CORONAL DIAGNOSTIC SPECTROMETER
SoHO

CDS SOFTWARE NOTE No. 47

Version 2 October 1997

The Component Fitting System (CFIT) for IDL

Version 2

S. V. H. Haugan
Institute of Theoretical Astrophysics
University of Oslo

S.V.H.Haugan@astro.uio.no

Contents

1 Introduction

The Component Fitting System is designed to assist anyone using IDL to fit a set of functions to a data set. It is inspired by what I perceive to be the underlying philosophy of XSPEC[1], that the hard part of complex spectral fitting problems should not be to describe the model being fitted, but to find starting values leading to the best fit, and to decide which results are sensible and which aren't.

Instead of writing many specialized IDL functions, one for fitting a Gaussian with linear background, one with two Gaussians and constant background or, say, a Voigt profile plus a cubic background etc. etc., the Component Fitting System is designed to minimize the need for programming altogether. In fact, specifying the components to be fitted should be as simple as listing them up! Then, the user may spend his time worrying about the *correctness* of the results.

1.1 Version 2 enhancements

A number of new features (most of them in XCFIT_BLOCK) have been added in Version 2 – see especially Sections 3.2 and 3.5, as well as Appendix B.

Modified paragraphs in other sections have been flagged with a marginal note (V2), like this paragraph. V2

I've also made it easier to generate one-sigma error estimates in CFIT_BLOCK – use XDOC, look for the SIGMA, MAKE_SIGMA and ERROR_ONLY keywords. Also, look at the program CFIT_ERRDEMO for a test of the estimated sigmas (based on synthetic data).

2 Analyzing single spectra

One of the main programs in the Component Fitting System is XCFIT, an interactive program for designing the model to be fitted. It needs three parameters; two arrays X and Y containing the data to be fitted, and a third parameter containing the component fit structure (CFIT structure).

You should normally also supply an array of statistical *weights* for each point, being inversely proportional to the square of the estimated standard error for each point, through the keyword WEIGHTS.

2.1 A demonstration with dummy data

To start a demonstration run of XCFIT, make sure the three parameters (and the WEIGHTS array) are undefined, i.e.,

```
IDL> delvar,x,y,fit,weights
IDL> xcfit,x,y,fit,weights=weights
```

When the input parameter X is not defined, a dummy spectrum will be created, with a constant background and three Gaussian components, plus noise. An array with weights to be used in the

[1]Part of the XANADU software package, see http:heasarc.gsfc.nasa.gov/docs/xanadu/xanadu.html

fitting procedure is also created. The fit structure will be initialized with a zero order polynomial if it is not defined.

2.1.1 Adding components

Now, move around inside the spectrum/residual plots with the middle mouse button, positioning the focus point close to the center of the *left* emission line.

Select [`Add component`] : [`..showing absolute position.`]

The middle part of the display is updated to show the new component along with the background. The program makes some initial guesses for the fit parameters, and the resulting function is plotted on top of the spectrum. No χ^2 fit has yet been performed with the new component – press [`Redo fit`] to improve the fit, and the `Value` column for each component is updated with the values at the end point of the χ^2 minimization.

Now, it seems like this is a good fit for this particular line and this particular part of the spectrum, so we press [`Use as initial state`] : [`(value -> initial)`] to store the current values as initial values for the two components. The initial values are updated on the screen.

We'll also edit the name of the Gaussian component by pressing the button labeled [`gauss`] – an `XINPUT` dialog box appears and we alter the name to e.g., "Left", since this seems to be the leftmost component.

Now we focus on the emission line on the right. Move to the position of the peak, and add another component here.

We'll see how well this emission feature is matched by a single Gaussian by pressing [`Redo fit`] again. Obviously, the result is not very good, and in fact we got a much more "credible" result with the initial values, revealing the second component of the blend, so we use the [`Reset values`] : [`(initial -> values)`] button to undo the best fit calculation.

Now, move to the peak that's visible in the residual (should be at about 503.1) and add a third Gaussian.

Although the results from the initial value guesses could indicate a fourth component, we'll try [`Redo fit`] once first. We see that the residual now reflects mostly noise, and we can name the last two components e.g., "Center" and "Right".

2.1.2 Hints for analyzing heavily blended spectra

Concentrate on the most clear-cut parts of the spectrum first. Each time you add a component and press [`Redo fit`], use your gut feeling to decide whether the "best fit" is actually good (i.e., whether the component now mathches an actual component in the spectrum). If it looks good, don't hesitate to press [`Use as initial state`]. If it doesn't, *don't* press it. Consider instead adding another component before retrying. This will keep you from straying into a wilderness of misplaced lines in case adding the next component makes the fitting process freak out. You can always reset the values whatever the initial values are.

2.1.3 Leaving out/removing components

To visualize the effect of leaving out one of the components, press [Include: ON] to turn a component off, and see what happens with the resulting fit. If you wish to remove a component permanently, make sure it shows [Include: OFF], and then press [Purge components].

2.1.4 Finishing up

When you're satisfied with a fit model, exit the XCFIT program using [Exit] : [Save changes]. First, however, you should consider pressing [Redo fit] and then use the button [Use as initial state] : [(value -> initial)]. Do this if you suspect the current values to be better suited as initial values for further processing (of other, similar spectra) rather than the initial values that were estimated by XCFIT by guessing.

2.1.5 Re-using the designed fit structure

Now, if you'd like to use the designed fit model inside e.g., a program, use the command:

```
IDL> print_cfit,fit,/program
```

This will print out a series of lines that may be inserted directly into an IDL program to reconstruct the structure describing the fit. The structure is called a *component fit structure* (a CFIT structure). Note that the *current* parameter values stored in FIT will become the new initial values unless you set /INITIAL flag of the PRINT_CFIT routine. The CFIT structure produced by the printed statements may then be applied to other data sets, in a batch job (with routines CFIT or CFIT_BLOCK), or interactively (with e.g., XCFIT or XCFIT_BLOCK).

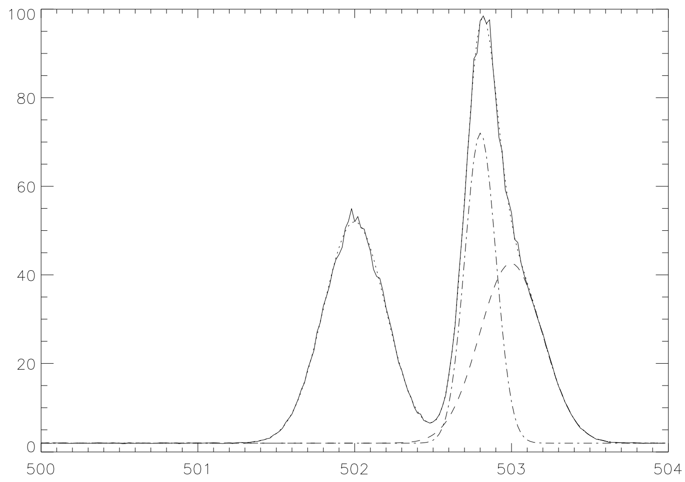
2.1.6 Evaluating the fitted function for plotting etc.

In order to evaluate the function represented by a CFIT structure, use the routine EVAL_SFIT. I.e., after fitting three components to the dummy spectrum and then exiting from XCFIT, do the following:

```
IDL> ps,'cfit-dummy-fit.eps'
IDL>
IDL> help,x,y,fit,/str       ; Just to show you what we've got
X               FLOAT     = Array(200)
Y               FLOAT     = Array(200)
** Structure <4011efa8>, 4 tags, length=2944, refs=1:
   IGAUSS4         STRUCT    -> COMPONENT_STC_3 Array(1)
   IGAUSS3         STRUCT    -> COMPONENT_STC_3 Array(1)
   IGAUSS2         STRUCT    -> COMPONENT_STC_3 Array(1)
   BG              STRUCT    -> COMPONENT_STC_1 Array(1)
IDL> eval_cfit,x,yy,fit
IDL> plot,x,y
IDL> oplot,x,yy,linestyle=1  ; Overplot the fitted function
IDL>
IDL> one_gauss = {gauss:fit.(0), bg:fit.bg} ; New CFIT structure with
IDL> eval_cfit,x,yy,one_gauss               ; only one gaussian plus the
IDL>oplot,x,yy,linestyle=2                 ; background
```

```
IDL>
IDL> one_gauss = {gauss:fit.(1), bg:fit.bg} ; The other gaussian of the
IDL> eval_cfit,x,yy,one_gauss              ; "blend"
IDL> oplot,x,yy,linestyle=3
IDL> psclose
```

and the postscript file looks like this:



2.2 `Xlib` error, IDL process freezes

For some reason, running `XCFIT` sometimes causes `Xlib` errors to occur (at least on DEC Alphas running OSF UNIX 4.0.1) giving error messages along the lines of "`Xlib: unexpected async reply (sequence 0x20f4)!`" or "`Xlib: sequence lost`" or similar. The reason for this is not known, as it has proved difficult to establish exactly what triggers it. The only "cure" seems to be to stop IDL by pressing `^Z` and then kill the process. Killing the patient is not considered a great form of treatment, though.

The frequency of this error has been reduced significantly by inserting a `PRINT` statement (!) in one of the `XCFIT` routines. The error is more easily provoked if you add a component after leaving your display idle for a while. If you find a way to produce this error *consistently*, could you please experiment a little bit more to see if there are any actions that will keep it from happening (like pushing various buttons before doing something else, or perhaps running any other widget program first, etc), and report any results to `s.v.h.haugan@astro.uio.no`.

2.3 *Nominal* vs *actual* parameter values

Notice that the `[Add component]` menu lets you choose between Gaussian components showing the *absolute line position*, or components showing the *velocity* (in km/s) relative to some lab wavelength. In some situations representing the line position by it's velocity makes it easier to asess the feasibility of a result.

This has been accommodated in the Component Fitting System without having two Gaussian "component types" as such, by allowing a linear transformation between a parameter's *nominal* value and it's *actual* value.

Given the two linear transformation coefficients TRANS_A and TRANS_B, the actual value is given by

$$\text{Actual} = \text{Nominal} * \texttt{TRANS\_A} + \texttt{TRANS\_B} \tag{1}$$

The χ^2 fit is always done with respect to the *actual* values, and the "presentation" is normally done in the *nominal* values.

So, implementing a component showing the line positions as a velocity is simply done by setting

$$\begin{aligned} \texttt{TRANS\_A} &= \pm\frac{c}{\lambda_0} \\ \texttt{TRANS\_B} &= \lambda_0 \end{aligned} \tag{2}$$

where the $\pm$ choice seems to be determined by whether you're an observer favouring a plus sign, giving positive velocity when *distance* increases (use [`blueshift <=> negative velocity`]) or a theoretician favouring the minus sign, giving positive velocity when *height* increases ([`blueshift <=> positive velocity`]).

The lab wavelength is initially taken as the position of the focus at the time of adding the component. This is, of course, seldom exactly the correct value, and you'll need to adjust the transformation parameters. TRANS_B is the most sensitive, while in most cases the error in TRANS_A corresponds to an error in c of order one percent or less.

It is of course also possible to measure velocities in furlongs per fortnight if you so desire – simply express the light speed in these units and plug into Equation 2 above.

2.4 Background polynomials of order N

By default, XCFIT uses a constant background (i.e., zero order polynomial) when the input CFIT structure is initially undefined. If you wish to use an Nth order polynomial, just define the CFIT structure before starting XCFIT:

```
IDL> fit = {bg:mk_comp_poly(N)}
IDL> xcfit,x,y,fit
```

The MK_COMP_POLY function can create a polynomial component structure of any order. *Always* use the tag name "BG" for the background – this is to allow automatic handling of the tag in the future. The names of other tags are arbitrary.

If you've already created your fit structure, you can replace the background by using the function REP_TAG_VALUE, e.g.,

```
IDL> fit = rep_tag_value(fit,mk_comp_poly(N),'bg')
```

This will replace the tag BG with a polynomial component structure of order N.

2.5 Fitting broad spectral regions

When analyzing broad spectral regions, it is sometimes better to fit a smaller part of the spectrum at a time, because the number of lines becomes very large, because XCFIT becomes sluggish when dealing with too many data points, or because the background is difficult to fit over a broad spectral range.

There are (at least) two ways of dealing with this. You may divide your data arrays into smaller pieces and analyze them separately. Or, you may set the WEIGHTS array to zero outside the region you're studying at the moment. The first option cures all of the problems. The last option, however, gives you the benefit of being able to see the adjacent parts of the spectrum while leaving the background estimate unaffected by non-fitted lines, but it doesn't speed up the operation of XCFIT.

3 Analyzing blocks of data

Now that we've demonstrated how to analyze single spectra, we'll take a look at how you can analyze blocks of data to produce velocity maps etc.

Given a block of data where the *first* dimension is the dispersion direction, you can use the programs XCFIT_BLOCK and CFIT_BLOCK to design and/or apply a fit to the whole data block in one operation.

3.1 CFIT_BLOCK

Given a block of spectra in an array with dimensions (LAMBDA,X,Y), with corresponding arrays containing the wavelengths, weights etc, this routine applies a component fit to all the individual spectra, yielding a resulting array (PARAMETERS,X,Y).

If the wavelength calibration is constant over the whole data array, the array containing the wavelengths may have only one dimension, of the same size as the first dimension of the data array.

The first dimension of the result will accommodate all parameters (stored consecutively as they appear in the fit structure) and the χ^2 value (actually, it's the *reduced* χ^2 value) of the fit at that point. That is, for a fit with N parameters, the first dimension will have N + 1 elements.

3.1.1 Analyzing simple cases

Let's say wish to create analyze the He I 584.33A line window from a CDS NIS Synoptic observation:

```
IDL> a = readcdsfits("s3920r04")
IDL> xcds_cosmic,a
```

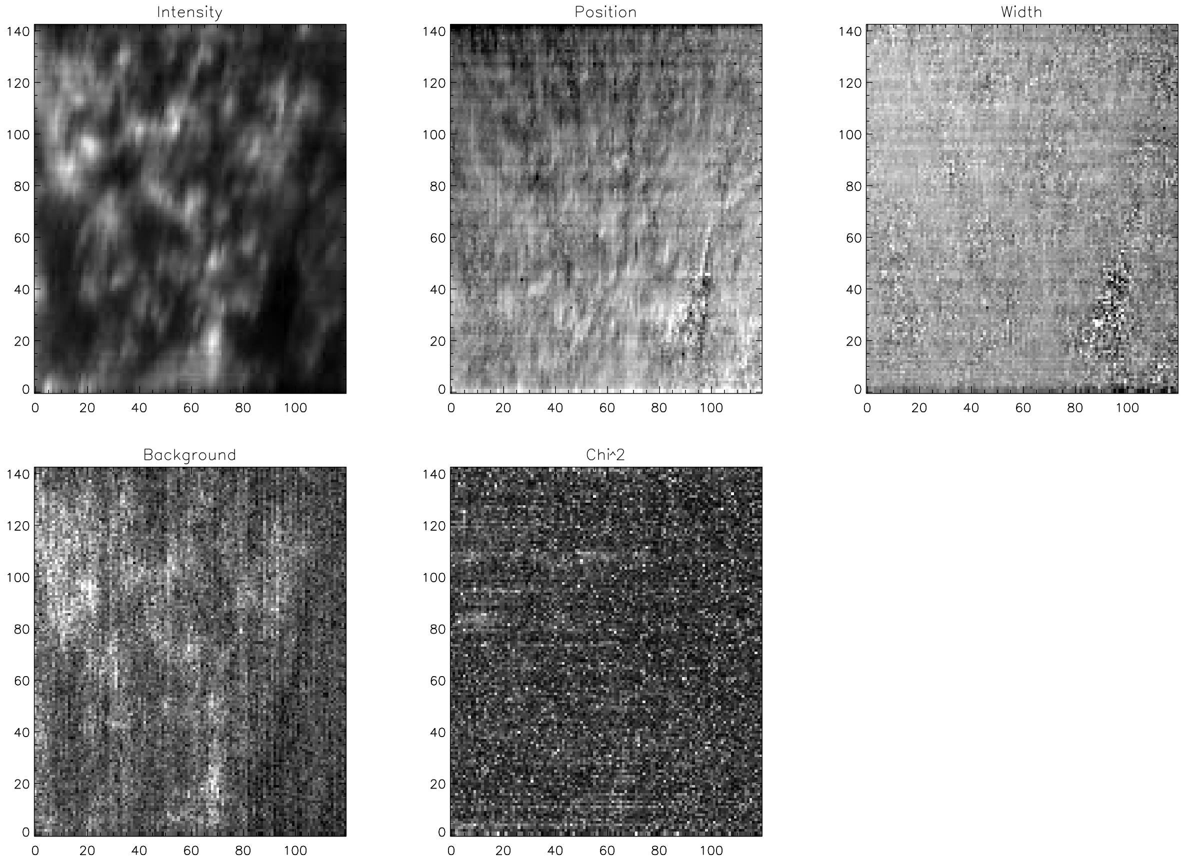
Removing cosmic rays is always a good thing to ensure sensible results. Inside XCDS_COSMIC, execute CDS_CLEAN_EXP automatic CR removal, then manually verify the He I window. There's at least one partially removed CR at exposure index 27, some pixels at exposure index 61, and some at 63. Do not fill the missing pixels. When doing line fitting, filling in pixels (at least in the final analysis) amounts to "inventing" data.

We proceed as follows:

```
IDL> ; Get a sample spectrum and a wavelength array from the He I window
IDL> ; in one line
IDL> sp = gt_spectrum(a,window=0,xix=30,yix=30,lambda=lam)
IDL>
IDL> ; Now, create a fit structure - adding one gaussian (absolute position),
IDL> ; Press Redo fit, then Use as initial state.
IDL> ;
IDL> xcfit,lam,sp,fit
% XCFIT: WARNING: No weights supplied - constant weights used
% Program caused arithmetic error: Floating underflow
IDL>
IDL> ; Below are the statements necessary to reproduce the fit structure
IDL> ; inside a program, or simply to ensure that you'll get the same
IDL> ; result in our demonstration case.
IDL> ;
IDL> print_cfit,fit,/program
% Compiled module: PRINT_CFIT.
IGAUSS2 = mk_comp_gauss([68.1385,584.365,0.543508],$
    max_arr=[100000,584.769,0.995819],min_arr=[0.0001,583.832,0.110647],$
    trans_a=[1,1,0.424661],trans_b=[0,0,0],$
    const=[0b,0b,0b])
IGAUSS2.name = 'gauss'
BG = mk_comp_poly([1.4854],$
    max_arr=[3.40282E+38],min_arr=[-3.40282E+38],$
    trans_a=[1],trans_b=[0],$
    const=[0b])
BG.name = 'Polynomial'
fit = { IGAUSS2 : IGAUSS2,$
        BG : BG}

<The output from print_cfit contains a lot of stuff that's not really
 necessary to our particular example. In fact, the following will do: >

IDL> fit = { igauss2:mk_comp_gauss([68.1385,584.365,0.543508]),$
IDL>         bg:mk_comp_poly([1.485]) }
IDL>
IDL> ; Fetch data array from QLDS
IDL> ;
IDL> da = gt_windata(a,0)
IDL>
IDL> ; Form some kind of statistical weight array - notice that negative
IDL> ; weights would have quite weird implications - the fit would get
IDL> ; better as the "negative" weighted errors get bigger!
IDL> ;
IDL> wts = 1./(da > 1)
IDL>
IDL> ; Now we do the fitting. Notice that the wavelength tilt is not
IDL> ; compensated in this example. -100 is the value for MISSING points.
IDL>
IDL> cfit_block,lam,da,wts,fit,-100,result,residual,/double

<CFIT_BLOCK will inform about it's processing by printing the percentage
 done once in a while. Note that processing can take quite a while -
 maybe this is a good time to get that cup of coffee?>

100%

IDL> help,result,residual
RESULT          FLOAT     = Array(5, 120, 143)
RESIDUAL        FLOAT     = Array(18, 120, 143)

IDL> ps,'cfit-he-1.eps',/encapsulated
IDL> !P.multi = [0,3,2]
IDL> setflag,missing=-100
IDL> plot_image,reform(result(0,*,*)),title='Intensity'
IDL> plot_image,sigrange(reform(result(1,*,*)),frac=.999),title='Position'
IDL> plot_image,sigrange(reform(result(2,*,*)),frac=.999),title='Width'
```

```
IDL> plot_image,sigrange(reform(result(3,*,*)),frac=.999),title='Background'
IDL> plot_image,sigrange(reform(result(4,*,*)),frac=.999),title='Chi^2'
IDL> psclose
```

The resulting postscript file looks as follows:

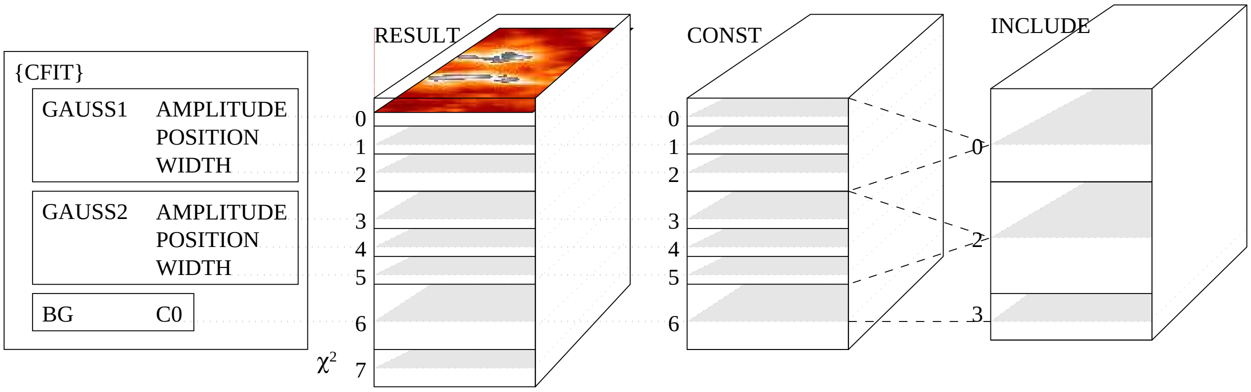


The reason for using SIGRANGE is that at some points in the data structure, the fit may not converge very well, so the results may be off. This is usually caused by cosmic ray hits in positions that make constraining the position/height/width of a line difficult, or by having just too poor signal/noise ratio to constrain a line's position/width parameters reliably. Also, at some positions, there are simply not enough valid data points (due to cosmic ray hits) to make a valid fit.

You may also notice the north-south gradient of the line position caused by the NIS line tilt.

It is possible to supply a "suggestion" result array to CFIT_BLOCK. The supplied results will then be used as initial values for the fit at each point, unless the keyword USE_RESULT is explicitly set to zero. If defined, the RESULT array should have the same size on input as it will have on output, i.e., taking into account that one extra "slot" is used for the χ^2 value.

It is also possible to supply CFIT_BLOCK with arrays controlling where specific components should be turned "off" (INCLUDE), and where specific parameters are to be kept constant (CONST). The first dimension of these arrays should be of size N_COMPONENTS and N_PARAMETERS, respectively. The relationship between the CFIT structure and the RESULT, CONST, and INCLUDE arrays can be illustrated with the following sketch:



3.2 V2: Easy handling – The Analysis Structure `CFIT_ANALYSIS`

With version 2 of the Component Fitting System, the concept of an *analysis structure* (`CFIT_ANALYSIS`) has been introduced. It is a structure containing everything associated with the analysis of a block of data in CFIT (as well as some auxiliary information that's used by e.g., `XCFIT_BLOCK`).

To create an analysis structure containing the data, weights, etc., use e.g.,

```
IDL> ana = mk_analysis(lam,da,wts,fit,-100)

IDL> help,lam,da,wts
LAM             UNDEFINED = <Undefined>
DA              UNDEFINED = <Undefined>
WTS             UNDEFINED = <Undefined>

IDL> help,ana,/str
** Structure CFIT_ANALYSIS, 18 tags, length=128:
   FILENAME        STRING    ''
   DATASOURCE      STRING    ''
   DEFINITION      STRING    ''
   LABEL           STRING    ''
   HISTORY_H       LONG               801
   LAMBDA_H        LONG               802
   DATA_H          LONG               803
   WEIGHTS_H       LONG               804
   FIT_H           LONG               805
   MISSING         DOUBLE         -100.00000
   RESULT_H        LONG               806
   RESIDUAL_H      LONG               807
   INCLUDE_H       LONG               808
   CONST_H         LONG               809
   ORIGIN_H        LONG               810
   SCALE_H         LONG               811
   PHYS_SCALE_H    LONG               812
   DIMNAMES_H      LONG               813
```

Note that the input variables to `MK_ANALYSIS` have become undefined (you may avoid this by setting the keyword `NO_COPY=0` in the call). For more information on the use of `MK_ANALYSIS`, look at the on-line documentation with `XDOC`.

With the analysis structure initialized as above, the call to `CFIT_BLOCK` is considerably less cumbersome than in the previous section:

```
IDL> cfit_block,analysis=ana,/double  ;; Can be shortened, e.g., ...block,ana=ana
```

All the data blocks associated with a CFIT_ANALYSISare stored using handles; to get to the arrays use e.g.,

```
IDL> handle_value,ana.data_h,data       ; Fetch the data cube
IDL> handle_vlaue,ana.result_h,result   ; Fetch the result cube
```

3.2.1 Saving and restoring CFIT_ANALYSIS structures

Since handles are used in the CFIT_ANALYSIS structures, using the standard IDL commands SAVE and RESTORE will not save and restore the arrays associated with the structures.

Instead, you should use the routines SAVE_ANALYSIS and RESTORE_ANALYSIS, e.g.:

```
IDL> ana.filename = 'my_analysis.ana'
IDL> save_analysis,ana
:
:
IDL> ana = restore_analysis('my_analysis.ana')
```

If the CFIT_ANALYSIS tag FILENAME is an empty string, a PICKFILE widget will appear to let the user select a file name. This also happens when RESTORE_ANALYSIS is called without any arguments.

It is possible to restore/revert to the "last saved" version of an analysis structure by using

```
IDL> ana = restore_analysis(ana)
```

3.2.2 Deleting CFIT_ANALYSIS structures

To make sure that the storage space and the handles are freed when throwing away an analysis structure, use DELETE_ANALYSIS to delete it.

3.3 XCFIT_BLOCK

Since χ^2 fitting methods are always "local" in the sense that they'll only find the best fit if the initial values are fairly close to the optimal solution, one cannot expect to ever have any fully automatic method that will work in all cases. The program XCFIT_BLOCK allows the user to view results of the fitting process and manipulate starting values etc at "trouble spots", so as to (perhaps) eliminate all non-sensible results.

With the results from the previous section, start XCFIT_BLOCK with

```
IDL> xcfit_block,lam,da,wts,fit,-100,result,residual,inc,const
```

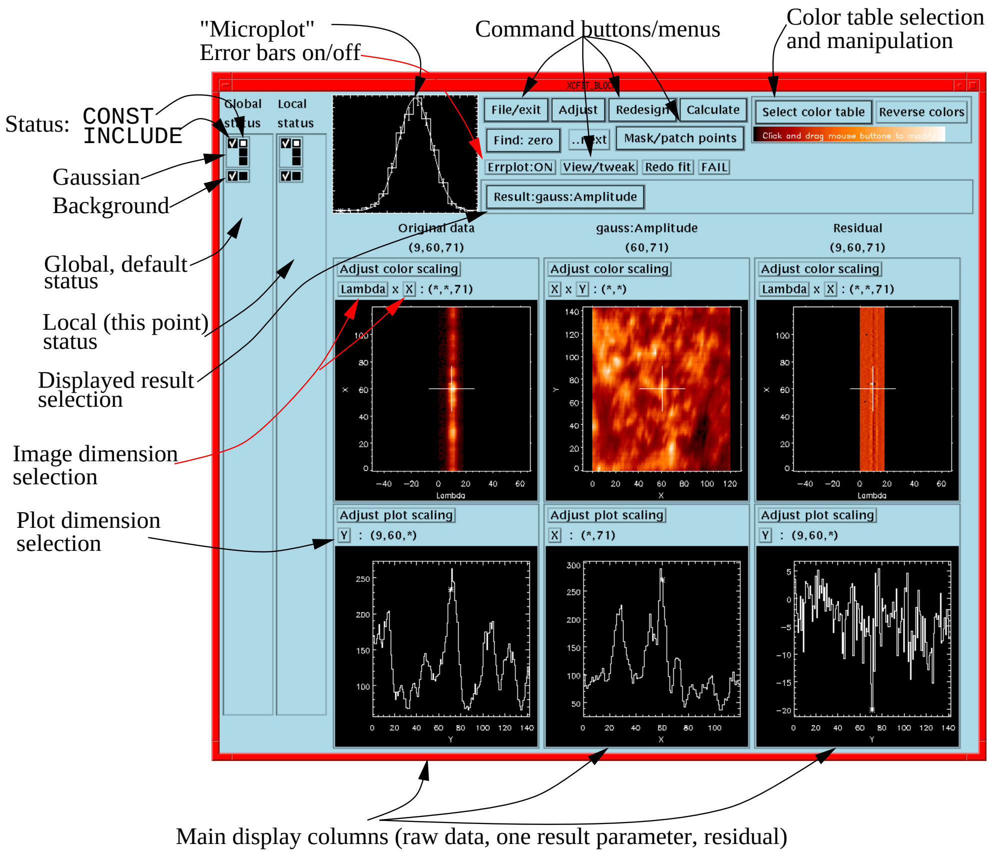
or, with a CFIT_ANALYSIS structure, simply V2

```
IDL> xcfit_block,analysis=ana  ; Can be shortened, e.g., xcfit_block,ana=ana
```

where INC and CONST are used to store information on where specific components are left out and where specific parameter values are to be kept constant.

3.3.1 An overview of XCFIT_BLOCK

When you start XCFIT_BLOCK it looks approximately like this:



The three main display columns in the bottom part of the display are, from left to right, the original data, the fit result (one parameter shown at a time), and the residual data array.

You may view the data cubes in any way you like, try pushing the buttons just above the image displays to change the dimensions currently displayed, or the button just above the profile plots to change the dimension being plotted (image dimension selection/plot dimension selection).

To *move around* in the displayed data, use the middle mouse button to click on the displayed images or plots. To move *one pixel at a time*, click *outside* the image/plot boundary. All the display columns will be focused on the *same physical point* in the data cubes even though the displayed dimensions may vary. To zoom out/in, use the left/right buttons.

To view different result parameters in the middle column, select from the pulldown menu [Result:...].

To the left in the widget are two status columns showing the INCLUDE and CONST values for all the components/parameters. Each group of "buttons" represents one component - a component V2
is included if the checkbox (left) is checked, and a parameter is constant if it's crossed out. You may alter the status by clicking on the buttons.

The left status column shows the global, default values. The right status column reflects the status at the current point - if you move around in the data block, the status will be changed. V2

The CONST box corresponding to the currently displayed result parameter is also highlighted in the status columns on the left. V2

To the left of the command buttons is a "microplot" window, showing the spectrum at the current point, with the currently fitted function on top, as well as error bars. This is a "quick glance" at how well the fitting process has managed to fit the data. You may zoom in/out/refocus with the right/left/middle mouse buttons. The error bars may be turned on and off by pressing the [Errplot:..] button. V2

Some of the command buttons are:

[File/exit] **save/restore options**
The options [Save], [Save as..], [Restore last saved] and [Restore other] deal with saving and restoring the current analysis result through SAVE_ANALYSIS/RESTORE_ANALYSIS. V2

[File] : [View/edit History]
This option allows editing the analysis "history", which is a text array on the handle HISTORY_H in the CFIT_ANALYSIS structure. The history is saved/restored along with the other elements in the analysis structure. V2

[Adjust] : [Adjust (global) MIN/MAX values, names etc]
This button starts XCFIT in the same mode as when you press [View/tweak], but if you alter the MIN/MAX values, or the component names, variable names etc, this will be permanently changed in the fit structure. If you alter the status of the constant flag (CONST) for any parameter, you will be asked whether the new status should be applied to all points in the data cube. Also, you should be aware that if you leave components flagged with non-included components, or with constant parameters, this will be imposed on all the data array points when you do a recalculation from global initial values. You may, however, not add, remove or change the order of any components when selecting this option.

[Adjust] : [Update (global) initial value for...]
This option allows modification of the initial value for the currently displayed result – it may be set to the *median* or the *average* of the current result. Only valid (non-failed), fitted (non-constant) values are taken into consideration. V2

[Redesign] : [Discard all results, redesign fit structure]
Use this button to start XCFIT in the "normal" mode where you can change the CFIT structure by adding, removing (purging), and sorting components. This will, however, leave XCFIT_BLOCK in the blue as to which parts of any calculated results correspond to which components/parameters; Unless you use exit options [Flag as FAILED/IMPOSSIBLE] or [Discard changes] to leave the structure completely unchanged, *all results and residuals will be discarded.*

[Calculate] : [Recalculate based on current result]
This option runs CFIT_BLOCK over your data, using the current RESULT, INCLUDE and CONST arrays as input. Normally, it's quicker to recalculate a fit from current results than to recalculate from global initial values (since the starting points will normally be much closer to the final values).

[Calculate] : [Recalculate from global initial values]
This option runs CFIT_BLOCK over your data, after resetting the current RESULT, INCLUDE and CONST arrays to contain the INITIAL value and INCLUDE/CONST values of the current fit structure. Normally, it's quicker to recalculate a fit from current results than to recalculate from global initial values (since the starting points will normally be much closer to the final values).

[View/tweak]
Pushing this button starts XCFIT, showing the data and the corresponding fit from the current point in the data array. You can modify permanently the INCLUDE and CONST status for any component/parameter for this point. You may also adjust the MIN/MAX limits, INITIAL values etc to circumvent problems with finding a good fit, but these values are not stored individually for each point, and will revert to the global values stored in the original CFIT structure.

[FAIL]

If the fitting process for any reason (like cosmic rays etc) breaks down completely at some point, and no tweaking of initial values etc can produce a good fit, you can declare the fit at this point as FAILED. This will flag the values of all the result parameters with the MISSING value, and at the same time declares all the variables as *constant* at this point. This will also signal to `CFIT_BLOCK` that it should not worry about trying to fit this point again.

[Find: ...]

This pulldown menu (and the [..next] button next to it) is useful for finding "odd" points where V2 some aspect of the fitting process has gone wrong. Selecting e.g., [Find max value] will jump to the point with the highest value for the currently displayed result. Pressing [..next] will jump to the next highest value, etc.

Note that points where `CFIT_BLOCK` failed to converge are marked by a χ^2 value of zero - they are thus easy to find by displaying the χ^2 value and then finding all zero points.

[Mask/patch points]

This pulldown menu provides a way of selecting points in the data block with special characteristics V2 (high or low S/N ratio, many MISSING pixels etc) – and then applying various "patches" to those points. See Section 3.5 for a short discussion – also try selecting [Mask/patch points] : [Edit masking program] from the menu.

3.3.2 Tweaking the results, identifying trouble spots

Returning to our example, select [Result:..] : [gauss] : [Width]. This will make the middle column display the width of the fitted Gaussian. To find extreme values, which often indicate that something is wrong, select [Find: ..] : [Find max value]. The program should now jump to V2 the point (27,43).

Adjust the image dimensions of the original data column to show [Lambda] x [Y]. We see that this data point is positioned at a cosmic ray hit, providing very few points to constrain the Gaussian parameters. Press [View/tweak] to start `XCFIT` with the spectrum taken from this point. It appears difficult to fit this spectrum without having a somewhat high value for the line width. The result is (probably) not valid – some of the pixels included in the fit are probably affected by the cosmic ray. When you exit `XCFIT` you should use the option [Flag as FAILED/IMPOSSIBLE]. This will flag all the results at this point (and the residual) as MISSING.

The result at the current focus point can also be flagged as failed by simply pressing [FAIL], without starting `XCFIT`.

Another option is to make the line position and/or width *constant* at this point – i.e., fixing the width position/width at the global initial value (assuming that this is a good approximation). this may be done either in `XCFIT` or by modifying the local `CONST` status directly in the status V2 columns on the left.

With more complicated line fitting involving multiple components, failures in the fitting process are more common. There are several ways of identifying them.

One method is to select the [Result:..] : [Chi^2] to display the χ^2 results. First of all, if the fitting process fails (as detected by `MCURVEFIT`) at some point, the χ^2 value of that point will be set to *zero*, which is lower than any possible "physical" value. Select [Find: ..] : [Find zero] to V2 find the first point where the χ^2 result is zero (and [..next] to find the next one that's zero, etc).

Try adjusting e.g., initial values so that the fit converges. When you use [`View/tweak`], the initial values are adjusted only temporarily, and for *this particular point* only. When you exit `XCFIT`, only the parameter values and the status of individual components/parameters (INCLUDE and CONST) are stored.

Some times, the min/max values of the line position must be narrowed in to avoid the line jumping outside the spectrum during the line fitting process.

In `XCFIT`, if you're satisfied with the result, use the [`Save changes`] exit option. If you've fumbled the initial values etc and would like to start over, use the [`Discard changes`] exit, and restart `XCFIT`. If it seems like a lost cause for some reason (like CR hits), flag the fit as failed.

Some types of failures cannot be identified by simply looking at the χ^2 values. Some times, like when there's too little signal to constrain a Gaussian, the line position and width ends up with values that makes the line cover only one or two particularly "noisy" pixels. These situations can often be spotted when viewing the Gaussian widths.

The process of identifying and correcting these "misfits" is definitely one of the most time-consuming parts of making useful analysis results. The programs described here will hopefully speed up the process a lot, though.

3.4 Some hints to cope with difficult data sets

"Difficult" data sets typically have

- Close blends, and often at the same time:
- Few pixels/degrees of freedom relative to the number of parameters to be fitted.
- Signs of multiple flows in some places, but not everywhere
- Components that are impossible to ignore in some places but vanishingly small in other places

Some techniques have proven to be useful when working with difficult data sets giving lots of wrong, or nonsensical results.

Try to freeze the line widths and positions of "weak" lines at reasonable values (use [`Adjust`]) by setting the initial values, and switching them to `Fit:OFF`). Ignore the question about whether or not to apply the `CONST` changes. Any answer will do, because the next thing you should do is to recalculate *from global initial values.*

Then, press [`Adjust`] and un-freeze (thaw) the line position. When you exit `XCFIT` after doing that, you should answer *yes* to the question about applying the changes to all points. Now, do a recalculation *from current values.*

Finally, try thawing up the line width as well. This procedure often cures the problem of a Gaussian shrinking in width to "eat up" one or two pixels that are simply above average due to noise. Some times leaving the line width in a frozen state is the best thing – you can't really use the results for anything, anyway, but you avoid "noise" in the maps of the width and position parameters.

In other situations, the best thing is to apply the constraints to your data manually. E.g., wherever a line amplitude is below a fixed threshold, set the width/position to "good" values and freeze them only at those points, redo the fit calculation from current results, then thaw the parameters if you wish etc.

To do such manual operations, the following example may be useful:

```
IDL> xcfit_block,lam,da,wts,fit,-100,result,residual,inc,const
IDL>
IDL> ; We've discovered the problem, and found that the first component
IDL> ; gives nonsensical results for the width/position parameters when
IDL> ; the amplitude goes below about 6
IDL>
IDL> ; Find the points spelling trouble
IDL> ix = where(result(0,*,*,..) lt 6 and result(0,*,*,..) ne -100)
IDL>
IDL> ; Patch those points
IDL> cfit_bpatch,result,ix,1,0    ; velocity set to zero
IDL> cfit_bpatch,result,ix,2,0.56 ; width
IDL> cfit_bpatch,const,ix,1,1b    ; Freeze the velocities at those points
IDL> cfit_bpatch,const,ix,2,1b    ; and the widths
IDL>
IDL> xcfit_block,lam,da,wts,fit,-100,result,residual,inc,const ; Done!
```

Notice that some times blended components switch their identities during the fitting process. This can be detected manually by checking the absolute line positions, and then corrected by swapping all the line parameters for the two lines at those points.

3.5 V2: Easy handling – masking and patching points

With version 2 of the Component Fitting System, patching the result as in the previous section may be done from within `XCFIT_BLOCK`. The options under the [`Mask/patch points`] menu allows you to edit a program defining a mask that includes "special" points in the data sets. Press [`Mask/patch points`] : [`Edit masking program`] to edit the program. When you are editing the program (in `XTEXTEDIT`) you may push the button [`Test program`] to see what points are masked by the current program (they will be flickered).

After editing the program, the mask is calculated, and will only be calculated again when you press [`Re-execute masking program`]. Now, select options from the [`Patch masked points`] menu to manipulate the `RESULT`, `INCLUDE` and `CONST` status for those points.

The patching is done based on the global status stored in the fit structure - you may want to use e.g., [`Adjust`] : [`Adjust (global) MIN/MAX values..`] before actually doing the patching – or modifying the `INCLUDE` or `CONST` value by clicking on the buttons in the status column to the left.

3.6 Working with large data sets

Since recalculations for large data sets often take quite a while, it is often convenient to undersample your data set while testing out various strategies for analyzing the data.

I.e., if you have a 3-dimensional data set, and undersample each "physical" dimension by a factor of two (*not* the wavelength dimension), then the recalculation time for the "working" data set will be one fourth of the original, but you'll still see examples of the most typical problems of this data set.

For one example on how to undersample your data set, see the skeleton program in Appendix A. In some cases you may, of course, want to undersample only one of the dimensions, not all of them.

3.7 Using the ORIGIN and SCALE keywords

To display data cubes with correct wavelengths, positions, pixel sizes, times etc. in the image display section of XCFIT_BLOCK, use the ORIGIN and SCALE keywords. They have the same meaning as in e.g., PLOT_IMAGE, but they must always have as many dimensions as the supplied data array.

Using the SCALE keyword you can get correct pixel sizes in cases where the spatial resolution is different in different directions.

In order to avoid scaling displayed pixels based on e.g., wavelength vs physical position, however, an extra keyword called PHYS_SCALE must be used. If you have e.g., a data set with dimensions (LAMBDA, SOLAR_X, SOLAR_Y) where the SOLAR_X spacing is twice the SOLAR_Y spacing, use SCALE=[0.23,2,1] and PHYS_SCALE=[0b,1b,1b]. This signals that the wavelength scaling factor should not be used against the two other scaling factors.

The ORIGIN, SCALE, and PHYS_SCALE values are supplied through handles ORIGIN_H, SCALE_H V2
and PHYS_SCALE_H in the CFIT_ANALYSIS structure. You may also supply dimension *names* in the form of an array of strings through the handle DIMNAMES_H.

4 Useful procedures

The Component Fitting System involves a large number of functions/procedures, but only a few of them are of relevance to the users. The most important procedures are:

CFIT
: This is the routine used to calculate the best χ^2 fit of the function described by FIT to the data given in (X,Y).

XCFIT
: A "front end" to CFIT(), allowing design and modification of the FIT structure.

CFIT_BLOCK
: A "batch" procedure to apply a component fit to a block of data in one operation. The first dimension of the data array (DA) should be the dispersion direction.

XCFIT_BLOCK
: A "front end" to CFIT_BLOCK visualizing the original data, the results, and the residuals to help identify points where the fitting process does not converge well.

PRINT_CFIT
: Prints out the "current content" of a CFIT structure, with one line per parameter, or as a set of IDL statements to be included in programs.

EVAL_CFIT
: Evaluates the function described by a CFIT structure according to it's current contents.

CFIT_BPATCH
: Shortcut used to manipulate a set of points in RESULT, CONST or INCLUDE arrays. See the example in Section 3.4.

UPDATE_CFIT
: Update the VALUE tags of the component parameters in a CFIT structure from an array of values.

SORT_CFIT
: Sort some of the components in a CFIT structure on the (*actual*) value of a given parameter, in ascending or descending order. Also used to purge non-included components.

MK_COMP_<name>
: Makes a structure for component type <name>.

For a more detailed explanation on the use of each routine, use e.g., XDOC to look at the documentation headers of each routine.

5 Technical information

The Component Fitting System (CFIT) is extremely flexible, allowing any number of components with any number of parameters to be fitted simultaneously. Virtually any type of component can be implemented and used in combination with other types of components.

5.1 Data structures

At the heart of the system is the *component fit structure* (CFIT structure), which has one tag for each component in the fit.

The tags desribing the components are themselves *component structures*, with the following tags (shown for a Gaussian component, which has 3 parameters).

```
** Structure COMPONENT_STC_3, 7 tags, length=840:
   NAME            STRING    'gauss'
   FUNC_NAME       STRING    'comp_gauss'
   FUNC_STRING     STRING    'g'
   MULTIPLICATIVE  BYTE         0
   INCLUDE         BYTE         1
   DESCRIPTION     STRING    Array(10)
   PARAM           STRUCT    -> PARAMETER_STC Array(3)
```

The NAME is the name (nothing to do with it's *type*) of this component, which can be edited by the user (e.g., "He I" etc). The FUNC_NAME is the *name of the function that will evaluate a component of this type*. This function is written to the specifications set forth in the CURVEFIT procedure.

The FUNC_STRING tag has to be a (preferably short) string which is *unique* (one for each component *type*). It is used to compose the name of a procedure that evaluates and adds together the components of a fit.

The tag MULTIPLICATIVE is reserved for future use with multiplicative components (like absorption etc). When nonzero, the value will indicate the *number of preceeding components* that should be multiplied with the results of *this* component.

The INCLUDE tag is set to zero when the component is turned off, i.e., when the result of this component should *not* be added to the sum making up the combined fit.

The DESCRIPTION simply contains a 10-line description of the component.

The tag PARAM contains the specification of each parameter for this component. PARAMETER_STC is as follows:

```
** Structure PARAMETER_STC, 9 tags, length=208:
   NAME            STRING    'Amplitude'
   DESCRIPTION     STRING    Array(10)
   INITIAL         FLOAT          27.9153
   VALUE           FLOAT          78.1525
   CONST           BYTE         0
   MAX_VAL         FLOAT          100000.
   MIN_VAL         FLOAT      0.000100000
   TRANS_A         FLOAT          1.00000
   TRANS_B         FLOAT          0.00000
```

The NAME and DESCRIPTION tags should be obvious. The CONST tag is nonzero when the parameter is kept constant, the other tags are used to store initial values, current values, max/min values (as *nominal* values). The TRANS_A and TRANS_B tags contain the linear transformation coefficients between the *nominal* and *actual* values (see Section 2.3).

5.1.1 The "short fit" (SFIT) structure

A "short fit" (SFIT) structure is an anonymous structure used internally in the Component Fitting System, as an intermediate level in the evaluation of a CFIT structure, to speed up the execution of the fitting procedures. It consists of the following tags:

```
** Structure <400eaee8>, 13 tags, length=168, refs=1:
   COMPILEDFUNC    STRING    'cf_g_p0_'
   COMPILED        BYTE         0
   FUNCTS          STRING    Array(2)
   MULTIP          BYTE      Array(2)
   INCLUDE         BYTE      Array(2)
   N_PARMS         LONG      Array(2)
   A_ACT           FLOAT     Array(4)
   ACT_INITIAL     FLOAT     Array(4)
   TRANS_A         FLOAT     Array(4)
   TRANS_B         FLOAT     Array(4)
   MAX_ARR         FLOAT     Array(4)
   MIN_ARR         FLOAT     Array(4)
   CONST           BYTE      Array(4)
```

The COMPILEDFUNC tag contains the name of the compound IDL procedure that may be used to evaluate the compound fit (if/when the compilation has been performed successfully). The COMPILED tag is 0b if the procedure has not been written and compiled, 1b if the compilation has been done successfully, or 2b if the compilation somehow failed (to avoid futile retries).

5.1.2 Compiled compound procedures (cf_g_...._.pro)

The Component Fitting System includes a procedure (COMPILE_SFIT) that is able to write and compile IDL procedures for evaluation of functions with any combination of components. This is done whenever possible to avoid the overhead of interpreting a structure each time a fit is to be evaluated.

The FUNC_STRING tag in the component structure is used to compose the name of such a compound procedure. Given that the FUNC_STRING for a Gaussian component is "g", and for a zero order polynomial it is "p0", a component fit structure consisting of 3 Gaussians and one zero order polynomial (in that order) corresponds to a procedure called CF_G_G_G_P0_, written according to the specifications of CURVEFIT.

If the environment variable IDL_COMPILE_DIR is set, the procedure will be written to that directory (which should, of course, appear in your !PATH). Otherwise, the procedure will be written in the current directory.

Since no extra information is passed on to these compound procedures, they cannot be used to evaluate a fit with one or more components turned off (i.e., INCLUDE=0). In such cases, (or in cases where compilation does not succeed), the function EVAL_SFIT is used to evaluate the fit, based on the contents of the corresponding SFIT structure, for a given set of parameter values. The loss of speed is on the order of ten percent.

5.2 Implementing new components

As mentioned previously, it's fairly easy to add new types of components to the Component Fitting System, and to mix them with other component types in any way you like.

Let's say we want to add a component type VOIGT, i.e., a line with a Voigt profile instead of a simple Gaussian – taking four parameters. To implement the new component type, two steps are required:

First, write a procedure that will evaluate a component of this type. The name for this procedure should be COMP_VOIGT. It should follow the requirements of any CURVEFIT "function". I.e., it should be a procedure accepting parameters X, A, F, and an optional parameter PDER. See the CURVEFIT documentation for further information.

Second, create a procedure called MK_COMP_VOIGT that creates a *component structure* describing a Voigt component. MK_COMP_VOIGT may use the routine MK_COMPONENT_STC to create the basic structure (with 4 component parameters). All MK_COMP_<cname> functions should accept (at least) the following parameter and keywords:

VALUE: The (only) parameter, an array containing the *nominal* values for all the parameters.

TRANS_A: The linear transformation A coefficients for all parameters.

TRANS_B: The linear transformation B coefficients for all parameters.

MAX_ARR: The array of maximum *nominal* values.

MAX_ARR: The array of minimum *nominal* values.

CONST: An array with the CONST status for all the parameters.

The supplied values (if present) should be inserted into the component structure.

Also, the routine should initialize the tags FUNC_NAME ("comp_voigt") and FUNC_STRING (e.g., "v" for a Voigt profile). Likewise, it should insert clear DESCRIPTIONs of the component as a whole, and each of the parameters, so that there is no confusion as to the exact meaning of the parameter values (include e.g., assumptions about units etc if necessary, or a formula showing how the parameters are used.)

When this is done, the new component type is ready for use. Note that this includes the use of e.g., PRINT_CFIT,..,/PROGRAM, as long as the function naming conventions are followed. Some modifications of the XCFIT program will be needed, however, to allow addition of the new component type through the [Add component] menu.

6 Concluding remarks about χ^2 fitting.

Having presented a package for χ^2 fitting, this would be a good place to state some of the requirements that should be fulfilled in order to take the results of a χ^2 fit at face value.

First of all, it is assumed that all your data points y_i contain "true" values $y(x_i)$ (for a given set of model parameters), plus Gaussian noise with a given standard deviation σ_i (which may vary from point to point, of course). Under this assumption, the result of a χ^2 fit gives you the *most likely* values of the model parameters, as long as the global χ^2 minimum has been found. So far so good.

But these assumptions lead to the following conclusions: If you've found the "correct" model for your data, you should be able to retrieve an estimate of the gaussian noise by subtracting the model from your data. I.e., the residuals $r_i = y(x_i) - y_i$ should be *featureless*, with no systematic structure other than a varying amplitude, reflecting the variations in the standard deviation of the noise. The normalized residuals, $n_i = \frac{r_i}{\sigma_i}$ should, strictly speaking, be "white" noise with an overall standard deviation of one.

A Example skeleton program for analyzing CDS NIS data

Note: This method is now outdated, but it is included for "educational purposes" – see Appendix B for a better method

Below is an example program which is quite useful as a skeleton for an analysis "session". It is a "main" program in it's own right, to use it (assuming it's stored under the file name setupan.pro) simply write

```
IDL> a=readcdsfits(<filename>)
IDL> xcds_cosmic,a                 ; Remove CR
IDL> win=0                         ; Analyze first window in a
IDL> .run setupan
IDL> xcfit_block,lam,da,wts,fit,-100,result,residual,include,const
```

and that's it. The variables LAM, DA and WTS are initialized, and the other variables are left undefined. The program is available at: http://www.uio.no/~steinhh/setupan.pro.

```
; http://www.uio.no/~steinhh/setupan.pro
;
; Extract data from QLDS called "a". Assumed to be debiased and
; cleaned for cosmic rays
;
; "win" must contain the window number of the window to be analyzed
; "sub" may contain the downsampling factor

;
; Routine for downsampling (up to 4-dimensional) data blocks
;
PRO downsamp,da,sub,missing
  sz = size(da)

; Final sizes of 2nd, 3rd (4th) dimension after resampling

  fsz = (sz(2:sz(0))/sub) > 1

  IF sub GT 1 THEN BEGIN

     ;; Subsample FIRST dimension by averaging

     ix = lindgen(fsz(0))*sub
     dad = [da(*,ix,*,*),da(*,ix+1,*,*)] ;; Sub *must* be > 1...
     FOR ii = 2,sub-1 DO dad = [dad,da(*,ix+ii,*,*)]

     dad = dimreform(dad,[sz(1),sub,fsz(0),sz(3:sz(0))])
     da = average(dad,2,missing=missing)

     ;; Subsample SECOND dimension by averaging
     iy = lindgen(fsz(1))*sub
     dad = [da(*,*,iy,*),da(*,*,iy+1,*)]
     FOR ii = 2,sub-1 DO dad = [dad,da(*,*,iy+ii,*)]

     IF sz(0) EQ 4 THEN dad = reform(dad,sz(1),sub,fsz(0),fsz(1),sz(4)) $
     else dad = reform(dad,sz(1),sub,fsz(0),fsz(1))

     da = average(dad,2,missing=missing)

     ;; Subsample DEL_TIME dimension if present

     if sz(0) eq 4 then begin
        it = lindgen(fsz(2))*sub
        dad = [da(*,*,*,it),da(*,*,*,it+1)]
        FOR ii = 2,sub-1 DO dad = [dad,da(*,*,*,it+ii)]
```

```
        dad = reform(dad,sz(1),sub,fsz(0),fsz(1),fsz(2))
        da = average(dad,2,missing=missing)
     end

  END
END

if n_elements(win) ne 1 then message,"WIN must contain window number"

da = gt_windata(a,win)

;
; This is the sampling factor (set to e.g., 2 when testing)
;
if n_elements(sub) ne 1 then sub = 1

downsamp,da,sub,-100

;
; The calculation of weights have to be modified for calibrated data
;
wts = 1./(da > 1)

;; Assuming "a" contains non-calibrated data, the following lines
;; may be used to analyse calibrated data, with correct weights except
;; for an overall scaling factor.
;

IF keyword_set(calibrate) THEN BEGIN
   print,"Using calibrated data"
   aa = copy_qlds(a)

   vds_calib,aa

   med_uncalib = median(da(where(da NE -100)))

   da = gt_windata(aa,win)
   delete_qlds,aa

   downsamp,da,sub,-100
   med_calib = median(da(where(da NE -100)))
   print,"Calibration factor:"+trim(med_calib/med_uncalib)
END


sz = size(da)

if sz(0) eq 4 then begin
   detx = a.detdesc(win).detx + dimrebin(lindgen(sz(1),1,1,1),sz(1:4))
   dety = a.detdesc(win).dety + dimrebin(lindgen(1,1,sz(3),1)*sub,sz(1:4))
end else begin
   detx = a.detdesc(win).detx + dimrebin(lindgen(sz(1),1,1),sz(1:3))
   dety = a.detdesc(win).dety + dimrebin(lindgen(1,1,sz(3))*sub,sz(1:3))
end

;; Which detector are we using?

det = (['N1','N2'])(a.detdesc(win).dety LT 512)

; Calculate tilt for all pixels - values taken from TILT_NIS1_DEMO and
; TILT_NIS2_DEMO

IF det EQ 'N1' THEN tilt = 0.0075942d - 6.135603e-6*detx $
ELSE                tilt = 0.00397 - 4.763135e-6*detx + 4.764016e-9 * detx^2

;; Find the median detector y position - use as zero point for tilt correction
```

```
med_dety = median(dety(0,0,*))

ndetx = detx + (dety-med_dety) * tilt

lam = float(pix2wave(det,ndetx,/nolimit))

END
```

B V2: Analyzing CDS data

In order to extract and analyze a data window from a CDS QLDS called "a", simply use the following statements:

```
IDL> ana = mk_cds_analysis(a,window_i [,downsampling] [CALIBRATION FLAGS])
IDL> xcds_analysis,ana=ana
```

and simply start building the fit structure from scratch. See XDOC,'MK_CDS_ANALYSIS' for a description of how various calibration flags are applied.

Another option is to use XCDS_ANALYSIS, which lets the user define a CDS "Analysis Definition" (ADEF) – allowing multiple windows to be analyzed as one block, (semi-)automatic cosmic ray handling, etc. The calling sequence is:

```
IDL> xcds_analysis,adef,ana,qlds=a
```

XCDS_ANALYSIS may be invoked from DSP_MENU as well.

Once an ADEF has been designed, it may be applied to other data sets simply by referring to them by their fits file names, e.g.,

```
IDL> ana = apply_cds_adef(adef,'s4747r00')
```

(assuming, of course, that READCDSFITS is able to find the fits file in question).

Analysis definitions use handles, and should therefore be saved/restored/deleted by using SAVE_CDS_ADEF, RESTORE_CDS_ADEF, and DELETE_CDS_ADEF respectively.

C Future enhancements

This section is more or less a list of "things to do" for the future, included here so that anyone willing to spend time making improvements to the Component Fitting System knows at least some of my intentions. If anyone has further ideas, or would like to do some work, please contact me.

In no particular order:

- It would be nice to have *visual* warnings in XCFIT when max/min limits are touched
- Automatic undersampling on/off in XCFIT_BLOCK – would also have option to CONGRID the undersampled result to full size, and then recalculate from those values.
- Multiplicative components. As it is, the Component Fitting System only encompasses additive components (although components may have negative values, of course). But it would be nice to have true "absorption" components, multiplying *emission* components with a value between 1 and 0.

  This is possible to implement by using the tag MULTIPLICATIVE as described in Section 5.1. Requires changes in EVAL_SFIT as well as COMPILE_SFIT.
- The speed of the fitting can be improved significantly if some clever method of making initial guesses could be implemented (like guessing the background level, then the amplitude of the lines). Also, the robustness of the fitting routine would improve.
- Automatic application of limits (on χ^2) for good vs bad fits, and applying both fixed and intelligent "guesses" as initial values, then keeping the best result. Maybe also automatic variation of initial values at trouble spots, "stepping through" from min to max values etc, to improve robustness for batch application.
- Automatic detection of unconstrained components due to e.g., cosmic rays etc, invalidating the results at that point.