Monte Carlo tests using *XSPEC*

Simon Vaughan (Leicester), January 26, 2006

1 Hypothesis testing

Imagine looking at an image and searching for a source at a given position. You measure the flux at the expected position and compare this to a prediction, based on a model of the background perhaps. In order to assess whether this is real detection or not you want to gauge the probability that the flux in the image could be just a chance fluctuation of the background – the false alarm probability. (In the statistics literature a false alarm is known as a "type I error.") If this probability is very small then you would conclude that the detection is solid because it is very improbable that the flux in the image could be caused by a random background fluctuation. This is a hypothesis test. The starting hypothesis (or "null hypothesis") is that there is no source, the flux is entirely due to background. To carry out the test you need to know the probability of getting the observed flux assuming the null hypothesis is true, called the p-value[1]. If this is tiny then the hypothesis is rejected and you need an alternative (e.g. there is a source present). [2]

The tricky part of this is to know the appropriate probability distribution function (PDF) for the test statistic assuming the null hypothesis. This PDF is sometimes known as the reference distribution. From the data you form a statistic – meaning a single quantity derived from the data – such as the flux. But this is just a number. In order to convert this to a probability you need a reference distribution to calibrate the test statistic. By comparing the value of the test statistic you got from the data to the reference distribution you can find the p-value. Often you may be able to find the formula for the appropriate null hypothesis PDF in a textbook. The most used statistics, such as χ^2, have well known PDFs under quite general conditions, and one can find the p-value for a given χ^2 quite easily, e.g. using the formula for $p(\chi^2)$.

This is straightforward so long as you know the reference distribution. If you are dealing with a new or difficult statistic, or are working with a standard statistic but in a situation that does not satisfy the criteria necessary to use the textbook reference distribution, this will not be the case. You can either figure out the formula for the reference distribution (which may be difficult or impossible) or use a Monte Carlo method.

2 Monte Carlo methods

Monte Carlo methods use random (or quasi-random) data to solve problems. In the context of hypothesis testing, one generates randomised data based on the null hypothesis model, taking care to make the fake data as realistic as possible, and uses them as a "control" sample with which to calibrate the test statistic. In the source detection example one would simulate a large ensemble of images, by randomising the background level, and measure the flux at the source position in each one. The frequency distribution (histogram) of the fluxes produced from the fake data is a Monte Carlo estimate of the reference PDF. As well as making sure the data are simulated as accurately as possible one must also simulate a large number of datasets so that the histogram of the test statistics from the simulations converges on the true PDF. Even if there is no analytical expression for the reference distribution, it is always possible to find it using the Monte Carlo method so long as one can simulate a sufficient number of (realistic) fake data.

[1] The probability value or p-value is the probability of observing a value of the test statistic as extreme or more extreme than the value actually observed, given that the null model holds. Small p-values are taken as evidence against the null model: i.e. p-values are used to calibrate tests.

[2] Strictly speaking what we are doing here is calculating the likelihood of obtaining the data assuming the null hypothesis is true. If this likelihood is tiny we have evidence that this assumption is faulty. Only if this more parsimonious model can be soundly rejected (tiny p-value) do we have groupns to favour the more complex model (this is simply following *Occam's razor*).

Monte Carlo methods are extremely powerful and conceptually simple. The drawback is that they may require a large amount of computer processing time to generate and analyse a large quantity of simulated data.

3 Application to the F-test

Maybe you are trying to test for an emission line in a spectrum; adding the line to the model improves the fit a bit, but you don't know whether the improvement should be considered significant or not. You could use the F-test but one of the assumptions behind the F-test is not valid in this case[3]. Normally you would measure the F-statistic and compare this with a reference distribution – this tells you how unexpected your value of F is. (In this case the reference distribution for the F-test is the Fisher-Snedecor distribution.) The reference distribution gives you a probability, or p-value, for the given value of F. If the probability is small (let's say $p = 0.001$) then you conclude this result is unlikely to have occurred by chance, so it must be a significant detection (people often quote this by inverting the false alarm probability: $100 \times [1 - p] = 99.9$ per cent confidence). But, as we just said, the case of adding a line violates one of the fundamental assumptions behind the F-test and so you cannot use the textbook reference distribution to go from an F-value (what you measure from the data) to a p-value (how significant it is). But we can solve the problem using a Monte Carlo approach.

4 An overview of the method

The general idea is as follows. First you need to define exactly what it is you want to test. If you want a clear answer you need a clear question! In the case of line detection, perhaps you are comparing a power law to a power law plus an emission line. The null hypothesis is that the simpler of these two models is true – in this example the null hypothesis is that the spectrum is just a power law. The alternative hypothesis is that the spectrum is a power law plus an emission line. The way you go about making a hypothesis test is to measure some test statistic from the data. Maybe you used the F-test and measured an F-value. (The F-value comes from the decrease in chi-square when you add a line to the model, and the number of degrees of freedom.) But you don't know the reference distribution to turn this into a probability. What you need to do is make a large batch of fake data for which the null hypothesis is true, and measure the same test statistic for each of the fake datasets. So you make a fake dataset, measure the test statistic, make another one, measure the test statistic, etc. etc. If you keep doing this you will build up the distribution of the test statistic assuming the null hypothesis is true (because all your fake data are produced using the null hypothesis). In the "power law vs. power law plus line" example what we do is simulate a spectrum of a power law, then fit the data using a power law with and without a line, and then measure the F-value. Over and over again. As you perform more simulations you build up a clearer picture of the distribution of F-values (if the null hypothesis is true).

You can then ask the question: how many simulations show a larger test statistic (e.g. F-value) than the one I got for my real data? Did the value I got for my test statistic appear in many of the simulations or only very rarely? Maybe the F-value was 6.71 from the real data. And when we ran the $1,000$ simulations we found only 3 out of the $1,000$ had a value bigger than this. We could conclude there is a $3/1000$ chance of getting an F-value like the one observed if the null hypothesis is true. This is the false alarm probability, and since it is quite small we may interpret it as indicating the null hypothesis is false, and we

[3]There are two conditions that must be satisfied for the F-test to follow its expected theoretical reference distribution. These are that the two models being compared are *nested*, and that the null values of the additional parameters are not on the boundary of possible parameter space. This second condition is violated when testing for a line (or any other additive component) because the null value of one of the new parameters (normalisation) is zero, which is the boundary of the parameter space. You should also have enough counts per bin to be able to use χ^2 properly as well. See Protassov et al. (2002).

therefore favour the alternative hypothesis. In other words we could say the line is detected at 99.7 per cent confidence (because $997/1000$ simulations showed a smaller F-value).

5 Outline of a simple MC method

A simple Monte Carlo significance test works along the following lines:

1 Define the null and alternative hypotheses

2 Choose a test statistic: call it T

3 Measure the test statistic of the real data: call it T_0

4 Definite loop. For each $i = 1, 2, \ldots, N$:

  4a Produce a simulated data set: D_i

  4b Measure the test statistic from the simulated data: T_i

5 Calculate where T_0 falls in the distribution of T_i

The p-value is fraction of the T_i values that exceed the measured T_0 value: $p = n[T_i \geq T_0]/N$. Inverting this the significance is $1-p = n[T_0 > T_i]/N$. Ensure N is large or this will not be a very accurate representation (the error on the p value is $\sqrt{p(1-p)/N}$, which comes from the binomial formula).

It is vital that you make a *fair* measurement of the test statistic from the simulated data – you must be careful not to bias this measurement based on your prior experience of the real data. What ever you did to the real data, you must also do to the simulated ("control") data, otherwise you are not performing a fair like-for-like test.

Let me give an example. You think there may be an emission line in your spectrum, but you don't know the redshift so have no prior assumption about where the line could be (anywhere within the energy range of the spectrum is equally valid). You proceed to fit a model without a line, then you fit a model with a line, keeping the line energy free to vary over the entire spectrum. In effect, you searched the entire accessible energy range for a line, and found a possible detection at, let's say 5.5 keV. You then measure the improvement in the fit upon having a line here compared to having no line. This improvement is quantified using your test statistic. Now, when you measure your test statistic in step $4b$ you need to measure it *exactly* as you did the real data, i.e. you must search the entire spectrum for a line and measure the improvement in the fit between the having a line at the new best-fitting line energy and having no line at all. What you must not do is compare the improvement in the fit between having a line at 5.5 keV and having no line. This biases the test: when you looked at the real data you did not only test for a line at 5.5 keV, you looked over the whole spectrum, you should therefore treat the simulated data in exactly the same way. Otherwise you are treating the two sets of the data (real and simulated) differently and you cannot use the simulation results to infer anything about the real data. Richard Feynman demonstrated the problem of confusing *a posteriori* and *a priori* knowledge:

> You know, the most amazing thing happened to me tonight. I was coming here, on the way to the lecture, and I came in through the parking lot. And you won't believe what happened. I saw a car with the license plate ARW 357. Can you imagine? Of all the millions of license plates in the state, what was the chance that I would see that particular one tonight? Amazing!

The trick with all Monte Carlo tests is to make sure you simulate something sensible and do to the simulated data as you originally did to the real data. If you don't generate or analyse your simulations properly, your test will not be valid. Garbage in, garbage out!

6 Script files for *XSPEC*

That's the theory dealt with. Now for a worked example of running a Monte Carlo test using *XSPEC*.

XSPEC will allow you to run a sequence of commands from a script. This means you can automate the process of generating and fitting a large sequence of fake data. If you are trying to examine 1,000 spectral simulations this really is the only way to do it. Basically all you do is write the appropriate commands into an *XSPEC* script file (`*.xcm`) and then you can run it from within *XSPEC*. *XSPEC* uses TCL (Tool Command Language) to control its operation, so you can also add basic control structures (like loops) and input/output commands to your script. In this way an *XSPEC* script with a few TCL commands is quite a powerful tool.

A slight technical problem is that you may need more than just *XSPEC*. If your real data were grouped have have 20 counts per bin, then you must do this to your fake data too. But this needs to be done by GRPPHA – outside of *XSPEC*. What I do is break down the process into a set of basic tasks, each of which has its own script. I use one *XSPEC* script to produce N fake datasets. Then I use a shell script (from the UNIX command line, outside of *XSPEC*) to run GRPPHA on each of the fake data files. Then I have another *XSPEC* script to load each of the (now grouped!) fake data files in turn, fit them, and save the results to a text file. Then I use a Q script, or a Fortran program (or whatever) to examine the results and see how the simulated distribution of the test statistic compares to the 'real' number. That is four scripts in total.

In order to get a feel for how *XSPEC* scripts look and work, Appendix A contains an example script taken from the *XSPEC* web pages. Please spend some time looking at this. Now, it may look a bit frightening, but if you spend some time thinking about it some of the commands should be familiar. Look how the loop is handled, with the:

```
for {set i 1} {$i < 4} {incr i} {
...
}
```

construct. This loops over the commands between the curly brackets, each time for $i = 1, 2, 3$. The data are loaded with the command:

```
  data file$i
```

which loads in the files called `file1.pha, file2.pha, file3.pha` at each cycle round the loop. The central part of the script sets up a model, fits it, and puts the best fitting parameters in a place ready to save. The values are saved using the command:

```
  puts $fileid  "$dp1 $dp2 $dp3"
```

which puts the values of the variables `dp1, dp2` and `dp3` into a line of the text file which itself is labelled `fileid`. Then the model is forgotten and the script returns to the start of the loop ready for another pass. Once all the loops are done it closes the text file and ends. Don't worry too much about the complicated looking bits in the middle. It is usually best to figure these bits out when and if they are needed.

There are a number of ways to run an *XSPEC* script like this. One is to simply use the @ symbol, like you would a normal *XSPEC* `*.xcm` file.

```
xspec>  @script.xcm
```

A better way is to use the following from the UNIX command line

```
unix>  xspec - script.xcm
```

This will start *XSPEC* and run the script. If you end the script with an `exit` command it will then return you to the UNIX command line.

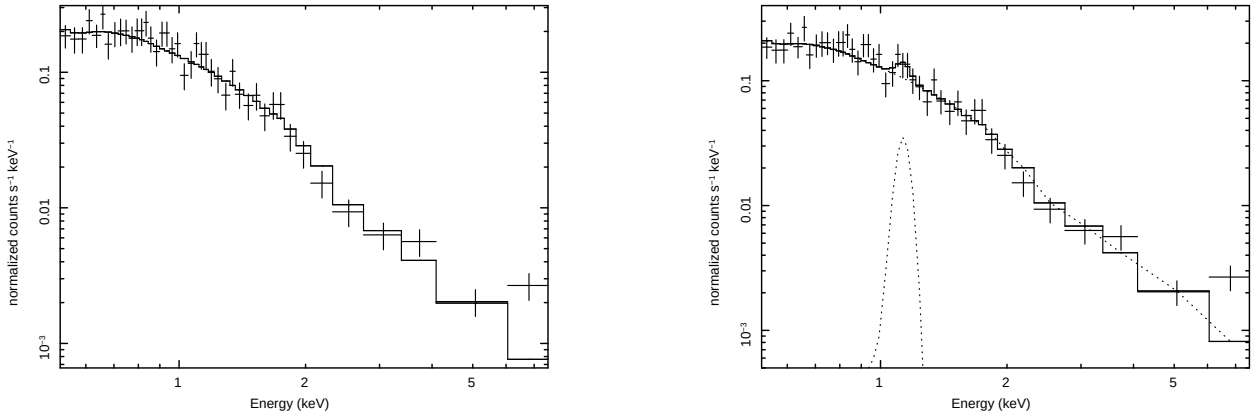

Figure 1: The "real" data of the example. The left hand panel shows the spectrum fitted with the power law model (null hypothesis) the right hand panel shows the same data fitted with the power law plus line (alternative hypothesis).

7 A worked example

Let us try running through an example using some data. All the data and scripts used in the following analysis are available from this address:

`http://www.star.le.ac.uk/~sav2/stats/mc.html`

Figure 1 shows an X-ray spectrum. called `source.pha`. This has been grouped such that each channel contains at least 20 events, and all data below 0.5 keV or above 10 keV have been ignored. There are $N = 56$ good channels used. The simplest expectation is that the spectrum is an absorbed power law, so this is our null hypothesis: call this H_0. The alternative is that there is a narrow emission line, so the alternative hypothesis is that the spectrum is an absorbed power law plus line: call this H_1. This spectrum perfectly well fitted with a simple absorbed power law ($\chi^2_{H0} = 37.64$, with $m_0 = 3$ free parameters, or $D_0 = N - m_0 = 40$ dof), but adding a line improves the fit slightly ($\chi^2_{H1} = 34.59$, with $m_1 = 5$ free parameters, or $D_1 = N - m_1 = 38$ dof). Now, the improvement in the fit is $\Delta\chi^2 = 3.05$ for $\Delta m = m_1 - m_0 = 2$ additional free parameters (or a change of $\Delta D = D_0 - D_1 = 2$ dof). The F-value is:

$$F = \frac{\Delta\chi^2}{\Delta D} / \frac{\chi^2_1}{m_1} = \frac{\chi^2_0 - \chi^2_1}{(D_0 - D_1)(\chi^2_1/D_1)} \tag{1}$$

In this case, plugging in the numbers gives $F = 1.675$. How do we we judge whether this is significant? Using the "textbook" distribution gives $p = 0.200$ But because we are testing for an additive component, an assumption behind the F-test is not valid, and so we should not believe these results (Protassov et al. 2002). We need to run some simulations to work out the actual distribution of F in this case. The implementation I use works as follows:

1. `sim-rand.xcm` – produce simulated spectra
2. `groupall` – group simulated spectra
3. `fit.xcm` – fit simulated spectra
4. `mc_dist.qin` – examine results

The following sections examine each of these in turn.

8 Spectral simulations: `sim.xcm`

The first step is to produce lots of fake spectra drawn from the null hypothesis, i.e. simulate lots of absorbed power laws with the same observational set-up as the real data. In order to produce the simulations we need to know the model (absorbed power law fit) plus the observational details such as the exposure time how we grouped the channels, the energy range used in the fit, and the response matrices. In this case the energy range was $0.5 - 10$ keV and the exposure time was $t = 5000$ sec.

There's an important subtlety to bear in mind when generating the simulations. We could use the parameters from the best fit of the null hypothesis model (e.g. absorbed power law) to generate the fake data. Each fake spectrum would be one random realisation of that model with the same parameters (e.g. absorption column, slope, etc.). But the result then depends on the validity of that model. The final result would be the reference distribution assuming those parameters are **exactly** correct. Clearly, unless our data are fantastically good, we will have some uncertainty in the model parameters, so we should be careful to include this. One simple way is to randomise the model parameters before each simulation, drawing them at random from a Gaussian distribution centred on the best fit, with a width equal to the error bar. (Strictly, if we have more than one free parameter then the distribution will be a multivariate Gaussian and the size and shape come from the covariance matrix.) If the fit is secure we can usually trust the assumption that the probability distribution for the parameters is Gaussian, so randomly picking them from a Gaussian is a reasonable thing to do. Our simulations then sample the probable range of parameter space allowed by the fit to the (original) data. The model randomisation can be accomplished using the `tclout simpars` to generate and output randomised parameters (based on the covariance matrix at the best fit).

The script `sim-rand.xcm` is an *XSPEC* script will generate fake spectra. The first half of the script is a loop that repeats `$nn` times, where `$nn` is a parameter defined near the beginning of the script. The original data is loaded and fitted with the null hypothesis model. Then the first loop begins. Each time round the loop the command:

```
tclout simpars
```

is used to draw one set of randomised model parameters. These are saved to an ASCII file using the line

```
puts $fileid "$xspec_tclout"
```

where `$fileid` is a parameter containing the the ASCII file name, as defined near the beginning of the script. Once this loop is finished we have an ASCII file containing `$nn` sets of randomised model parameters with which to produce the simulated spectra,

The second half of the script is a loop that reads in the model parameters from the ASCII file (produced in the first half of the script) and simulates a spectrum using these parameters. The response matrices (both RMF and ARF) are defined in the lines:

```
proc faker {parm1 parm2} {
  fakeit none & xmmpn.rmf & xmmpn.arf & y &  & ${parm1} & ${parm2} & /*
}
```

which defines exactly how the simulations are done. Normally the command `fakeit` prompts the user for various inputs (background file, response matrix, ancillary response, whether to use Poisson statistics, output prefix, output filename and exposure time) but by defining a procedure that calls `fakeit` we can run it in a non-interactive way. Obviously the response files should be exactly those used for the real data. In this case we are using the `fakeit none` command to produce a fake spectrum with no background ("`none`"). If you need to include a background spectrum then you can do so by inserting the filename in place of the "`none`" in the `fakeit` command. The name of the output file name and the exposure time are input as parameters `par1` and `par2` when this command is called.

The model to be simulated is defined as follows. First the values from the ASCII file are read and given to variables:

```
    set parms [gets $fileid]
    set par1 [lindex $parms 0]
    set par2 [lindex $parms 1]
    set par3 [lindex $parms 2]
```

then these variables are used to define the model parameters:

```
    newpar 1 $par1
    newpar 2 $par2
    newpar 3 $par3
```

The model itself is already defined (retained in memory after fitting the real data).

With the randomised model defined we can use the `faker` procedure to generate one fake spectrum.

```
  faker sim_$i\.fak 5000
```

The input parameters are the file name and the exposure time. The file name is `sim_$i\.fak`, where i is the file number ($i = 1, 2, \ldots$). The exposure time is 5000 (i.e. same as the real observation).

We can run this script in the following way from the command line (outside of *XSPEC*):

```
unix> xspec - sim-rand.xcm
```

The end product is `$nn` fake spectra saved with files names `sim_1.fak`, `sim_2.fak`, etc.

9 Grouping the fake spectra: `groupall`

The fake spectra produced by *XSPEC* are not grouped, whereas the real data we analysed were group to have at least 20 counts per bin. Therefore, do be fair to the fake data, we should apply the same grouping. We can do this using a Bash script called `groupall` which runs from the UNIX/Linux command line and simply calls GRPPHA for each and every file produced by `sim.xcm`. To run it, you first need to make sure it is executable.

```
unix> chmod +x groupall
```

This modifies the file status so it is executable. You only need to do this once. Then you can run it from the command line using:

```
unix> groupall sim
```

where `sim` is the root name of the fake data files (they are all of the form `sim_$i\.fak`. This is just so you can call the files something different if you want to (by changing the line in `sim.xcm`). On some machines you may need to call `groupall` slightly differently, depending on the set-up:

```
unix> ./groupall sim
```

The output filenames are the same, except with a `.g` attached to the end (e.g. `sim_1.fak.g`).

The lines that perform the grouping are:

```
# bin spectrum (until N >= 20 ct/bin)
    grppha infile=$fname outfile=$outp chatter=1 \
     comm="group min 20 & exit" >& grppha.log
```

Obviously if you binned the real spectrum differently you should change this line to bin the fake spectrum in exactly the same way.

10 Fitting the data: `fit.xcm`

Now you should have a set of grouped, fake spectra based on the null hypothesis. Each of these needs to be fitted with both models (H_0 and H_1) and from the resulting fits we need to calculate an F value for each one. Then we will have a distribution of F values.

However, performing automated fitting in *XSPEC* is tricky. Anyone who has spent time fitting spectra by hand knows how fiddly the process can get. One major problem is that the χ^2 minimisation routine can get stuck in local minima, rather than finding the global minimum. We do not have time to spend hours checking every fake spectrum by hand, so the script needs to have a few tricks in it to make sure this is not a serious problem. For simple models this can be done reasonably well. If you are fitting a very complicated model with lots of free parameters then automatic fitting may not be feasible. One trick to stop the fitting getting lost it to make sure the starting parameters are sensible. Also, it is important to keep the initial step size sensibly small (when you enter a parameter in *XSPEC* you enter its value, its step size and the hard & soft minima bounds and hard & soft maximum bounds). If the step size is too large the fitting may get lost quickly.

The script `fit.xcm` is an *XSPEC* script rather like `sim.xcm` except instead of simulating data based on a pre-specified model, it will load the data and fit the model. It performs a loop over $i = 1, 2, \ldots, N$, each time carrying out the following steps:

1 load fake spectrum i

2 Ignore channels as per the real data

3 Null hypothesis (H_0):

- 3a Define null hypothesis model with starting parameters
- 3b Fit the model to the data
- 3c Use the `error` command to 'shake' the fit from local minima
- 3d Record the χ^2_0 and D_0 values

4 Alternative hypothesis (H_1):

- 4a Define alternative hypothesis model with starting parameters
- 4b Fit the model to the data
- 4c Use the `error` command to 'shake' the fit from local minima
- 4d Keep hold of the χ^2_1 and D_1 values

5 Save both sets of χ^2 and D values to a text file

Actually this particular script is slightly more intellegent than this. Adding a line at an arbitrary energy and finding the best-fitting energy is a troublesome exercise. It is difficult to be sure of finding the global best-fit (minimum χ^2). This is made more robust by the way the line is included in the model using the `addline` command as follows:

```
addline 1 gaussian nofit
```

This inserts a narrow Gaussian line at the position of the largest data-model residual. In this way the line starts off in a sensible place in the spectrum, a good initial guess to the global best-fit position (certainly better than inserting the line at some arbitrary energy). The line parameters are then adjusted using:

```
newpar 4,,0.01 0.2 0.5 5.0 10.0
newpar 5,,0.01 0.0 0.0 1.0 1.0
newpar 6,,1e-6 0.0 0.0 1.0 1.0
freeze 5
```

This fixes the step sizes and lower/upper bounds of the line energy, width and normalisation parameters, but leaves their values as they are. The last line freezes the line width at zero so that the line stays narrow. The model is then fitted using

```
fit 100 0.01
```

and as a double check that the parameter solution is stable the script calculates the 90 per cent error bars are some of the parameters using:

```
error stopat 10 0.01 max 10.0 2.706 1-2,4-5
```

Sometimes during the process of calculating errors a better fit is found. A final call of the `fit` command should ensure we're at the global minimum.

Note that because of the amount of fitting involved (actually, fitting, then checking, then fitting again, etc.) this script can take a few minutes per spectrum! This means the time required to run a reasonably large Monte Carlo test is quite significant. It is therefore important to check the whole process works well on a small sample (say 50) then, if everything is working as it should, run it with a much larger sample overnight, or during a weekend.

As with other *XSPEC* scripts, a nice way to run it is from the command line:

```
unix> xspec - fit.xcm
```

After a while the script will finish and you will be left with a text file called `fit_result.dat` with all the important details in. There should be five columns and one row for each simulation. The five rows list the following:

$$i \;\; \chi^2_{H0} \;\; D_{H0} \;\; \chi^2_{H1} \;\; D_{H1} \tag{2}$$

which is everything we need to calculate the distribution of $\Delta\chi^2$ or F.

11 Examining the distribution: `mc_dist.qin`

Now you have all the data you need in a text file, which can be processed using IDL, Fortran, MS Excel or whatever you like. I have a Q script which performs the most obvious tasks. The first lines of the script define the 'real' F-value you are wanting to calibrate (i.e. from the real data). This is written into the script as:

```
; -- F-value from real data
F0=1.675
```

Obviously, you will want to change this if using this script on your own data. With this set you can then run the script from within Q as you normally would. Start Q with:

```
unix> q
```

assuming that Q is initialised correctly. Then run the script:

```
Q> load mc_dist.qin
```

The result is a plot of the cumulative distribution of the F-values from the simulations. Also marked is where the true value falls. See figure 2. The script also outputs a single number, the fraction of F-values from the simulations that are smaller than that of the real data, i.e. $n[F_0 > F_i]/N$. This is the significance level. In this case the $F = 1.675$ value was frequently exceeded. The resulting p-value is 0.581 ± 0.002 based on $1,000$ simulations. Since this is not small we have no evidence to reject the null hypothesis model. We must therefore accept there is no evidence for a line. Job done.

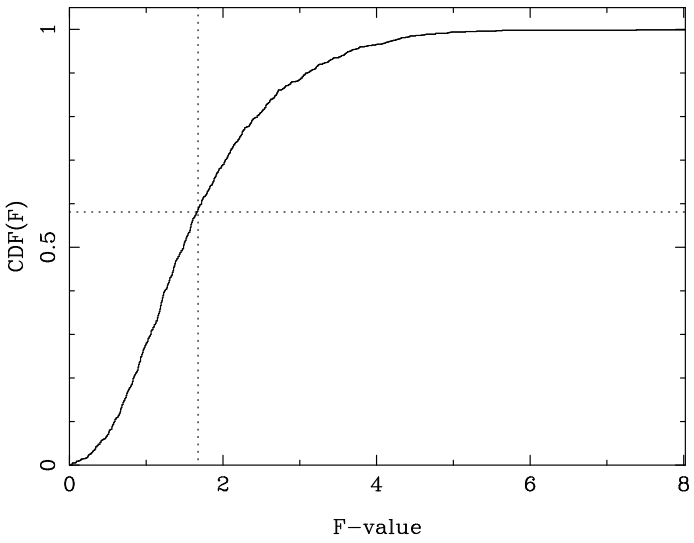

Figure 2: Cumulative distribution of F-values from 1000 simulations. Also marked is the position of the F-value from the real data – this occurs at 58.1 per cent.

References

Protassov R. et al., 2002, ApJ, 571, 545

A Example *XSPEC* script

As an example of an *XSPEC* script using some TCL commands, below is the example script from the *XSPEC* web pages:

```
# This script gives an example of how one might use the power of tcl's
# scripting language in an XSPEC session.  In this example, XSPEC loops
# thru 3 data files (file1, file2 and file3) and fits them each to the
# same model 'wabs(po+ga)'.  After the fit the values of 4 parameters
# for each data set is saved to a file.

# Return TCL results for XSPEC commands.
set xs_return_result 1

# Keep going until fit converges.
query yes

# Open the file to put the results in.
set fileid [open fit_result.dat w]

for {set i 1} {$i < 4} {incr i} {

# Get the file.
  data file$i

# Set up the model.
  mo wabs(po+ga)
  /*

# Fit it to the model.
  fit

log tmp.log
show param
log none
set f [open tmp.log r]
foreach j { 1 2 3 4} {
set reg "^ *$j.* (\[0-9.eE+-\]+) *\$"
seek $f 0
while {! [regexp $reg [gets $f] p dp$j]} {}
}
close $f

# Print out the result to the file. This writes the current version
# of chi-squared and the values of parameters 1, 2, and 4.
# puts $fileid "$i [new 0] [show param 1] [show param 2] [show param 4]"

  puts $fileid  "$dp1 $dp2 $dp3"

# Reset the model.
  model none
}

# Close the file.
close $fileid
```