Module 4

Modelling and analysing special spectral features – A unified approach

Lecture viewgraphs

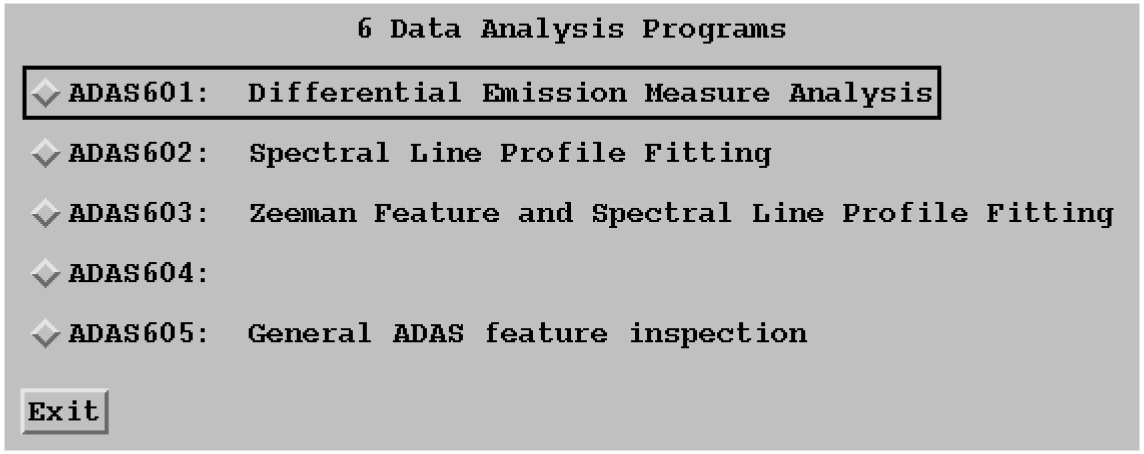
Hugh Summers, Christopher Nicholas, Andy Meigs, Martin O'Mullane and Alessandra Giunta

University of Strathclyde

Contents

1.1 Preliminaries

Spectral fitting and analysis is part of the capabilities of all fusion laboratories. It is also a natural adjunct to an atomic data and analysis structure. ADAS, at an early stage in its development, added some spectral analysis codes in series 6.


6 Data Analysis Programs
ADAS601: Differential Emission Measure Analysis
ADAS602: Spectral Line Profile Fitting
ADAS603: Zeeman Feature and Spectral Line Profile Fitting
ADAS604:
ADAS605: General ADAS feature inspection
Exit

Basic analysis of spectra by fitting the individual line profiles and background in a spectral interval is the minimum provided in ADAS by ADAS602. Also, a dedicated code, ADAS603, for handling Zeeman features was prepared built on the *xPaschen* code of IPR Forschungszentrum Juelich.

At JET, some spectral intervals of special importance and/or complexity became the focus of dedicated fitting and analysis methods (e.g. CXSFIT). Some, to which ADAS developers have contributed, are maintained as independent codes by ADAS.

1.2 Preliminaries

In the planning the principal themes for ADAS-EU, it was decided to develop a general approach to spectral fitting, closely linked to the atomic population modelling capabilities of ADAS, including complex composite features.

ADAS and its databases primarily operate as a forward modelling system, establishing populations of ions, ionisation state, emissivities etc. as a function of key plasma parameters such as T_e, N_e, magnetic field, beam energy and so on.

By contrast, spectral analysis is usually a reverse analysis system, fitting amplitude, width and shape of spectrum lines individually and independently of their origin. These extractions are later examined and contrasted with predictions.

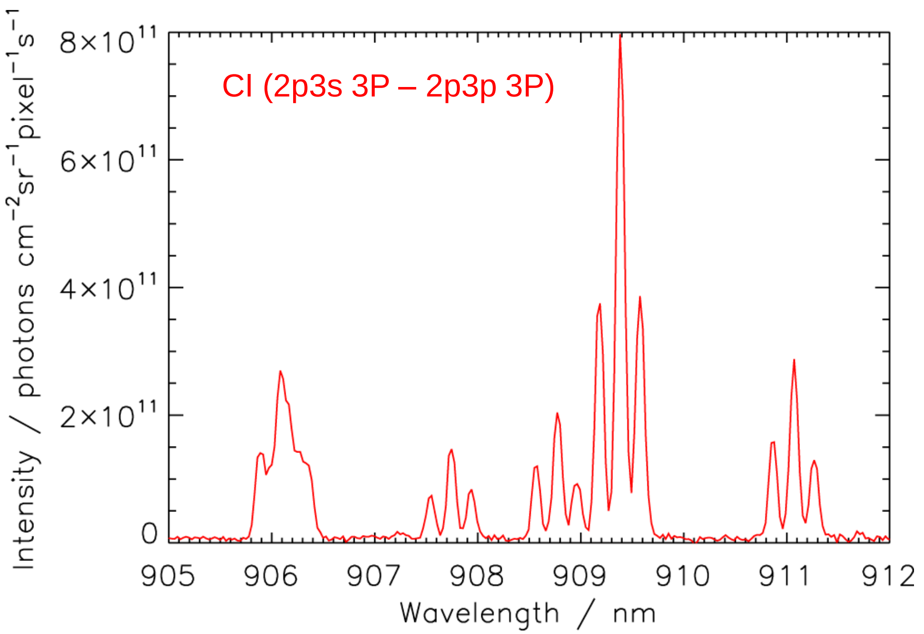
A derived data format of ADAS, such as an *adf15* dataset for an ion, is (or closely related to) an abstract numerical representation of the relative intensities of a whole set of spectrum lines as a function of the plasma parameters, which are the ultimate objective of reverse spectral analysis. With the *adf40* feature emissivity coefficients for complex heavy ions, this is even more obvious. Equally, this is true of ADAS codes such as ADAS305.

We view these ADAS datasets and ADAS codes as special spectral features or more exactly as ADAS special feature generators, which can be used as a whole in spectral analysis for the direct extraction of plasma parameters in the fitting process.

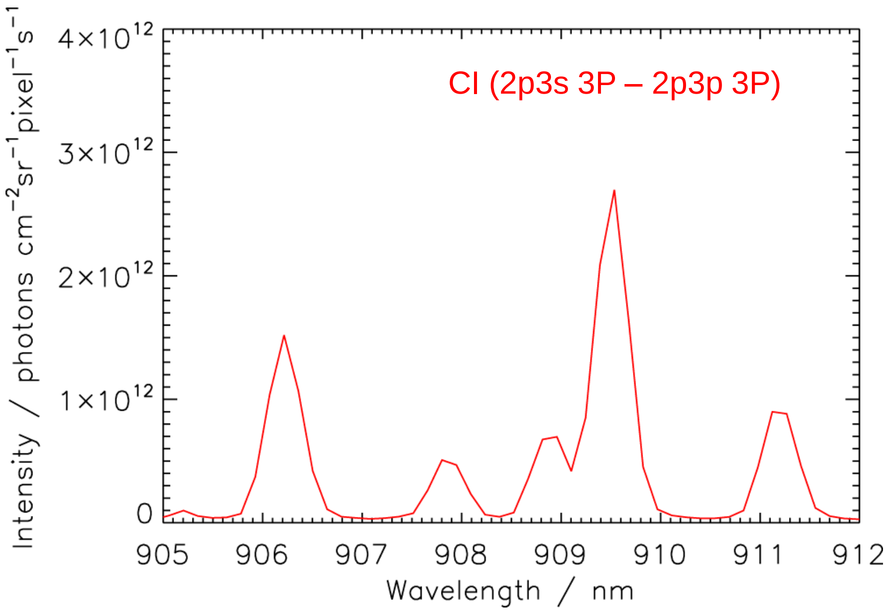
1.3 Preliminaries

The development is quite complex in design, using different coding techniques (object-oriented) from the remainder of ADAS. Also, the whole system is still in the final stages of completion and incorporation.

In this brief introduction, the Zeeman spectra below from JET will be used illustratively.



Zeeman split carbon line emission, from JET pulse #75989 (R = 2:8 m, t = 46:25 s, diagnostic KT3C using 1200 lines mm^{-1} grating) recorded at sufficient resolution to resolve several component lines of the feature.



Zeeman split carbon line emission, from JET pulse #70574, (R = 2:68 m, t = 60:28 s, diagnostic KT3C using 300 lines mm^{-1} grating) at lower resolution. Individual components are no longer resolved, instead they appear blended together.

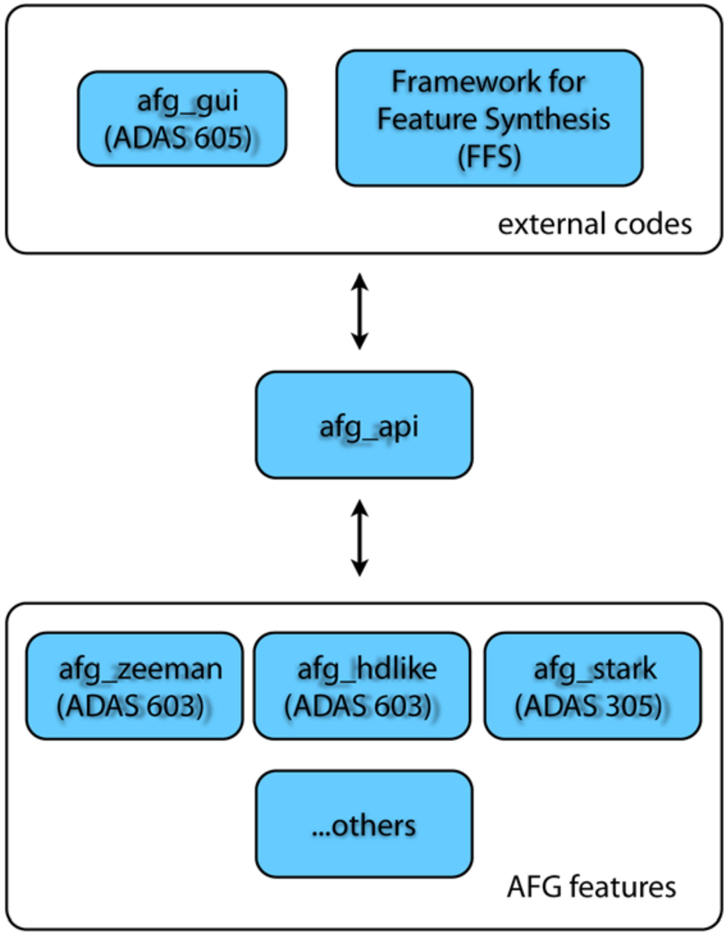
2.1 Accessing ADAS special features - AFG

The main drive behind the *ADAS Feature Generator* (*AFG*) is to ease access to the ADAS special feature routines such that they are easily incorporated into any external modelling code (see *FFS* later). *AFG* enables this to be done through a series of simple commands — now common to all of the ADAS special features. This is the *Applications Program Interface* (*API*).

The *API* provides a common access point to the ADAS special features

The approach means that all external programs can access ADAS special features in a structured, consistent manner for each of the special features.

AFG is made more accessible via a graphical user interface (*GUI*). This code, which also fulfils a pedagogical role, is known as ADAS605.



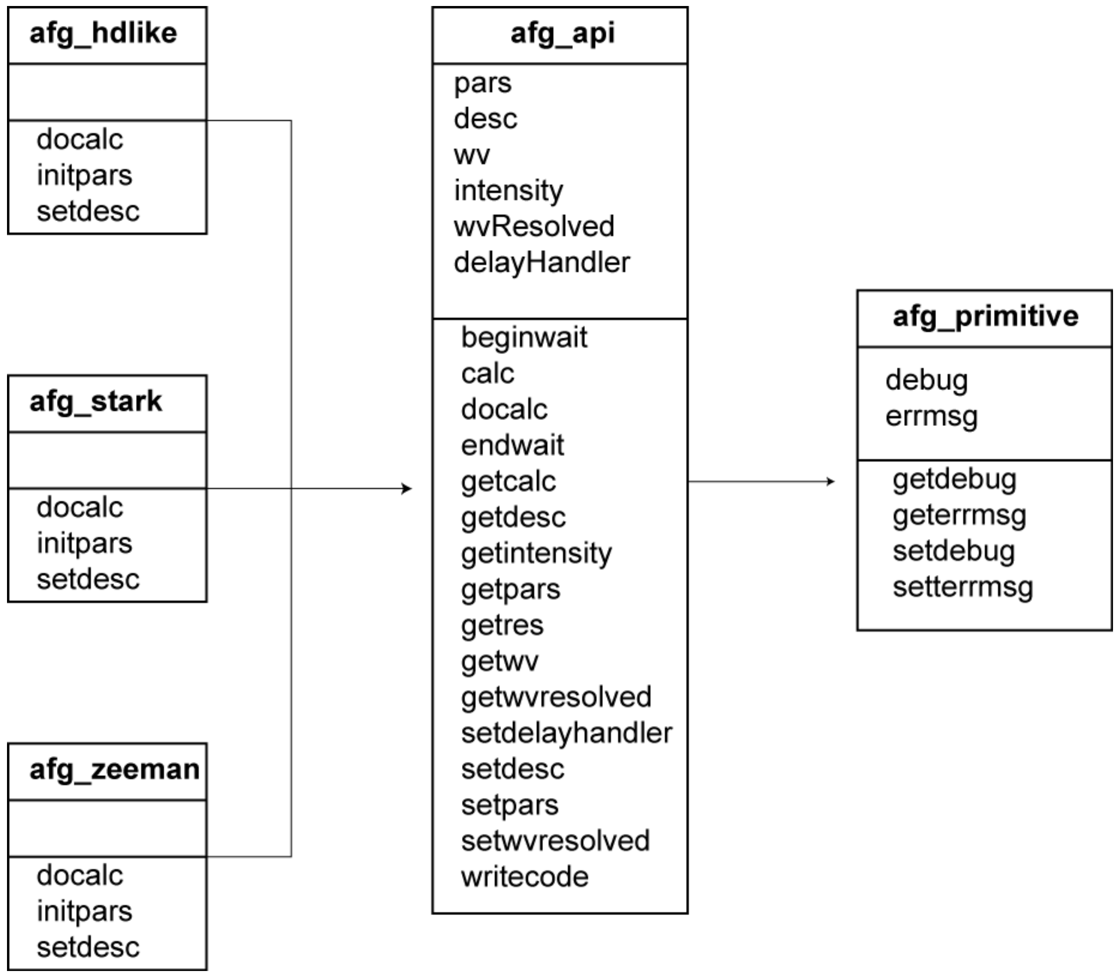
2.2 Class diagram for AFG

AFG is constructed using the *object oriented programming* capabilities of IDL.

As illustrated on the right, each *class* is represented by a rectangular box, split into three compartments.

The upper division is the *class name*, the middle section contains the *class attributes* (data members) and the lower segment holds the names of the available *class methods* (operations).

The arrows, pointing from one class to another, indicate the *inheritance hierarchy*.



2.3 AFG description structures, accessed through the ‘getdesc’ method

Command line interaction with AFG, retrieving the description structure for the Zeeman feature.

```
IDL> zeeman = obj_new('afg_zeeman')
IDL> desc = zeeman->getdesc()
IDL> help, desc, /str
** Structure <95adba4>, 3 tags, length=1060, data
        length=1060, refs=2:
   NAME            STRING    'Zeeman Feature'
   TEXT            STRING    'ADAS implementaion of Zeeman
                                    features base'...
   PARAMETERS      STRUCT    -> <Anonymous> Array[1]
```

Examination of an AFG description structure, for the Zeeman feature, at the command line.

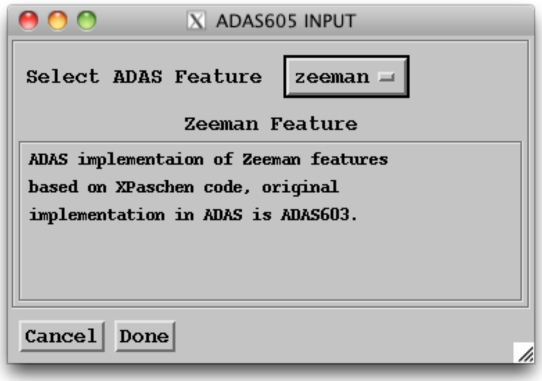
```
IDL> help, desc.parameters, /str
** Structure <96877a4>, 4 tags, length=1036, data
        length=1036, refs=2:
   POL             STRUCT    -> <Anonymous> Array[1]
   OBSANGLE        STRUCT    -> <Anonymous> Array[1]
   BVALUE          STRUCT    -> <Anonymous> Array[1]
   FINDEX          STRUCT    -> <Anonymous> Array[1]
```

AFG feature parameter sub-structure for the magnetic field strength, for the Zeeman feature.

```
IDL> help, desc.parameters.bvalue, /str
** Structure <96989b4>, 8 tags, length=60, data
        length=60, refs=2:
   DESC            STRING    'Magnetic field strength (T)'
   TYPE            STRING    'float'
   UNITS           STRING    'T'
   MIN             FLOAT           0.00000
   MAX             FLOAT           20.0000
   DISPTYPE        STRING    'continuous'
   LOG             INT             0
   ALTERSLIMITS    INT             0
```

3.1 ADAS605 – input and processing screens

In keeping with the pedagogical role of the interactive ADAS screens, a graphical user interface has been prepared for *AFG* called ADAS605.



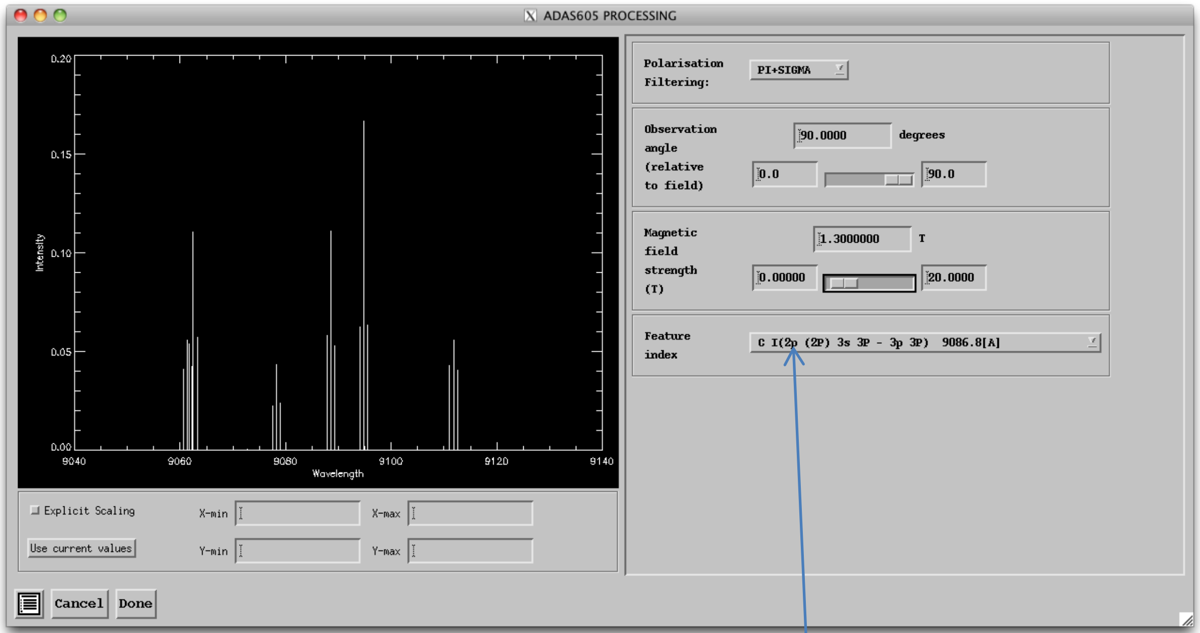
Select ADAS Feature: zeeman

Zeeman Feature

ADAS implementaion of Zeeman features based on XPaschen code, original implementation in ADAS is ADAS603.

ADAS605 has been designed to use the *AFG API* intensively, such that the interface is highly dynamic (i.e. its appearance is very much dependent upon the feature under consideration).

The *processing screen* is split into two main segments; the left hand side is consistently the same regardless of the feature selected — it is a *graphical display area*.



X-min:
X-max:
Y-min:
Y-max:

Polarisation Filtering: PI+SIGMA

Observation angle (relative to field): 90.0000 degrees (0.0 – 90.0)

Magnetic field strength (T): 1.3000000 T (0.00000 – 20.0000)

Feature index: C I(2p (2P) 3s 3P - 3p 3P) 9086.8[A]

The right hand side is comprised of a set of *control widgets* to alter the special feature parameters and will therefore adapt to the particular feature selected from the input screen.

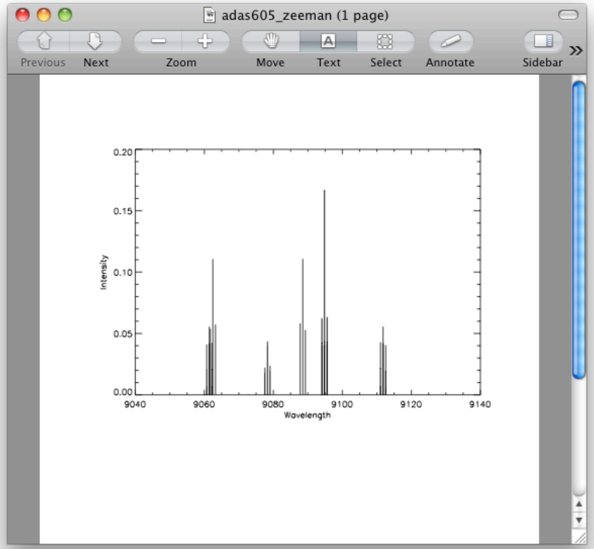
The control panel is not predefined in a static fashion. Instead, ADAS605 is examining the parameter description structures returned from method calls to the *API*.

CI (2p3s 3P – 2p3p 3P)

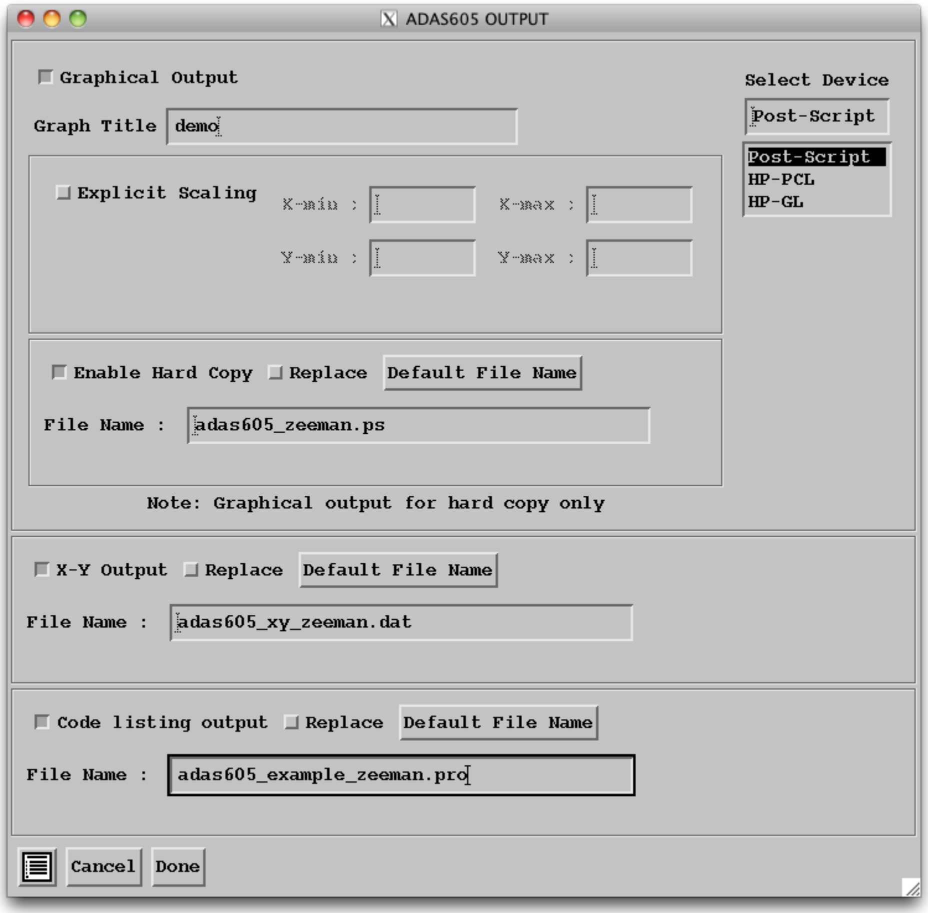
3.2 ADAS605 – output screen

The *output screen* provides the familiar ADAS graphical and text outputs, along with a further output called *Code listing output*.

graphical output



text output of x-y values



☐ Graphical Output

Graph Title demo

Select Device: Post-Script

- [✓ Post-Script]
- HP-PCL
- HP-GL

☐ Explicit Scaling   X-min :   X-max :   Y-min :   Y-max :

☐ Enable Hard Copy ☐ Replace Default File Name

File Name : adas605_zeeman.ps

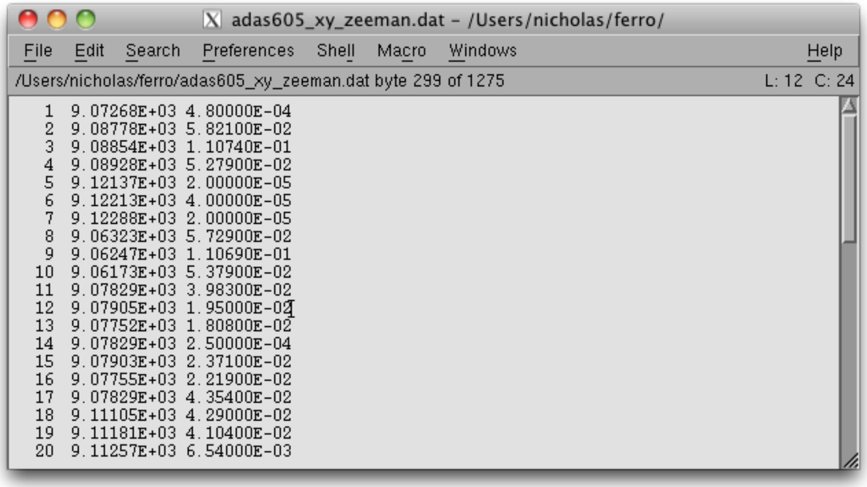
Note: Graphical output for hard copy only

☐ X-Y Output ☐ Replace Default File Name

File Name : adas605_xy_zeeman.dat

☐ Code listing output ☐ Replace Default File Name

File Name : adas605_example_zeeman.pro



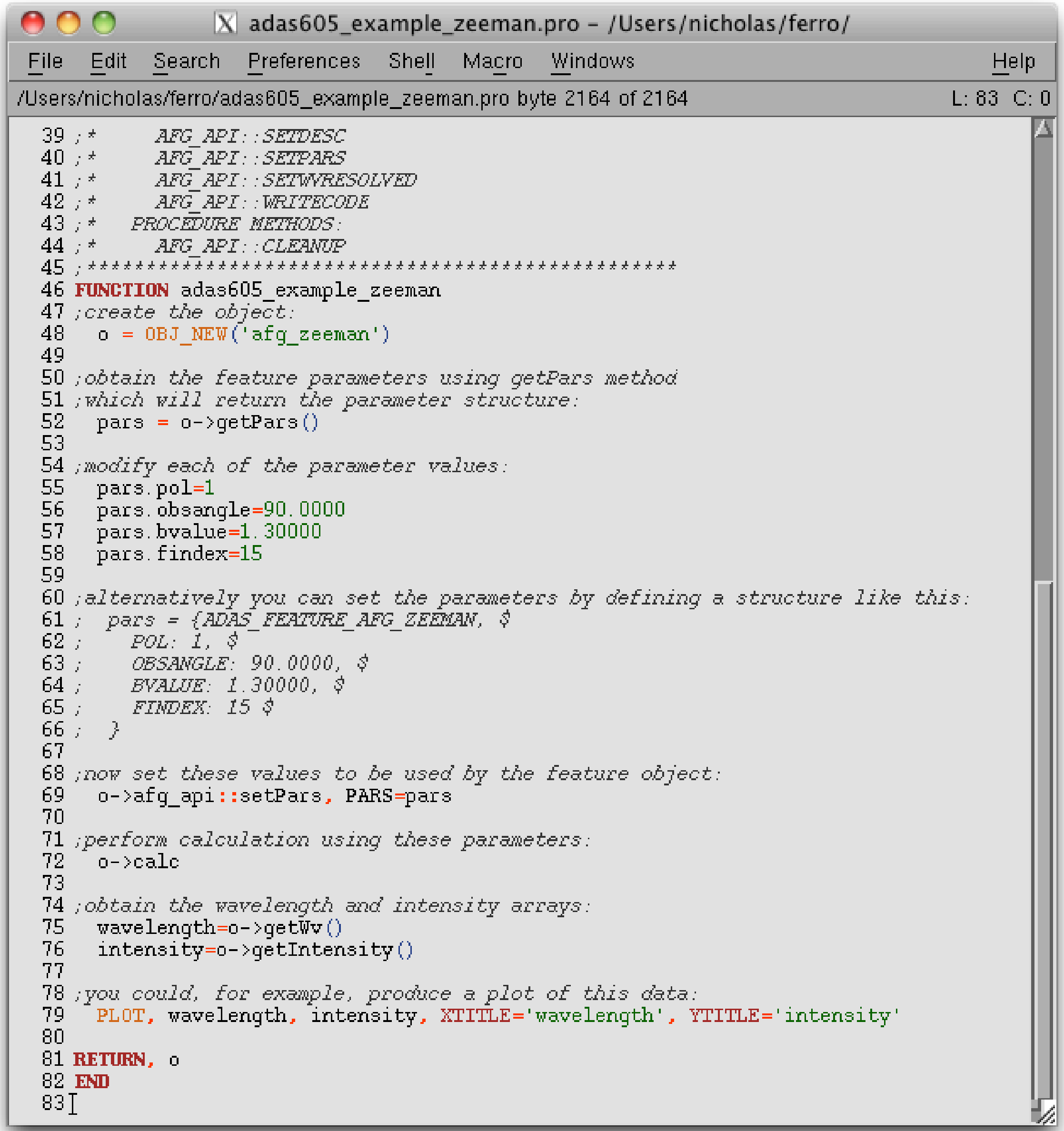
/Users/nicholas/ferro/adas605_xy_zeeman.dat byte 299 of 1275   L: 12 C: 24

```
  1  9.07268E+03 4.80000E-04
  2  9.08778E+03 5.82100E-02
  3  9.08854E+03 1.10740E-01
  4  9.08928E+03 5.27900E-02
  5  9.12137E+03 2.00000E-05
  6  9.12213E+03 4.00000E-05
  7  9.12288E+03 2.00000E-05
  8  9.06323E+03 5.72900E-02
  9  9.06247E+03 1.10690E-01
 10  9.06173E+03 5.37900E-02
 11  9.07829E+03 3.98300E-02
 12  9.07905E+03 1.95000E-02
 13  9.07752E+03 1.80800E-02
 14  9.07829E+03 2.50000E-04
 15  9.07903E+03 2.37100E-02
 16  9.07755E+03 2.21900E-02
 17  9.07829E+03 4.35400E-02
 18  9.11105E+03 4.29000E-02
 19  9.11181E+03 4.10400E-02
 20  9.11257E+03 6.54000E-03
```

3.3 ADAS605 code listing output

AFG will auto-generate the appropriate IDL source code (including in-line comments) to generate the feature using the *API* directly, rather than via the *GUI*.

It is envisaged that production of this template source code will serve as an entry point to most users looking to utilise AFG in their own codes.


adas605_example_zeeman.pro - /Users/nicholas/ferro/
File
Edit
Search
Preferences
Shell
Macro
Windows
Help
/Users/nicholas/ferro/adas605_example_zeeman.pro byte 2164 of 2164
L: 83 C: 0

```
;*      AFG_API::SETDESC
;*      AFG_API::SETPARS
;*      AFG_API::SETWVRESOLVED
;*      AFG_API::WRITECODE
;*    PROCEDURE METHODS:
;*      AFG_API::CLEANUP
;*****************************************************
FUNCTION adas605_example_zeeman
;create the object:
  o = OBJ_NEW('afg_zeeman')

;obtain the feature parameters using getPars method
;which will return the parameter structure:
  pars = o->getPars()

;modify each of the parameter values:
  pars.pol=1
  pars.obsangle=90.0000
  pars.bvalue=1.30000
  pars.findex=15

;alternatively you can set the parameters by defining a structure like this:
;  pars = {ADAS_FEATURE_AFG_ZEEMAN, $
;    POL: 1, $
;    OBSANGLE: 90.0000, $
;    BVALUE: 1.30000, $
;    FINDEX: 15 $
;  }

;now set these values to be used by the feature object:
  o->afg_api::setPars, PARS=pars

;perform calculation using these parameters:
  o->calc

;obtain the wavelength and intensity arrays:
  wavelength=o->getWv()
  intensity=o->getIntensity()

;you could, for example, produce a plot of this data:
  PLOT, wavelength, intensity, XTITLE='wavelength', YTITLE='intensity'

RETURN, o
END

```

4.1 Setting up spectral fitting with complex features

In order to model complex spectra, it is useful to consider a composite structure in which various model elements are assembled together to represent the various features present in the data.

In a mathematical sense, these model elements provide a set of basis functions for the model.

A useful analysis system requires a reasonable set of these elements, from basic spectral feature representations, such as a Gaussian line, to complex features coming from specialised modelling codes.

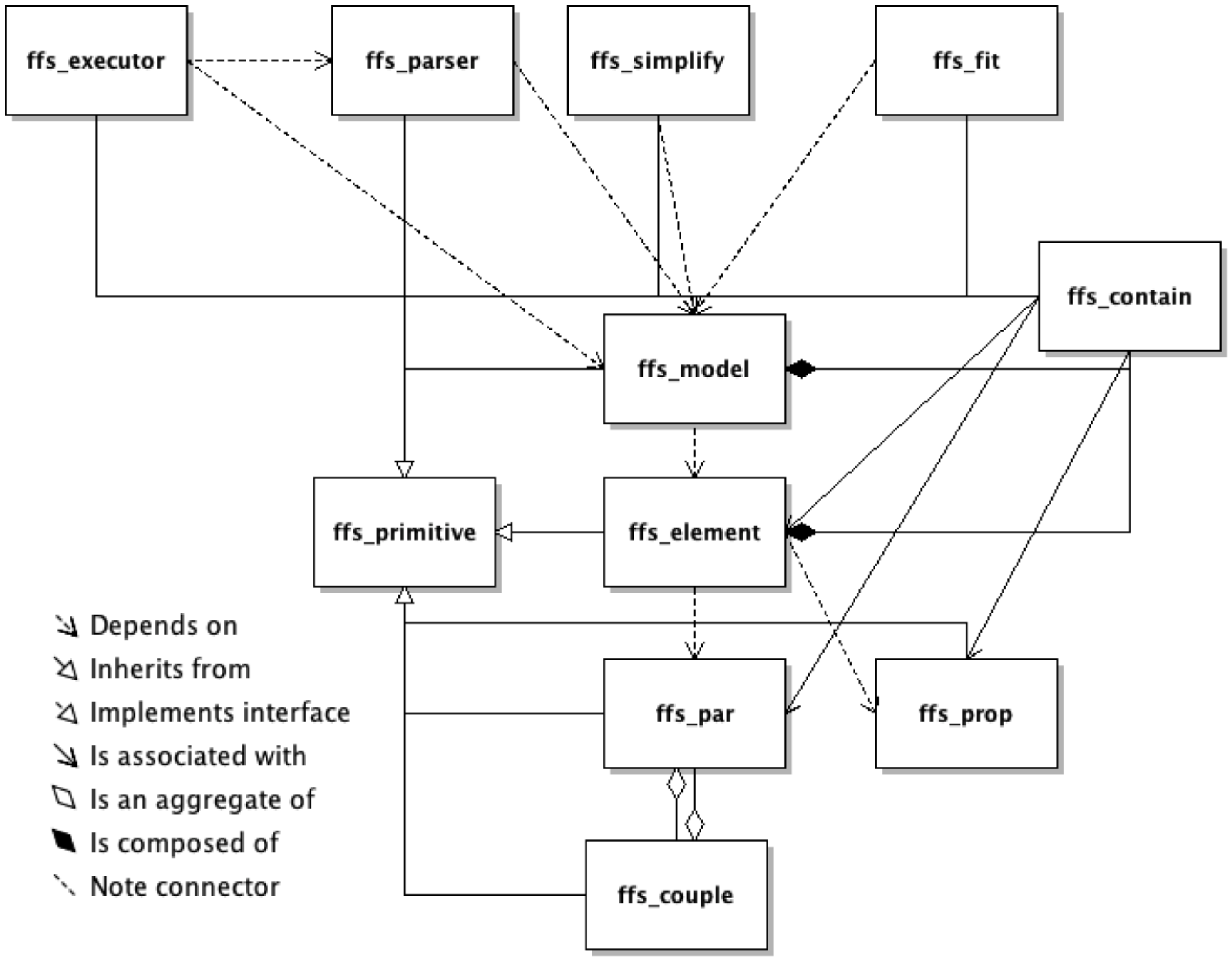
ADAS (via *AFG*) provides the complex features coming from specialised modelling codes.

Attention is given to the mathematical formulations for the calculation of the most commonly occurring features and their partial derivatives.

The analytic representations of the partial derivatives provide substantial improvements in performance for the fitting algorithm.

Each of the functions defined here (including intermediates such as the broadening functions) are implemented, programmatically in the *Framework for Feature Synthesis* (*FFS*) system.

4.3 Class diagram for FFS

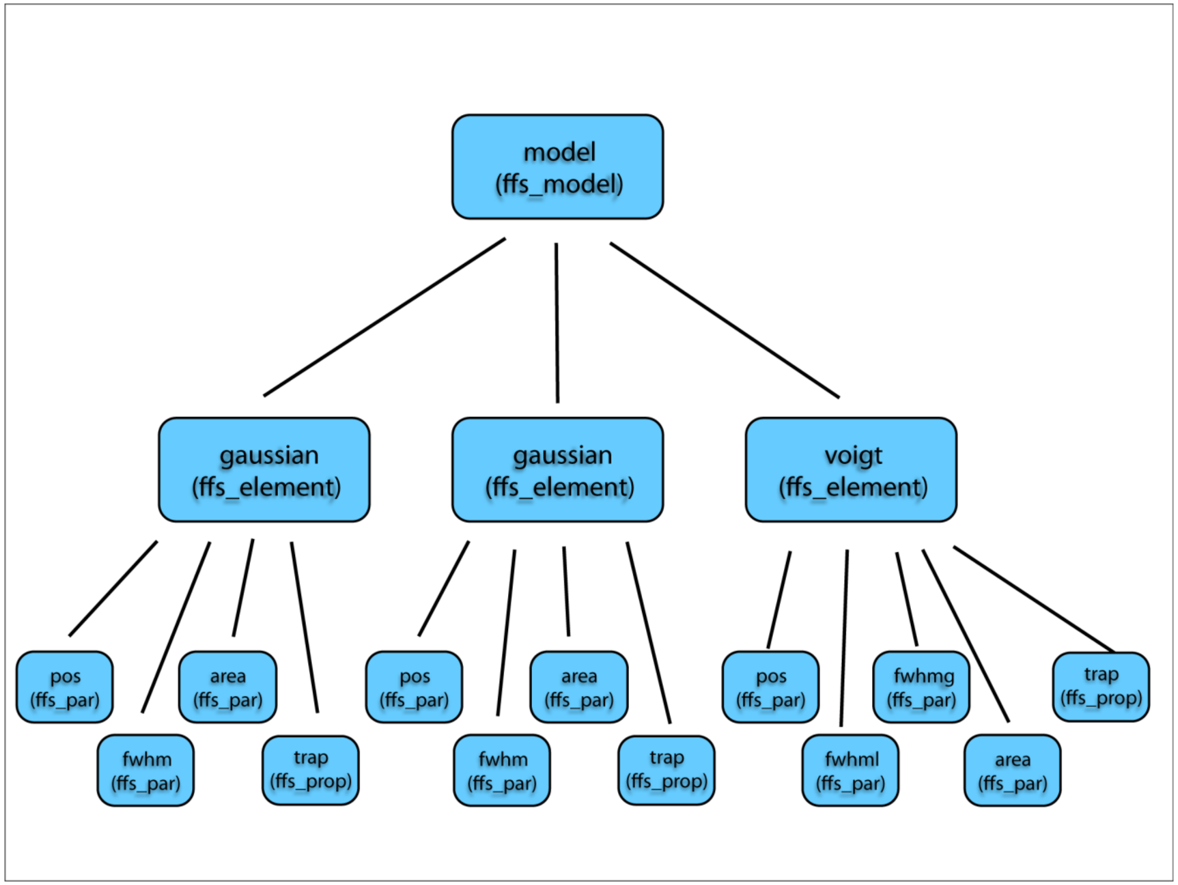


4.4 FFS

On the right, the model-element-par hierarchy for a simple model in *FFS*, consisting of two Gaussian lines and a Voigt line shape is shown. The Gaussian shapes have three parameters: position, full-width at half maximum and area. The Voigt has four: position, Lorentzian component of width, Gaussian component of width and area. The three elements, in this case also have a property ‘trap’.

The model on the right has only a single layer of elements in the tree structure—elements that are independent of each other

FFS is not limited to this case — there is support for operator elements that take the output of one or more of the other elements as input. To manage this in a generalised way, the *ffs_element* class caters for the storage of ‘child elements’ i.e. those elements on which it is dependent. By storing a reference to a ‘root’ element, the *ffs_model* element can then initiate recursive traversal of the tree to ensure that element results are calculated in the correct order.



4.5 FFS element class

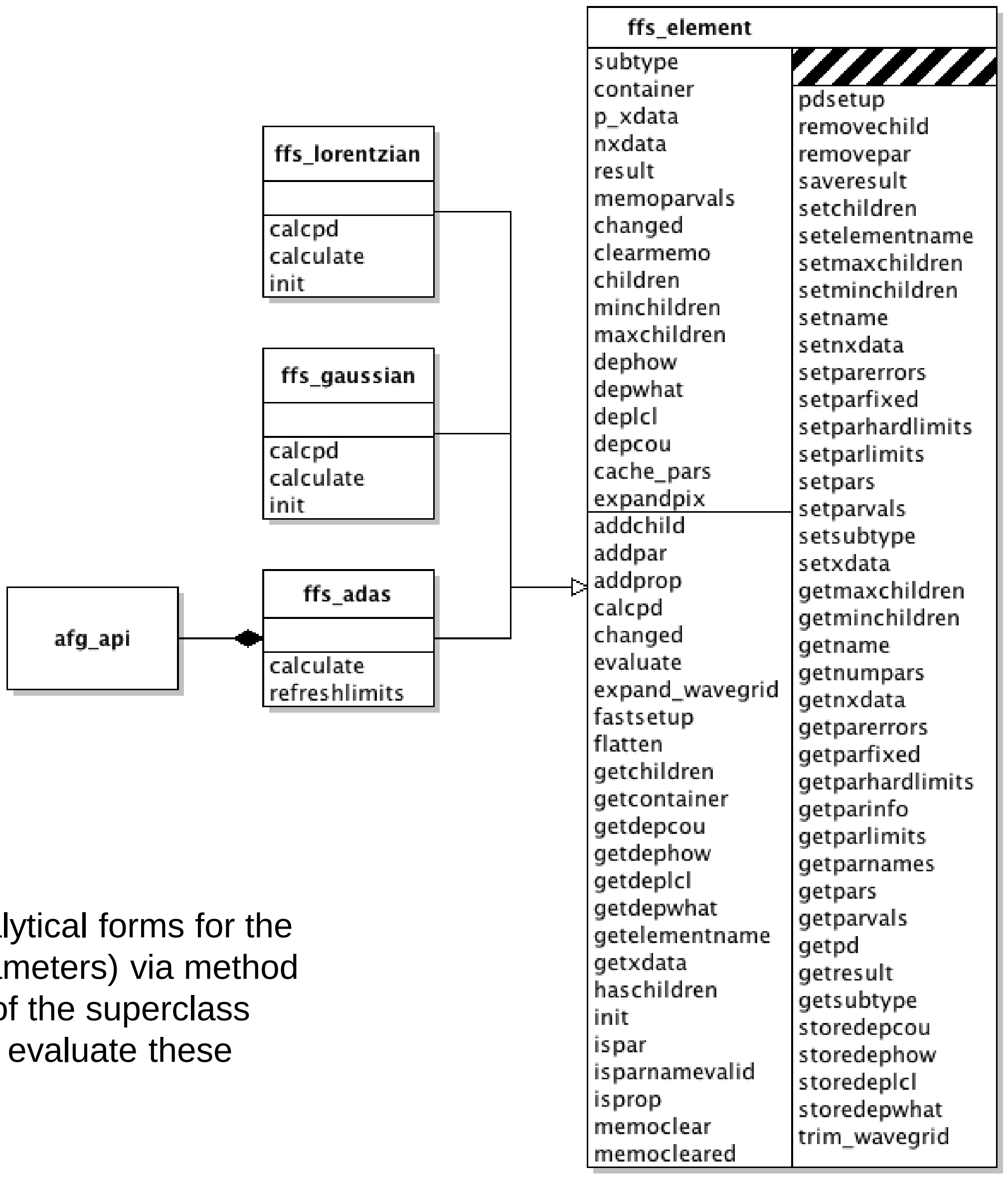
The *FFS* element class and some example sub-classes are shown on the right. Note that the class data entries are intentionally blank for the subclasses — they only have inherited data members.

In terms of implementation, *ffs_element* is an abstract class, from which *FFS* component features should inherit.

On the right is shown a few example features gaining access to the plethora of methods available from the superclass.

Note that the subclasses are, in all cases, required to supply a ‘calculate’ method which overrides the abstract method in the *ffs_element* superclass. This method provides the means to evaluate the spectral component.

If available, the *ffs_element* subclasses also provide analytical forms for the partial derivatives of the element (with respect to its parameters) via method ‘*calcpd*’. If not, then a call to ‘*calcpd*’ will result in usage of the superclass implementation, which uses a finite difference method to evaluate these quantities.



4.6 MDF model definition language and syntax

To specify the construct for an arbitrarily complex model spectra, it was helpful to set out a *Model Definition Language* (*MDL*).

The expressions defining elements take the form shown on the right where 'elementclass' is the name of the class type of the element.

```
(elementclass[-optinput] [operands] elementname)
```

'operands' (as noted above) is optional and can in fact be a list of element definition expressions.

```
(model
   (+
      (broaden_gauss
         (+
            (gaussian g)
            (lorentzian l)
         )
      brdg)
      (background-linear bg)
   )
example)
```

The model is defined by a set of nested element definition expressions, each enclosed in a set of brackets. The expressions themselves are of prefix notation i.e. an operator followed by a set of operands. It should be noted that one, or indeed all, of the operands can be further *MDL* expressions.

FFS provides a system for handling complex coupling between parameters, specified by the model definition. In the coupling expression form, shown on the right,'parname' is the name of the parameter being coupled and 'elementname' is the name of the element to which it belongs.

```
(couple elementname.parname cexpression)
```

4.7 The model optimisation

It is possible that some models will possess combinations of elements that can be readily reduced to a more optimum representation.

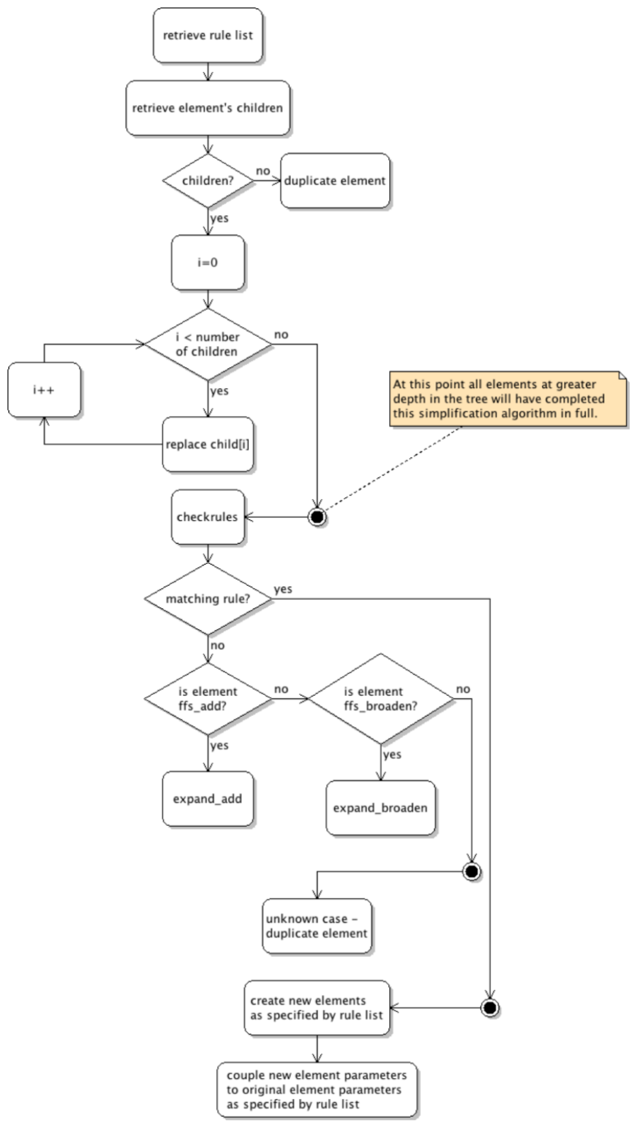
This is the case if, for example, there is a well known analytic solution for an operator element acting on some other element, that provides more efficient function evaluation.

The possibility of using analytic expressions for the parameter partial derivatives is an important such case.

A simple illustration is below.

```
(model                            | (model
  (+                              |   (+
    (broaden_gauss                |     (gaussian new_gauss)
      (+                          |     (voigt new_voigt)
        (gaussian g)              |     (background-linear new_bg)
        (lorentzian l)            |   )
      )                           | optimized)
    brdg)                         |
    (background-linear bg)        |
  )                               |
example)                          |
```

The model optimisation, by the the *ffs_ simplify* routine, in practice, uses a reference table of rules defining more efficient representations for a set of operator element-element pairs. The whole scheme is shown on the right.


retrieve rule list
retrieve element's children
children?
no
duplicate element
yes
i=0
i < number of children
no
i++
yes
replace child[i]
At this point all elements at greater depth in the tree will have completed this simplification algorithm in full.
checkrules
matching rule?
yes
no
is element ffs_add?
no
is element ffs_broaden?
no
yes
yes
expand_add
expand_broaden
unknown case - duplicate element
create new elements as specified by rule list
couple new element parameters to original element parameters as specified by rule list

4.8 FFS_FIT

Non-linear least squares fitting:

The fitting algorithm implemented for use in this work, is a version of that developed by Marquardt. The method has become one of the most widely used in optimisation problems. The advantage of this algorithm is that it manages to smoothly vary between two methods of minimising a function mentioned above: steepest descent and the Gauss-Newton method. The two methods complement each other in that each is effective under conditions that are less favourable for the other.

A custom fitting code:

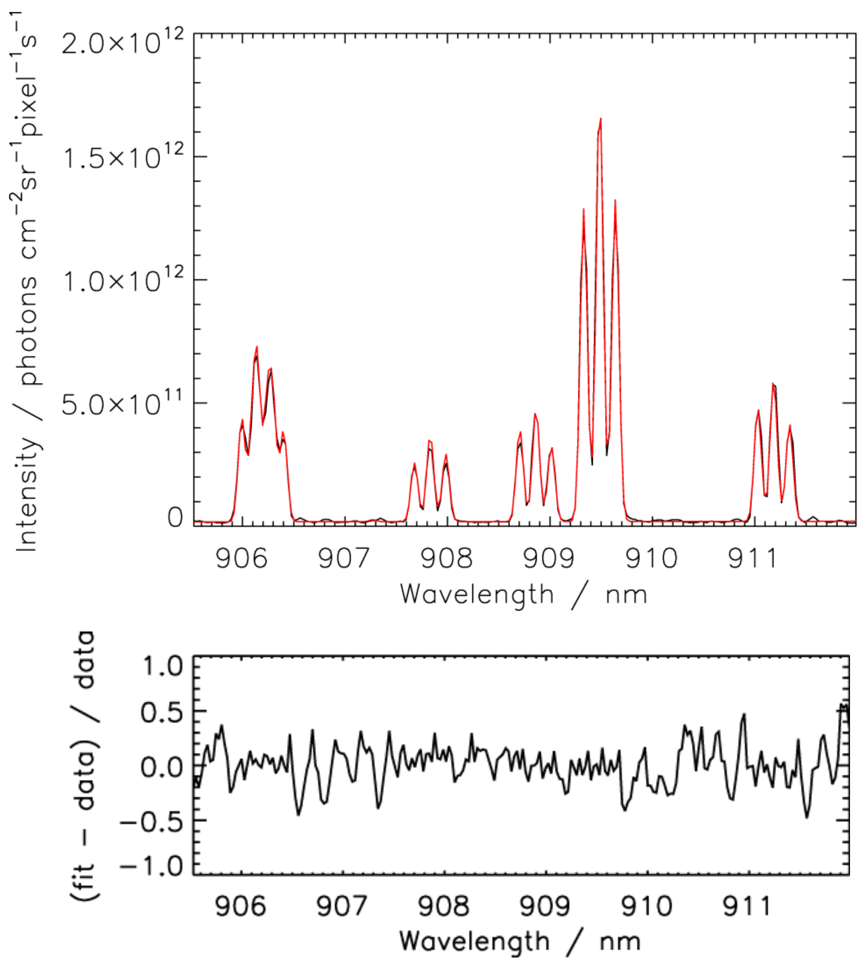
FFS is a computational framework for provision of complex spectral model specification. The spectral fitting process itself, however, is handled by a separate module to allow for flexibility for the user. Despite this design decision, it should be noted that FFS was originally intended to be used in conjunction with the readily available fitting program MPFIT. This package has, at its core, the Levenberg-Marquardt algorithm detailed above. MPFIT provides some additional machinery around the core algorithm such as setting some parameters in the model to be fixed, or imposing boundary constraints, basic parameter coupling and suggested step sizes (for numerical partial derivatives). These facilities influenced the parameter structure for FFS, thus it remains compatible with the routine, but FFS retains control of these properties as they are considered to be part of the model definition, rather than the concern of a fitting program.

Batch fitting:

It is often the case that an experiment will record multiple spectra—usually a series of spectra in time, space or both. It is desirable to fit all of the spectra in a systematic, automated scheme to identify trends in the derived parameters across the series. To this end, scripts exist as part of FFS which cycle through the frames of spectra performing fits using a single model definition.

4.7 An example

```
(model zeeman
    (+
        (shift-lambda
            (+
                (*
                    (broaden_gauss
                        (adas-zeeman cizeemanpi)
                    bgpi)
                cimultpi)
               (*
                    (broaden_gauss
                        (adas-zeeman cizeemansig)
                    bgsig)
                cimultsig)
            )
        sh)
        (background-linear backg)
    )
)

(setval sh.lambda 0.01)
(setval backg.m -8.0e9)
(setval backg.c 1.0e11)

(setval bgpi.fwhm 0.1)
(setlimits bgpi.fwhm 0.05 1.0)

(setval cizeemanpi.findex 15)
(setval cizeemanpi.obsangle 90.0)
(fixed cizeemanpi.obsangle)
(setval cizeemanpi.bvalue 2.0)
(setlimits cizeemanpi.bvalue 1.0 4.0)
(setval cizeemanpi.pol 2)
(setval cimultpi.factor 1.0e13)
(setlimits cimultpi.factor 1.0e11 1.0e15)

(setval bgsig.fwhm 0.1)
(setlimits bgsig.fwhm 0.05 1.0)
(couple bgsig.fwhm = bgpi.fwhm)

(setval cizeemansig.findex 15)
(setval cizeemansig.obsangle 90.0)
(fixed cizeemansig.obsangle)
(setval cizeemansig.bvalue 2.0)
(setlimits cizeemansig.bvalue 1.0 4.0)
(couple cizeemansig.bvalue = cizeemanpi.bvalue)
(setval cizeemansig.pol 3)
(setval cimultsig.factor 1.0e13)
(setlimits cimultsig.factor 1.0e11 1.0e15)
```



C I (2s22p3s 3P – 2s22p3p 3P) emission at 909 nm as recorded by KT3C at JET, pulse #78658 (R = 2:749 m, t = 61:3 s). The fitted model is shown in red, with the fit residual shown in the lower plot

5.1 Conclusions

The outcome of this work is the provision of two useful tools for analysis of atomic and molecular spectra. Firstly, the ADAS Feature Generator (*AFG*) facilitates ease of access to a range of special feature models within ADAS.

Secondly, the Framework for Feature Synthesis (FFS) package accomplishes its task by providing a highly flexible, modular model representation that can draw upon ADAS provided special features (via AFG) in addition to a basis set of familiar mathematical functions.

The FFS implementation goes beyond that offered by existing packages by allowing for specification of arbitrarily complex functional dependence between parameters via the model definition language.

A model 'simplification' module optimises any inefficient representations using the derived analytic solutions to form a new model, yet connects the new parameter set, via the coupling system, to the originally specified set.

The system has proved effective across the range of studies considered and the design of the framework remains flexible enough such that it can be used in support of analysis of any fusion or astrophysical spectra — a new special feature model is easily integrated by means of a lightweight AFG wrapper to an external modelling code.

The system is not yet complete. A number of areas are receiving further attention and the scope of models is being increased.