logarithmic.
- **ylog** (*bool*) – (Optional) Set the Y-axis to be logarithmic.
- **title** (*str*) – (Optional) Set the title of the plot.
- **show** (*bool*) – (Optional) Show the plot (True) or return the plot object (False).

Returns Matplotlib plt object.
Return type matplotlib.pyplot

plot_data(*ax*, *xlog=False*, *ylog=False*)
Generate plot of the data, model and background spectrum. This is a subfunction of plot() and chiplot().

Parameters
- **ax** (*matplotlib.axes.Axes*) – Axis object from Matplotlib.
- **xlog** (*bool*) – (Optional) Set the X-axis to be logarithmic.
- **ylog** (*bool*) – (Optional) Set the Y-axis to be logarithmic.

update(*chi='dchi'*)
Update data plot information for this instrument.
Parameters **chi** (*str*) – (Optional) Type of residual. Either 'dchi' (data-model)/error or 'rel' (data-model)/model.

class pyspex.plot.**PlotDataInst**
Class containing the data plot information for this instrument.

Variables
- **nreg** (*int*) – Number of regions.
- **reg** (*list*) – List of regions with data plot information

update(*inst*, *tchi='dchi'*)
Update the data plot information for this instrument.

Parameters
- **inst** (*int*) – Instrument number.
- **tchi** (*str*) – (Optional) Type of residual. Either 'dchi' (data-model)/error or 'rel' (data-model)/model.

class pyspex.plot.**PlotDataReg**
Class containing the data plot information for a region.

Variables

- **n** (*int*) – Number of bins
- **elow** (*numpy.ndarray*) – Bin lower boundary
- **ectr** (*numpy.ndarray*) – Bin center value
- **eupp** (*numpy.ndarray*) – Bin upper boundary
- **pdata** (*numpy.ndarray*) – Data value
- **pmodel** (*numpy.ndarray*) – Model value
- **pback** (*numpy.ndarray*) – Background value
- **perror** (*numpy.ndarray*) – Error value
- **tints** (*numpy.ndarray*) – Exposure time
- **tarea** (*numpy.ndarray*) – Area value
- **comp** (*list*) – List of model components for a component plot
- **ncomp** (*int*) – Number of model components
- **chi** (*numpy.ndarray*) – Residual value
- **chierr** (*numpy.ndarray*) – Residual error

chitype(*tchi*)
Calculate the residuals for the chi plot window.
Parameters **tchi** (*str*) – (Optional) Type of residual. Either 'dchi' (data-model)/error or 'rel' (data-model)/model.

update(*inst*, *reg*, *tchi='dchi'*)
Update the data plot information for this region.
Parameters
- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.
- **tchi** (*str*) – (Optional) Type of residual. Either 'dchi' (data-model)/error or 'rel' (data-model)/model.

Effective area plots

class pyspex.plot.**PlotArea**
Class for plotting effective area.

Variables

- **ninst** (*int*) – Number of instruments.
- **inst** (*list of objects*) – List of instrument objects.

plot(*xlog=False*, *ylog=False*, *title='Effective Area'*, *show=True*)
Plot the effective area.
Parameters
- **xlog** (*bool*) – (Optional) Set the X-axis to be logarithmic.
- **ylog** (*bool*) – (Optional) Set the Y-axis to be logarithmic.
- **title** (*str*) – (Optional) Set the title of the plot.
- **show** (*bool*) – (Optional) Show the plot (True) or return the plot object (False).

Returns Matplotlib plt object.
Return type matplotlib.pyplot

update()
Update the effective area information for the plot.

class pyspex.plot.**PlotAreaInst**
Instrument definition and region list for area plots.

Variables

- **nreg** (*int*) – Number of regions.
- **reg** (*list*) – List of region area plots.

update(*inst*)
Update the area information in the regions of this instrument.
Parameters **inst** (*int*) – Instrument number to update.

class pyspex.plot.**PlotAreaReg**
Effective area information for each region.

Variables

- **xc** (*numpy.ndarray*) – Energy bin center value (keV)
- **xb** (*numpy.ndarray*) – Energy bin upper boundary value (keV)
- **area** (*numpy.ndarray*) – Effective area (m^2)

update(*inst*, *reg*)
Update effective area information for this region (reg) in instrument (inst).
Parameters
- **inst** (*int*) – Instrument number.
- **reg** (*int*) – Region number.

6.3.6 Ascdump classes

Running an ascdump command to retrieve internal model parameters will return an object with the desired values. On this page, the structure of the object returned for a certain ascdump type is explained. Each section shows the variable names of the parameters such that the numbers can be accessed.

PLAS: Basic plasma properties

class pyspex.ascdump.**Plas**
Output for the command ascdump plas.

Variables

- **t** (*float*) – Electron temperature (keV).
- **elden** (*float*) – Electron density/1E20 (/m**3).
- **hden** (*float*) – Hydrogen density/1E20 (/m**3).
- **ed** (*float*) – Electron/Hydrogen density.
- **pdion** (*float*) – (Electron + Ion)/Hydrogen density.
- **denmas** (*float*) – Mass density / (n_H * m_p).
- **cs** (*float*) – Sound speed (km/s).

get(*isect*, *icomp*)
Obtain the plasma parameters from SPEX.
Parameters
- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

ABUN: Elemental abundances

class pyspex.ascdump.**Abun**
Output for the command ascdump abun.

Variables

- **nz** (*int*) – Number of elements.
- **atom** (*numpy.ndarray*) – Atomic number.
- **name** (*numpy.ndarray*) – Element name.
- **relabn** (*numpy.ndarray*) – Abundance in solar units.
- **absabn** (*numpy.ndarray*) – Absolute abundance.
- **charge** (*numpy.ndarray*) – Average charge.

get(*isect*, *icomp*)
Obtain the abundance information from SPEX.

Parameters

- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

ICON: Ion concentrations

class pyspex.ascdump.**Icon**
Ascdump output containing the ion concentrations.

Variables

- **nline** (*int*) – Number of entries
- **atom** (*numpy.ndarray*) – Atomic number.
- **ion** (*numpy.ndarray*) – Ionisation stage.
- **name** (*numpy.ndarray*) – Element name.
- **roman** (*numpy.ndarray*) – Ionisation stage (Roman).
- **charge** (*numpy.ndarray*) – Ion charge.
- **conrel** (*numpy.ndarray*) – Relative ion concentration.
- **conabs** (*numpy.ndarray*) – Absolute ion concentration.

get(*isect*, *icomp*)
Obtain the ion concentrations from SPEX.

Parameters

- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

RATE: Total ionisation, recombination and charge transfer rates

class pyspex.ascdump.**Rate**
Output for the command ascdump rate.

Variables

- **nline** (*int*) – Number of entries.
- **atom** (*numpy.ndarray*) – Atomic number.
- **ion** (*numpy.ndarray*) – Ionisation stage.
- **name** (*numpy.ndarray*) – Element name.

- **roman** – Ionisation stage (Roman).
- **irate** (*numpy.ndarray*) – Ionisation rate.
- **rrate** (*numpy.ndarray*) – Recombination rate.
- **ceirate** (*numpy.ndarray*) – Charge transfer ionisation rate.
- **cerrate** (*numpy.ndarray*) – Charge transfer recombination rate.

get(*isect*, *icomp*)
Obtain the rates from SPEX.
Parameters
- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

RION: Ionisation rates per atomic subshell

class pyspex.ascdump.**Rion**
Ionisation rates per subshell.

Variables

- **nline** (*int*) – Number of entries
- **atom** (*numpy.ndarray*) – Atomic number
- **name** (*numpy.ndarray*) – Element name
- **ion** (*numpy.ndarray*) – Ionisation stage
- **roman** (*numpy.ndarray*) – Ionisation stage (Roman)
- **shell** (*numpy.ndarray*) – Shell number
- **shname** (*numpy.ndarray*) – Shell name
- **phion** (*numpy.ndarray*) – Photon ionisation rate (/s/ion)
- **cpion** (*numpy.ndarray*) – Compton ionisation rate (/s/ion)
- **elion** (*numpy.ndarray*) – Electron ionisation rate (/s/ion)
- **phheat** (*numpy.ndarray*) – Photon heating (keV/ion)
- **cpheat** (*numpy.ndarray*) – Compton heating (keV/ion)
- **elcool** (*numpy.ndarray*) – Electron cooling (kev/ion)

get(*isect*, *icomp*)
Obtain the ionisation rates per subshell.
Parameters
- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

GRID: Energy and wavelength grid

class pyspex.ascdump.**Grid**
Output the last used energy/wavelength grid used in the model.

Variables

- **nbin** (*int*) – Number of energy bins.
- **ibin** (*numpy.ndarray*) – Bin number.
- **elow** (*numpy.ndarray*) – Low energy boundary of bin (keV).
- **eupp** (*numpy.ndarray*) – Upper energy boundary of bin (keV).

- **ewidth** (*numpy.ndarray*) – Width of the bin (keV).
- **emean** (*numpy.ndarray*) – Mean energy of the bin (keV).
- **wave** (*numpy.ndarray*) – Wavelength of the bin (Angstrom).

get(*isect*, *icomp*)
Obtain the energy grid from SPEX.

Parameters
- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

CLIN: Continuum, line and total spectrum

class pyspex.ascdump.**Clin**
Output the continuum, line and total flux for each energy bin.

Variables
- **nbin** (*int*) – Number of energy bins.
- **ibin** (*numpy.ndarray*) – Bin number.
- **emean** (*numpy.ndarray*) – Mean energy of the bin (keV).
- **fluxcon** (*numpy.ndarray*) – Continuum flux (10^{44} ph/s/keV).
- **fluxlin** (*numpy.ndarray*) – Line flux (10^{44} ph/s/keV).
- **flux** (*numpy.ndarray*) – Total flux (10^{44} ph/s/keV).

get(*isect*, *icomp*)
Obtain the continuum, line and total fluxes from SPEX.

Parameters
- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

LINE: Line energies, wavelengths and total line emission

class pyspex.ascdump.**Line**
The line energy and wavelength, as well as the total line emission (photons/s) for each line contributing to the spectrum, for the last plasma layer of the model. Also given is the natural line width and the Doppler broadening (including thermal and turbulent broadening), expressed as a FWHM in keV.

Variables
- **nline** (*int*) – Number of lines in output.
- **id** (*numpy.ndarray*) – Line ID number
- **atom** (*numpy.ndarray*) – Atomic number
- **name** (*numpy.ndarray*) – Element name
- **ion** (*numpy.ndarray*) – Ionisation stage
- **roman** (*numpy.ndarray*) – Ionisation stage (Roman)
- **al** (*numpy.ndarray*) – Lower level configuration
- **au** (*numpy.ndarray*) – Upper level configuration
- **ener** (*numpy.ndarray*) – Line energy (keV)
- **wave** (*numpy.ndarray*) – Line wavelength (Ang)
- **flux** (*numpy.ndarray*) – Line strength in photons/s

- **width** (*numpy.ndarray*) – Natural line width (FWHM)
- **dopp** (*numpy.ndarray*) – Doppler line width (FWHM)
- **tau** (*numpy.ndarray*) – Optical depth at line center (calculated)
- **pescl** (*numpy.ndarray*) – Single flight line escape factor (calculated)
- **pescc** (*numpy.ndarray*) – Single flight continuum escape factor (calculated, including Thomson scattering)
- **pesc** (*numpy.ndarray*) – Single flight line escape factor including continuum extinction (calculated)
- **epsilon** (*numpy.ndarray*) – Destruction probability (calculated in newlin)
- **sort** (*numpy.ndarray*) – Sorting array containing the sorted indices.

get(*isect*, *icomp*, *sort='ener'*, *erange=0.1, 10.0*, *fluxlim=1e+35*)

Obtain the line list from SPEX.

Parameters

- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.
- **sort** (*str*) – Column to sort.
- **erange** (*Tuple*) – Energy range for output.
- **fluxlim** (*float*) – Minimum line emissivity to show (in photons/s).

EBAL: Energy balance contributions

class pyspex.ascdump.**Ebal**

Output the energy balance contributions of each layer (only photoionized plasmas). (Heating and cooling in W/m**3)

Variables

- **niter** (*int*) – Number of iterations.
- **icond** (*float*) – I
- **ncedec** (*float*) – CE
- **hden** (*float*) – Hydrogen density (1E20/m**3)
- **elden** (*float*) – Electron density (1E20/m**3)
- **ed** (*float*) – Hydrogen / Electron density
- **t** (*float*) – Electron temperature (keV)
- **dif** (*float*) – Delta
- **heat** (*float*) – Total heating
- **cool** (*float*) – Total cooling
- **heatcom** (*float*) – Heating by Compton scattering
- **heatffa** (*float*) – FF absorption heating
- **heatphi** (*float*) – Photo-electron heating
- **heatcio** (*float*) – Compton ion. heating
- **heataug** (*float*) – Auger electron heating
- **heatdex** (*float*) – Collisional de-excitation heating
- **heatext** (*float*) – External heating
- **coolcom** (*float*) – Compton scattering cooling

- **coolion** (*float*) – Collisional ionisation cooling
- **coolrec** (*float*) – Recombination cooling
- **coolffe** (*float*) – FF emission cooling
- **coolexc** (*float*) – Collisional excitation cooling
- **cooldr** (*float*) – Di-electronic recombination cooling
- **cooladi** (*float*) – Adiabatic cooling

get(*isect*, *icomp*)

Obtain the energy balance for each iteration of the model.

Parameters

- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

CON: Ions contributing to the continuum

class pyspex.ascdump.**Con**

List of the ions that contribute to the free-free, free-bound and two-photon continuum emission, followed by the free-free, free-bound, two-photon and total continuum spectrum for the last plasma layer of the model.

Variables

- **ff_nion** (*int*) – Number of ions contributing to free-free continuum.
- **ff_atom** (*numpy.ndarray*) – Atomic numbers of ions contributing to free-free continuum.
- **ff_ion** (*numpy.ndarray*) – Ion numbers of ions contributing to free-free continuum.
- **ff_name** (*numpy.ndarray*) – Ion names of ions contributing to free-free continuum.
- **fb_nion** (*int*) – Number of ions contributing to free-bound continuum.
- **fb_atom** (*numpy.ndarray*) – Atomic numbers of ions contributing to free-bound continuum.
- **fb_ion** (*numpy.ndarray*) – Ion numbers of ions contributing to free-bound continuum.
- **fb_name** (*numpy.ndarray*) – Ion names of ions contributing to free-bound continuum.
- **tf_nion** (*int*) – Number of ions contributing to two-photon continuum.
- **tf_atom** (*numpy.ndarray*) – Atomic numbers of ions contributing to two-photon continuum.
- **tf_ion** (*numpy.ndarray*) – Ion numbers of ions contributing to two-photon continuum.
- **tf_name** (*numpy.ndarray*) – Ion names of ions contributing to two-photon continuum.
- **nbin** (*int*) – Number of spectral bins.
- **ener** (*numpy.ndarray*) – Mean energy (keV).
- **ff** (*numpy.ndarray*) – Free-Free continuum (1E44 ph/s/keV).
- **fb** (*numpy.ndarray*) – Free-Bound continuum (1E44 ph/s/keV).
- **tf** (*numpy.ndarray*) – Two-photon continuum (1E44 ph/s/keV).

- **tot** (*numpy.ndarray*) – Total continuum (1E44 ph/s/keV).

get(*isect*, *icomp*)
Obtain the continuum contributions from SPEX.

TCL: Layer added continuum, line and total spectrum

class pyspex.ascdump.**Tcl**
Output the continuum, line and total flux for each energy bin summed for all plasma layers of the model.

Variables

- **nbin** (*int*) – Number of energy bins.
- **ibin** (*numpy.ndarray*) – Bin number.
- **emean** (*numpy.ndarray*) – Mean energy of the bin (keV).
- **fluxcon** (*numpy.ndarray*) – Continuum flux (10^{44} ph/s/keV).
- **fluxlin** (*numpy.ndarray*) – Line flux (10^{44} ph/s/keV).
- **flux** (*numpy.ndarray*) – Total flux (10^{44} ph/s/keV).

get(*isect*, *icomp*)
Obtain the continuum, line and total fluxes from SPEX.

Parameters
- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

TCON: Ions contributing to the continuum (added layers)

class pyspex.ascdump.**Tcon**
List of the ions that contribute to the free-free, free-bound and two-photon continuum emission, followed by the free-free, free-bound, two-photon and total continuum spectrum added for all plasma layers of the model.

Variables

- **ff_nion** (*int*) – Number of ions contributing to free-free continuum.
- **ff_atom** (*numpy.ndarray*) – Atomic numbers of ions contributing to free-free continuum.
- **ff_ion** (*numpy.ndarray*) – Ion numbers of ions contributing to free-free continuum.
- **ff_name** (*numpy.ndarray*) – Ion names of ions contributing to free-free continuum.
- **ff_layer** (*numpy.ndarray*) – Number of layers in the model
- **fb_nion** (*int*) – Number of ions contributing to free-bound continuum.
- **fb_atom** (*numpy.ndarray*) – Atomic numbers of ions contributing to free-bound continuum.
- **fb_ion** (*numpy.ndarray*) – Ion numbers of ions contributing to free-bound continuum.
- **fb_name** (*numpy.ndarray*) – Ion names of ions contributing to free-bound continuum.
- **fb_layer** (*numpy.ndarray*) – Number of layers in the model
- **tf_nion** (*int*) – Number of ions contributing to two-photon continuum.

- **tf_atom** (*numpy.ndarray*) – Atomic numbers of ions contributing to two-photon continuum.
- **tf_ion** (*numpy.ndarray*) – Ion numbers of ions contributing to two-photon continuum.
- **tf_name** (*numpy.ndarray*) – Ion names of ions contributing to two-photon continuum.
- **tf_layer** (*numpy.ndarray*) – Number of layers in the model
- **nbin** (*int*) – Number of spectral bins.
- **ener** (*numpy.ndarray*) – Mean energy (keV).
- **ff** (*numpy.ndarray*) – Free-Free continuum (1E44 ph/s/keV).
- **fb** (*numpy.ndarray*) – Free-Bound continuum (1E44 ph/s/keV).
- **tf** (*numpy.ndarray*) – Two-photon continuum (1E44 ph/s/keV).
- **tot** (*numpy.ndarray*) – Total continuum (1E44 ph/s/keV).

get (*isect*, *icomp*)
Obtain the continuum contributions from SPEX.

POP: Level populations

class pyspex.ascdump.**Pop**
The occupation numbers as well as upwards/downwards loss and gain rates to all quantum levels included.

Variables

- **nlev** (*int*) – Number of levels
- **num** (*numpy.ndarray*) – Level number
- **atom** (*numpy.ndarray*) – Atomic number
- **name** (*numpy.ndarray*) – Element name
- **ion** (*numpy.ndarray*) – Ion charge
- **roman** (*numpy.ndarray*) – Ionisation stage (Roman)
- **conf** (*numpy.ndarray*) – Configuration of level
- **ener** (*numpy.ndarray*) – Energy of level (keV)
- **occ** (*numpy.ndarray*) – Population (sum=1)
- **occlte** (*numpy.ndarray*) – Pop/PopLTE
- **cascade** (*numpy.ndarray*) – Cascade (per s)
- **excite** (*numpy.ndarray*) – Excitation (per s)
- **down** (*numpy.ndarray*) – Down loss (per s)
- **up** (*numpy.ndarray*) – Upward loss (per s)
- **arr** (*numpy.ndarray*) – Radiative recombination (per s)
- **adr** (*numpy.ndarray*) – Dielectronic recombination (per s)
- **aii** (*numpy.ndarray*) – Innershell ionisation (per s)
- **cxrr** (*numpy.ndarray*) – Charge exchange (per s)

get (*isect*, *icomp*)
Obtain the occupation numbers from SPEX.

ELEX: Electron collision excitation and de-excitation

class pyspex.ascdump.**Elex**

Output the collisional excitation and de-excitation rates due to collisions with electrons.

Variables

- **nline** (*int*) – Number of transitions.
- **il** (*numpy.ndarray*) – Number of lower level
- **iu** (*numpy.ndarray*) – Number of upper level
- **atom** (*numpy.ndarray*) – Element number (atomic number)
- **name** (*numpy.ndarray*) – Element name
- **ion** (*numpy.ndarray*) – Ionisation stage
- **roman** (*numpy.ndarray*) – Ionisation stage (Roman)
- **al** (*numpy.ndarray*) – Configuration of lower level
- **au** (*numpy.ndarray*) – Configuration of upper level
- **upsilon** (*numpy.ndarray*) – Upsilon
- **gbar** (*numpy.ndarray*) – Gbar
- **exrate** (*numpy.ndarray*) – Excitation rate (/atom/s)
- **dexrate** (*numpy.ndarray*) – De-excitation rate (/atom/s)

get(*isect*, *icomp*)

Obtain the collisional excitation and de-excitation rates due to collisions with electrons from SPEX.

Parameters

- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

PREX: Proton collision excitation and de-excitation

class pyspex.ascdump.**Prex**

Output the collisional excitation and de-excitation rates due to collisions with protons.

Variables

- **nline** (*int*) – Number of transitions.
- **il** (*numpy.ndarray*) – Number of lower level
- **iu** (*numpy.ndarray*) – Number of upper level
- **atom** (*numpy.ndarray*) – Element number (atomic number)
- **name** (*numpy.ndarray*) – Element name
- **ion** (*numpy.ndarray*) – Ionisation stage
- **roman** (*numpy.ndarray*) – Ionisation stage (Roman)
- **al** (*numpy.ndarray*) – Configuration of lower level
- **au** (*numpy.ndarray*) – Configuration of upper level
- **exrate** (*numpy.ndarray*) – Excitation rate (/atom/s)
- **dexrate** (*numpy.ndarray*) – De-excitation rate (/atom/s)

get(*isect*, *icomp*)
Obtain the collisional excitation and de-excitation rates due to collisions with protons from SPEX.

Parameters
- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

RAD: Radiative transition rates

class pyspex.ascdump.**Rad**
Output the radiative transition rates from each level

Variables
- **nline** (*int*) – Number of transitions.
- **il** (*numpy.ndarray*) – Number of lower level
- **iu** (*numpy.ndarray*) – Number of upper level
- **atom** (*numpy.ndarray*) – Element number (atomic number)
- **name** (*numpy.ndarray*) – Element name
- **ion** (*numpy.ndarray*) – Ionisation stage
- **roman** (*numpy.ndarray*) – Ionisation stage (Roman)
- **al** (*numpy.ndarray*) – Configuration of lower level
- **au** (*numpy.ndarray*) – Configuration of upper level
- **prob** (*numpy.ndarray*) – Transition probability

get(*isect*, *icomp*)
Obtain the radiative transition probablilities from SPEX.

Parameters
- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

TWO: Two-photon emission

class pyspex.ascdump.**Two**
Output the two-photon emission transition rates from each level.

Variables
- **nline** (*int*) – Number of two-photon transitions.
- **il** (*numpy.ndarray*) – Number of lower level
- **iu** (*numpy.ndarray*) – Number of upper level
- **atom** (*numpy.ndarray*) – Element number (atomic number)
- **name** (*numpy.ndarray*) – Element name
- **ion** (*numpy.ndarray*) – Ionisation stage
- **roman** (*numpy.ndarray*) – Ionisation stage (Roman)
- **al** (*numpy.ndarray*) – Configuration of lower level
- **au** (*numpy.ndarray*) – Configuration of upper level
- **prob** (*numpy.ndarray*) – Transition probability

get(*isect*, *icomp*)
Obtain the two-photon transition probablilities from SPEX.

Parameters
- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

TIME: Recombination timescale per ion

class pyspex.ascdump.**Time**

Recombination time scale per ion according to the Bottorf et al. (2000) definition, and relative ion concentrations. Note that the recombination time scale depends upon the hydrogen density, so do not forget to set the relevant density in the model.

Variables
- **nline** (*int*) – Number of two-photon transitions.
- **atom** (*numpy.ndarray*) – Element number (atomic number)
- **name** (*numpy.ndarray*) – Element name
- **ion** (*numpy.ndarray*) – Ionisation stage
- **roman** (*numpy.ndarray*) – Ionisation stage (Roman)
- **texp** (*numpy.ndarray*) – Time scale (s)
- **conrel** (*numpy.ndarray*) – Relative concentration

get(*isect*, *icomp*)

Obtain the ionisation/recombination time scale for each ion from SPEX.

Parameters
- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

REC: Recombination and inner-shell ionisation for each level

class pyspex.ascdump.**Rec**

Outputs for each atomic level the populating contributions from radiative, dielectronic and charge exchange recombination, as well as inner-shell ionisation.

Variables
- **nline** (*int*) – Number of two-photon transitions.
- **ilev** (*numpy.ndarray*) – Level number.
- **atom** (*numpy.ndarray*) – Element number (atomic number)
- **name** (*numpy.ndarray*) – Element name
- **ion** (*numpy.ndarray*) – Ionisation stage
- **roman** (*numpy.ndarray*) – Ionisation stage (Roman)
- **tran** (*numpy.ndarray*) – Transition type
- **arr** (*numpy.ndarray*) – Radiative recombination rate (1E-20m**3/s).
- **adr** (*numpy.ndarray*) – Di-electronic recombination rate (1E-20m**3/s).
- **cxrr** – Charge exchange recombination rate (1E-20m**3/s).
- **aii** – Inner-shell ionisation rate (1E-20m**3/s).

NEI: History of ionisation parameter and temperature

class pyspex.ascdump.**Nei**
Output the history of ionisation parameter and temperature for NEI models.

Variables

- **nbin** (*int*) – Number of histogram bins.
- **u** (*numpy.ndarray*) – Ionisation parameter (1E20 s/m**3).
- **kt** (*numpy.ndarray*) – Temperature (keV).

get(*isect*, *icomp*)
Obtain the history of the ionisation parameter from SPEX.

Parameters

- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

HEAT: Plasma heating rates

class pyspex.ascdump.**Heat**
Plasma heating rates (only for photoionized plasmas).

Variables

- **heat** (*float*) – Total heating
- **heatcom** (*float*) – Heating by Compton scattering
- **heatffa** (*float*) – Heating by free-free absorption
- **heatphi** (*float*) – Heating by photo-electrons
- **heatcio** (*float*) – Heating by Compton ionisation
- **heataug** (*float*) – Heating by Auger electrons
- **heatdex** (*float*) – Heating by collisional de-excitation
- **heatext** (*float*) – Heating by external source
- **cool** (*float*) – Total cooling
- **coolcom** (*float*) – Cooling by inverse Compton scattering
- **coolion** (*float*) – Cooling by electron ionisation
- **coolrec** (*float*) – Cooling by radiative recombination
- **coolffe** (*float*) – Cooling by free-free emission
- **coolexc** (*float*) – Cooling by collisional excitation
- **cooldr** (*float*) – Cooling by dielectronic recombination
- **cooladi** (*float*) – Cooling by adiabatic expansion

get(*isect*, *icomp*)
Obtain plasma heating rates from SPEX.

Parameters

- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

COL: Ionic column densities

class pyspex.ascdump.**Col**
Output the ionic column densities.

Variables

- **nline** (*int*) – Number of lines.
- **atom** (*numpy.ndarray*) – Atomic number.
- **ion** (*numpy.ndarray*) – Ionisation stage.
- **name** (*numpy.ndarray*) – Element name.
- **roman** (*numpy.ndarray*) – Ionisation stage (Roman).
- **column** (*numpy.ndarray*) – Column density (1E28/m**2).
- **logcol** (*numpy.ndarray*) – Recombination rate (/m**2).

get(*isect*, *icomp*)
Obtain the ionic column denstities from SPEX.

Parameters

- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.

TRAN: Transmission for absorption lines and edges

class pyspex.ascdump.**Tran**

In two subsequent objects, the transmission and equivalent width of absorption lines (self.line) and absorption edges (self.edge) are listed for the hot, pion, slab, xabs and warm models.

Variables

- **line** (pyspex.ascdump.Tranline (page 268)) – Object containing the transmission and equivalent widths of the absorption lines.
- **edge** (pyspex.ascdump.Tranedge (page 269)) – Object containing the transmission and equivalent widths of the absorption edges.

get(*isect*, *icomp*, *sortn='ener'*)
Obtain the transmission and equivalent width from absorption lines and edges from SPEX. This works for the hot, pion, slab, xabs, and warm models.

Parameters

- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.
- **sortn** (*str*) – Sort the data based on column: energy (ener), wavelength (wav), ion (ion), line power (powe), natural line width (wid).

class pyspex.ascdump.**Tranline**
The transmission and equivalent width of absorption lines for the hot, pion, slab, xabs and warm models.

Variables

- **nline** (*int*) – Number of lines
- **id** (*numpy.ndarray*) – Line ID number
- **atom** (*numpy.ndarray*) – Atomic number
- **ion** (*numpy.ndarray*) – Ionisation stage
- **name** (*numpy.ndarray*) – Element name.

- **roman** (*numpy.ndarray*) – Ionisation stage (Roman).
- **ener** (*numpy.ndarray*) – Line energy (keV)
- **ew** (*numpy.ndarray*) – Equivalent width (keV)
- **tau** (*numpy.ndarray*) – Optical depth tau
- **avoigt** (*numpy.ndarray*) – Natural line width
- **sorted** (*numpy.ndarray*) – Sorted index array

get(*isect*, *icomp*, *sortn='ener'*)

Obtain the transmission and equivalent width from absorption lines from SPEX. This works for the hot, pion, slab, xabs, and warm models.

Parameters

- **isect** (*int*) – Sector number of the component.
- **icomp** (*int*) – Component number.
- **sortn** (*str*) – Sort the data based on column: energy (ener), wavelength (wav), ion (ion), line power (powe), natural line width (wid).

class pyspex.ascdump.**Tranedge**

The transmission and equivalent width of absorption edges for the hot, pion, slab, xabs and warm models.

Variables

- **nedge** (*int*) – Number of edges
- **atom** (*numpy.ndarray*) – Atomic number
- **ion** (*numpy.ndarray*) – Ionisation stage
- **shell** (*numpy.ndarray*) – Atomic shell
- **shname** (*numpy.ndarray*) – Shell name
- **name** (*numpy.ndarray*) – Element name.
- **roman** (*numpy.ndarray*) – Ionisation stage (Roman).
- **ener** (*numpy.ndarray*) – Edge energy
- **tau** (*numpy.ndarray*) – Optical depth tau
- **ew** (*numpy.ndarray*) – Equivalent width (keV)

get(*isect*, *icomp*)

Obtain the transmission and equivalent width from absorption edges from SPEX. This works for the hot, pion, slab, xabs, and warm models.

WARM: Column densities, ionisation parameters and temperatures

class pyspex.ascdump.**Warm**

Column densities, ionisation parameters and temperatures of the warm model.

Variables

- **nline** (*int*) – Number of lines.
- **atom** (*numpy.ndarray*) – Atomic number.
- **ion** (*numpy.ndarray*) – Ionisation stage.
- **name** (*numpy.ndarray*) – Element name.
- **roman** (*numpy.ndarray*) – Ionisation stage (Roman).
- **xi** (*numpy.ndarray*) – Log Xi (1E-9 Wm).
- **t** (*numpy.ndarray*) – Log T (keV).

- **col** (*numpy.ndarray*) – dN/d ln Xi.
- **nxil** (*int*) – Number of Xi grid points.
- **xilgrid** (*numpy.ndarray*) – Xi (1E-9 Wm).
- **dndlnxi** (*numpy.ndarray*) – dN/d ln Xi

get(*isect*, *icomp*)
Obtain the column densities, ionisation parameters and temperatures of the warm model from SPEX.

6.3.7 Logging commands

class pyspex.log.**Log**
Class to save and execute commands and output from the terminal.

Variables

- **do_output** (*bool*) – Parameter indicating whether log output is on.
- **do_save** (*bool*) – Parameter indicating whether log save commands is on.

close(*logtype*)
Close the save or output file.

Parameters **logtype** (*str*) – String indicating the type of log to close ('save' or 'out').

execute(*filename*)
Execute commands listed in an ascii file with a .com extension.

Parameters **filename** (*str*) – Filename of the ASCII file with the SPEX commands (including .com extension).

Return type int

output(*outfile*, *status=None*)
Save the terminal output to ascii file. The status is optional and either overwrite or append.

Parameters

- **outfile** (*str*) – Filename of the output file (including .out extension).
- **status** (*str*) – (Optional) Overwrite or append an existing file name. String can be 'overwrite' or 'append'.

save(*savefile*, *status=None*)
Save the commands for a session in an ascii file. The status is optional and either overwrite or append.

Parameters

- **savefile** (*str*) – Filename of the output command file (including .com extension).
- **status** (*str*) – (Optional) Overwrite or append an existing file name. String can be 'overwrite' or 'append'.

CHAPTER

SEVEN

HELP & TROUBLESHOOTING

7.1 Commandline help

If you are unsure about the syntax of a command or you forgot the exact name of a command, then the command-line help function in SPEX can help you. Simply entering `help` on the commandline explains the basic usage of `help`:

```
SPEX> help
Usage:
   SPEX> help <command>

For a list of available help files, type 'help index'.

See also the SPEX Reference manual and the SPEX Cookbook on:
http://www.sron.nl/spex
```

7.1.1 List of commands

When you type `help index`, SPEX will display all the commands for which a help text is available. From there you can pick the command that you want to see a simple help text for:

```
SPEX> help index
Help files are available for these commands:

  abundance     elim    log         simulate
  ascdump       error   model       step
  bin           fit     multiply    syserr
  calculate     help    obin        system
  comp          ibal    par         use
  data          ignore  plot        var
  dem           index   sector      vbin
  distance      ions    shiftplot   watch
  egrid

Usage:
  SPEX> help <command>
```

7.1.2 Print help for command

When you know the command that you need help for, just type `help <command>`, for example we show the help for the abundance command below:

```
SPEX> help abundance
This command changes the used set of elemental abundances.

Usage:
  SPEX> abundance <set>

where <set> is the set of abundances used. Allowed sets:

  reset    - default set, currently Lodders et al. 2009
  ag       - Anders & Grevesse 1989
  allen    - Allen 1973
  ra       - Ross & Aller 1976
  grevesse - Grevesse et al 1992
  gs       - Grevesse & Sauval 1998
  lodders  - Lodders proto-Solar 2003
  solar    - Lodders Solar photospheric 2003

To show the current abundance set in use:
  SPEX> abun show
```

7.2 Solving SPEX problems

In case you encounter a problem with SPEX, like a command does not do what you want, you get unexpected results or in the worst case the program crashes, then you arrived at the right page. Here we provide a checklist which may either allow you to solve the problem yourself or provide us with enough information to help you.

1. Does SPEX give you an error message? Although not all error messages clearly identify the problem, they do contain information about what is wrong. If the error message is not obvious, then entering the error message in Google or any other search engine may reveal solutions for your problem found by other people.
2. If there is no clear error message, then it is important to know at which step in the process the problem appears to occur. Are you unable to start a program, or does the program quit after reading in files? The exact moment can give important hints about what may go wrong. If the problem occurs in SPEX, you could create a .com file with the commands that cause the issue and see at which command the error occurs.

   SPEX can also provide information about which subroutine SPEX is running. With the command watch sub true, SPEX will print out the subroutines for all following commands. Putting this line in the top of your .com file will make it easier to identify where the program fails.
3. Check the *Find known issues* (page 274) page how to search our Github page for similar problems. There may be very recent problems reported that are not indexed by search machines yet.
4. When the points above do not lead you to a solution, please submit an issue to our issue tracker on Github. The *Report issues* (page 274) page contains instructions on how to do that.

7.2.1 Advanced debugging

If SPEX crashes with a segmentation fault, then it may help to run a debugger on the SPEX program. We do not expect you to do this for each problem, but we may ask you to run SPEX trough a debugger if you submitted an issue.

A very general debugger is the GDB program, which is available on most platforms. On the command line, you can enter:

```
linux:~/spex> gdb $SPEX90/bin/spex

GNU gdb (Debian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from /spex/bin/spex...done.
(gdb)
```

When GDB is started, you can type run to start SPEX:

```
(gdb) run
Starting program: /spex/bin/spex
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
 Welcome user to SPEX version 3.05.00

 NEW in this version of SPEX:
11-06-2018 Added Ext_Rate column to new spo files
18-12-2018 SPEX is now using the GPL license

 Currently using SPEXACT version 2.07.00. Type `help var calc` for details.

SPEX>
```

In the next step, you can enter the SPEX commands that lead to the problem. Having the commands in a .com file would be most practical:

```
SPEX> log exe bug
```

The GDB program will run slower than usual. When the error occurs, GDB will show more detailed error messages related to the crash. If you want even more information, you can give the bt command to backtrace where the error occurs:

```
(gdb) bt
```

This output can be a bit intimidating, but it should show in which line of the code things go wrong. This kind of information is very helpful to us in solving programming errors.

7.3 Find known issues

If you encounter a problem with SPEX, it is good to check whether the problem is already known. One way to check this is, for example, entering the error message in a search engine (like Google). This may yield articles about the SPEX problem that you have, which may point to a solution.

It is also good to check our Github issue tracker. The problem that you have may have been reported before (and did not end up in a search engine yet).

If you cannot find information about your issue with SPEX, then please see the *Report issues* (page 274) page to submit an issue to us. We will try to help you as quickly as possible.

7.4 Report issues

If you have a problem with SPEX that you are unable to solve using our *Solving SPEX problems* (page 272) page and the problem does not appear when you follow *Find known issues* (page 274), then you can submit an issue to our Github issue tracker. Please read the text below before clicking the link on the bottom of this page.

Also feature requests and questions about how to use the software are welcome. However, keep in mind that we have limited menpower. We may not be able to implement a new feature in a short time.

When you submit an issue, please provide the following information (where applicable):

- The name of the program causing the problem or that you want to discuss.
- The name of your operating system (e.g. Ubuntu Linux, Mac OSX, etc.).
- A careful explanation of the problem (e.g. What is wrong? At which point does it occur?, etc.)
- A copy of the error message (if available).
- If possible, a command file or script that (re)produces the problem.
- If necessary, a link to spectrum or response files that we could use to reproduce the problem.

The more information about the problem we have, the better we can help you. You can imagine that just telling us “Program X does not work!” (it really happens) does not contain enough information for us to find out what is going on. So please try to explain the problem in as much detail as possible in your first post.

Please keep in mind that your issue is likely to appear online for everybody to read. This way, people with similar SPEX problems can benefit from the answer that we give.

Go to our Github issue tracker.

CHAPTER

EIGHT

SPEX THEORY

8.1 SPEX Atomic Code & Tables

8.1.1 The SPEX Atomic Code & Tables (SPEXACT)

The SPEX package is a spectral fitting program with integrated models that are mostly based on one atomic database and a set of routines to calculate all the atomic processes in the plasma. Although the development of the atomic database and code is performed mostly in parallel with the other SPEX development, it can sometimes be confusing which version of the atomic database was actually used in an analysis. Especially after the major update of the atomic database in SPEX 3.0 and the option to calculate models using the 'old' database and routines from SPEX 2.0, there is a need to name and version the 'core' routines of SPEX separately. This has become SPEXACT (SPEX Atomic Code & Tables). In principle, models calculated using the same SPEXACT version should produce the same results.

Please note that the SPEXACT database and routines are an integrated part of SPEX and are not distributed separately.

Main version number definition

SPEXACT v1: Is essentially the MEKAL model that was developed in the 1980's and is distributed with Xspec. This model is no longer developed or supported, but can be regarded as the first version of SPEXACT. It is no longer included in SPEX.

SPEXACT v2: In SPEX version 1 & 2, the original MEKAL model was extended and updated into SPEX. The version number of this SPEXACT version is the same as the version number of SPEX when it was released. For example, the SPEXACT version in SPEX version 2.04.00 is also 2.04.00. In SPEX version 3, these routines are still the default and are used in `var calc old` mode.

SPEXACT v3: These are the newly developed atomic database and corresponding routines that were officially released with SPEX version 3.00.00. This database and its routines are not (yet) the default in SPEX, but can be used when the `var calc new` command is given. The second number in the SPEXACT version is the same as the SPEX version it was released in. The third number can in principle be different from the SPEX version and indicates the minor version of the database.

8.1.2 Plasma model in SPEX 3.0

The core of the plasma model

The old plasma code used by in version 2.0 and below is essentially the same plasma code as developed originally by Rolf Mewe and colleagues, with relatively minor updates to the atomic data (like wavelength improvements, corrections of a few typo's, improvements for Fe XVII).

Its basis were pre-calculated and parametrized line emissivities for each spectral line, as a function of temperature, with for relevant lines empirical density corrections. For some transitions, like the He-like triplets, the calculations were rather complex and required several correction factors to account for the full density dependence.

In the new approach presented here, the basic plasma processes are evaluated for each individual level, and then the occupation numbers of the excited states are calculated for the full ion, solving a matrix equation. This has the great advantage that with the same effort a multitude of processes can be taken into account, including effects of photo-excitation and photo-ionisation. From the occupation numbers and the radiative transition probabilities it is then straightforward to calculate the emitted spectrum.

In order to keep the code fast and flexible, we have chosen for a procedure to parametrise all relevant processes, and using simple analytical formulae with a limited number of parameters for each process. This is beneficial both in terms of computation time and storage demands and formed the basis for the succes of Mewe's original work.

The production of the relevant files is not yet complete, but in the first release of version 3.0 we incorporate the data for the H, He, and Li iso-electronic sequences, and some data for the other sequences, including the Fe-L ions. For an overview of what is in the code see Section *Ions for which updated calculations are available* (page 276) below.

By default, the plasma code is the old version 2.0 code, but by giving the command "var calc new", for the ions for which new data are available, the new code is used. This leads in principle to higher accuracy and many more spectral lines. A disadvantage is of course that the computations become somewhat slower. For spectral fitting, one could envision a procedure where in a first run the old code is used to get close to the best parameters, and then to refine using the new plasma core.

If you want to use the new plasma model, it is important to make sure the ionisation balance is the new *u17* balance. This new balance has improved collisional ionisation rates and allows to calculate properly inner-shell transitions that are needed for the new calculations. See Urdampilleta et al. (2017) for details.

Finally, it is advised that the users have a look to the various ascii-output options that are available for the plasma models, allowing to get deeper insight into the physical process and parameters that are being used.

Ions for which updated calculations are available

Below we list for each iso-electronic sequence what data is available for the new plasma calculations. Each line corresponds to one ion, with *ii, iz and jz* corresponding to the iso-electronic sequence, nuclear charge and ionisation stage, respectively. Then for a number of process we list two quantities: N, the number of entries we have (e.g., number of energy levels or transition rates), and n_{max} the maximum principal quantum number for wich we use data for that ion and process. Note that n_{max} refers to the highest level included for a transition between two levels.

The processes incorporated and shown in the table are:

1. *levels* – Energy levels
2. *Arad* – Radiative transition probabilities (Einstein coefficients)
3. *Atwo* – Two-photon processes
4. *El col* – Electron collisional excitation
5. *Pr col* – Proton collisional excitation
6. *Aug* – Auger rates (auto-ionisation, needed for dielectronic recombination)
7. *CX* – Charge exchange recombination
8. *RR* – Radiative recombination
9. *II* – Inner-shell processes (this is merely the number of quantum states after ionisation; the number of fluorescent lines will be larger)

8.1.3 Absorption model theory

Introduction

In most astrophysical situations we have to take into account absorption of photons between the emitting region and the observer. Apart from a few standard models like the ones by Morrison & McCammon (1983) (our `absm` model) and Rumph et al. (1994) (our `euve` model) we have constructed our own absorption models based upon the atomic database used by SPEX.

Essentially, we adopt a few steps, which will be described below. First, we produce a set of column densities, in different ways for the diferent absorption models (see *Different types of absorption models* (page 277)). Next, using a dynamical model for the absorber we calculate its transmission (*Dynamical model for the absorbers* (page 278)). For these calculations, we use our atomic database as described in *Atomic database for the absorbers* (page 279).

A basic assumption in all the absorption models is that there is no re-emission, i.e. we look through an absorbing nmedium that has a very small solid angle as seen from the X-ray source. This allows essentially to calculate the transmission simply as $e^{-\tau(E)}$ with $\tau(E)$ the optical depth.

Thomson scattering

The above approach also allows us to include Thomson-scattering in the transmission. Any source photon aimed at the observer but that suffers from Thomson scattering is scattered out of the line of sight and hanec in this approximation is not seen by the observer. We have included not simply the Thomson cross-section σ_T but have taken the Klein-Nishina correction into account (see Rybicki & Lightman 1986, eqn. 7.5 (exact expression) and 7.6 (approximations)). The evaluation of the exact formula for the cross section is non-trivial, as terms up to the third power in $x = E/m_e c^2$ cancel; we have extended the low-energy polynomial approximation (7.6.a) of Rybicki & Lightman by comparing to quadruple precision calculations using the exact formula, and made the following approximation that has a relative accuracy of better than 3×10^{-4} for all energies, when evaluated using single precision arithmetics:

$$\sigma = \begin{cases} \sigma_T(1 - 2x + 5.2x^2 - 13.3x^3 + 32.685x^4) & x < 0.05; \\ 0.75\sigma_T \left[\frac{1+x}{x^3}\left\{\frac{2x(1+x)}{1+2x} - \ln(1+2x)\right\} + \frac{\ln(1+2x)}{2x} - \frac{1+3x}{(1+2x)^2}\right] & 0.05 < x < 5000; \\ 0.375\sigma_T(\ln(2x) + 0.5)/x & x > 5000. \end{cases}$$

Different types of absorption models

We have a set of spectral models available with different levels of sophistication and applicability, that we list below.

Slab model

The *slab* model calculates the transmission of a slab of material, where all ionic column densities can be chosen independently. This has the advantage that a spectrum can be fit without any knowledge of the ionisation balance of the slab. After a spectral fit has been made, one may try to explain the observed column densities by comparing the with predictions from any model (as calculated by , Cloudy, XSTAR, ION or any other existing (photo)ionization code).

Xabs model

In the *xabs* model, the ionic column densities are not independent quantities, but are linked through a set of runs using a photo ionization code. See the description of the *xabs* model for more details about this. The relevant parameter is the ionization parameter $\xi = L/nr^2$, with L the source luminosity, n the hydrogen density and r the distance from the ionizing source. The advantage of the *xabs* model over the *slab* model is that all relevant ions are taken into account, also those which would be detected only with marginal significance using the *slab* model. In some circumstances, the combined effect of many weak absorption features still can be significant. A disadvantage of the *xabs* model happens of course when the ionization balance of the source is different from the ionization balance that was used to produce the set of runs with the photo ionization code. In that case the *xabs* model may fail to give an acceptable fit, while the *slab* model may perform better.

Warm model

In the *warm* model, we construct a model for a continuous distribution of column density $N_{\rm H}$ as a function of ξ. It is in some sense comparable to the differential emission measure models used to model the emission from multi-temperature gas. Here we have absorption from multi-ionization gas. Depending upon the physics of the source, this may be a better approximation than just the sum of a few *xabs* components. A disadvantage of the model may be (but this also depends upon the physics of the source), that all dynamical parameters for each value of ξ are the same, like the outflow velocity and turbulent broadening. If this appears to be the case in a given source, one may of course avoid this problem by taking multiple, non-overlapping *warm* components.

Hot model

In the *hot* model, we link the different ionic column densities simply by using a collisional ionsation (CIE) plasma. It may be useful in situations where photoionisation is relatively unimportant but the source has a non-negligible optical depth. A special application is of course the case for a low temperature, where it can be used to mimick the absorption of (almost) neutral gas.

Pion model

Finally we have in the pion model, which does a self-consistent photo ionization calculation of the slab of material.

Dynamical model for the absorbers

For each of the absorption models described in the previous section, we have the freedom to prescribe the dynamics of the source. The way we have implemented this in is described below.

The transmission $T(\lambda)$ of the slab is simply calculated as

$$T(\lambda) = \exp[-\tau_c(\lambda) - \tau_l(\lambda)]$$

with τ_c and τ_l the total continuum and line optical depth, respectively. As long as the thickness of the slab is not too large, this most simple approximation allows a fast computation of the spectrum, which is desirable for spectral fitting.

In particular UV observations of AGN show that the absorption lines can often be decomposed into multiple velocity components. In the X-ray band these are not always fully resolvable, which led us to the following approach. Each absorption line is split into different velocity components, using

$$\tau_l(v) = \sum_i \tau_i \exp\left[-(v - v_i)^2/2\sigma_{\rm v}^2\right]$$

(or the equivalent generalisation to the Voigt profile). Further, we take

$$v_i = v_0 + i\,\Delta v,$$

$$\tau_i = K \exp\left[-v_i^2/2\sigma_\mathrm{b}^2\right],$$

where v_0 is the average velocity of the blend (a negative value corresponds to a blue-shift or outflow), Δv is the separation between the velocity components, and the r.m.s. width of the blend σ_b is in general larger than the intrinsic width σ_v of the components (do never confuse both σ's!). The normalization K is defined in such a way that $\sum \tau_i = \tau_0$. Finally, the total optical depth τ_0 is given by

$$\tau_0 = 0.106 f N_{20} \lambda / \sigma_{\mathrm{v},100}.$$

Here f is the oscillator strength, λ the wavelength in Å, $\sigma_{\mathrm{v},100}$ the velocity dispersion in units of 100 km/s and N_{20} the total column density of the ion in units of 10^{20} m^{-2}.

This dynamical structure offers the user a broad range of applicability. However, we advise the user to use the extension with σ_b with caution! Always start with the most simple case. The default values for are defined in such a way that $\sigma_\mathrm{b} = 0$. This will produce the "normal" case of single absorption lines. In that case, the velocity separation Δv is an irrelevant parameter.

Finally, we make a remark on the r.m.s. line width of individual lines, σ_v. In our code, this *only* includes the turbulent broadening of the lines. The thermal broadening due to motion of the ions is included by adding it in quadrature to the tutbulent broadening. The only exception is the *slab* model, where of course due to the lack of underlying physics the thermal broadening is unknown, and therefore in using the slab model one should be aware that σ_v also includes a thermal contribution.

8.1.4 Atomic database for the absorbers

The continuum opacities are taken from Verner & Yakovlev (1995). Line opacities and wavelengths for most ions are from Verner et al. (1996), with the following additions and exceptions:

K-shell transitions

For some hydrogenic ions (C, N, O, Ne, Mg, Si, S and Ar) we added the transitions from principal quantum numbers 11–20 to the ground using our own SPEX database.

C-K transitions

Three inner-shell K-shell transitions for C I were calculated by Raassen using the Cowan code, but here only the oscillator strengths were calculated, and therefore calculated equivalent widths for large optical depths will be too weak. We added two C IV lines from the work of Nahar et al. (2001).

N-K transitions

For N I the K-shell transitions were calculated by Raassen using the Cowan code. We adjusted the wavelength of the strongest 1s–2p line acording to measurements. However, for the 1s–2p lines towards $1s2s^22p^4\ ^4P$, we used oscilator strengths and Auger widths from Garcia et al. (2009).

Inner-shell K-shell transitions for N II were also calculated by Raassen using the Cowan code, but here only the oscillator strengths were calculated, and therefore calculated equivalent widths for large optical depths will be too weak. The exception to this are the 1s–2p absorption lines to the $1s2s^22p^3\ ^3S_1$, $^3P_1$ and $^3D_1$ levels, for which we use Garcia et al. (2009).

For N III we added the 1s–2p absorption lines towards the $1s2s^22p^2\ ^2S$, ^2P and $^2D_{3/2}$ levels from the paper by Garcia et al. (2009). Wavelengths are theoretical, so may be somewhat off.

For N IV we added the 1s–2p absorption lines towards the $1s2s^22p\ ^1P_1$ and $1s2p^3\ ^1P_1$ levels from the paper by Garcia et al. (2009). Wavelengths are theoretical, so may be somewhat off.

For N V we added the 1s–2p absorption lines towards the $1s(2S)2s2p(3P)\ ^2P$ doublet and $1s(2S)2s2p(1P)\ ^2P$ doublet from the paper by Garcia et al. (2009). Wavelengths were corrected according to the measurements of Beiersdorfer et al. (1999).

For N VI we added the 1s–np lines for $n = 5 - 7$ from our SPEX database.

O-K transitions

We included the inner shell K-shell transitions of oxygen ions (O I – O VI) from earlier work of Behar (HULLAC calculations, private commumication).

We adjusted the wavelength of the strongest line to 22.370 ± 0.010 Å, taken from Gu et al. (2005).

The two 1s–2p $^2P_{1/2,3/2}$ lines of O IV were adjusted to 22.741 and 22.739 Å, respectively following Gu et al. (2005). The lines to the $^2D_{3/2}$ and $^2S_{1/2}$ terms were not adjusted as no values are given by Gu et al. (2005).

For O III, Gu et al. (2005) give only a single line instead of the three dominant lines to $^3D_1$, $^3S_1$, $^3P_1$. The oscillator-strength weighted average wavelength for these three lines using Behar's HULLAC calculations is 23.058 Å, compared to 23.065 Å as given by Gu et al. (2005). Other sets of calculation differ much from this, up to 0.05–0.10 Å Olalla et al., 2002; Pradhan et al., 2003; Juett et al., 2004; so we keep the Behar values lacking more reliable calculations.

For O II, Juett et al. (2004) identified a feature in observed Chandra HETGS spectra towards several sources with the 1s–2p line from this ion; their measured wavelength is 23.351 ± 0.003 Å. This identification seems to be justified given the Auger electron measurements of Krause (1994) and Caldwell et al. (1994) as cited by Garcia et al. (2005). The relevant transitions are from the ground $^4S_{3/2}$ to the $^4P_{5/2}$, $^4P_{3/2}$, and $^4P_{1/2}$ levels; the oscillator strength weighted wavelength from Behar is 23.3012 Å. Therefore we shift the wavelengths of these transitions by $+0.0498$ Å in order to match the Juett et al. (2004) value.

Finally, in a similar way we shift the strong 1s–2p doublet from to match the weighted average of the value found by Juett et al. (2004) with the Chandra HETGS (23.508 ± 0.003 Å) and in Mrk 421 with RGS (23.5130 ± 0.0022 Å) to (23.5113 ± 0.0018 Å). For all O I lines, we have updated the oscillator strengths and transition rates to values obtained by Ton Raassen using Cowan's code, and benchmarked to an adjusted wavelength scale.

Lines from O VII up to $n = 100$ were calculated by extrapolation of the data for $n \leq 10$.

Ne-K to Ca-K transitions

The strongest K-shell transitions from Li-like to F-like ions of Ne, Mg, Al, Si, S, Ar, and Ca were taken from Behar & Netzer (2002).

Fe-K transitions

K-shell transitions in Fe II – Fe XXIV were included from the series of papers by Palmeri et al. (2003), Mendoza et al. (2004), and Bautista et al. (2004).

In addition, the strongest 1s–3p transitions in Fe XXIII were added from Behar & Netzer (2002).

L-shell transitions

Fe-L transitions

For the L-shell ions of iron (Fe XVII - Fe XXIV) we used HULLAC calculations. The wavelengths of these L-shell transitions were adjusted according to Phillips et al. (1999). Also the L-shell ions of Si VIII – Si XII, S X – S XIV and Ar XV were calculated using the HULLAC code. In addition, we added the 2p–3d inner shell resonance lines of Fe I - Fe XVI from Behar et al. (2001). These inner shell resonance lines occur mainly in the 16 - 17 Å band and appear to be extremely important diagnostic tools, since the normal resonance lines of these lowly ionized iron ions have their wavelengths mainly in the inaccessible EUV band.

Ni-L transitions

L-shell transitions of Ni I – Ni XVIII were calculated by Raassen using Cowan's code, with the neglect of line broadening due to auto-ionizations.

M-shell transitions

For M-shell transitions of Fe ions, we have used data calculated by Ehud Behar using HULLAC (Fe I – Fe XVI). The calculations for (Fe VI – Fe XVI) were replaced on November 21, 2008 by updated calculations by Ming Feng Gu using FAC and compared with the NGC 3783 spectrum in Gu et al. (2006). They represent slight (~ 0.03 Angstrom) shifts from Behar's initial calculations. Also, they exhibit typically 2–3 times higher total transition rates (Auger rates).

8.1.5 Non-equilibrium ionisation (NEI) calculations

Our current implementation of time-dependent, non-equilibrium ionisation spectra is based upon the work of Kaastra & Jansen (1993), KJ hereafter. The method consists essentially upon calculating the transition matrix (containing ionisation and recombination rates) for a grid of temperatures, calculating the eigenvalues and eigenvectors of these matrices and also the inverse eigenvector matrix, and storing those on disk. The ionisation balance is calculated then by dividing the temperature history in several steps, with the steps defined in such a way that the temperature over each time interval is close to one temperature grid point. Then for each step the time evolution of the plasma is calculated using eigenvalue decomposition and using the pre-calculated coefficients for the relevant temperature. For more details see KJ.

Here we make a few changes with respect to KJ. First, KJ adopted a prescribed history of the Hydrogen density, and calculate the corresponding electron density at each time from the ion concentrations at that time. This was motivated by the argument that for a fixed plasma element the number of Hydrogen atoms remains constant over time, while the number of electrons depends upon the degree of ionisation and may be time-dependent, thus not known before the ionisation balance has beeen solved completely. But the situation is slightly more complicated. If the plasma is not (nearly) completely ionised, then for normal abundances there must be a considerable population of neutral Hydrogen or Helium. But in that case charge exchange reactions become important, and should be taken into account in the transition matrix. This however is impossible to incorporate in the KJ method, since in this case there are 3 free parameters ($n_{\rm e}/n_{\rm H}$, $n_{\rm e}/n_{\rm He}$ and T) instead of 1 (T), and storage of matrices etc. becomes physically impossible. Knowing that the method of KJ becomes inaccurate for low ionisation rates we decided to leave the prescribed $n_{\rm H}$ concept and return to the prescribed $n_{\rm e}$ concept. This has the advantage that the equations for the ionisation balance of each chemical element can be calculated independently from the other elements; KJ need to calculate the concentrations of all elements for each time step in order to determine the conversion from Hydrogen to electron density.

Another rationale behind the above-mentioned modification is the following. For self-similar supernova remnants, a class of problems where non-equilibrium ionisation applies, the hydrodynamics is inconsistent with a time- or place varying electron/Hydrogen ratio: that would introduce another scale length and destroy the assumptions behind self-similarity. Thus, for SNR models it is necessary to assume that the plasma is (almost) completely ionised in order to assure self-consistency. We will neglect small changes in the electron/Hydrogen ratio of SNRs therefore.

The second modification concerns the temperature grid. Contrary to KJ, we use a keV grid instead of a K grid. Further, since we now include 30 chemical elements instead of 15, we had to reduce the step size h of the grid from 0.05 in $\log T$ to 0.10, in order not to get disk storage problems. This change however is compensated by a better interpolation technique used here. KJ just rounded all temperatures to the nearest grid point. Here we interpolate the calculated concentrations between two temperature grid points linearly, which is one order of magnitude (h) better. A rough estimate learns that the error in the method of KJ per decade is nh^2 which gives for $n = 1/h$ an error of order 0.05, while the current implementation has typical errors per decade of nh^3 which gives a typical error of 0.01 per decade.

8.1.6 Non-thermal electron distributions

In the current version of SPEX it is possible to include the effects of non-thermal (NT) electron distributions. Such NT distributions affect the spectrum in two ways: by changing the ionization balance and the emitted spectrum. Currently we take both effects into account for all our plasma models. The only exception is the NEI model, for non-equilibrium ionisation. The way we calculate the ionisation balance does not allow us to include the NT effects in the ionisation balance, but we do take it into account in the spectrum.

The way we implemented it is as follows. Our atomic data (like collision strengths) are all parameterized in a way that allow for analytical integration over simple electron distributions like delta-functions, Maxwellians or power laws. But there are other processes like radiative recombination where such an analytical approach fails. However, one way or the other, we use expressions for Maxwellian-averaged rates in all our calculations. In principle, it is possible to decompose any electron distribution as a linear combination of Maxwellians. The total rate, for example a recombination rate, is then the sum of the rates caused by the individual Maxwellian components.

Therefore, in all our rate calcualtions we build the total rate based upon such a linear combination of Maxwellians. This is done for all relevant processes (ionization, recombination, excitation, etc.)

In all our thermal models, there is an ascii-parameter called "file"; if this value is defined (i.e. when a file name of an existing file is entered), it will read the parameters of the Maxwellians from an ascii-file with that filename. If there is not such a file, or if the filename is reset by entering the `par file aval none` command, no file will be taken into account (i.e., we have a simple, single Maxwellian again).

The (ascii) file should have the following format. On the first line, the number of Maxwellians is given. Then for each Maxwellian there is a separate line containing two numbers: the first number is the temperature of the Maxwellian, in units of the main temperature of the plasma; the second number is the total number of electrons in this Maxwellian, relative to the main component. It is wise to have the parameters of the main component as the first line of this list.

Let us give an example. Suppose we have a plasma with three components: 2, 20 and 200 keV Maxwellians, with electron densities of 3000, 300 and 30 m^{-3}, respectively. In this case the parameters of the *cie* model should be: temperature 2 keV, electron density 3000 m^{-3}, and the corresponding file should be as follows:

```
3
1 1
10 0.1
100 0.01
```

The first line tells us that there are three Maxwellians. The second line contains the parameters of the first Maxwellian, that we scale here to 1 1 (it is the one with temperature 2 keV and electron density 3000 m^{-3}). The third lines contain the second Maxwellian, which has a 10 times higher temperature but also a 10 times lower electron density as the first component. Finally, the fourth line contains the parameters of the third Maxwellian, which has a 100 times higher temperature and a 100 times lower electron density as the first component.

8.1.7 Supernova remnant model theory

For the calculation of SNR models we follow Kaastra & Jansen (1993). But also here we make some modifications in the implementation.

The X-ray spectrum of a SNR is characterised in lowest order by its average temperature T, ionisation parameter U and emission measure Y. If one calculates a specific model however, these average values are not known a priori. A relevant set of input parameters could be the interstellar density, the shock radius and the explosion energy. These are the parameters used by KJ. A disadvantage is that this choice is not suitable for spectral fitting: changing one of the parameters gives correlated changes in T, U and Y, which is an undesirable feature in spectral fitting. KJ solved this by inputting a guess of T, U, and Y and using these guess parameters as the fitting parameters. After fitting, the model had to be calculated another time in order to find the relation between guessed and true parameters. here we take another choice. We will select input parameters which scale with T, U and Y respectively but are connected to the SNR parameters independent of the hydrodynamical model. The first parameter is the shock temperature

$$T_s,$$

the second the shock ionisation parameter defined by

$$U_s \equiv n_{\rm e} t$$

where $n_{\rm e}$ is the electron density before the shock (for a completely ionised gas), and t the age of the remnant. Further we define the shock reduced emission measure by

$$Y_s \equiv n_{\rm e} n_{\rm H} r_s^3 / d^2$$

where r_s is the main shock radius, and d the distance to the source. We have introduced here also the electron density n_e instead of the more natural Hydrogen density, because this allows an efficient use of the model for spectral fitting: if one fits a spectrum with variable chemical abundances, then using the current scaling we need to calculate (for a given T_s, U_s and Y_s) the relative ion concentrations only once; for other abundances, the relative concentrations may be simply scaled without a need to redo completely the ionisation balance calculations. Finally we introduced the distance in the definition of Y_s in order to allow an easy comparison with spectra observed at earth. These 3 input parameters T_s, U_s and Y_s will serve as input parameters for all hydrodynamical models. They are linked to other parameters as follows:

$$f_{\rm p} k T_s = f_{\rm T} \epsilon_\rho m_{\rm p} r_s^2 / t^2 \tag{8.1}$$

where ϵ_ρ is a dimensionless factor representing the average mass in units of a proton mass per Hydrogen atom ($\rho = \epsilon_\rho m_{\rm p} n_{\rm H}$ with $m_{\rm p}$ the proton mass), and $f_{\rm p}$ is another dimensionless factor, representing the ratio of electron plus ion density over the Hydrogen density. Further, $f_{\rm T}$ is a model-dependent dimensionless scaling factor which is given by

$$f_{\rm T} = \frac{f_v^2}{\eta - 1}$$

with f_v the velocity scaling discussed below and η the density jump at the main shock front, which depends upon the Hydrodynamical model. Another scaling law links the gas velocity to the temperature:

$$v_2 = f_v r_s / t$$

with v_2 the post-shock gas velocity (not to be confused with the shock velocity!). The velocity scaling factor is given by

$$f_v \equiv \frac{(n-3)(\eta-1)}{(n-s)\eta}$$

with the understatement that for the Sedov and Solinger model one should take formally the limit $n = 5$.

The explosion energy E_0 is linked to the other parameters by

$$E_0 = \frac{\alpha r_s^5 \rho}{t^2},$$

where α is the dimensionless energy integral introduced by Sedov. The value of α depends both upon the hydrodynamical model and the value for s and n, the density gradient scale heights of the interstellar medium and the stellar ejecta.

If we express T_s in keV, U_s in 10^{20} m^{-3}s, Y_s in 10^{20} m^{-5}, r_s in units of 10^8 m, $n_{\rm H}$ in 10^{20} m^{-3}, d in units of 10^{22} m and t in s, we obtain from (8.1) the value for the proton mass of $m_{\rm p}$ = 1.043969×10^5 keV^2(10^{16} m^2)$^{-1}$. Using this value and defining $f_m = f_T \epsilon_\rho m_{\rm p} / f_{\rm p}$, the scaling laws for SNRs may be written as

$$n_{\rm e} = \frac{T_s^{1.5} U_s^3}{f_m^{1.5} Y_s d^2}$$

$$t = \frac{f_m^{1.5} Y_s d^2 n_{\rm e}}{T_s^{1.5} U_s^2 n_{\rm H}}$$

$$r = \frac{f_m Y_s d^2 n_{\rm e}}{U_s^2 T_s n_{\rm H}}$$

$$v = \frac{f_v T_s^{0.5}}{f_m^{0.5}}$$

$$E_0 = [1.6726231 \times 10^{33}\ \mathrm{J}] \frac{\alpha \epsilon_\rho f_m^{0.5} Y_s^2 d^4 n_\mathrm{e}}{U_s^3 T_s^{0.5} n_\mathrm{H}}$$

$$M = [8.409 \times 10^{-14}\ \mathrm{M}_\odot] \frac{\epsilon_\rho \beta \eta f_m^{1.5} Y_s^2 d^4 n_\mathrm{e}}{U_s^3 T_s^{1.5} n_\mathrm{H}}$$

where M is the mass integral, and β is the dimensionless mass integral (defined by $\beta = 4\pi \int \frac{\rho}{\rho_s} (\frac{r}{r_s})^2 \mathrm{d}\frac{r}{r_s}$).

8.2 Modelling and fitting

This chapter explains the procedures and tricks used in SPEX for spectral modelling and fitting.

8.2.1 Calculating models on grids

Model spectra are generally calculated for an energy or wavelength grid. In SPEX, the default energy grid has 8192 bins between 0.001 and 100 keV. This grid is used when calculating models without an instrument defined, for example, when you plot a model spectrum using the `plot type model` command (*Plot types* (page 123)). The resolution of this grid is usually enough, but when it is not, the egrid command can help to increase the resolution or broaden the energy range (*Egrid: define model energy grids* (page 88)).

Energy grids when loading data

When data is loaded into SPEX, then the energy grid is automatically adapted to the loaded response matrix. Because response matrices already contain a model energy grid and a channel grid, the SPEX energy grid should match the one from the response, at least within the energy range defined in the response matrix. This way, the model is always calculated at a resolution that is set by the response matrix.

Special cases

For some models, like PION, the broad-band spectrum is very important for the calculation. In those cases, the energy band should be extended. Recommended is a logarithmic grid between $10^{-6} - 10^6$ keV with a step size of 0.005 (see *Pion: SPEX photoionised plasma model* (page 163)). Thus, the default grid has 200 bins per decade. Which energy grid should be used depends on your settings and wishes. The range (lower and upper energy) matter for the physical outcome of the PION model, due to the effects of (inverse) Compton scattering of high- and low energy photons.

When an observed spectrum and response are loaded, we could stick with just the energies contained in the matrix. However, for the PION model also energies outside this range are relevant, because the photons there contribute to heating/cooling/ionisation etc. That is the reason why the energy grid is extended for this model to span the full 1E-6 to 1E6 range. We realise that the 200 bins per decade of that extension is an arbitrary choice, but we think for most cases that is accurate enough to represent the IR or gamma-ray range.

This extension of the energy grid happens in SPEX always, as soon as you read data, even if you do not have the PION model. This is because SPEX cannot anticipate what model components the user may want to add or delete to the full model that is being used. But of course it is harmless for all other models.

You might argue that sometimes you may also want to have higher resolution than 200 bins per decade in the optical or UV range, for instance in those cases where you might have joint HST/COS data with high resolution. Fortunately, whenever the corresponding UV data (response & spectrum) is added, the way SPEX determines its energy grid as described above would include the high-resolution UV part automatically at the desirable high resolution.

Checking the energy grid

Whenever you read data (a single spectrum, or from multiple instruments), SPEX creates a new energy grid on which the model is evaluated, before folding with the response matrices. This energy grid is the combination of all energies on the photon axis (not to be confused with the count axis) of the response matrices of all instruments involved, supplemented with a grid with a spacing of 0.005 in log energy spanning between 1E-6 and 1E6 keV.

To see what happens in a specific case, you can plot the model spectrum, put on the x-axis the energy in keV, and choose `plot uy bin` which gives you the bin number. Also issue the command `plot ux kev` to re-determine the min/max energies of the grid. This should show the binning of your response matrix/matrices as well as the extension.

Another method to inspect the energy grid is running the `egrid save` command (*Egrid: define model energy grids* (page 88)). This will save the full energy grid to file.

8.2.2 Different types of spectral components

SPEX distinguishes three different types of model components, called *additive*, *multiplicative*, and *hybrid* components respectively. Additive components have a normalisation that determines the flux level, for example power-law, CIE, and other emission models. Multiplicative components operate on additive components, for example absorption models like HOT, XABS and AMOL. The pion model is a hybrid model that models both the absorption and emission from an intervening slab of plasma.

The redshift component is treated as a multiplicative component, since it operates on additive components.

Additive components can be divided into two classes: simple components (like power law, blackbody etc.) and plasma components, that use our atomic code. For the plasma components it is possible to plot or list specific properties, while for the simple models this is not applicable.

Multiplicative components can be divided into 3 classes. First, there are the absorption-type components, like interstellar absorption. These components simply are an energy-dependent multiplication of the original source spectrum. has both simple absorption components as well as absorption components based upon our plasma code. The second class consists of shifts: redshift (either due to Doppler shifts or due to cosmology) is the prototype of this. The third class consists of convolution-type operators. An example is Gaussian velocity broadening.

For more information about the currently defined spectral components in, see Chapter *Overview of spectral components* (page 137).

8.2.3 Sectors and regions

Introduction

In many cases an observer analyses the observed spectrum of a single X-ray source with a single instrument. There are however situations with more complex geometries or multiple instruments involved that require a more complicated setup of model components and data.

Example 1: An extended source, where the spectrum may be extracted from different regions of the detector, but where these spectra need to be analysed simultaneously due to the overlap in point-spread function from one region to the other. This situation is e.g. encountered in the analysis of cluster data with ASCA or BeppoSAX.

Example 2: For the RGS detector of XMM-Newton, the actual data-space in the dispersion direction is actually two-dimensional: the position z where a photon lands on the detector and its energy or pulse height E as measured with the CCD detector. X-ray sources that are extended in the direction of the dispersion axis ϕ are characterised by spectra that are a function of both the energy E and off-axis angle ϕ. The sky photon distribution as a function of (ϕ,E) is then mapped onto the (z,E)-plane. By defining appropriate regions in both planes and evaluating the correct (overlapping) responses, one may analyse extended sources.

Example 3: One may also fit simultaneously several time-dependent spectra using the same response, e.g. data obtained during a stellar flare.

Example 4: When one wants to fit the instrumental and/or particle background simultaneously with the source spectrum, the particle background needs to be folded with a different response function.

These examples can be modeled and fitted using the SPEX concept of sectors and regions.

Sectors

Sectors are a way in SPEX to organize or group model components that are related to each other. For example, we may put components related to the source spectrum and components related to the (particle) background into separate sectors.

A sector may also contain the components for different sources on the sky. For example if there is a bright point source superimposed upon the diffuse emission of the cluster, we can define two sectors: an sector for the cluster emission, and sector for the point source. Both sectors might even overlap, as this example shows!

Another example: the two nearby components of the close binary α Centauri observed with the XMM-Newton instruments, with overlapping point-spread-functions of both components. In that case we would have two sectors that each describe one of the double star's components.

The model spectrum for each sector may and will be different in general. For example, in the case of an AGN superimposed upon a cluster of galaxies, one might model the spectrum of the point-like AGN sector using a power law, and the spectrum from the surrounding cluster emission using a thermal plasma model.

Sectors can be defined using the command *Sector: creating, copying and deleting of a sector* (page 111) in combination with the *Comp: create, delete and relate spectral components* (page 82) command.

Regions and instruments

The observed count rate spectra are extracted in practice in different regions of the detector. It is necessary here to distinguish clearly the model sectors and detector regions. A detector region for the XMM EPIC camera would be for example a rectangular box, spanning a certain number of pixels in the x- and y-directions. It may also be a circular or annular extraction region centered around a particular pixel of the detector, or whatever spatial filter is desired. For the XMM RGS it could be a specific "banana" part of the detector-coordinate CCD pulse-height plane (z,E).

Note that the detector regions need not to coincide with the sectors, neither should their number to be equal! A good example of this is again the example of an AGN superimposed upon a cluster of galaxies. The sky sector corresponding to the AGN is simply a point, while, for a finite instrumental psf, its extraction region at the detector is for example a circular region centered around the pixel corresponding to the sky position of the source.

Also, one could observe the same source with a number of different instruments and analyse the data simultaneously. In this case one would have only one sky sector but more detector regions, namely one for each participating instrument.

Regions and instruments have a very similar role in SPEX. The subtle difference is that regions usually have the same or similar response properties, while different instruments have different responses. Although this difference exists, in practice instruments and regions are often mixed.

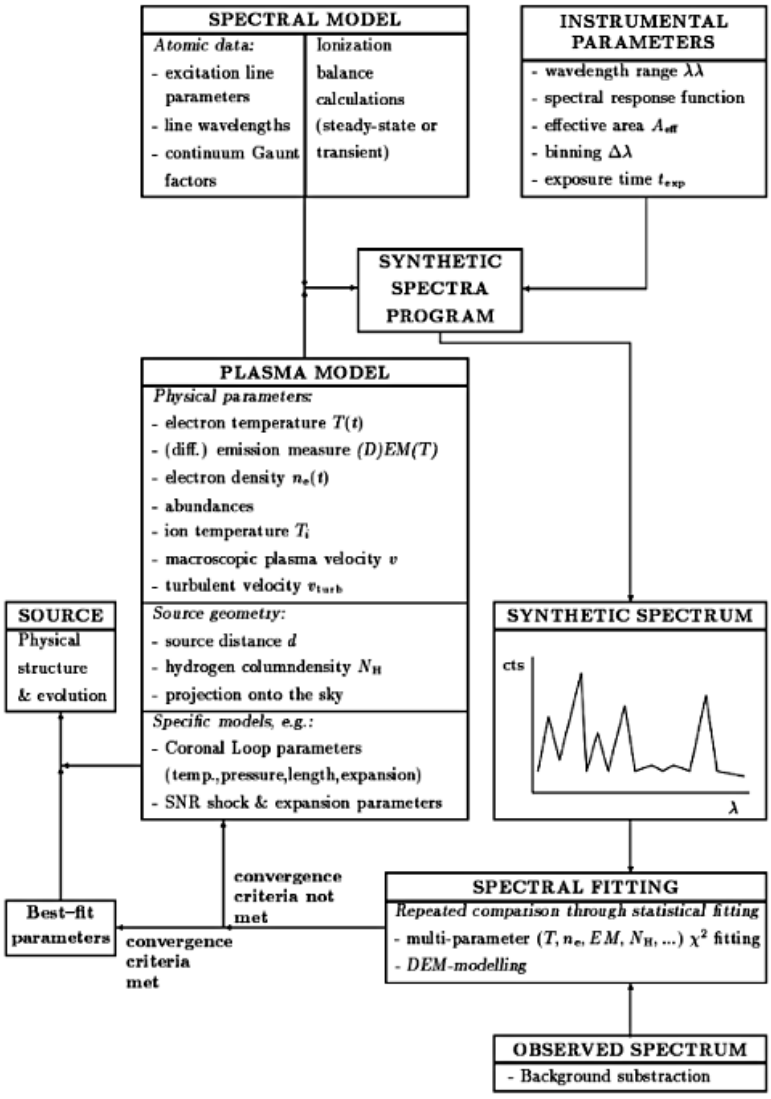
Including sectors and regions in trafo

If certain model components need to be applied to particular spectra, the `trafo` program (*Trafo* (page 181)) can be used to predefine the sector and region numbers in the .res and .spo files. This can be helpful in the case of modelling particle background simultaneously with the source spectrum. In trafo, two sectors can be defined to contain the source and background components, to make sure that the background components are folded with a different response than the source components. In addition, model spectra from both sectors can be added and applied to one detector region. This example is explained in more detail in *Modeling particle background* (page 32).

8.2.4 Spectral Fitting

To infer relevant physical parameters from observed X-ray spectra, physical models are needed. The models need to calculate the expected X-ray spectrum based on physical parameters, like the electron temperature, emission measure and density distributions, ion and elemental abundances, mass motions, and the nature of the ambient radiation field. Unfortunately, the physical models cannot be compared directly to the measured spectra, because the instruments used to obtain the spectra change the spectrum for example by adding background components (instrument noise), change the continuum shape due to the sensitivity of the mirror (effective area), and blur spectral lines due to the instruments spectral resolution.

To take these instrumental components into account, the usual procedure is to apply a forward modelling technique by convolving theoretical model spectra with the instrumental response and to vary the model parameters in order to optimize the fit of the model to the observational data. A common approach is to consider first a simplified plasma model for the X-ray source, neglecting much of the complexity of the temperature and density structure and of the effects of opacity, and to synthesize such models into successively more sophisticated approximations of the source model.



In Figure *1* (page **??**) we give in a processing flow diagram schematically the process of spectral modelling for the case of optically thin coronal plasmas. The synthetic spectra program is fed with input parameters from the spectral model (atomic data for ionization and line and continuum excitation), the instrumental model, and from the assumed plasma model for the source. The synthetic spectra code generates spectra which can be compared to the observations and tested by means of statistical fitting procedures.

8.3 Optimal definition of respons matrices

The SPEX FITS format allows us to bin spectra and response matrices on arbitrary grids, which enables us to bin spectra and responses in an optimal way. In Kaastra & Bleeker (2016) a theoretical framework is developed in order to estimate the optimum binning of X-ray spectra. Expressions are derived for the optimum bin size for the model spectra as well as for the observed data using different levels of sophistication. It is shown that by not only taking into account the number of photons in a given spectral model bin but also its average energy (not necessarily equal to the bin center) the number of model energy bins and the size of the response matrix can be reduced by a factor of 10-100. The response matrix should then not only contain the response at the bin center, but also its derivative with respect to the incoming photon energy. In the coming sections we describe practical guidelines how to construct the optimum energy grids as well as how to structure the response matrix. Finally a few examples are given to illustrate the present methods.

8.3.1 Proposed file formats

In Kaastra & Bleeker (2016) it was shown how the optimum data and model binning can be determined, and what the corresponding optimum way to create the instrumental response is. Now I focus upon the possible data formats for these files.

For large data sets, the fastest and most transparent way to save a response matrix to a file would be to use direct fortran or C write statements. Unfortunately not all computer systems use the same binary data representation. Therefore the FITS-format has become the *de facto* standard in many fields of astronomical data analysis. For that reason I propose here to save the response matrix in FITS-format.

A widely used response matrix format is NASAs OGIP format. This is used e.g. as the data format for XSPEC. There are a few reasons that we propose *not* to adhere to the OGIP format here, as listed below:

1. The OGIP response file format as it is currently defined does not account for the possibility of response derivatives. As was shown in the previous sections, these derivatives are needed for the optimum binning. Thus, the OGIP format would need to be updated.
2. In the later versions of OGIP new possibilities of defining the energy grid have been introduced that are prone to errors. In particular the possibility of letting grids start at another index than 1 may (and has led) often to errors. The software also becomes unncessarily complicated, having to account for different possible starting indices. Moreover, the splitting into arf and rmf files makes it necessary to check if the indexing in both files is consistent, and also if the indexing in the corresponding spectra is consistent.
3. There are some redundant quantities in the OGIP format, like the `areascal` keyword. When the effective area should be scaled by a given factor, this can be done explicitly in the matrix.
4. As was shown in this work, it is more efficient to do the grouping within the response matrix differently, splitting the matrix into components where each component may have its own energy grid. This is not possible within the present OGIP format.

Proposed response format

I propose to give all response files the extension `.res`, in order not to confuse with the `.rmf` or `.arf` files used within the OGIP format.

The file consists of the mandatory primary array that will remain empty, plus a number of extensions. In principle, we could define an extension for each response component. However, this may result into practical problems. For example, the fitsio-package allows a maximum of 1000 extensions. The documentation of fitsio says that this number can be increased, but that the access time to later extensions in the file may become very long.

In principle we want to allow for data sets with an unlimited number of response components. For example, when a cluster spectrum is analysed in 4 quadrants and 10 radial annuli, one might want to extract the spectrum in 40 detector regions and model the spectrum in 40 sky sectors, resulting in principle in at least 1600 response components (this may be more if the response for each sky sector and detector region has more components).

Therefore I propose to use only three additional and mandatory extensions.

The first extension is a binary table with 4 columns and contains for each component the number of data channels, model energy bins, sky sector number and detector region number (see table below).

The second extension is a binary table with 5 columns and contains for each model energy bin for each component the lower model energy bin boundary (keV), the upper model energy bin boundary (keV), the starting data channel, end data channel and total number of data channels for the response group (see table below).

The third extension is a binary table with 2 columns and contains for each data channel, for each model energy bin for each component the value of the response at the bin center and its derivative with respect to energy (see table below). SI units are mandatory (i.e. m^2 for the response, $\mathrm{m}^2\ \mathrm{keV}^{-1}$ for the response derivative).

First extension to the response file

keyword	description
EXTNAME (=RESP_INDEX)	Contains the basic indices for the components in the form of a binary table
NAXIS1 = 16	There are 16 bytes in one row
NAXIS2 =	This number corresponds to the total number of components (the number of rows in the table)
NSECTOR =	This 4-byte integer is the number of sky sectors used.
NREGION =	This 4-byte integer is the number of detector regions used.
NCOMP =	This 4-byte integer is the totalnumber of response components used (should be equal to NAXIS2).
TFIELDS = 4	The table has 4 columns; all columns are written as 4-byte integers (TFORM=1J)
TTYPE1 = NCHAN	First column contains the number of data channels for each component. Not necessarily the same for all components, but it must agree with the number of data channels as present in the corresponding spectrum file.
TTYPE2 = NEG	Second column contains the number of model energy grid bins for each component. Not necessarily the same for all components.
TTYPE3 = SECTOR	Third column contains the sky sector number as defined by the user for this component. In case of simple spectra, this number should be 1.
TTYPE4 = REGION	Fourth column contains the detector region number as defined by the user for this component. In case of simple spectra, this number should be 1.

Second extension to the response file

keyword	description
EXTNAME (=RESP_COMP)	Binary table with for each row relevant index information for a single energy of a component; stored sequentially, starting at the lowest component and within each component at the lowest energy.
NAXIS1 = 20	There are 20 bytes in one row
NAXIS2 =	This number must be the sum of the number of model energy bins added for all components (the number of rows in the table).
TFIELDS = 5	The table has 5 columns
TTYPE1 = EG1	The lower energy (keV) as a 4-byte real of the relevant model energy bin
TTYPE2 = EG2	The upper energy (keV) as a 4-byte real of the relevant model energy bin
TTYPE3 = IC1	The lowest data channel number (as a 4-byte integer) for which the response at this model energy bin will be given. The response for all data channels below IC1 is zero. Note that IC1 should be at least 1 (i.e. start counting at channel 1!).
TTYPE4 = IC2	The highest data channel number (as a 4-byte integer) for which the response at this model energy bin will be given. The response for all data channels above IC2 is zero.
TTYPE5 = NC	The total number of non-zero response elements for this model energy bin (as a 4-byte integer). NC is redundant, and should equal IC2-IC1+1, but it is convenient to have directly available in order to allocate memory for the response group.

Third extension to the response file

keyword	description
EXTNAME (=RESP_RESP)	Contains the response matrix elements and their derivatives with respect to model energy. The data are stored sequentially, starting at the lowest component, within each component at the lowest energy bin, and within each energy bin at the lowest channel number.
NAXIS1 = 8	There are 8 bytes in one row
NAXIS2 =	This number must be the sum of the NC values, added for all model energy bins and all components (the number of rows in the table).
TFIELDS = 2	The table has 2 columns
TTYPE1 = Response	The response values $R_{ij,0}$, stored as a 4-byte real **in SI units** i.e. in units of m^2.
TTYPE1 = Response_Der	The response derivative values $R_{ij,1}$, stored as a 4-byte real **in SI units** i.e. in units of $m^2\ keV^{-1}$.

Any other information that is not needed for the spectral analysis may be put into additional file extensions, but will be ignored by the spectral analysis program.

Finally I note that the proposed response format is used as the standard format by version 2.0 of the SPEX spectral analysis package.

Proposed spectral file format

There exists also a standard OGIP FITS-type format for spectra. As for the OGIP response file format, this format has a few redundant parameters and not absolutely necessary options.

There is some information that is absolutely necessary to have. In the first place the *net source count rate* S_i (counts/s) and *its statistical uncertainty* ΔS_i (counts/s) are needed. These quantities are used e.g. in a classical χ^2-minimization procedure during spectral fitting.

In some cases the count rate may be rather low; for example, in a thermal plasma model at high energies the source flux is low, resulting in only a few counts per data channel. In such cases it is often desirable to use different statistics for the spectral fitting, for example maximum likelihood fitting. Such methods are often based upon the number of counts; in particular the difference between Poissonian and Gaussian statistics might be taken into account. In order to allow for these situations, also the *exposure time* t_i per data channel is needed. This exposure time needs not to be the same for all data channels; for example, a part of the detector might be switched off during a part of the observation, or the observer might want to use only a part of the data for some reason.

Further, in several situations the spectrum will contain a contribution from background events. The observed spectrum can be corrected for this by subtracting the *background spectrum* B_i scaled at the source position. In cases where one needs to use Poissonian statistics, including the background level, this subtracted background B_i must be available to the spectral fitting program. Also for spectral simulations it is necessary to include the background in the statistics.

The background can be determined in different ways, depending upon the instrument. For the spectrum of a point-source obtained by an imaging detector one could extract the spectrum for example from a circular region, and the background from a surrounding annulus and scale the background to be subtracted using the area fractions. Alternatively, one could take an observation of an *empty* field, and using a similar extraction region as for ther source spectrum, one could use the empty field observation scaled by the exposure times to estimate the background.

The background level B_i may also contain noise, e.g. due to Poissonion noise in the background observation. In some cases (e.g. when the raw background count rate is low) it is sometimes desirable to smooth the background to be subtracted first; in this case the *nominal background uncertainty* ΔB_i no longer has a Poissonian distribution, but its value can nevertheless be determined. In spectral simulations one may account for the background uncertainty by e.g. simply assuming that the square of the background signal-to-noise ratio $(B_i/\Delta B_i)^2$ has a Poissonian distribution.

For spectral simulations, however, one should be able to know the *expected* background. This cannot be derived from the observed (Poissonian) background if by chance a data channel has zero background counts. To alleviate that problem, we have introduced an additional column in the spectral file that represents the exposure ratio of the background region divided by that of the source region, where the exposure is the product of exposure time times extraction area.

Furthermore, there may be systematic calibration uncertainties, for example in the instrumental effective area or in the background subtraction. These systematic errors may be expressed as a fraction of the source count rate (such that the total *systematic source uncertainty* is $\epsilon_{si}S_i$) and/or as a fraction of the subtracted background (such that the total *systematic background uncertainty* is $\epsilon_{bi}B_i$). Again, these systematic errors may vary from data channel to data channel. They should also be treated different than the statistical errors in spectral simulations: they must be applied to the simulated spectra *after* that the statistical errors have been taken into account by drawing random realisations. Also, whenever spectral rebinning is done, the systematic errors should be averaged and applied to the rebinned spectrum: a 10% systematic uncertainty over a given spectral range may not become 1% by just rebinning by a factor of 100, but remains 10%, while a statistical error of 10% becomes 1% after rebinning by a factor of 100.

In the previous sections we have shown how to choose the optimal data binning. The observer may want to rebin further in order to increase the significance of some low-statistics regions in the spectrum, or may want to inspect the unbinned spectrum. Also, during spectral analysis or beforehand the observer may wish to discard certain parts of the spectrum, e.g. data with bad quality or parts of the spectrum that he is not interested in. For these reasons it is also usefull to have the proposed rebinning scheme available in the spectral file.

I propose to add to each data channel three logical flags (either true or false): first, if the data channel is the *first channel* of a group (f_i); next, if the data channel is the *last channel* of a group (l_i); and finally, if the data channel is *to be used* (u_i). A channel may be both first and last of a rebinning group (f_i and l_i both true) in the case of no rebinning. The first data channel $i = 1$ always *must* have f_i true, and the last data channel l_i true. Whenever

there are data channels that are not used (u_i false), the programmer should check that the first data channel after this bin that is used gets f_i true and the last data channel before this bin taht is used gets l_i true. The spectral analysis package needs also to check for these conditions upon reading a data set, and to return an error condition whenever this is violated.

Finally, I propose to add the nominal energies of the data bin boundaries c_{i1} and c_{i2} to the data file. This is very useful if for example the observed spectrum is plotted outside the spectral fitting program. In the OGIP format, this information is contained in the response matrix. I know that sometimes objections are made against the use of these data bin boundaries, expressed as energies. Of course, formally speaking the observed data bin boundaries often do not have energy units; it may be for example a detector voltage, or for grating spectra a detector position. However, given a proper response. However given a corresponding response matrix there is a one-to-one mapping of photon energy to data channel with maximum response, and it is this mapping that needs to be given here. In the case of only a single data channel (e.g. the DS detector of EUVE) one might simply put here the energies corresponding to the FWHM of the response. Another reason to put the data bin boundaries in the spectral file and not in the response file is that the response file might contain several components, all of which relate to the same data bins. And finally, it is impossible to analyse a spectrum without knowing simultaneously the response. Therefore, the spectral analysis program should read the spectrum and response together.

As a last step we must deal with multiple spectra, i.e. spectra of different detector regions that are related through the response matrix. In this case the maximum number of FITS-file extensions of 1000 is a much smaller problem then for the response matrix. It is hard to imagine that anybody might wish to fit more than 1000 spectra simultaneously; but maybe future will prove me to be wrong. An example could be the following. The supernova remnant Cas A has a radius of about 3". With a spatial resolution of 10", this would offer the possibility of analysing XMM-EPIC spectra of 1018 detector regions. At the scale of 10" the responses of neighbouring regions overlap so fitting all spectra simultaneously could be an option.

Therefore it is wise to stay here on the conservative side, and to write the all spectra of different detector regions into one extension. As a result we propose the following spectral file format.

After the null primary array the first extension contains the number of regions for the spectra, as well as a binary table with the number of data channels per region (see table below). This helps to allocate memory for the spectra, that are stored as one continuous block in the second extension (see table below).

First extension to the spectrum file

keyword	description
EXTNAME (=SPEC_REGIONS)	Contains the spectral regions in the form of a binary table
NAXIS1 = 4	There are 4 bytes in one row
NAXIS2 =	This number corresponds to the total number of regions (spectra) contained in the file (the number of rows in the table)
TFIELDS = 1	The table has 1 column, written as 4-byte integer (TFORM=1J).
TTYPE1 = NCHAN	Number of data channels for this spectrum.

Second extension to the spectrum file

keyword	description
EXTNAME (=SPEC_SPECTRUM)	Contains the basic spectral data in the form of a binary table
NAXIS1 = 28	There are 28 bytes in one row
NAXIS2 =	This number corresponds to the total number of data channels as added over all regions (the number of rows in the table)
TFIELDS = 12	The table has 12 columns.
TTYPE1 = Lower_Energy	Nominal lower energy of the data channel c_{i1} in keV; 4-byte real.
TTYPE2 = Upper_Energy	Nominal upper energy of the data channel c_{i2} in keV; 4-byte real.
TTYPE3 = Exposure_Time	Net exposure time t_i in s; 4-byte real.
TTYPE4 = Source_Rate	Background-subtracted source count rate S_i of the data channel in counts/s; 4-byte real.
TTYPE5 = Err_Source_Rate	Background-subtracted source count rate error ΔS_i of the data channel in counts/s (i.e. the statistical error on Source_Rate); 4-byte real.
TTYPE6 = Back_Rate	The background count rate B_i that was subtracted from the raw source count rate in order to get the net source count rate S_i; in counts/s; 4-byte real.
TTYPE7 = Err_Back_Rate	The statistical uncertainty ΔB_i of Back_Rate in counts/s; 4-byte real.
TTYPE8 = Exp_Rate	ratio of the exposures (time $\times$ area) for the background region divided by that of the source region; dimensionless; 4-byte real.
TTYPE9 = Sys_Source	Systematic uncertainty ϵ_{si} as a fraction of the net source count rate S_i; dimensionless; 4-byte real.
TTYPE10 = Sys_Back	Systematic uncertainty ϵ_{bi} as a fraction of the subtracted background count rate B_i; dimensionless; 4-byte real.
TTYPE11 = First	True if it is the first channel of a group; otherwise false. 4-byte logical.
TTYPE12 = Last	True if it is the last channel of a group; otherwise false. 4-byte logical.
TTYPE13 = Used	True if the channel is to be used; otherwise false. 4-byte logical.

8.4 Definition of the micro-turbulent velocity in SPEX

As of SPEX version 3.06.00, we are using a different definition of velocity broadening for emission models. Regarding parameter name, we moved from V_{mic}, or micro-turbulent velocity, to root-mean-square velocity V_{rms}. The two definitions are related to each other as follows:

$V_{mic} = \sqrt{2} V_{rms}$

Warning: The change of definition means that the V_{rms} that you measure with SPEX 3.06 will be a factor of $\sqrt{(2)}$ smaller than the V_{mic} that you previously measured with earlier SPEX versions.

8.4.1 Why change the definition?

The V_{rms} is defined such that it is equivalent to the Gaussian σ, which makes it a more intuitive quantity for most colleagues in the field. This new definition makes the emission models consistent with the velocity broadening parameters in the SPEX absorption models. In addition, this definition is also used by the APEC code in XSPEC. To avoid confusion when results from the two codes are compared, we decided to change the V_{mic} parameter to V_{rms}.

8.4.2 Microturbulent velocity in SPEX versions <=3.05.00

In previous versions of SPEX before 3.06.00 the turbulent velocity was called `vmic`. The micro-turbulent velocity (V_{mic}) was defined following a definition historically used often in spectroscopy. In this definition, the Doppler broadening (ΔE_D) is defined as:

$\Delta E_D = \frac{E_0}{c}\sqrt{\left(\frac{2kT_{ion}}{Am_p} + V_{mic}^2\right)}$

As is shown in Zhuravleva et al. (2012; Section 7.2, Eq. 24), this relation can be written also in terms of Gaussian σ:

$\Delta E_D = \frac{E_0}{c}\sqrt{2(\sigma_{therm}^2 + \sigma_{turb}^2)}$,

where $\sigma_{therm} = \sqrt{\frac{kT}{Am_p}}$ is the thermal line broadening. From these equations, it is easy to see that $V_{mic} = \sqrt{2}\sigma_{turb}$, where we call σ_{turb} the root-mean square (RMS) of the line-of-sight velocity (called *vrms* in the current SPEX version).

8.4.3 Historic note

The definition $V_{mic} = \sqrt{2}\sigma_{turb}$ historically originates from spectroscopy. It was chosen because it made the Maxwellian function and Gaussian function easier to calculate. It may also be related to the definition of the error function (ERF), in which the width is also a factor $\sqrt{2}$ different from the Gaussian σ. Still, SPEX internally calculates the line broadening using the error function which uses the V_{mic} definition, but the number that is reported changed to V_{rms}.

In the early days of MEKAL and SPEX (from the '70s onward), this V_{mic} definition was probably chosen to speed up the calculations. This definition is still used often in theoretical work, so be prepared to convert your measured V_{rms} to ΔE_D or V_{mic} if you want to do a meaningful comparison.

CHAPTER

NINE

EXERCISES

In this section, we list a number of exercises that students can use to learn SPEX. The level slowly increases from basic to more advanced spectral modeling. Before doing the exercises, we advise to read *How to run SPEX* (page 3).

9.1 Powerlaw

Before doing this exercise, we advise you to read *How to run SPEX* (page 3).

The spectrum in the files `powerl.spo` and `powerl.res` was recorded from a source at 6 kpc distance.

1. Load the spectrum into SPEX and plot it.
2. The calibration of the instrument is not very accurate for energies below 0.3 keV and above 10 keV. Ignore those parts of the spectrum.
3. How many data bins do you have in the spectrum? Now try to apply optimal binning to the spectrum. How many bins are left?
4. Set up an absorbed powerlaw model. Do not forget to set the distance to the source.
5. Next step: fit the spectrum. Is it a good fit?
6. Calculate the errors on all free parameters and save your results in a text file.
7. It may be wise to save the commands that you entered in SPEX to a command file, so you can repeat the analysis where needed. Try to make such a command file.

Learning goals:

After having done this spectrum, you should know:

- How to read a spectrum in SPEX (using *Data: read response file and spectrum* (page 83)).
- How to use the basic plot functionalities (using *Plot: Plotting data and models* (page 106)).
- How to ignore parts of the spectrum (using *Ignore: ignoring part of the spectrum* (page 95)).
- How to rebin a spectrum (using *Bin: rebin the spectrum* (page 81)).
- How to set-up a simple spectral model (using *Comp: create, delete and relate spectral components* (page 82)).
- How to set the distance (using *Distance: set the source distance* (page 86)).
- How to do spectral fitting (using *Fit: spectral fitting* (page 91)).
- How to determine errors on parameters (using *Error: Calculate the errors of the fitted parameters* (page 89)).
- How to save your commands & results (using *Log: Making and using command files* (page 100)).

9.2 Powerlaw with a Gaussian line

The files `powgaus.spo` and `powgaus.res` contain an absorbed powerlaw spectrum with a Gaussian line. From an optical observation of the source we know that this source has a redshift of z = 0.0345. There is also Galactic foreground absorption. The source has been observed with the same instrument as exercise 1.

1. Load the spectrum into SPEX, select the proper energy range and rebin the spectrum properly. Set up a model with the right components (a gaussian is added with the command `com gaus`) and fit the spectrum.
2. You may want to fit first with the line flux set to zero (freeze or fix some parameter of the line, make a fit, and then fine-tune the line by releasing the relevant parameters.

   You can find the absorbed and unabsorbed fluxes and luminosity just above the fit statistics. The energy limits can be changed by the command `elim`. To change the energy range over which the fluxes are calculated, type `elim 0.2 10.`. This changes the range to 0.2–10 keV.
3. What is the 2–10 keV luminosity of the source? What is the difference between absorbed and unabsorbed flux? Find out which of the columns in the table (listed above the fit statistic values) is absorbed.
4. What is the energy of the centroid of the line? Calculate the equivalent width of the line (by hand). (equivalent width is the ratio $\frac{F_\ell}{F_c}$, where F_ℓ is the photon flux of the line in unit $\mathrm{photons\ s^{-1}}$ and F_c is the flux per unit of energy of the continuum at the energy of the line in unit $\mathrm{photons\ s^{-1}\ keV^{-1}}$)
5. Calculate the errors on all free parameters. What is the error in the equivalent width you calculated?

Learning goals:

After having done this spectrum, you should know:

- How to make complex spectral models
- How to freeze or thaw parameters (using *Par: Input and output of model parameters* (page 104))
- How to obtain fluxes & luminosities
- How to calculate the equivalent width of a line

9.3 Statistics, binning and more

The files `nustar.spo` and `nustar.res` contains the observed (not simulated!) spectrum of the pulsar PSR J1813-1246 as observed by the NuSTAR satellite. You can adopt a distance of 2.5 kpc to this source. You can assume that the spectrum is a simple power law with Galactic foreground absorption.

1. Load the spectrum into SPEX, and rebin the spectrum properly. No need to ignore any data here.
2. Define the proper model and make a fit.
3. We will make a few alternative fits. Make a table and note down the following numbers for each fit starting with your current one: the best-fit parameters and their error bars of the three free parameters of your model; the correlation between these parameters; and the fit statistic with its expected values; your current fit is model A.

Parameter	Model A	Model B	Model C	Model D
Norm				
Γ				
Luminosity				
corr norm-Γ				
corr norm-n_H				
corr $n_\mathrm{H} - \Gamma$				
C-stat or χ^2				
Exp. C-stat or dof				

4. Same as before, but now use the 3-80 keV luminosity as free parameter instead of the norm (change "type" for the power law model and the status of some parameters). This is model B.
5. As above, but now instead the 3-20 keV luminosity (model C).
6. As above, but now change the fit statistic from C-stat to χ^2 (model D). Be shocked by the bias in fit parameters that you will see!

Learning goals:

After having done this spectrum, you should know:

- How to change from normalisation ot luminosity.
- See the importance of "pivot" point for the luminosity / normalisation.
- Understand the danger of using χ^2-fitting.

9.4 Stellar Spectra

We examine a high resolution spectrum of a nearby star at a distance of 10 pc. The files `corona.spo` and `corona.res` contain the spectrum and instrumental response of this source. This spectrum has been simulated using a RGS-like response. We do not provide real data files, because of their size and because of the time constraints of the course. You can use `plot_rgs.com` for plotting, but it is more instructive and easier to plot manually!

1. Read the data into SPEX and plot it in unit Å. The data ranges roughly from 8 to 38 Å. Hint: you may issue successively the commands `plot set all` and `plot line disp t` to get a connecting line through the data for better visibility.
2. Determine the wavelength of the two strongest lines in the spectrum and use this table to identify them: http://www.sron.nl/~kaastra/leiden2018/line_new.pdf. Can you give a rough estimate of the temperature (in keV) of the corona from the fact that you see these lines? Note: The temperatures listed in the line list are in log Kelvin ($k_b = 8.617 \times 10^{-8}$ keV K^{-1}).
3. Load a model with just one component called CIE (`com cie`) which means Collisional Ionization Equilibrium. Fit the spectrum and calculate the error. Was your temperature estimate right?
4. There is one line that is not fitted correctly. Use the line table from exercise 2 to see which element is emitting this line. Set this element to `thawn` and fit again. Is the fit better?
5. Now you can also free the iron and oxygen abundance and fit the spectrum again. What are the values for the abundance? The values in the fit are relative to solar abundances (more details can be obtained with the command `asc ter 1 1 abun`, see *Ascdump: ascii output of plasma properties* (page 79)). Calculate the errors. Are the fitted abundances consistent with solar abundances?

   These kind of spectra (with a low continuum) often produce wrong abundances because of the low S/N of the continuum and the bias produced by χ^2 statistics (Humphrey, Liu & Buote, 2008). In SPEX we can switch from C-statistics to χ^2 statistics to test this. This is accomplished by issuing the command `fit stat chi` before doing another fit.
6. Fit the spectrum using `fit stat chi`. Are the abundances the same as before? Calculate the errors. Are the new values consistent with solar abundances? Switch back to C-statistics after this step.
7. Fix all abundances and free the ion temperature. Fit the spectrum and calculate the error on the ion temperature. Is it well constrained? Find out what the ion temperature does with the lines for values 1, 10, 100, and 1000 (change the value in the model and calculate the model by typing `calc`. Do not fit!). What happens?
8. Plot the spectrum from 21 Å to 23 Å. These lines are called the oxygen triplet. The lines depend strongly on electron density. Play around with the electron density in the same way as you did with the broadening (ion temperature). What happens with these lines? Calculate the error. Can you give an upper limit for the density of the plasma?

9. There are a lot of lines in this spectrum. You can learn also which lines belong to a certain element by putting its abundance to 0 and calculate. Do this for O, Ne and Fe. If you like, you can also try it out for other elements. Finally, SPEX can also provide a list with all the present ions: `asc ter 1 1 icon`.

Note: SPEX can provide a lot of information about the plasma or an absorber through the command `asc ter`. See the SPEX manual for more output options.

Learning goals:

After having done this spectrum, you should know:

- How to plot a spectrum in wavelength units.
- How to identify spectral lines.
- How to get physical parameters from a thermal X-ray spectrum.
- How to measure abundances from an X-ray spectrum.
- How to measure electron density from an X-ray spectrum.
- How to use the "asc" command in SPEX (*Ascdump: ascii output of plasma properties* (page 79)).

9.5 Supernova remnants

Up to now we have only fitted an object which was in collisional ionization equilibrium (CIE), but when there are plasma shocks in a (low density) medium, equilibrium might not be reached yet. This is often the case in supernova remnants. We will illustrate this with the following spectrum: `nei.spo`. Again the response is the same as `corona.res`. Adopt a source distance of 3 kpc, and fix the Galactic foreground absorption to 5×10^{24} m^{-2}. Define your spectral model.

1. Fit the spectrum with a CIE model. Is the fit acceptable?
2. With the parameter `rt`, which is the ratio between the temperature in ionization balance and spectral temperature, we can obtain a better fit. Set the parameter to thawn, but be aware that this ratio is not allowed to get too close to 0! Is the fit acceptable?
3. In SPEX there is also a component which can fit a non-equilibrium spectrum called `neij`. The most important parameter is U. It is defined as follows: $U = \int_{t_0}^{t_n} n_e dt$. When U is big, it means the ionization is in equilibrium. Fit the spectrum with `neij`. What is the temperature after the shock?
4. Now vary the pre-shock temperature. Does that make any difference?
5. In order to see the effect of Non-Equilibrium Ionisation (NEI), make the parameter "U" of the `neij` model 10 times smaller and 10 times larger than your best-fit value (leave all other parameters the same!), calculate the spectrum using the "calc" command (no fitting here!) and plot the spectrum. You will see large differences.

Learning goals:

After having done this spectrum, you should know:

- How to check your data for Non-Equilibrium Ionisation (NEI) effects by using the parameter RT.
- How to use proper NEI models and get a basic understanding of these spectra.

9.6 Relativistic lines

The file `agnrel.spo` contains a spectrum of an AGN at 1 Mpc distance. The response matrix to be used is `corona.res`. The two features that are visible in the spectrum are the O VIII Lα ($\lambda = 18.969$ Å)and N VII Lα ($\lambda = 24.781$ Å) lines. Because the emission from the disk is influenced by general relativistic effects, the lines appear to be broad and asymmetric. In SPEX we can model these lines by convolving a delta line with a so-called Laor profile (`com laor`) and learn a lot about the geometry of the system.

In this exercise we fix the N_H to 1×10^{20} cm^{-2}. The continuum emission can be well described with a powerlaw. The response file is `corona.res`.

1. First start with the most simple model components: `pow` and Galactic foreground absorption. Plot the data. Hint: it may be useful to plot the flux as counts/m^2/s/Å versus wavelength in Å. Try first to get a reasonable model spectrum for the continuum. You may do this by ignoring the part of the spectrum where you see lines.
2. Set up next the full model by adding lines with `delt` and relativistic broadening by `laor`. Do not forget to use all data again, and it may be wise at this stage to freeze the parameters for the continuum (but do not forget to thaw them for your final fits).
3. Fit the spectrum. Hint: do not thaw all parameters at the same time, but step by step.

   In the `laor` component you can free the parameters `r1`, `r2`, `q` and `i`. You can find the line energies of the oxygen and nitrogen lines also in the SPEX line list. Is the fit acceptable? Look at the residuals of the fit and check wether the lines are well fitted. If not, try to find out which parameter of the `laor` component should be altered.
4. The inner radius of the disk `r1` tells you immediately the spin of the black hole. A non-spinning black hole has $r_1 = 6$, a maximally spinning black hole has $r_1 \to 1$. What can you tell about the spin of the black hole?
5. Vary the parameters `r1`, `r2`, `q` and `i` to see how they influence the line profile (give them manually a different value and issue the "calc" command followed by the "plot" command).

Learning goals:

After having done this spectrum, you should know:

- How to create a complex model with relativistic emission lines.
- The usefullness of freezing parameters or omitting initially parts of the spectrum to get a first-step estimate of the spectrum.
- How X-ray spectra tell you something about spinning black holes.

9.7 AGN winds

In many cases the emission from the central region around the black hole is partly absorbed by the disk and/or wind. In the file `agn.spo` we have a spectrum showing a lot of absorption lines. We set the distance to the source to 1 Mpc and the N_H to 1×10^{20} cm^{-2}. Response file: `corona.res`.

1. This spectrum is too complicated to fit in one run. Therefore start fitting with just an absorbed power-law model to get the slope and normalization right.
2. Identify the absorption lines near 17.7 Å, 18.6 Å, 19.0 Å and 21.6 Å using the SPEX line list.
3. Now add a component called `slab` (*Slab: thin slab absorption model* (page 171)) to your model and free the ions which you identified in 2. You may want to increase the column density per ion to about 10^{20} (Note: the parameter in `slab` is logarithmic). Do the same for the ions C VI and N VII. Is it a good fit?
4. Near 15 Å there are a lot of lines associated with iron. Free the column density of Fe XIV to Fe XVII and fit again. Is the fit acceptable yet?

5. You can obtain a table with the optical depth of all the lines with the following command: `asc ter 1 3 tran` (replace `3` in this command with your component number for `slab`). Write down the optical depth of the O VII and O VIII edge or save the output. Where are the O VII and O VIII edge in the plot? Also save or write down the column densities of O VII and O VIII. You need those later.

6. Remove the `slab` component from your model and add a component called `xabs`, which is a more physical model (*Xabs: photoionised absorption model* (page 177)). Fit the spectrum again.

7. If we zoom in on the lines, we see that they are blueshifted. Fit the parameter `zv`. Provide a reasonable starting value for `zv` first. What is the speed of the absorber?

8. Create the table with optical depths again with the command `asc ter 1 3 tran` and `asc ter 1 3 col` to get the column densities for every ion. Do you see differences with the `slab` model?

9. The ionization parameter `xi` is defined as follows: $\xi = \frac{L_X}{nr^2}$. Determine the luminosity (L_X) of the source (L_X is the luminosity between 1 and 1000 Rydberg, where 1 Rydberg = 13.6 eV) and calculate the density (n) if the wind is at 1 pc from the source. Take care of units and logs.

   Usually the density is estimated from variability arguments, which then allow one to put a limit on the distance of the absorber.

10. The impact of the outflow on its environment can be assessed as follows. Assume that the outflow has constant velocity v (measured from your spectrum). The outflow is not fully spherical. Usually people assume that the solid angle sustained by the outflow $\Omega \sim 1$.

    Then the total mass loss is $\dot{M} = \Omega m_{\rm H} n r^2 v$ and the kinetic energy carried away per second is $L_{\rm K} = \frac{1}{2} v^2 \Omega m_{\rm H} n r^2 v$.

    Calculate the kinetic energy carried away and compare to the ionising luminosity (you should do this offline from SPEX, but use the numbers from your fit). Is the kinetic energy significant compared to the radiated energy?

Learning goals:

After having done this spectrum, you should know:

- How to use models for photo-ionised plasmas
- How you can get more info out of SPEX by using the "asc" commands
- Some basic parameters of black hole winds that can be derived from X-ray spectra

CHAPTER

TEN

CHANGELOG

What is new in SPEX version 3.00 compared to SPEX 2.06? Most of the changes are actually "under the hood" of the SPEX program and are only noticed when model fits are compared, but we also provide useful new model additions. A more detailed overview of the changes in the plasma models can be found in Section *Plasma model in SPEX 3.0* (page 275). The most important changes are summarized here:

- The first version of the new atomic database prepared by Ton Raassen is included in this release. It contains hundreds of thousands of energy levels from hydrogen to zinc. The new database is not yet enabled by default (see command `var calc new`). Using the new line database requires more memory and CPU time then before, which may not be needed in every case. Enable the new line list if you need more accuracy, for example when you fit high-resolution spectra.
- We have optimized the calculation of the radiative recombination, di-electronic recombination, two-photon, and proton excitation processes to be able to calculate a spectrum from the new atomic data within a reasonable time. The results from full calculations were approximated by functions that can be calculated much faster, which allows the fitting of complicated spectra with models containing a large range of physical parameters of the plasma.
- The photo-ionization model `pion` has been substantially improved. It is now possible to model several absorption layers with including their radiative transfer. Also the emission part is implemented. However, use with care since this model is still experimental.
- We included natural & Voigt broadening to SPEX emission models.
- We introduce a new charge exchange model (`cx`) developed by Gu et al. (2016).
- We added the `ebv` component in SPEX to model interstellar extinction.
- There are now interfaces to include your own model in SPEX. Use `com user` for additive models and `com musr` for multiplicative models. Examples of how to use this model component are shown in the revised SPEX Cookbook. It also allows you to call Xspec models from SPEX.
- We added a line emission/absorption model with a Voigt profile (`com line`) to model individual lines.
- Contour plots of χ^2 values from the `steppar` command can now be plotted using the `stepcontour` task. The `steppar` command can now export the results to an ASCII file that `stepcontour` can convert into a quick contour plot (or QDP file).
- The SPEX syntax has been slightly changed. From now on, all ranges can be written with a ':' in between to improve the consistency of the syntax. For example, changing the X axis of a plot can now be written as: `plot rx 0.1:10`. Affected commands are `plot`, `egrid`, `elim`, `dem chireg`, `step axis`, and `par ... range`. If possible, the old syntax was also kept alive. Check the manual for details.
- C-statistics, the Bryans et al. (2009) ionization balance and the proto-solar abundances by Lodders et al. (2009) are now the default in SPEX. The abundance table and the ionisation balance can now be queried using the commands `abun show` and `ibal show`, respectively.

10.1 Version 3.01.00

- To optimize for calculation speed, in the new line list, one can select ions that should be calculated the old way (`ions old`) or the new way (`ions new`). The syntax is the same as the `ions ignore` and `ions use` commands.
- Added sorting option to the command `asc ter ... line` to sort lines based on emissivity.
- In version 3.00.00 (using the new line database, `var calc new`), we had also included dielectronic satellite lines of He-like ions of the type 1s2 => 2s 2p2, which actually are trielectronic recombination lines. The 2s 2p2 levels decay rapidly through radiation of a photon (2p to 1s transition) to the 1s 2s 2p level, which decays further by a similar transition to the 1s2 2s ground state of Li-like iron. Therefore, our model predicts relatively strong satellite lines from these transitions, stronger than present in corresponding EBIT spectra. We have resolved this issue by for this moment omitting transitions from these triply excited states from the calculations. Note they were/are absent in SPEX version 2.0 or all APEC versions.
- The new CIE model did not reproduce the x/y line ratio in He-like iron very well. The x & y lines are the two intercombination lines in this ion. We traced this problem down to some confusion caused by different notations for line levels. The NIST database, the Cowan code, Chianti use different notations for the same transition: some use 1s2s(3S)2p 2P1/2, others 1s2s(1S)2p 2P1/2 for the same transition (same if you compare the values of the corresponding energy levels). Some others couple the 2s and 2p electrons first, leading to P states, and then attach the 1s electron to it, leading to again different notations. This confusion led to an interchange of collisonal and radiative rates between some levels, thereby affecting the x/y line intensities. This has now been fixed.
- The innershell levels and related auger and radiative transitions for Fe XIX to Fe XXIII are added using the data from Palmeri et al. (2003).

10.2 Version 3.02.00

- Updated pion model (details will be described soon).
- Re-installed the tri-electronic recombination transistions (details will be described soon).
- Introduced the SPEXACT naming and versioning for the atomic code and tables in SPEX.

10.3 Version 3.03.00

In SPEX 3.03.00:

- Fixed bug related to opening many fits files.
- Fixed bug in data merge regarding the deallocation of the derivative array.
- Fixed bug in the calculation of the average exposure time in data show.

In SPEXACT 3.03.00:

- Fixed bug in generation of atomic data files.
- Added Auger rates for O V, Ne VII, Fe XIII.
- Ionization limits added for Mg IX, Si XI, S XIII, Ar XV.
- Atomic data extended including autoionisation for Be-like ions.
- Added correction for auto-ionization for excitation and inner-shell ionisation to doubly excited levels.
- Update and bugfix for CX model (H-H collision).
- Updated ionisation balance for U16.

10.4 Version 3.04.00

In SPEX 3.04.00:

- The ionisation balance is now by default Urdampilleta et al. (2017) The actual data did not change, but the paper was published in 2017. The command `ibal u16` still works, but gives a warning message that the actual command is `ibal u17` from now on.
- Fixed bug in stepcontour program related to logarithmic grids.

In SPEXACT 3.04.00:

- Fixed bugs in the CIE, NEIJ, CX and PION models.
- Included radiative recombination cooling following Mao et al. (2017).
- Updates of PION model.
- Updated Auger rates for Be-like to C-like Fe.
- Ni XXI of the O-like added and some Fe ions extended.
- Now we multiply the emission measure by n_e/n:math:_*H* in the pion model. This will result in typically 20% more emission from the pion model.
- Added the tmod option to the pion model (see [sec:pion]), allowing to set the temperature and not solve for energy balance (useful for hot stars, the WHIM etc.).
- Allowed to use multiple solutions in the pion model (fmod and soln parameters).
- Allowed for external heating source in the pion model (exth parameter).
- Extended the pion model with all elements from Z=1 to Z=30.
- Fe XX levels modified to Nist5.
- Bug fix: Now we use the proper argument 2 (y) in the call to the integrated Voigt function sivf. The Lorentzian component is now 2x narrower than before, which does affect the CIE emission spectra. It does not affect "old" xabs, hot calculations etc.
- Extended data for Fe XXI, Fe XXII, and Fe XXIII.

10.5 Version 3.05.00

In SPEX 3.05.00:

- Bug in sector copy fixed.
- Fixed additional page problem in postscript output of plot.
- Introduced W-statistics (not recommended for use).
- Xabsinput can now handle SEDs with more than 1024 bins.
- New version of trafo that adds the Ext_rate column to a .spo file. This column is necessary to properly simulate a spectrum when there is a background spectrum present. The Ext_rate column shows the ratio of the backscales of the source and background spectrum.
- Simulate command syntax changed.
- When calculating errors of multiple parameters with one error command, parameters of the best fit found across all parameters are written to `spex_lower_chi.com`, if a better minimum was found.
- Spectra dumped with `plot adum` now contain a 'NO' to separate the spectra.
- The sector command can now dump a model to a text file that can be read in by the SPEX file model. This could be helpful when one wants to model the background without evaluating the model each time.
- Added correlation information in the output of the `par show` command.

- Added the instrument normalisation to the output of `par write`.
- Replaced a few proprietary math routines with open source alternatives.
- Applied GPL license to SPEX and prepared source code for publication.
- Added first version of SPEX tests to the source code directory.
- Fixed bug in the check of the number of free instrument normalisations in the model.

SPEXACT 3.05.00:

- Bug and stability fixes for the pion model.
- Bugfix in the temperature grid of the cooling-flow model. Mass deposition rates are now consistent with other cooling-flow models.
- Stability issue with the free-bound calculation resolved.
- Fixed small issue in the charge exchange model for Fe XXIII.

10.6 Version 3.06.00

In SPEX 3.06.00:

- The new SPEX manual can now be found at: https://spex-xray.github.io/spex-help/
- Added a Python interface to SPEX (Experimental, see *Python Interface* (page 197))
- User can now control the random seed in simulations (see *Simulate: Simulation of data* (page 113)).
- User can now control the output of the command `par show` (see *Par: Input and output of model parameters* (page 104)).
- Next to Levenberg-Marquardt, we also added the simplex and simulated annealing fit methods (see *Fit: spectral fitting* (page 91)).
- Introduced the `ions mute` and `ions unmute` commands (see *Ion: select ions for the plasma models* (page 96)).
- The help texts in SPEX now contain links to the online documentation (Ctrl-<click> to open in browser).

In SPEXACT 3.06.00:

- Model calculations now in double precision.
- Added new dust extinction models (see *Amol: interstellar dust absorption model* (page 137)).
- Added adiabatic cooling to the pion model.
- Added new cooling by Stofanova et al. (submitted).
- Added cooling by di-electronic recombination.
- The definition of microturbulent velocity in emission models has been changed to Vrms (see *Definition of the micro-turbulent velocity in SPEX* (page 293)).
- The `ebv` model now contains a switch to put the Milky way bump on and off.
- Atomic data:
  - Fe-L data from Gu et al. (2019).
  - O II, Mg VI, S X, Ar XII, Ca XIV, Ti XVI, Cr XVIII
  - Absorption lines of Ne II, Ne III, Mg I and Si I have been updated using Juett et al. (2006) for Ne. The Mg I and Si I have been updated using calculations with FAC and COWAN respectively.

10.7 Version 3.06.01

This SPEX release mainly fixes a few important bugs found after the release of SPEX 3.06.00.

In SPEX 3.06.01:

- Added par_show function to pyspex for use with Jupyter Notebook.
- Added functionality to ascdump, both in SPEX and in pyspex. The ascdump command in SPEX has now the `set range` and `set flux` options to limit the output to a certain energy range or to lines with a minimum flux or optical depth.
- Added a command to remove one (or a couple of) lines from emission spectra. With the `ions mute line` command, a line can be selected for removal from the spectrum. This is especially helpful when a line shows effects that SPEX does not model, but can be fitted with a customized Gaussian/line profile.

In SPEXACT 3.06.01:

- The collisional excitation cross sections for H I at low and intermediate temperatures were too low. After carefull comparison, we have chosen to replace the 1s-nl and 2s-nl transitions (n.le.5) with the ones as given in the Chianti database. That is based on the Anderson et al. 2002 erratum with a sensible extrapolation to higher energies.

  Warning: The update above affects both emitted H I spectra, and the total cooling rate at the lower temperatures (few eV) in the `pion` model. That cooling rate may differ at some temperatures by a factor of 3-4. Therefore results obtained with the most recent SPEX version will differ from those of version 3.05 and earlier.

- Fixed a problem with Li-like Fe-L data (remove high multipole transitions arisen from a bug in FAC).
- Fixed a bug in the NEIJ model which produced wrong ion concentrations from SPEX version 3.05.
- Added pyroxene and DNA to amol model.
- Updates to ions: C I, N II, O II, Ne IV, Cr IV, Cr V, Mn V, Mn VI.

CHAPTER

ELEVEN

CREDITS

We would like to express our special thanks to numerous people who have substantially contributed to the development of various aspects of this work, in alphabetical order:

Frans Alkemade, Gert-Jan Bartelds, Ehud Behar, Alex Blustin, Elisa Costantini, Max Duysens, Jacobo Ebrero, Irma Eggenkamp, Andrzej Fludra, Yan Grange, Ed Gronenschild, Liyi Gu, Theo Gunsing, Jörgen Hansen, Wouter Hartmann, Kurt van der Heyden. Fred Jansen, Jelle Kaastra, Jim Lemen, Duane Liedahl, Junjie Mao, Pasquale Mazzotta, Missagh Mehdipour, Rolf Mewe, Hans Nieuwenhuijzen, Bert van den Oord, Ken Phillips, Ciro Pinto, Ton Raassen, Remco van der Rijst, Makoto Sawada, Hans Schrijver, Karel Schrijver, Tiemen Schut, Janusz Sylwester, Rob Smeets, Katrien Steenbrugge, Jeroen Stil, Igone Urdampilleta, Dima Verner, Jacco Vink, Frank van der Wolf, Piotr Zycki.

During the years the many people have contributed as follows:

11.1 Software

Most software is written by Kaastra. Nieuwenhuijzen contributed to the software design.

De Plaa maintains the software on different platforms.

The philosophy of the fitting procedure is based on the experiences obtained with the fitting of EXOSAT data, especially on the software package ERRFIT designed by Jansen and Kaastra.

Tiemen Schut has initiated the parallelisation of important parts of the code.

This product includes software developed by the University of Chicago, as Operator of Argonne National Laboratory. In particular code from the MINPACK package.

11.2 Plasma model

The line and continuum emission routines were written by Kaastra, based on previous routines by Mewe, Gronenschild, van den Oord, and Lemen (1981-1986).

The original line and continuum emission data are based on the work by Mewe and Kaastra.

Raassen has calculated and compiled a large amount of atomic data for the models.

Nieuwenhuijzen, Hansen, Duysens and Van der Rijst contributed to the extension of the atomic database.

Phillips contributed to improvements of the wavelengths of strong lines.

Liedahl contributed to the Fe-L calculations.

The photo-ionization cross-sections have been provided by Verner. Behar and Raassen contributed to the atomic data for the absorption models.

Missagh Mehdipour helped with the pion model.

Mazzotta and Urdampilleta worked on the ionization balance.

Mao incorporated new radiative recombination data.

Gu developed the charge exchange models.

The non-equilibrium ionization balance routine was developed by Jansen, based on earlier work by Hans Schrijver (1974) and Gronenschild (1981). It was extended and updated together with Sawada (2014).

11.3 SPEX models

The SNR models are based on the work by Kaastra and Jansen (1992), with further extensions and testing by Stil, Eggenkamp and Vink.

Costantini and Pinto contributed to the dust models.

Lemen, Smeets, and Alkemade developed an earlier version (1986-1990) of a spectrum synthesis program.

The original DEM modelling was based on the work by Sylwester and Fludra (1980), while the current DEM regularisation algorithm has been designed by Karel Schrijver and Alkemade. Bartelds contributed to the testing of these models.

Piotr Zycki provided us the *refl* model code.

11.4 Documentation

The first version (1.0) of the documentation has been prepared – with starting help by Gunsing – by van der Wolf and Nieuwenhuijzen. The second version was set-up by Kaastra with contributions from Steenbrugge, van der Heyden and Blustin. The later versions were managed by de Plaa.

The cookbook contains contributions from De Plaa, Ebrero, Grange, Pinto, Kaastra, Steenbrugge, Gu, and Mehdipour.

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

T

U

V

W