Special features and spectral analysis for fusion plasmas

A thesis submitted to The Department of Physics

of the University of Strathclyde

for the degree of Doctor of Philosophy.

Christopher H Nicholas

February 2011

© Copyright 2011

Abstract

In the magnetic confinement fusion domain, spectral analysis is a principal tool for establishing behaviour and performance characteristics of devices. It can assist, *inter alia*, in determining, with spatial and temporal resolution, key parameters such as electron and ion temperatures and densities, radiated power, impurity transport, impurity concentrations, internal magnetic and electric fields. The spectrometer complement on devices such as JET, ASDEX-U and LHD is very large, spanning wavelength ranges from x-ray to infra-red. Similar instrumentation will again be present on the next step in the world fusion program, ITER, currently under construction in France. Spectral analysis, at its most powerful, uses related spectral lines and features for diagnostic inference. In this thesis, such groupings are called *special features*. Their sensitivity is determined by the response of the emitting atoms to their physical environment. In broad concept, this thesis is concerned with special features and their diagnostic exploitation. To achieve this end, the thesis reviews special features and focusses on a number of representative types. It explores and expands the atomic physics link so that the special feature may be realised as a mathematical/computational construct for use in spectral modelling and fitting. In implementing this, it works closely with the Atomic Data and Analysis Structure (ADAS) project and its databases. The thesis seeks to empower special feature analysis by implementing generalised computational structures — *AFG* (ADAS Feature Generator) and *FFS* (Framework for Feature Synthesis). These allow both a pedagogical insight into the capabilities of each special feature as well as practical execution of optimised spectral fitting and plasma parameter extraction. The methods, based on object-oriented programming, are universal including aspects such as self-generating graphical user interfaces and an algebra of parametric feature creation. Demonstration of these methods is on selected JET spectra and 'shot' sequences. It is hoped that the work of this thesis will provide an advanced tool to match the spectral instrumentation capabilities from current fusion devices through to ITER. The implementation will be made available to the fusion community in an ADAS release in due course.

Contents

List of Figures iv

List of Tables viii

Acknowledgements ix

1 Introduction 1

1.1 Traditional Analysis Systems 6

1.2 Object-oriented Modelling for Numerical Fitting 8

1.3 Improving on Existing Software 10

1.4 Thesis Outline 11

2 Special Feature Modelling for Nuclear Fusion 13

2.1 Introduction 13

2.2 Issues of Orientation — Zeeman and Stark Spectra 15

2.3 Thermal Emission and Resolution 23

2.4 Stark Broadening and Series Limits: Balmer and Paschen Series . . . 27

2.5 Diatomic Molecular Emission 29

2.6 Heavy Species Envelope Emission 30

2.7 Helium-like Soft X-ray Resonance and Satellite Lines 32

2.7.1 The Population Calculations 35

2.7.2 Electron-impact Rate Coefficients 39

2.7.3 Computational Details 41

3 ADAS Special Feature Application Programming Interface 44

3.1 ADAS Feature Generator (AFG) 44

3.2 ADAS605 — GUI to AFG 52

4 Combinations of Functions for Spectral Fitting **59**
4.1 Introduction . . . 60
4.2 Functions Considered . . . 60
4.2.1 Un-broadened Line . . . 60
4.2.2 Gaussian . . . 61
4.2.3 Doppler . . . 63
4.2.4 Lorentzian . . . 64
4.2.5 Voigt . . . 65
4.2.6 Linear Background . . . 69
4.2.7 Addition Operator . . . 69
4.2.8 Scale Factor Operator . . . 70
4.2.9 Shift Operator . . . 71
4.2.10 AFG . . . 71
4.3 Practical Examples . . . 72
4.3.1 Convolution of two Normalized, Un-shifted Gaussian Functions 72
4.3.2 Convolution of N-Gaussian Profile with Gaussian . . . 73
4.4 Framework for Feature Synthesis . . . 75
4.5 Model Definition Language . . . 78
4.6 Parameter Coupling . . . 79
4.7 Optimisation of the Model . . . 85
4.8 Non-linear Least Squares Fitting . . . 91
4.9 A Custom Fitting Code . . . 94
4.10 Batch Fitting . . . 96
4.11 Validation of Results . . . 97
4.12 Analytic / Numerical fitting Speed Comparison . . . 102

5 Experimental Analysis **105**
5.1 Initial Validation . . . 106
5.2 An Illustration from SOHO-CDS . . . 106
5.3 Divertor Detachment Experiment at JET . . . 114
5.4 Zeeman Split Features . . . 120
5.5 Diatomic Molecular Spectra in the JET Divertor . . . 125
5.6 VUV Divertor Impurity Spectra . . . 128
5.7 He-like Argon Spectra from TEXTOR . . . 131

6 Conclusions and Future Work **134**

A Mathematical Notes **149**

A.1 Convolution — Definition and Basic Properties 149

A.2 Area of Convolved Functions 149

A.3 Raw Moments of Convolved Functions 150

A.3.1 First Raw Moment 150

A.3.2 Second Raw Moment 151

A.4 Gaussian Line Widths 151

A.5 Delta Function 153

A.6 Error Function 153

A.6.1 Definition 153

A.7 Complementary Error Function 153

A.7.1 Definition 153

A.7.2 Derivative 155

A.8 Complex Error Function 157

A.8.1 Definition 157

A.8.2 Derivative 157

B FFS MDL Listings **160**

C FFS Simplification Rules **173**

D FFS Core Routine Methods **177**

E ADAS Glossary **217**

E.1 Acronyms 217

E.2 ADAS Main Program Series 218

E.3 ADAS Data Formats 222

List of Figures

1.1 Zeeman split carbon line emission, from JET pulse #75989. 3
1.2 Zeeman split carbon line emission, from JET pulse #70574. 3
1.3 Zeeman split carbon feature from JET pulse #75898, diagnostic KS8. 4

2.1 Simulation of the Zeeman multiplet feature for CI (2p3s ^{3}P – 2p3p ^{3}P) with ADAS603. 18
2.2 The ADAS305 neutral beam simulation for a deuterium plasma. . . . 24
2.3 A simulated spectrum showing the Balmer series in deuterium. 29
2.4 A simulated BeD molecular spectrum as produced using CALCAT (via AFG). 30
2.5 $\mathcal{F}$ – $\mathcal{PEC}$ for tungsten ions, ionisation stages W^{54+} to W^{73+}. 31
2.6 Fundamental dielectronic recombination (DR) data preparation and special feature assembly. 42

3.1 Accessing ADAS special features. 45
3.2 Class diagram for AFG. 47
3.3 Command line interaction with AFG, retrieving the description structure for the Zeeman feature. 47
3.4 Examination of an AFG description structure, for the Zeeman feature, at the command line. 47
3.5 AFG feature parameter sub-structure for the magnetic field strength, for the Zeeman feature. 48
3.6 Command line interaction with AFG, altering a parameter values structure. 49
3.7 Command line interaction with AFG, altering parameter values directly. 49
3.8 Examination of the AFG subordinate structure for the Stark feature from the command line. 50
3.9 AFG ‘subordinate’ structure for the Stark feature. 51
3.10 Examination of the AFG group structure for the Stark feature from the command line. 51

3.11 ADAS605 input screen: feature selection. 52
3.12 Screenshots showing examples of the IDL widgets generated by ADAS605. 53
3.13 ADAS605 processing screen. 54
3.14 ADAS605 processing screen when the Stark feature is selected. 55
3.15 ADAS605 output screen. 56
3.16 ADAS605 'graphical output'. 57
3.17 ADAS605 'X-Y output'. 57
3.18 ADAS605 'code listing output'. 58

4.1 Class diagram for FFS. 75
4.2 The model-element-par hierarchy for a simple model in FFS. 76
4.3 FFS element class and some example subclasses 77
4.4 MDL — element definition syntax. 78
4.5 MDL — model definition syntax. 79
4.6 MDL — main coupling syntax. 80
4.7 MDL — coupling expression syntax 80
4.8 Flowchart showing the algorithm for retrieving analytic partial derivatives through coupled parameters. 82
4.9 Example of an MDL coupling statement. 83
4.10 The coupling statement shown in 4.9 after pre-parse for use by coupling object. 83
4.11 Debug output from *ffs_couple* as it parses the inner expression of the coupling statement displayed in Fig. 4.10. 83
4.12 Debug output from *ffs_couple* as it parses the outer expression of the coupling statement displayed in Fig. 4.10. 84
4.13 The 'simplification' of a user specified model. 86
4.14 Rule list structure example. 87
4.15 Optimisation procedure. Note that the operation *replace child* follows the algorithm defined by this flowchart i.e. this is a recursive method call. 88
4.16 Dealing with an addition element branch. 89
4.17 Dealing with a broadener element branch. 90
4.18 Basic line function and partial derivative calculation verification (numeric versus analytic). 98

4.19 Broadened Gaussian line function calculation verification (simplified versus original representation). 100
4.20 The 'simplification' of the Gaussian broadener acting on a Gaussian. . 100
4.21 Broadened Gaussian line function calculation verification (simplified versus original representation) at the edge. 101

5.1 Initial validation of FFS — fitting two noisy Gaussians with a linear background. 107
5.2 OIV multiplet as recorded by NIS-2 before the loss of SOHO, with fitted models from FFS and ADAS602 overlaid. 111
5.3 MDL statements to form the custom line shape for 'post-loss' SOHO-CDS data. 112
5.4 OIV multiplet as recorded by NIS-2 after the loss of SOHO, with fitted models from FFS and ADAS602 overlaid. 113
5.5 Poloidal cross-section of JET showing magnetic flux surfaces and Langmuir probe positions in the divertor. 115
5.6 Balmer series spectrum as recorded by instrument KT3A at JET, pulse #70578 with FFS fit. 116
5.7 A colour-shaded representation of the electron density, as a function of position and time, as extracted from a set of KT3 measurements by FFS and a related graph of ion fluxes as measured by a divertor probe. 118
5.8 Electron density measurements versus radial position from FFS fits to spectroscopic data, compared with Langmuir probe measurements. . . 119
5.9 C I ($2s^2 2p3s\ ^3P$ – $2s^2 2p3p\ ^3P$) emission at ~ 909 nm as recorded by KT3C at JET, pulse #78658. 121
5.10 Comparison of colour-shaded representation of the magnetic field, as a function of position and time, resulting from FFS fits against results from *Flush*. 122
5.11 (a)KS8 line of sight at JET, (b) magnetic field versus major radius (comparison of FFS against *Flush*). 124
5.12 C III ($1s^2 2s3s\ ^3S$ – $1s^2 2s3p\ ^3P$) multiplet at ~ 465 nm as recorded by KS8 at JET, pulse #75898 with FFS fit. 124
5.13 Observed beryllium deuteride (BeD) spectrum from instrument KS3A during JET pulse #35687, with FFS fit. 126
5.14 Observed beryllium deuteride (BeD) spectrum from instrument KS3B during JET pulse #35689, with FFS fit. 127
5.15 Surface plot (in temperature in density) of $\mathcal{PEC}$ for C IV ($1s^2 2s$ – $1s^2 3p$). 129
5.16 Spectrum recorded by JET diagnostic KT7/2, JET pulse #35421, with FFS fit. 130

5.17 He-like Argon x-ray spectrum from *TEXTOR* shot #81156, with FFS fit. 133

List of Tables

3.1 Summary of requirements and outputs of the various ADAS special feature generation facilities. 45

3.2 AFG 'disptype' options. 48

4.1 The 'simplification' of the various broadener elements acting upon an un-broadened line element. 99

4.2 Partial derivative calculation performance test. 103

5.1 Correlation matrix resulting from the fit of FFS validation trial 1. 107

5.2 Correlation matrix resulting from the fit of FFS validation trial 2. 108

5.3 The fit parameters resulting from the fit to the (pre-loss) NIS-2 OIV multiplet ($2s^2 2p\ ^2P$ – $2s2p^2\ ^2P$). 109

5.4 The fit parameters resulting from the fit to the (post-loss) NIS-2 OIV multiplet ($2s^2 2p\ ^2P$ – $2s2p^2\ ^2P$). 110

5.5 Comparison of line intensities as determined by fitting the pre-loss data with FFS and ADAS602 with the theoretically calculated photon emissivity coefficients ($\mathcal{PEC}$) values from ADAS208. 110

5.6 Comparison of line intensities as determined by fitting the post-loss data with FFS and ADAS602 with the theoretically calculated photon emissivity coefficients ($\mathcal{PEC}$) values from ADAS208. 112

5.7 Transitions in Ar^{16+} relevant for satellite line feature 131

5.8 Transitions in Ar^{15+} relevant for satellite line feature 131

Acknowledgements

Firstly, I would like to thank EPSRC and UKAEA Culham for funding my research. I am most grateful to my supervisors, Prof. Hugh Summers, Dr Allan Whiteford and Dr Andrew Meigs for the time and effort that they committed to my project. All of them showed a great deal of enthusiasm and support throughout. I would like to particularly thank Allan and Andy for their invaluable advice on computational physics and application to experiment. I thank Hugh for many enthusiastic discussions on atomic physics and for providing additional support for my research from ADAS. I would also like to thank Dr Martin O'Mullane for his support and suggestions while I was at JET. Thanks also go to Allan, Andy, Hugh, Martin and Prof. Nigel Badnell for taking the time to read through my thesis, whilst being so busy with their own work — particularly Allan who did so, despite the fact that it was no longer his job.

I would also like to mention Alessandra Giunta for her help in the acquisition and analysis of SOHO-CDS data and for her friendship throughout my degree. Similarly, I would like to thank Craig Hamilton and Lawrence Campbell for many lunch break discussions that helped keep me sane at Strathclyde.

Outside of university, I am thankful to have had encouragement from my friends, particularly Raymond Lutz, Gregg Newton, Andrew Stark and James Thomson — even if it was often just to ask: "have you not finished yet?". My family have all been very supportive of me throughout and spurred me on when I needed it. I would particularly like to thank my parents Argyros and Elizabeth for their support.

Finally, and most importantly, I must thank my fiancée Pamela Jane. She has always provided me with encouragement while completing my thesis and managed to keep me motivated even at times when morale was low. I am certain that completion of my thesis would have been far more difficult without her.

— Christopher Nicholas
February 2011

Chapter 1

Introduction

The analysis of atomic spectra from both astrophysical and laboratory fusion plasmas is essential in our effort to understand and interpret their characteristics. Spectroscopy is particularly key since such plasmas often remain remote to analysis by other means. Most astrophysical sources are simply physically too far away to probe in any other fashion, but there are many scenarios, even in the laboratory, in which the plasma remains out of reach. The passive, non-interactive nature of spectroscopy indeed may make it useful in these scenarios. For instance, in the case of a magnetically confined fusion device, insertion of diagnostic Langmuir probes is detrimental to plasma stability due to high levels of erosion of the probe at plasma temperatures — spectroscopy does not suffer in this way.

At a basic level, analysis of spectra can identify the species present within the plasma — an immediate and valuable result. However, spectroscopic data can often reveal far more than that. Atomic spectra may supply us with information about the local plasma conditions at the location of the emitter. It should be noted that this requires that the plasma is 'optically thin' i.e. that emitted photons are not readily absorbed in their path from their source to detector. The absorption of photons within the plasma volume will reduce the intensity of the spectra at the wavelength corresponding to that photon energy. Conversely, the population of other energy levels will have been increased by the absorption, which could in turn result in further photon emission at different energies due to spontaneous or stimulated emission from this level. If the resulting emission is to be modelled, these opacity effects must be taken into consideration, but analysis of the spectra cannot directly yield localised measurements. The most interesting spectra, from a diagnostic point of view, are those which are away from 'thermodynamic equilibrium' i.e. their radiation field does not exhibit a feature-less Planckian (black-body) distribution. High temperature, optically thin plasmas have a

large radiation deficit and so, fall in to the category of non-thermal plasmas. They emit a structured spectra, composed of lines.

It has been stated that physical parameters in plasmas can be inferred from the spectra, to exemplify this, consider the centroids of spectral lines. They may, for example, deviate from their expected position due to the doppler effect i.e. due to local plasma flow. Again, associated with the doppler effect, the distribution of particle velocities (due to thermal motion) means that spectral lines will not appear as single points of intensity, but instead a shape defined by the thermal velocity distribution is observed. This is referred to as a 'broadening mechanism' of which there are many e.g. magnetic and electric fields experienced by emitting atoms can also cause broadening of lines (through perturbation of the atomic structure of the emitter). Understanding, and consequently modelling, these mechanisms allow us to infer local measurements of the fields from the spectra. Of course, such line shapes are essentially the result of several emission lines in a very short wavelength interval blending together. In some cases however, instrumentation is of sufficient resolution to resolve individual components of these shapes, whereas lower resolution apparatus will show an envelope of emission. This means that models must be able to take into account such differences even when the underlying spectra are the same. Examples of Zeeman-split Carbon line emission are shown when recorded at sufficient resolution to see the individual components (Fig. 1.1) and then the same feature when recorded at a lower resolution (Fig. 1.2).

As well as the various line splitting / broadening mechanisms, it is also possible to characterise properties of the plasma by examination of the intensity of the spectral lines. By modelling the atomic population driving the emission, it is possible to deduce quantities, such as temperature, from the relative intensities of a set of lines from the same ion.

It should also be noted that just as the emission from atoms and molecules (and resultant spectra) are indicators of the local plasma conditions, the converse holds: a range of different plasma conditions result in a range of spectral features. It is possible (and likely) that along a given line of sight, several of these features may be present in a single spectrum, in some cases these occupy the same wavelength interval and are difficult to resolve. For example, with a horizontal, radial line of sight at a tokamak such as the Joint European Torus (*JET*) [1], emission of impurity species will be seen from the scrape-off layer (a region of particle flux in the outlying magnetic surfaces) near the inner and outer walls. The toroidal magnetic field in a tokamak decreases with major radius ($B_T \propto \frac{1}{R}$). This means that the emitters are situated in quite different magnetic fields and therefore experience differing Zeeman splitting patterns.

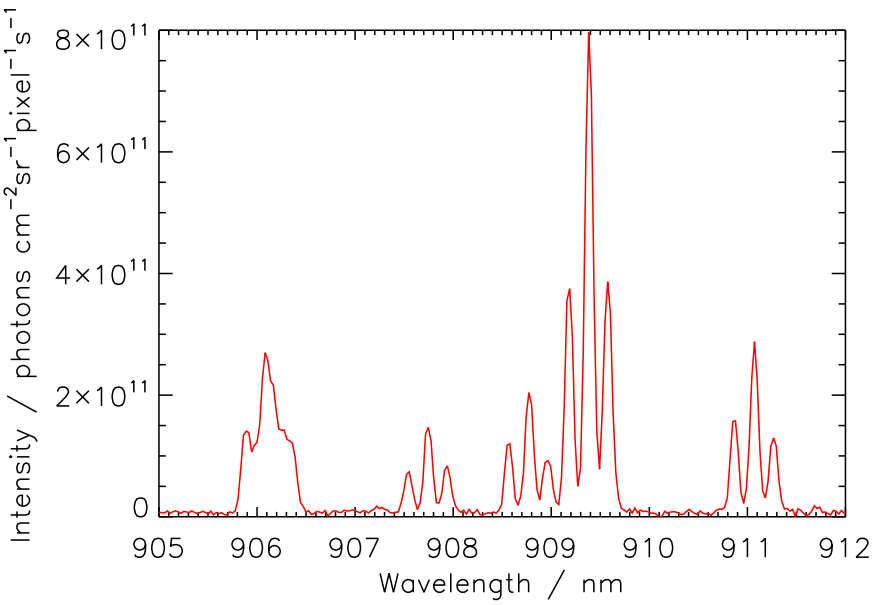


Figure 1.1: Zeeman split carbon line emission, from JET pulse #75989 (R = 2.8 m, t = 46.25 s, diagnostic KT3C using 1200 lines mm^{-1} grating) recorded at sufficient resolution to resolve several component lines of the feature.

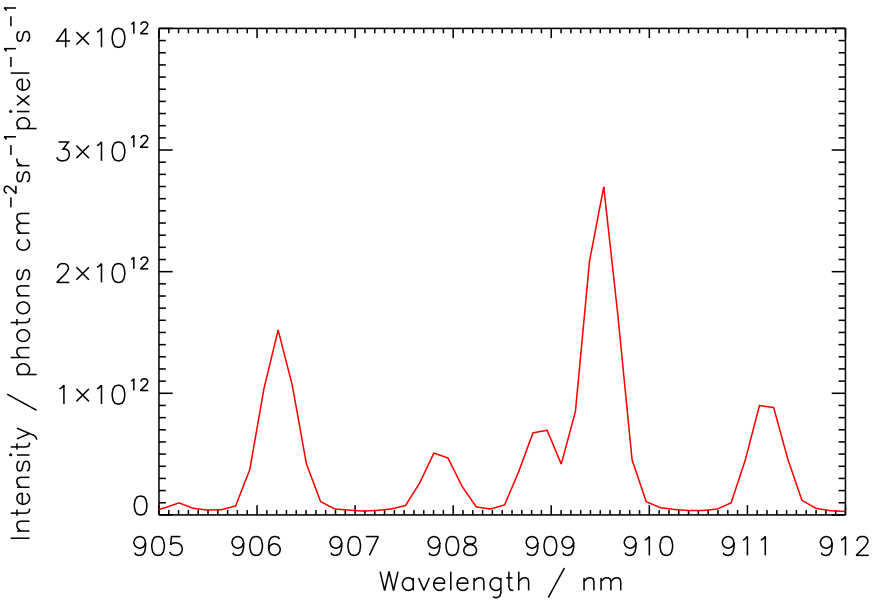


Figure 1.2: Zeeman split carbon line emission, from JET pulse #70574, (R = 2.68 m, t = 60.28 s, diagnostic KT3C using 300 lines mm^{-1} grating) at lower resolution than in the case of Fig. 1.1. Individual components are no longer resolved, instead they appear blended together.

Figure 1.3 shows this effect for CIII.

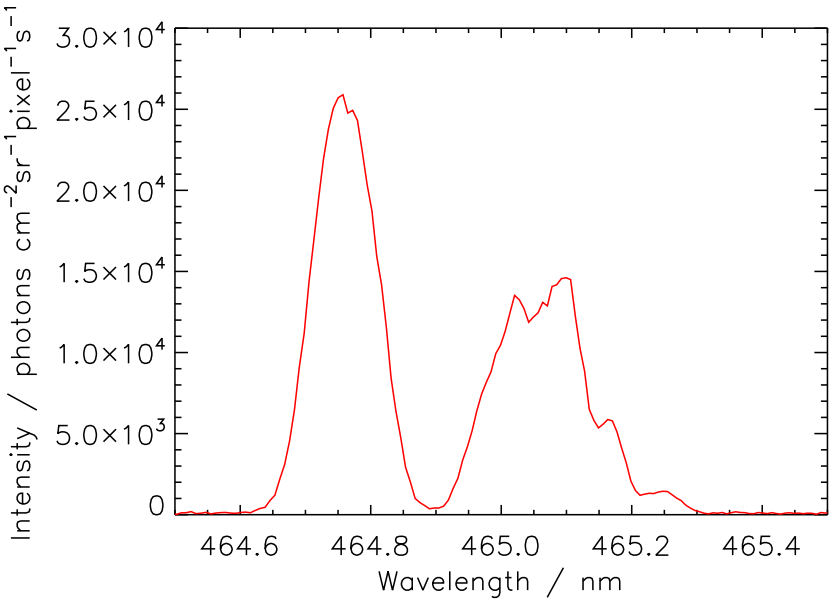


Figure 1.3: Zeeman split carbon feature from JET pulse #75898, diagnostic KS8. In this figure, the recorded spectral feature, is actually two overlapping Zeeman signatures, from similar emitters, but each situated in differing magnetic fields (see Section 5.4).

Spectroscopy is widely used in a vast number of plasma scenarios. This is exemplified by the extensive use on magnetically confined fusion plasma experiments. The plasma conditions vary significantly within the volume of such machines i.e. there are gradients in temperature, density and magnetic field. Temperatures in the core plasma are extremely high — in this environment, most elements are present as fully stripped ions and so, there is very little line emission from this region. Normally, spectra recorded from the core are mostly comprised of a Bremsstrahlung continuum. However, one of the methods of heating fusion plasmas is via injection of high energy neutral particles which impart their energy to the bulk hydrogen fuel by collision. The beam of neutrals undergo charge-exchange processes with the plasma ions. This means that a spectrometer with a line of sight passing through the neutral beam can measure the consequential emission, enabling a way of measuring plasma conditions in the core plasma.

There are cooler regions of the machine such as the plasma edge and the divertor. Careful control of the confining magnetic field in a tokamak can allow for formation of a special magnetic configuration in which two regimes exist — the inner being a set of closed, nested magnetic surfaces which are suitable for confinement, but outside this,

the flux surfaces are open, with the field lines directing particles to material surfaces. This system, whereby plasma flow along these outer surfaces (the scrape-off layer) is channeled to a surface remote from the bulk plasma forms the divertor. The reason for creating such a system is to remove impurities from the plasma and so, spectra from this area will display the presence of these species. Since there is a flow of the hot fusion plasma onto a collecting surface, there is the issue of heat load onto that surface. In order to alleviate this heat load, one of the techniques employed is the introduction of neutral gas which can provide radiative cooling and detachment of the plasma from the divertor plates — this can lead to strong deuterium recombination emission. Electron temperatures in the divertor are in fact low enough such that it is even possible to observe molecular emission.

Advanced models, considering the underlying atomic processes that produce the complex special features that are observed are of great diagnostic value. They characterise the spectra in terms of physical plasma parameters rather than more abstract parameters which define some mathematical line shape representation, which then need to be re-interpreted in terms of the parameters of interest. Special features are typically series of spectral emission lines which are interdependent on each other (i.e. they come from the same source). Consideration of the signature of the entire special feature, rather than the individual lines, has many benefits. One example, would be when a component of a line shape, say line width, is dependent upon multiple parameters, but the level of dependence on each differs between the individual spectral lines. In this case, only modelling the entire special feature will reliably resolve the parameter. Another example is when a feature is comprised of a large number of spectral lines at similar wavelengths such that the line shapes merge together; much of the detail of the feature is hidden as the various broadening mechanisms blend the lines together — this is seen in emission from heavy species elements. It is not practical in this case to decompose the feature into its individual line components in this case and so, a special feature model is required to fit the observed spectra. Such special features may also prove valuable in improving numerical stability of the minimisation algorithms utilised by reducing the number of model parameters. Special feature models may also offer more efficient computation of the function considered, the partial derivatives of this function (with respect to its parameters), or both.

Although the advantages of using special feature models is clear, often it has been the case that the development of computer programs to perform the analysis is focussed on a specific feature. The codes are purpose built and offer little flexibility to include additional simple lines or other special features in the same model. This can limit analysis to small spectral regions with such codes or, worse still, that their use is

omitted completely. Despite the fact that each of the special feature models are highly sophisticated and specialised, in reality they are essentially very similar; they are mathematical functions that are controlled by a defining set of parameters. Of course, this is also true for the simpler line representations too. In fact, it is best to consider the spectra in this way — an entity comprised of several component functions. This is the approach taken by this work — an effort has been made to abstract from the task of fitting a particular experiment and consider the problem more generally, while retaining the level of nuance required for the individual spectra at hand. Consideration is given to how to formulate a modular representation of spectra built from a series of basis functions and how to handle more complex combinations of them. Additional flexibility is provided by a system to mathematically couple together parameters in the system (across components) to constrain the model, define ad-hoc models, or both.

1.1 Traditional Analysis Systems

Many of the existing spectral fitting programs are focussed on a single aspect of spectroscopy, rather than taking a more global view to fitting spectral data. For example, the *Package for Interactive Analysis of Line Emission* (*PINTofALE*) [2] was originally developed to analyse spectra from optically thin coronal plasmas. *PINTofALE* aims to provide access to atomic databases — specifically *CHIANTI* [3], the *Astrophysical Plasma Emission Database* (*APED*) [4] and also the ***SPEC****tral* ***X****-ray and UV modeling, analysis and fitting package* (*SPEX*) [5] — and allows interactive comparison to observed data. The tools available for fitting spectra are, however, limited in scope. The interactive module, *fitlines*, supplied with *PINTofALE*, is only concerned with fitting line shapes with three parameters (e.g. Gaussian and Lorentzian profiles). Variants of these basic line shapes (with more than three parameters) are allowed, but only if the additional parameters remain fixed.

Although the theme of this work is generalised modelling and fitting of spectra, retention of focus on individual fitting problems is, nonetheless, not without merit; *CXSFIT* [6] strongly exemplifies this. *CXSFIT* is a spectral analysis tool that has been carefully tailored to specifically fulfil the required level of intricacy involved in fitting charge exchange spectra. Indeed, *CXSFIT* itself is based upon another code *KS4FIT*, which had been the mainstay of charge-exchange spectroscopy at JET. The resultant code is really the culmination of $\sim$ 20 years of experience of von Hellerman and associated JET researchers over this period. Charge exchange spectroscopy has proven very successful in current fusion devices that utilise a neutral beam injection (NBI)

heating system, for resolution of local measures of quantities such as temperature and impurity concentration and will remain so, looking towards ITER [7] such that there is yet some mileage to be had from a proven, specialised code like *CXSFIT*.

In order to produce accurate special feature models, they should be underpinned by the highest quality atomic data available. The *Atomic Data and Analysis Structure* (*ADAS*) [8] is the one of the most prominent atomic databases and consists of an archive of both fundamental and derived atomic data, coupled with a suite of computer programs (see Appendix E for a glossary of the ADAS program series numbers and file formats). The project supports analysis of radiating atoms and ions for a range of plasma conditions: laboratory fusion plasmas, the interstellar medium, the solar atmosphere and technological plasmas. The software suite includes an interactive system with an *Interactive Data Language* (*IDL*) [9] graphical front-end, which facilitates the expeditious generation and/or visualisation of the derived quantities or spectra. The nature of the interactive system means that it provides a learning environment to explore the behaviour of these derived quantities as a function of the plasma parameters. ADAS made some inroads into inclusion of spectral fitting and analysis capability with series 6. The intent was to create a series of dedicated spectral fitting codes, for a set of special features: ADAS602 would be a general purpose, Gaussian line based fitting code (designed, in the first instance, to analyse SOHO-CDS and SOHO-SUMER astrophysical data), ADAS603 would be concerned with fitting of Zeeman/Paschen-Back multiplets using the xPaschen code [10, 11], ADAS604 would deal with the Li-like satellite lines in the vicinity of He-like resonances and ADAS605 would tackle fitting diatomic molecular emission. ADAS602 and ADAS603 were realised as interactive codes with graphical user interfaces and used extensively in data analysis. Fitting and data analysis has been somewhat neglected since (ADAS604 and ADAS605 were never seen through to completion as intended). The set of special feature models themselves, however, are provided within ADAS. Historically the computer programs were written on a case by case basis and there was very little consistency between the way in which these programs are utilised. This meant that access to these important tools was hindered by the fact that any prospective user needed to understand the particular idiosyncrasies of the modelling code in question. Establishing a standard interface to the available special features results in a range of benefits — not least by providing ease of use for the operator. The unified approach that has been taken to achieve this (see Section 3.1) exposes the current special feature models for use by external analysis codes in a structured, consistent manner and, importantly, allows for easy inclusion of future models. Provision of standard access to the models is especially beneficial should multiple special features be required to

be added to a single model and is a necessary step towards creation of the extended-ADAS, generalised modelling and fitting program, FFS (see Section 4.4).

1.2 Object-oriented Modelling for Numerical Fitting

The need to abstract the modelling problem from a specific subset of data has been readily identified in other fields (e.g. modelling of diagenesis, in the geological domain [12, 13]). When it comes to the issue at hand (fitting spectral data), those of a computer science background quickly identify that the physics problem set before them is best served by a modular computational framework. Järvi [14], considered model spectra to be composed of a set of component elements. In terms of computation, an object-oriented approach [15] can parallel this structure, with a top level model object, holding a list of model element objects, which in turn hold a collection of objects representing the parameters of the model element underlying functions. This structure is very sensible and the benefits of the object-orientation are easy to see, e.g. the mathematical representation of each element is encapsulated within the object, which means that a user can intuitively build a fairly complex model from a basis of elementary modules, very quickly. The other advantage that this modular structure brings, is flexibility — new models can re-use the same base elements, with little or, in most cases, no 'hard-coding' of models. Other object-oriented concepts also prove useful, e.g. inheritance; all of the common functionality that each element posseses can be handled by an element superclass, from which all model components are derived — this simplifies the code for the derived classes (i.e. the concrete classes used to construct a model) greatly. There are some design decisions which have not been transferred to the current work (FFS). Firstly, a distinction has been made between 'composite' and 'elementary' components of the model. This seems unnecessary and is perhaps an over-abstraction of the problem. Additionally, the concept of 'composite' models, as discussed by Järvi, are restrictive — they only serve as container objects, with its evaluation function comprising of a summation of the results of the child model element function evaluations. Consideration has not been given to the more complex case of 'operator elements' (discussed in Section 4.2), i.e. container objects which are further functions of their child elements and not simply a linear combination of them. This is of particular importance for efficient calculation of partial derivatives for least-squares fitting (see sec. 4.7).

The parameter class hierarchy of Järvi's work will also not be adopted here, as it does not easily handle the required complexity. In order to deal with the issue of coupled

parameters (i.e. parameters that are related and therefore interdependent) the parameter hierarchy splits from a base parameter class to two main subclasses of 'stored' and 'dependant'. The 'stored' parameters are the set of truly adjustable fitting parameters, while the 'dependant' subclass represents those that are not — they are coupled to the value of another parameter of the system. The dependant parameter class however, only deals with one-to-one coupling. If the dependancy is more complex, then a further subclass must extend the dependant parameter class, over-riding the methods for retrieval of parameter value and derivative. While this solution works, it is not very flexible and requires that a class file is written whenever a coupling function is used for the first time. Consideration of arbitrary coupling between parameters is considered in Section 4.6.

Perhaps the closest to achieving a fully flexible generic modelling and fitting system is the *X-Ray Spectral Fitting Package* (*XSPEC*) [16]. This package has been developed at Nasa's Goddard Space Flight Center since 1983. The assertion made here (that an object-oriented approach to this problem is required) is further vindicated by the fact that this package underwent substantial re-design in 1998 to use this modern computational technique [17]. The latest edition of this application (v. 12 at time of writing) provides several features that are essential in achieving the goal. Firstly, a selection of model element objects are available, from which a model can be constructed. Secondly, Tool Command Language (Tcl) has been used to provide an interactive (command-line) interpreter for the custom *XSPEC* command language — this means that *XSPEC* has capability to define more complex relationships between model elements, than just a linear sum (as in v. 11). Again through the Tcl scripting language, there is support for coupling between model parameters, specified by polynomial expression in terms of other parameters, using a subclass of the model parser object. This is an evolution of the feature in v. 11, where coupling was limited to linear relations between parameters and the use of a separate scripting language to make the specifications eliminates the lack of flexibility in the extensible parameter class solution discussed above. *XSPEC* commands are used to set parameter bounds, fixed / free status etc. It must be noted that the *XSPEC* coupling syntax is somewhat obscure — particularly in that model elements can be named, but parameters cannot. This means that parameters are instead referenced by numerical index, in some arbitrary order defined by that element. The situation is worse if the elements are not named; the index is the position in the concatenated parameter list from all elements. Further to this, the parser assumes that all numerical characters in the range $[1, n]$, where n is the number of parameters, are considered to be references to parameters. If constants in this range are to be specified, they are indicated by the use of a decimal

point.

Routines for performing fitting are included with the package — currently Levenberg-Marquardt [18] and Minuit/Migrad methods [19, 20] are implemented — although annealing methods (see, for example, Kirkpatrick *et al* [21]) and genetic algorithms (originally developed by Holland [22], but see Besset [23] for implementation in Smalltalk and Java) existed in the older version of the code. *XSPEC* does not, however, make an attempt to use analytical partial derivatives where possible — a surprising omission given the associated performance increase (see Section 4.12 for the performance increase experienced in use for FFS). Additionally, no attention is given to optimisation of the user-defined model representation. Symbolic reduction of algebraic formulae has been tackled by two of the prominent, commercial applications Maple [24, 25] and Mathematica [26] and is examined, here, in Section 4.7. Similarly, no attempt is made to find the partial derivatives of the coupling expressions symbolically — this is discussed in Section 4.6.

1.3 Improving on Existing Software

The spectral analysis packages, mentioned here (which are just a small subset of those available) are essentially performing similar tasks, each with their own nuance as they have been tailored to successfully tackle a specific aspect of the modelling / fitting problem. It is more sensible for the core, shared characteristics of these to be unified as far as possible. It should be noted, however, that some of the tailored packages are very good at performing their specified task and are often highly efficient too. Therefore, in order to compete, performance may be compromised, but not to the extent that the advantages that unification offers are overshadowed by a lack of performance. The goal of the present work is to address this very issue and as such, it is the specialist codes that will provide the performance benchmark.

Some systems provide rich, complex modelling routines (e.g. ADAS), but often such models are not easily integrated into the packages that perform the analysis as they have a rather rigid framework, relying on intrinsic functionality. It is often difficult to incorporate external models into analysis packages without substantial re-development of them.

In order to create a successful modelling, fitting and analysis system, it is important to try to incorporate as many of the useful features of existing systems (like those considered above) but also to attempt to avoid as many of their respective negatives as possible. Clearly it is preferable to incorporate the best quality special features into

a model when possible, so the system must be capable of interfacing with external feature generation. Secondly, on this same issue, it is inevitable that the sources of such data (or generating algorithms) will be disparate — this means that an effective scheme must cope with the varied ways in which each is accessed. Many of the issues discussed can, at least partially, be solved by ensuring a high level of modularity. Ensuring modularity brings different benefits in different areas. Firstly, it allows the building and creation of a model to be separate from the fitting and analysis. Secondly it brings versatility to the modelling system; components are no longer locked down to those that are 'built-in' — each of the model elements must simply adhere to a defined interface. Tackling the problem in this manner has been considered by others (e.g. [14]) but the manner in which this has been implemented does not take the concept far enough and is again embroiled in the very same predicament that the modularisation was meant to avoid — rigidity. Other solutions such as *XSPEC* have embraced the object-oriented philosophy and identified the requirement of a separate engine to handle model definition and parameter specification via a scripting language. However, some aspects of the problem have been neglected, mainly regarding optimisation of the model and efficiency of partial derivative calculation for numerical fitting. In this respect, there is a danger that the performance advantage offered by dedicated codes may outweigh the flexibility of a modular framework.

It is intended that the packages AFG (Section 3.1) and FFS (Section 4.4) developed as part of this work embrace the philosophy of the object-oriented solutions, whilst retaining attention on the benchmark set by the various dedicated approaches.

1.4 Thesis Outline

This work describes the development of a flexible modelling system for spectral analysis, particularly in the fusion domain. Application of the package for fitting analysis of a range of spectral signatures, resulting from a diverse range of plasma scenarios, is explored.

Chapter 2 provides a survey of various special features commonly observed in spectra from fusion labs. Consideration is given to the modelling of these features by examination of the underlying atomic physics processes involved. Particular attention is given to modelling of the state populations.

Chapter 3 describes a method of (programmatically) unifying access to codes that produce the special features described in chapter 2 — the ADAS Feature Generator (AFG). This is an important step towards building a useful, modular spectral modelling

system. This chapter also details the accompanying graphical user interface (GUI), ADAS 605 which is comprised of self-generating graphical widget components.

Chapter 4 begins by considering (mathematically) the functions most commonly used to represent spectral lines and their partial derivatives. Examples of combinations of these functions are also inspected. This provides a basis from which the description of a modelling system, the Framework for Feature Synthesis (FFS), can be built. The computational development encompasses the specification of a Model Definition Language (MDL), support for arbitrarily complex model parameter coupling and a method by which poorly specified models can be reduced to optimal form. This chapter also discusses non-linear least-squares fitting using FFS, including *batch fitting* (of large numbers of similar spectra). Finally, there is a study as to the validity of the results of the code and the performance advantages associated with the use of analytic partial derivatives and the model optimisation system are investigated.

Chapter 5 follows the practical implementation of the FFS modelling system for fitting experimental data. The capabilities of FFS are tested progressively, from simple fits of manufactured test data through to exploitation of the coupling system in fusion experimental scenarios where speed of execution is important. Other examples see combined use of AFG and FFS — bringing the advanced special feature models into the analysis.

The final chapter summarises the completed work and looks towards further exploitation of the current capabilities of the system for spectral analysis as well as discussing potential future enhancements.

Chapter 2

Special Feature Modelling for Nuclear Fusion

2.1 Introduction

Whereas basic spectroscopy fits single lines, diagnostic spectroscopy establishes its ability to obtain plasma parameters by fitting connected sets of spectral lines. Lines are connected because of the excited population structure of the emitted atoms and ions. In conventional atomic physics nomenclature, a single spectral line is called a component and arises from a transition from one level of an atom to another. The levels of an atom group, according to angular momentum rules, into terms and then, into larger groups called configurations. The set of levels arising from a term is called a multiplet and the set of multiplets arising from a configuration is called a transition array. At issue, is the relative intensity of the components of a multiplet and of the multiplets within a transition array. These proportions follow from the populations of the levels of the terms and the terms of the configuration. It is these populations which are reactive to the plasma environment, especially electron density and temperature, through collisions. Here the population structure is used to describe the consequence of the interaction of the various collisional and radiative processes. The associated modelling is called collisional-radiative modelling. The objective of this thesis, which is spectral fitting with combinations of complex special features, is intertwined with the details of population modelling. This intertwining, can be illustrated by a simple example. Consider the O IV spectrum and the multiplet $2s^2 2p\ ^2P - 2s2p^2\ ^2P$ at $\sim$ 553 Å. One component is $^2P_{1/2} - {}^2P_{3/2}$ at 553.330 Å. For O^{+3} in a tokamak plasma, the populations of the levels of the upper term $2s2p^2\ ^2P$ are nearly in proportion to their statistical weights. Thus, the relative intensities of the components of the multiplet

have little connection with the plasma parameters. From a collisional point of view, this occurs because the energies of these levels are very close together and in a plasma, ion collisions are extremely efficient in distributing the level populations amongst each other. Plasma conditions are influential in establishing the situation, but the relative intensities are not sensitive. From a collisional-radiative point of view, this is in the so-called 'high-density' local thermodynamic equilibrium (LTE) regime. If the system is followed iso-electronically up to an ion such as, Fe^{+21}, the efficiency of radiative transitions increases and that of collisional transitions decreases and the balance shifts. Additionally, the separation of the levels of the term increase. The relative intensities move away from statistical, becoming sensitive to plasma conditions, and enter the so-called 'collisional-radiative' regime. For the iso-electronic system W^{+69} radiative transitions completely dominate and we enter the so-called 'low-density' (coronal) regime. Comprehensive spectral analysis is designed to be able to analyse in all these regimes and therefore must have in its support structure detailed population calculations. Here, ADAS [8] provides the background population modelling, but it is necessary to understand the balance of processes underpinning the population models to be able to best exploit the spectral analysis. In the following sections, the population modelling is explored in more detail and we consider its completeness for use in all practical scenarios.

It is intended that AFG (Section 3.1) and FFS (Section 4.4) deliver a diagnostic capability for all spectral features in the magnetically confined fusion domain. To structure these and connect them with the selection of features which follow in Section 5 we consider the plasma environments from the point of view of an emitting ion, subject to the collisionality of the thermal plasma in which it lies and also subject to external fields and driving mechanisms which influence it. External fields and directed driving processes such as neutral beams create a direction in the plasma and, as such, makes the orientation of the ion significant. We call this the *orientation problem*.

The bulk of population modelling applies to thermal plasma, without directional effects. The fusion plasma spans a wide range of temperature and consequently, ionisation stages of elements. The Hamiltonian of an ion of low charge state is dominated by the Coulomb interaction, giving a term structure for its energy levels. In high temperature zones of the fusion plasma, high ionisation states of heavier species occur. For such ions, relativistic corrections to the Hamiltonian modify the energy level structure, over the pure Coulomb case. The consequence is that fine-structure energy level separations become significant and collisions with them are differential. The population structure calculation must adjust to these different regimes. This is called the *resolution problem*. The terminology in this context is

somewhat different from spectral resolution, yet connected to it. It is convenient to deal with a number of resolutions called *ls* (term resolution), *ic* (intermediate coupling resolution), *ca* (configuration average resolution) and *bn* (bundle-n resolution). So, *ls* resolution applies to light element ions, the populations of terms are calculated and observations are of whole multiplets. In the *ic* case, the populations of levels are calculated and observations are of the separate individual components of multiplets. *ca* resolution deals with the sum of whole configurations and the observations are of unresolved transition arrays in low resolution spectroscopy. A yet coarser resolution is *bn* resolution where the populations of entire *n*-shells are treated as a whole. This latter resolution is relevant to very highly excited states of atoms and ions, so-called Rydberg states, which become increasingly H-like for all levels of the same n — effectively degenerate. Whereas spectroscopy is normally done at *ic* or *ls* resolution, precision of the population models which underpin the spectral intensities, do require subsidiary calculations at *ca* and *bn* resolution for completeness.

2.2 Issues of Orientation — Zeeman and Stark Spectra

At JET, in the coolest regions of the divertor, plasma electron temperatures, T_e, are as low as 0.5 eV. Consider emitting ions in conditions where the temperature range is $\sim$ 0.5 – 10 eV and the electron density is $\sim$ 10^{13} – 10^{15} cm^{-3}. Such a temperature range of thermal plasma implies ionisation stages of helium, beryllium, carbon and oxygen from neutral to doubly-ionised. The ion temperature, T_i, is of the same order as the electron temperature. Magnetic fields are $\sim$ 3 T which implies a Zeeman component separation of order $\sim$ 2.1 cm^{-1}. The doppler half-width, say for CI at 1 eV is $\sim$ 12.7 cm^{-1} and at 10 eV, $\sim$ 40.1 cm^{-1}. On the other hand, the term separations for CI are of order $\sim$ 11000 cm^{-1}, so the Zeeman perturbation is very small in comparison. This is insufficient for deviations of the magnetic sub-level populations from statistical. It is also noted that spectral analysis of the Zeeman split features normally focusses on a single transition. Therefore, population modelling is not influencing this situation and the requirements of the ADAS database are only the Zeeman / Paschen-Back splitting of individual lines. The handling of this scenario, by FFS-AFG is detailed in Section 5.4. The key parameters for extraction are magnetic field magnitude, magnetic field direction and doppler temperature. In the thermal central confined plasma, with highly ionised ions, the magnetic field, although guiding the ions, has no consequences for the electronic structure of the ions or the electron collisional processes populating them. The Zeeman splitting occurs due to interaction of the magnetic moment of the emitter

with the confining field. The magnetic moment arises from the orbital motion of the electron, as well as the fact that there is an intrinsic spin magnetic moment associated with both the electron and the nucleus. This interaction alters the Hamiltonian of the system, that is to say that it results in a new set of eigenfunctions and associated eigenvalues and therefore, a new set of energy levels. The interaction energy E^{mag} for a magnetic moment $\boldsymbol{\mu}$, in the presence of a magnetic field $\boldsymbol{B}$ is given by:

$$\begin{aligned} E^{\text{mag}} &= -\boldsymbol{\mu} \cdot \boldsymbol{B} \\ &= (\boldsymbol{\mu}_l + \boldsymbol{\mu}_s) \cdot \boldsymbol{B} \\ &= \frac{e}{2m_e} (\boldsymbol{L} + g_e \boldsymbol{S}) \cdot \boldsymbol{B} \\ &= \frac{e}{2m_e} (\boldsymbol{J} + (g_e - 1)\boldsymbol{S}) \cdot \boldsymbol{B} \end{aligned} \tag{2.1}$$

where $\boldsymbol{\mu}_l = -\frac{e}{2m_e}\boldsymbol{L}$ and $\boldsymbol{\mu}_s = -g_e\frac{e}{2m_e}\boldsymbol{S}$ are the magnetic moments due to the electron orbital angular momentum $\boldsymbol{L}$ and intrinsic electron spin angular momentum $\boldsymbol{S}$ respectively. The quantity g_e is the electronic g-factor: 2.0023193043622(15) [27]. Note that the interaction with the nuclear magnetic moment is small and so, neglected here.

For the case of magnetic field interaction which is relatively weak (compared with that of the spin-orbit interaction) states can be labelled as $|S\,L\,J\,M\rangle$ it is possible to express the magnetic interaction energy in terms of quantum numbers as:

$$E_e^{\text{mag}} = \mu_B g_L M B, \tag{2.2}$$

where g_L, the Landé g-factor:

$$g_L = 1 + (g_e - 1)\frac{J(J+1) + S(S+1) - L(L+1)}{2J(J+1)} \tag{2.3}$$

and μ_B, the Bohr magneton:

$$\mu_B = \frac{e\hbar}{2m_e} \tag{2.4}$$

have been introduced. The Landé g-factor arises from taking the scalar product in equation 2.1 as (when visualised as a vector model) the $\boldsymbol{L}$ and $\boldsymbol{S}$ precess around the total angular momentum vector $\boldsymbol{J}$.

Clearly, the interaction energy is directly related to the strength of the magnetic field, this in turn means that the spacing between components of the resulting Zeeman multiplet spectra is useful as a local diagnostic for magnetic field strength. The Zeeman component lines are of linear, left and right circular polarisation depending

upon whether $M = M'$, $M = M' + 1$ and $M = M' - 1$. The emissivities are then given by:

$$\begin{aligned} M = M' : &\quad \frac{1}{4\pi} \sin^2\theta A_{\gamma LS\,JM \to \gamma L'S'\,J'M'} N^{+z}_{\gamma LS\,JM} \\ M = M' + 1 : &\quad \frac{1}{8\pi}(1 + \cos^2\theta) A_{\gamma LS\,JM \to \gamma L'S'\,J'M'} N^{+z}_{\gamma LS\,JM} \\ M = M' - 1 : &\quad \frac{1}{8\pi}(1 + \cos^2\theta) A_{\gamma LS\,JM \to \gamma L'S'\,J'M'} N^{+z}_{\gamma LS\,JM} \end{aligned} \tag{2.5}$$

where θ is the angle between the line of sight and the magnetic field direction, $N^{+z}_{\gamma LS\,JM}$ is the population of the upper state of the transition and $A_{\gamma LS\,JM}$ is the spontaneous emission coefficient from that state [8]. Assuming statistical proportion populations for the magnetic sub-levels:

$$N^{+z}_{\gamma LS\,JM} = \frac{1}{(2J+1)} N^{+z}_{\gamma LS\,J} \tag{2.6}$$

If the magnetic field strength is of a higher magnitude, we can no longer consider the orbital angular momentum L and spin angular momentum S to be coupled as they become more strongly coupled (independently) to the external magnetic field. In this regime, the splitting differs from the weak magnetic field results and gives different spectral patterns. The name given to the strong magnetic spectral line splitting is the Paschen-Back effect. We can return to equation 2.1 for the interaction energy as before, but now that the spin-orbit interaction is negligible compared with the interaction with the external field i.e. no longer in an LS-coupling situation, the states may be labelled as $|S\,M_S\,L M_L\rangle$ and the interaction energy in terms of quantum numbers represented as:

$$E_e^{\text{mag}} = \mu_B B\,(M_L + g_e M_S)\,. \tag{2.7}$$

Figure 2.1 shows results from a simulation code that performs calculation of the Zeeman splitting i.e. calculates the locations of the wavelengths of the component lines. Another program then takes the zero width lines and applies Doppler thermal broadening effects to produce a more realistic spectra.

A markedly contrasting situation is the beam penetrated plasma and the neutral atoms of the beam. Neutral beams are principally used for heating fusion plasmas, but on occasion can be used in a diagnostic role. The beams are normally comprised of hydrogen isotopes, although helium is sometimes present. At JET, deuterium is typically the main neutral atom species, as it is the fuel. There are two types of beam source, designed to accelerate either positive or negative ions up to the required

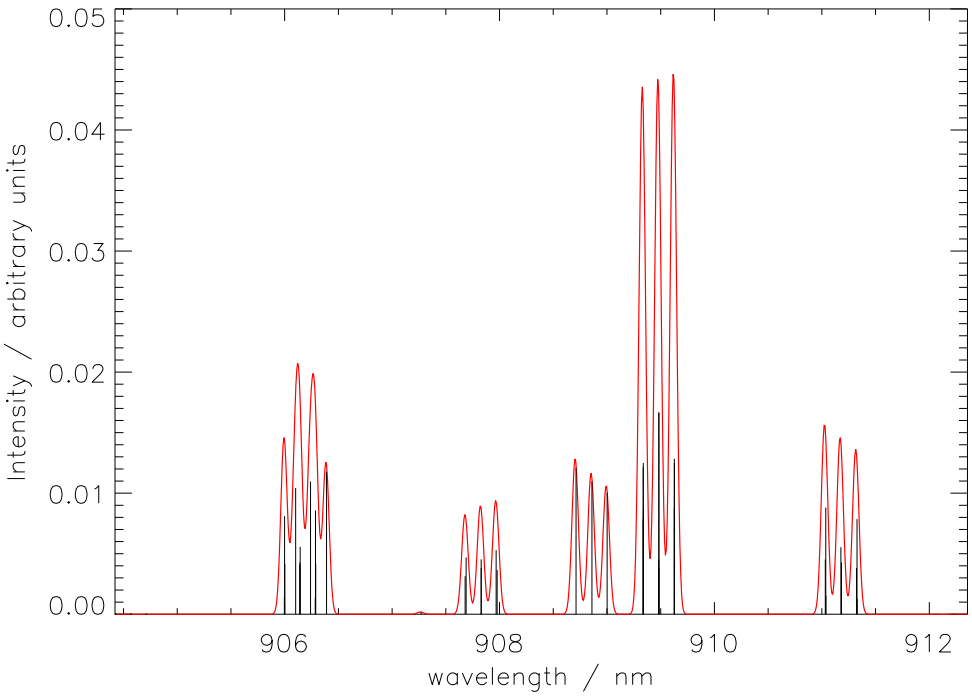


Figure 2.1: Simulation of the Zeeman multiplet feature for CI (2p3s ^{3}P – 2p3p ^{3}P) with ADAS603, for a magnetic field of 2.5 T with a viewing angle of 90 ° to the magnetic field. The calculated line intensities are then Doppler broadened using ADAS utility routine c5dplr. The width of the lines correspond to an electron temperature of 12 eV. The *un-broadened* components coming from ADAS 603 are over-plotted in black — note that this output has been scaled.

beam energy before neutralisation. The so-called 'positive-ion' sources have been used historically in fusion and provide beam particle energies of 30 – 80 keV amu^{-1}. The neutral beams from such positive ions sources have half and third energy components arising from molecular species present in the source discharge. Negative-ion sources, which will be used for ITER [28] have a single energy component in the beam and typically have particle energies > 100 keV amu^{-1}. Neutral hydrogen beam atoms can undergo a charge transfer reaction with the plasma ions and with impurity bare nuclei in the central plasma, which leads to so-called 'charge-exchange' emission. That is transitions between highly excited shells of the H-like impurity ions after the reaction, which are in the visible spectrum. A beam atom at 30 keV amu^{-1} encountering a plasma ion (deuteron) at an average thermal energy of $\sim$ 10 keV has a significantly different relative collision speed for co and counter collisions and so, different cross-sections. Subsequently, the receiver ion directed velocity is reflected in the doppler shift of its emission. These cross-section and Doppler effects lead to the necessity of special spectral fitting and inferences for charge-exchange spectroscopy. Dedicated fitting codes such as CXSFIT [6] handle charge-exchange spectroscopy, CXSFIT is effective and widely used.

The beam atoms also emit. For neutral hydrogen beams, Balmer-alpha is the principally observed feature, although Balmer-beta and Balmer-gamma are also observed. As the beam atoms are traversing the magnetic field at high velocity, they experience a Lorentz electric field $\mathscr{E} = \boldsymbol{v} \times \boldsymbol{B}$ which is of order $\sim$ 100 kV cm^{-1}. For a central plasma electron temperature > 5 keV electron collisions speeds are very fast and are in the high energy Born regime. For beam ions colliding with a plasma ion, the collision speed is close to the speed of the beam atom itself, which typically places the cross-section close to its peak, or just entering the Born regime. The electron collision cross-sections are of order $1/50$ of the ion collision cross-sections (for JET beam parameters). Whereas the electron collisions, because of their high speeds are essentially isotropic from the point of the target ions, for the ion collisions this is certainly not the case. From the point of view of the target, the ion colliders come from a rather narrow cone of attack. This cone of attack is differential between different masses of colliders. That is to say that there will be a wider cone for plasma protons and a narrower cone for the heavier impurity ions, such as carbon. So, in this situation the directional ion collisions dominate and the collisional rates are differential in collider mass. The Lorentz electric field is highly perturbative (linear Stark effect) on the nearly degenerate hydrogenic levels of each n-shell. The hydrogen eigenstates are distorted in the direction of the electric field and the isolated atom l-quantum number is no longer 'good'. The Stark states are labelled by the nkm quantum numbers, where m

is the usual azimuthal quantum number and k is parabolic arising from the Runge-Lenz constant of motion [29]. The electron cloud of negative k states are pulled in the direction of the electric field and positive k-states in the opposite direction. So, the degenerate isolated hydrogen atom states of a given n-shell are split into a number of separate levels, with a basic separation of $E^{\text{elec}} = \frac{3}{2}nk\mathscr{E}$ [30], called a Stark manifold. The set of transitions between the $n = 2$ and $n = 3$ Stark manifolds are shown in Fig. 2.2. The collision limit for this system is $n \sim 4$, so a population model is required for at least up to $n = 4$. The populations of the km sub-states are not in statistical proportions in general, although full statistical mixing can be approached in the fusion domain. The spectral emission from the hydrogen beam atom, because of its alignment to the field, is polarised and directional. So, a collisional-radiative population model must include calculation of the perturbed Stark states, transition probabilities between them and directed ion collisions of various mass particles with them. The polarisation and directional characteristics of the emission must be evaluated. This implies a large number of parameters entering the analysis of the beam emission. These include: magnetic field strength, direction, beam velocity, ion charge and temperature, ion mass, spectrometer line of sight and any instrumental polarisation characteristics. ADAS has such a collisional-radiative model — ADAS305, which is accessible through AFG (see Section 3.1).

Although the theory of Stark splitting of atomic states is outlined in standard quantum mechanics texts [30, 31], some expositional material is included here for completeness.

The atoms in the neutral beam injected into tokamak devices cross the confining magnetic field. When passing through this field, a Lorentz electric field $\mathscr{E} = \boldsymbol{v} \times \boldsymbol{B}$ exists in the reference frame of the atom. The electric field perturbs the Hamiltonian of the system, resulting in an alteration of the previously degenerate energy levels.

If we consider a hydrogen atom within an electric field $\mathscr{E}$ acting in the negative z-direction, the potential energy of the electron is given by:

$$U = -\frac{e^2}{4\pi\epsilon_0 r} - e\mathscr{E}z. \tag{2.8}$$

The time independant Schrödinger equation for the perturbed atom is:

$$E\boldsymbol{\psi} = \hat{H}\boldsymbol{\psi} \tag{2.9}$$

$$= \left(\frac{-\hbar^2}{2m_e}\nabla^2 + U\right)\boldsymbol{\psi}. \tag{2.10}$$

We now consider this problem in parabolic co-ordinates, such that the conventional

Cartesian axis values are given by:

$$x = \sqrt{\xi\eta}\cos\phi, \quad y = \sqrt{\xi\eta}\cos\phi, \quad z = \frac{1}{2}(\xi - \eta). \tag{2.11}$$

Note that $r = \sqrt{x^2 + y^2 + z^2} = \frac{1}{2}(\xi + \eta)$.

The Laplacian operator (acting on a function $\boldsymbol{\psi}$) is defined as:

$$\nabla^2\boldsymbol{\psi} = \frac{1}{h_1 h_2 h_3}\left(\frac{\partial}{\partial u_1}\left(\frac{h_2 h_3}{h_1}\frac{\partial}{\partial u_1}\right) + \frac{\partial}{\partial u_2}\left(\frac{h_1 h_3}{h_2}\frac{\partial}{\partial u_2}\right) + \frac{\partial}{\partial u_3}\left(\frac{h_1 h_2}{h_3}\frac{\partial}{\partial u_3}\right)\right)\boldsymbol{\psi}, \tag{2.12}$$

where h_1, h_2, h_3 are the scale factors given by $h_i = \sqrt{\sum_{j=1}^{n}\left(\frac{\partial x_j}{\partial p_i}\right)^2}$ for a set of parameterised functions, x_i (i.e. $x_i = f_i(p_1, p_2, \ldots p_3)$).

In this case, the scale factors are:

$$h_\xi = \frac{1}{2}\sqrt{\frac{\eta}{\xi} + 1}, \quad , h_\eta = \frac{1}{2}\sqrt{\frac{\xi}{\eta} + 1} \quad , h_\phi = \sqrt{\xi\eta}. \tag{2.13}$$

and by substitution into equation 2.12, the Laplacian becomes:

$$\nabla^2\boldsymbol{\psi} = \frac{4}{(\eta + \xi)}\left(\frac{\partial}{\partial\xi}\left(\xi\frac{\partial}{\partial\xi}\right) + \frac{\partial}{\partial\eta}\left(\eta\frac{\partial}{\partial\eta}\right) + \frac{1}{4}\left(\frac{1}{\xi} + \frac{1}{\eta}\right)\frac{\partial^2}{\partial\phi^2}\right)\boldsymbol{\psi}. \tag{2.14}$$

In this co-ordinate system, the potential energy is described by:

$$U = -\frac{e^2}{2\pi\epsilon_0(\xi + \eta)} - \frac{1}{2}e\mathscr{E}(\xi - \eta). \tag{2.15}$$

Re-arranging 2.10 to get $\left(\nabla^2 + \frac{2m_e}{\hbar^2}(E - U)\right)\boldsymbol{\psi} = 0$ and using 2.14 and 2.15, we can write the (time independant) Schrödinger equation as:

$$\begin{aligned}\left(\frac{\partial}{\partial\xi}\left(\xi\frac{\partial}{\partial\xi}\right) + \frac{\partial}{\partial\eta}\left(\eta\frac{\partial}{\partial\eta}\right) + \frac{1}{4}\left(\frac{1}{\xi} + \frac{1}{\eta}\right)\frac{\partial^2}{\partial\phi^2}\right)\boldsymbol{\psi} \\ + \frac{m_e}{2\hbar^2}\left(E(\xi + \eta) + \frac{e^2}{2\pi\epsilon_0} + \frac{1}{2}e\mathscr{E}(\xi^2 - \eta^2)\right)\boldsymbol{\psi} = 0.\end{aligned} \tag{2.16}$$

If the wavefunction is considered separable in ξ and η, i.e. that $\boldsymbol{\psi} = f(\xi)g(\eta)e^{im\phi}$, then

by substitution the following is obtained:

$$g(\eta)\frac{\partial}{\partial\xi}\left(\xi\frac{\partial f(\xi)}{\partial\xi}\right)+f(\xi)\frac{\partial}{\partial\eta}\left(\eta\frac{\partial g(\eta)}{\partial\eta}\right)-\frac{m^2}{4}\left(\frac{1}{\xi}+\frac{1}{\eta}\right)f(\xi)g(\eta) \tag{2.17}$$

$$+\frac{m_e}{2\hbar^2}\left(E(\xi+\eta)+\frac{e^2}{2\pi\epsilon_0}+\frac{1}{2}e\mathscr{E}(\xi^2-\eta^2)\right)f(\xi)g(\eta)=0. \tag{2.18}$$

Dividing through by $f(\xi)g(\eta)$ and separating terms of ξ from those of η:

$$\frac{\partial}{\partial\xi}\left(\xi\frac{\partial f(\xi)}{\partial\xi}\right)\frac{1}{f(\xi)}-\frac{m^2}{4\xi}+\frac{m_e}{2\hbar^2}\left(E\xi+\frac{e^2}{2\pi\epsilon_0}+\frac{1}{2}e\mathscr{E}\xi^2\right)$$
$$=-\frac{\partial}{\partial\eta}\left(\eta\frac{\partial g(\eta)}{\partial\eta}\right)\frac{1}{g(\eta)}+\frac{m^2}{4\eta}-\frac{m_e}{2\hbar^2}\left(E\eta-\frac{1}{2}e\mathscr{E}\eta^2\right) \tag{2.19}$$

To satisfy this equation, both sides must equal a constant, such that we obtain (after re-arrangement) a pair of ordinary differential equations:

$$\frac{d}{d\xi}\left(\xi\frac{df(\xi)}{d\xi}\right)+\frac{m_e}{2\hbar^2}\left(E\xi+2e^2\beta_1-\frac{m^2\hbar^2}{2m_e\xi}+\frac{1}{2}e\mathscr{E}\xi^2\right)f(\xi)=0 \tag{2.20}$$

and

$$\frac{d}{d\eta}\left(\eta\frac{dg(\eta)}{d\eta}\right)+\frac{m_e}{2\hbar^2}\left(E\eta+2e^2\beta_2-\frac{m^2\hbar^2}{2m_e\eta}+\frac{1}{2}e\mathscr{E}\eta^2\right)g(\eta)=0, \tag{2.21}$$

where $\beta_1=\frac{1}{4\pi\epsilon_0}-\frac{\hbar^2C}{e^2m_e}$ and $\beta_2=\frac{\hbar^2C}{e^2m_e}$.

The solution of these 1D-Schrödinger equations results in the eigenvalues, which are the set of parabolic quantum numbers n_1, n_2, m. Often the definition of parabolic quantum number $k=n_1-n_2$ is made such that the basic energy separation is given by:

$$E^{\text{elec}}=\frac{3}{2}nk\mathscr{E} \tag{2.22}$$

where $n=n_1+n_2+|m|+1$.

With the degeneracy removed, there are now more states between which transitions can be made, which give rise to more spectral lines. This is described as the original spectral line being split into Stark components and the group of lines are collectively identified as a Stark multiplet.

In the fusion neutral beam case, densities are such that the collision limit, from the point of view of the beam atoms, is $n\sim4$. Field ionisation starts to become significant around $n\sim10$, therefore it is ion-impact that is the dominant mechanism in this case. Thus a universal treatment of both magnetic and electric field effects can be handled by a perturbative treatment in the usual spherical nlm_lm_s representation. This is the

method exploited by the ADAS code, ADAS305. However, extended development is underway due to anxieties over asymmetries of the polarised components of the Stark feature in observed spectra. Field ionisation could be responsible for such asymmetries. The extensions to ADAS305 will mean that the full parabolic treatment, detailed above, will be used. These modifications will allow ADAS to tackle lower collisionality regimes such as beam–gas target collisions. Development is still at an early test phase and new JET spectra to probe the effects are not yet available.

In this work, it is the older, established version of ADAS305 that will be used to produce the theoretical spectra representing Stark multiplets. The ADAS program also takes into consideration experimental line of sight and Doppler shifts the multiplet accordingly. Figure 2.2 shows an example of such a synthetic spectrum modelling the Stark effect. The observed spectra are further complicated by the fact that the ion production process for the injectors actually results in three deuterium components, that is to say, D^+, D_2^+ & D_3^+. The difference in mass means that they are entering the device with different velocities and so, are Doppler shifted by different amounts. This means that the spectra will comprise of three overlapping Stark multiplets. It should be noted that there may be further experimental complications to account for in the simulation. It is quite possible that a fusion device may have multiple injectors, with different pathways. The angle at which these paths traverse the line of sight may differ and so to will the observed Doppler shifts of the resulting multiplets. This means that the observed spectra could show further overlap with the spectroscopic signal of another beam.

2.3 Thermal Emission and Resolution

Most spectroscopy is of thermal plasma, that is plasma for which a local Maxwellian electron temperature can be specified. It is these thermal electrons which drive the excitation of the impurity ions in the plasma. The population of the excited states of an ion are therefore calculated by including all the collisional rate coefficients and their inverse at the the local temperature and the radiative transitions, between a manifold of states. It is useful at this point to introduce the basic mathematical representations involved, such that notation and nomenclature is clear.

Consider the populations of ionisation stage z, separated into the metastable populations N_ρ^{+z}, indexed by the Greek letter ρ, and ordinary excited populations N_i^{+z}, indexed by the Roman letter i. The stage z has adjacent stages $z-1$ and $z+1$, its *child* and *parent*, with metastable populations labelled as N_μ^{+z-1} and N_ν^{+z+1} respectively. The time-

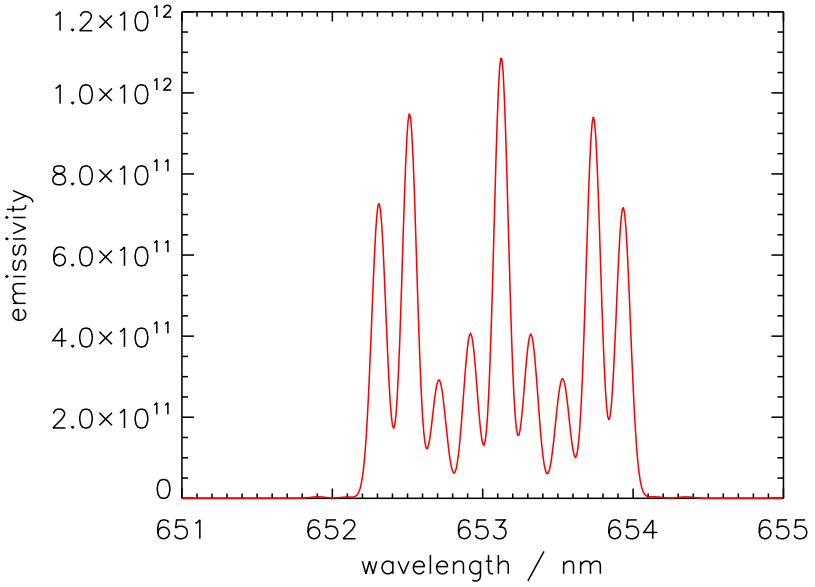


Figure 2.2: The ADAS305 simulation for a deuterium plasma (T_e = 4.44 keV, N_e = 2.49×10^{19} m^{-3}) with a deuteron beam of energy 40 keV amu^{-1}, density 4.27×10^{15} m^{-3}. The magnetic field B = 3.39 T, with direction defined by a normalised vector (0.788, 0.005, 0.615). Similarly, the observational line of sight is described by (0.870, -0.047, 0.491). Scale factors of 0.51 and 1.0 have been applied to the σ and π components respectively.

dependent equations 2.23 of the populations are written in matrix/suffix form[1], where we have omitted coupling to more distant ionisation stages.

$$\frac{d}{dt}\begin{bmatrix} N_\mu^{+z-1} \\ N_\rho^{+z} \\ N_i^{+z} \\ N_\nu^{+z+1} \end{bmatrix} = \begin{bmatrix} \mathcal{C}_{\mu\mu'} & N_e\mathcal{R}_{\mu\sigma} & 0 & 0 \\ N_e\mathcal{S}_{\rho\mu'} & C_{\rho\sigma} & C_{\rho j} & N_e r_{\rho\nu'} \\ 0 & C_{i\sigma} & C_{ij} & N_e r_{i\nu'} \\ 0 & N_e S_{\nu\sigma} & N_e S_{\nu j} & \mathcal{C}_{\nu\nu'} \end{bmatrix} \begin{bmatrix} N_{\mu'}^{+z-1} \\ N_\sigma^{+z} \\ N_j^{+z} \\ N_{\nu'}^{+z+1} \end{bmatrix} \tag{2.23}$$

This means that these equations are actually complete only for the stage z. Note that we have not shown explicitly the ordinary populations of stages $z-1$ and $z+1$ and that some of the sub-matrices are shown as script letters (eg. $\mathcal{C}_{\mu\mu'}$ and $\mathcal{R}_{\mu\sigma}$) whereas others are shown as standard letters (eg. $C_{\rho\sigma}$ and $S_{\nu j}$). Technically, this is because a 'quasi-static' assumption has been made about the ordinary populations of the stages $z-1$ and $z+1$ and the influence of their ordinary populations has been condensed onto their metastable populations. Note that the on-diagonal elements of C and $\mathcal{C}$ are negative quantities. C and $\mathcal{C}$ are linear in the electron density N_e. We wish to demonstrate this procedure for the ordinary populations of the stage z.

The quasi-static assumption is that $dN_i^{+z}/dt = 0$ which means that these ordinary populations are assumed in instantaneous statistical equilibrium with the various metastable populations[2]. This implies that

$$\begin{bmatrix} N_{\mu'}^{+z-1} \\ N_\sigma^{+z} \\ N_j^{+z} \\ N_{\nu'}^{+z+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -C_{ji}^{-1}C_{i\rho} & -N_e C_{ji}^{-1} r_{i\nu} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} N_\mu^{+z-1} \\ N_\rho^{+z} \\ N_\nu^{+z+1} \end{bmatrix} \tag{2.24}$$

and then

$$\frac{d}{dt}\begin{bmatrix} N_\mu^{+z-1} \\ N_\rho^{+z} \\ N_\nu^{+z+1} \end{bmatrix} = \begin{bmatrix} \mathcal{C}_{\mu\mu'} & N_e\mathcal{R}_{\mu\sigma} & 0 \\ N_e\mathcal{S}_{\rho\mu'} & \mathcal{C}_{\rho\sigma} & N_e\mathcal{R}_{\rho\nu'} \\ 0 & N_e\mathcal{S}_{\nu\sigma} & \mathcal{C}_{\nu\nu'} \end{bmatrix} \begin{bmatrix} N_{\mu'}^{+z-1} \\ N_\sigma^{+z} \\ N_{\nu'}^{+z+1} \end{bmatrix} \tag{2.25}$$

where we have the definitions of the effective metastable cross-coupling coefficient, effective recombination coefficient and effective ionisation coefficients between the

[1] In the following equations Einstein summation convention over repeated indices is adopted.

[2] We assume no direct populating mechanism from stage $z-1$ to ordinary excited state of stage z.

various metastables of stages z, $z-1$ and $z+1$:

$$\begin{aligned} Q^{cd}_{\sigma\to\rho} &\equiv \mathcal{C}_{\rho\sigma}/N_e &&= \left(C_{\rho\sigma} - C_{\rho j}C^{-1}_{ji}C_{i\sigma}\right)/N_e \\ A^{cd}_{\nu\prime\to\rho} &\equiv \mathcal{R}_{\rho\nu\prime} &&= r_{\rho\nu\prime} - C_{\rho j}C^{-1}_{ji}r_{i\nu\prime} \\ S^{cd}_{\sigma\to\nu} &\equiv \mathcal{S}_{\nu\sigma} &&= S_{\nu\sigma} - S_{\nu j}C^{-1}_{ji}C_{i\sigma}. \end{aligned} \tag{2.26}$$

Also there is formally an addition to the $\mathcal{C}_{\nu\nu\prime}$ term called the parent metastable cross-coupling coefficient

$$X^{cd}_{\nu\prime\to\nu} \equiv -\left(S_{\nu j}C^{-1}_{ji}r_{i\nu\prime}\right)/N_e \tag{2.27}$$

which we assume had already been incorporated. The superscript 'CD' denotes 'collisional-dielectronic' — a historic synonym for 'collisional-radiative' and parallels the naming conventions in the ADAS data format *adf11* used for such data.

The emission per unit volume per unit time in an observed spectrum line $j \to k$ with upper ordinary excited level population N^{+z}_j may be written as

$$A_{j\to k}N^{+z}_j = \sum_\sigma \mathcal{PEC}^{(exc)}_{\sigma,j\to k}N_eN^{+z}_\sigma + \sum_\nu \mathcal{PEC}^{(rec)}_{\nu,j\to k}N_eN^{+z+1}_\nu \tag{2.28}$$

identifying $\mathcal{PEC}^{(exc)}_{\sigma,j\to k}$, the usual *excitation photon emissivity coefficient*, driven by the metastable σ of the stage z, and $\mathcal{PEC}^{(rec)}_{\nu,j\to k}$, the *recombination photon emissivity coefficient*, driven by the metastable ν of the stage $z+1$.

A set of $\mathcal{PEC}$s for an ion, in a spectral interval, from above is the key input for spectral fitting. In general, the spectral interval will include spectral lines from more than one ionistation stage, so the relative intensities from different stages are part of the predicted model. These relative intensities are obtained from the ionisation balance of the ions of different stages of a given element and it is the $\mathcal{GCR}$ data which allows that balance to be calculated. That is to say that AFG models, discussed in Section 3.1, will draw $\mathcal{PEC}$ and $\mathcal{GCR}$ ionisation state data (A^{cd} and S^{cd}). In equilibrium ionisation balance, the fractional abundances of the various ionisation stages are a function of local electron temperature and density, as are the $\mathcal{PEC}$. In practice, in a fusion plasma, transport disturbs this simple ionisation balance. A parameter for the spectral fitting will be the degree of 'dis-equilibrium'. It is necessary to discuss in more detail the various states (i, ρ etc.) in the above equations. In *ic* resolution $i \equiv (\gamma_p S_p L_p)\ nlS\,LJ$ where γ_p is the configuration of the parent (core) state; in *ls* resolution $i \equiv (\gamma_p S_p L_p)\ nlS\,L$; in *ca* resolution $i \equiv (\gamma_p nl)$; in *bn* resolution $i \equiv (\gamma_p S_p L_p)\ n$ or $i \equiv (\gamma_p S_p L_p J_p)n$ for light elements and heavy elements respectively. ADAS

emissivity coefficients of format *adf15* occur in data sets of the various resolutions and include transitions between states of these various forms. The present work requires access to these datasets for the construction of an AFG model. It is appropriate to discuss briefly the completeness of various ADAS data with a view to spectral fitting. This is because a high resolution spectrometer operating at longer wavelengths will resolve components of a multiplet, that is to say transitions between J levels. However, from a population structure point of view, the separations from the term centres are very small and have negligible influence on the collisional cross-section problem. The component cross-sections are in simple algebraic proportions to the multiplet cross-sections, obtained from combinations of Wigner coefficients. It is appropriate then to carry out the population calculation in ls resolution. This may be 'unhelpful' from the spectral fitting perspective as the available dataset within ADAS may not supply the observed individual components of a given multiplet. The database must make a trade-off between including fully resolved data (which is not useful for the modelling of populations) and the issues associated with storing an even larger database than at present. The question arises whether to archive these data or perform on-the-fly calculation of the fine structure components from the coarser set.

2.4 Stark Broadening and Series Limits: Balmer and Paschen Series

The Balmer series is the group of spectral lines which are a result of principal quantum number transitions to $n = 2$, in hydrogen. The divertor region of (Joint European Torus) JET is able to exhibit an intense Balmer spectrum due to the lower temperature found here — allowing electrons to remain bound to their respective nuclei. The feature is even more prominent when neutral gas is introduced into the divertor, under detached operation (see Section 5.3), as recombination reactions take place. The feature is sensitive to both temperature and density and can be a useful diagnostic for these quantities in the divertor where direct measurement is difficult, or impossible by other means. Analysis of Balmer emission has been successfully carried out on spectroscopic data recorded from the JET divertor [32, 33].

The electron temperature can be inferred from the ratio of two lines in the series, or the ratio of a well resolved line to the continuum emissivity. However, analysis is best served by use of an accurate theoretical representation of the entire Balmer spectra fit to the experimental data in order to determine the density and temperature. ADAS (specifically ADAS311 in this case) provides the required capability to accurately

calculate the expected atomic level populations for a given input set of density and temperature. The resulting output can be archived and then used as base set of data for interpolation during rapid spectral fitting. Simultaneous fitting of several lines in the series should give higher confidence in the estimate of temperature.

The electron density can in fact be estimated via the Inglis-Teller relation $\log n_e = 23.06 - 7.5 \log n_{max}$ [34] where n_{max} is the value of the principal quantum number of the upper state in the transition that results in the last discernible Balmer line in the series. The density can also be determined from the width of the high-n Balmer series lines; their emission is subject to what is known as *pressure broadening*, which is due to interaction with other particles in the vicinity of the emitter. Specifically, it is the Coulomb interaction of ions and electrons with the emitting atoms leading to Stark splitting (see Section 2.2) of the atomic states of emitter. There are two limiting cases for this mechanism depending on whether the perturber is slow or fast moving. Fast moving perturbers lead to broadening collisions that are on a short timescale and so, are independent of each other and can be handled individually; the so called 'impact approximation'. Slow moving perturbers can be thought of as stationary from the emitter's point of view and so, the emitter can be considered to be located within an electric field created by all of the the 'stationary' perturbers in its proximity — the 'quasi-static approximation'. Typically the electrons have a much higher thermal velocity than the ions and so lend themselves to the dynamic impact approximation and the slower moving ions to the quasi-static ion micro-field approximation. It is possible to model the full Stark field as experienced by the emitter, considering the emitter in a quasi-static field created by surrounding perturber ions, with a high frequency fluctuation due to electron collisions. The Stark broadening of the Balmer lines results in Lorentzian profiles, with a characteristic width, dependant upon the density of the plasma. It is this property that makes the Balmer lines a useful density diagnostic. This is discussed extensively by Griem [35, 36] and by Oks [37].

Under the quasi-static micro-field regime, Griem derives that this width is given by:

$$w_L \approx \frac{12 Z_p \hbar}{Zm} \left(n_i^2 - n_f^2\right) n_e^{\frac{2}{3}} \tag{2.29}$$

In addition to this method of density inference, there also exist complex computer codes that model the micro-field in the vicinity of the emitting atom, such as the PPP Stark broadening code [38, 39]. A model spectrum, combining the use of ADAS population calculation data with Stark broadening from PPP is shown in Fig. 2.3.

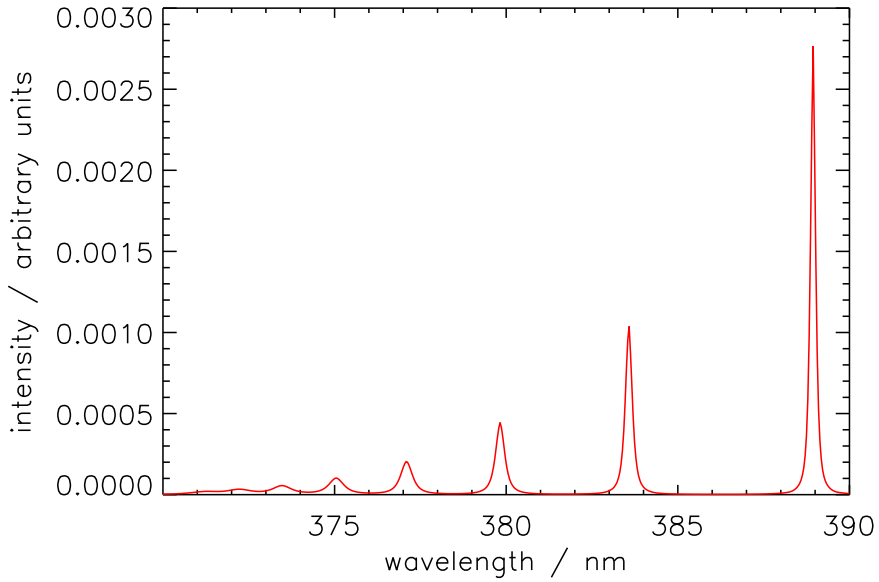


Figure 2.3: A simulated spectrum showing the Balmer series in deuterium (lines n=8→2 through to n=15→2 are present) $T_e = 2.15$ eV and $n_e = 1.0 \times 10^{14}$ cm^{-3}. ADAS provides accurate population modelling to establish the line intensities, whilst the PPP line broadening code has been used to generate the expected line shape. Note that no background baseline is included here.

2.5 Diatomic Molecular Emission

The cooler divertor temperatures also allow for the existence of molecules — molecular atoms will readily dissociate at higher temperature. The expected molecular emission species are homonuclear diatomic molecules of fuel nuclei (i.e. deuterium and tritium), impurity atoms (such as carbon and beryllium), or heteronuclear combinations thereof. Spectral emission from such species have been observed at JET, e.g., CD, C_2, BeD [40, 41, 42]. Under normal circumstances, the origin of such molecules is interaction with the material surfaces in the tokamak, via physical or chemical sputtering processes. However, for experimental purposes, the presence of these molecules may be deliberate; impurity gases (sometimes of more complex molecules, e.g. hydrocarbons, such as deuterated methane, CD_4, which are quickly catabolised into diatomics) are puffed into the plasma [43]. As the rotational and vibrational energies of a molecular system are very much dependent upon the masses of the constituent atoms, there is a marked difference in the ro-vibrational band structure associated with the various isotopologues of a given diatomic. Therefore, it has been suggested that examination of the observed superposition of BeD/BeT and CD/CT molecular features may be used to infer the D/T ratio [44]. Due to the complex nature

of the molecular signature, this sort of analysis calls for a forward modelled special feature. The specialised simulation code *CALCAT* [45], developed by H. Pickett at the Jet Propulsion Laboratory (JPL), can be used to acquire energies and relative intensities of transitions between the ro-vibronic states of the molecules under investigation. This program takes input of a range of molecular constants and the fitting parameters are the rotational and vibrational temperatures. The calculated vibrational band intensities can be scaled in Boltzmann proportions (of the upper vibrational states, according to the vibrational temperature), or scaled via a set of additional free parameters. The complete set of intensities can then be convolved with an instrumental function to produce a realistic synthetic spectra.

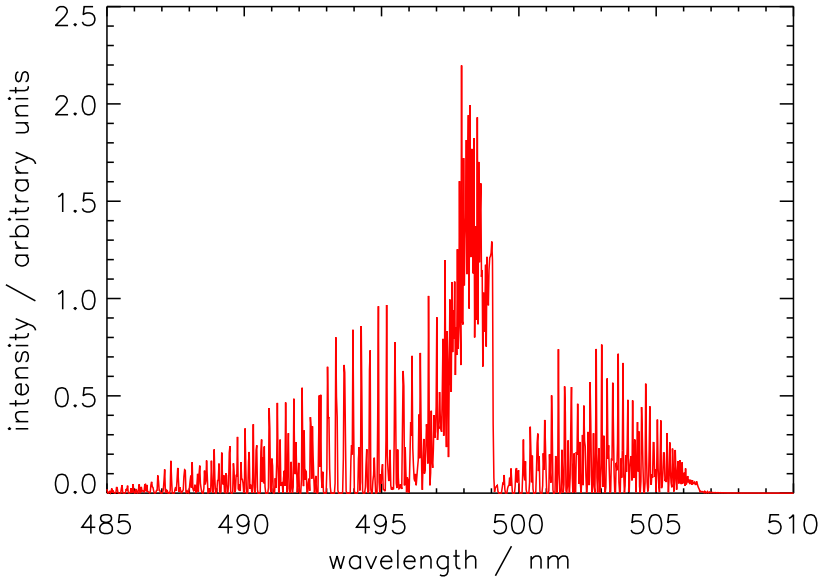


Figure 2.4: A simulated BeD molecular spectrum ($A^2\Pi$ – $X^2\Sigma$) as produced using CALCAT (via AFG). Rotational and vibrational temperatures are 0.3 eV. The result of the CALCAT program has been convolved with a Gaussian function, with full-width at half maximum of 3 Å to emulate instrumental broadening.

2.6 Heavy Species Envelope Emission

In future fusion devices (e.g. ITER) heavy elements may be used as wall material and tungsten will certainly be used as a divertor material due to its low sputtering yield [46]. In preparation for future fusion devices, experiments are carried out on materials such as tungsten on the current generation of machines. This is particularly true of *ASDEX-U* where the machine has evolved such that all of the

Plasma Facing Components (PFCs) are now tungsten [47]. This means that a special feature, modelling this type of emission, is highly relevant for application for current experiments(e.g. [48]) and beyond.

The heavy element impurities differ from lighter elements in that they are not fully ionised even at the high electron temperatures ($T_e \sim 10$ keV) found at the core of fusion devices. To further complicate matters, the elements often exist in ionisation stages which are very complex, containing hundreds of contributing levels and several thousand transitions in a fairly narrow wavelength range. The plethora of line transitions taking place with similar wavelengths results in spectral features with a great deal of line blending, making individual lines hard to distinguish; see, for example, the simulated spectrum in Fig. 2.5.

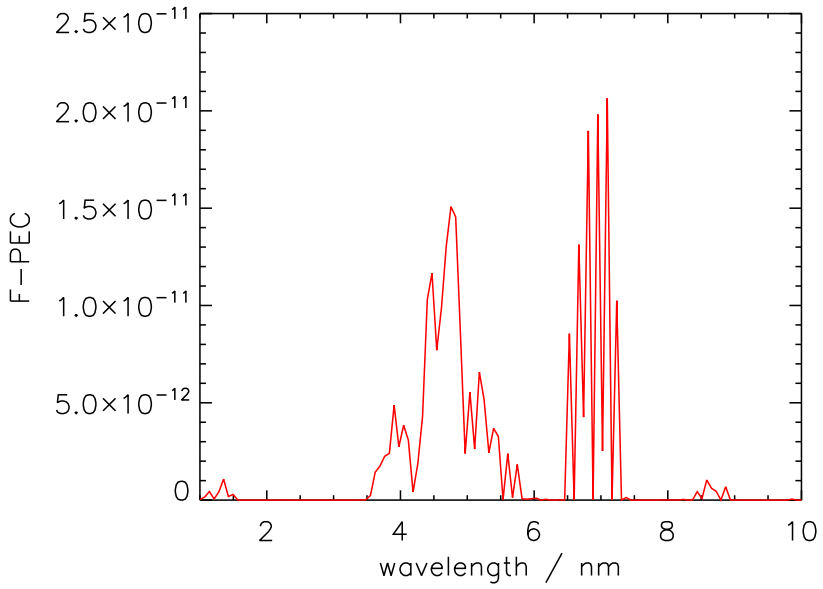


Figure 2.5: $\mathcal{F}-\mathcal{PEC}$ for tungsten ions, ionisation stages W^{54+} to W^{73+}. Equal abundance of each ionisation stage has been assumed. The vertical axis shows the value of the envelope feature photon emissivity coefficients ($\mathcal{F}-\mathcal{PEC}$). $T_e = 2$ keV and $N_e = 1 \times 10^{14}$ cm^{-3}.

These theoretical spectra are the result of collisional calculations over every ionisation stage of the element in question followed by collisional–radiative modelling to determine the emission. Due to the large number of line transitions for the heavy species, it becomes unsuitable to archive and utilise the usual photon emissivity coefficients ($\mathcal{PEC}$) data (archived in ADAS format *adf15*) for generation of the feature. Instead, it is more efficient to use what ADAS refers to as feature photon emissivity coefficients ($\mathcal{F}-\mathcal{PEC}$) which are archived in format *adf40*. To form these files, the

$\mathcal{PEC}$ are mapped on to a pixelated grid corresponding to a spectrometer pixel range - with the line intensities convolved with a normalised emission profile. By default, modest Doppler broadening, with a Maxwellian distribution for a set ion temperature, T_i, is applied. At each stage of the process, the data are archived, but the feature photon emissivity coefficients ($\mathcal{F} - \mathcal{PEC}$) are the most directly applicable to experimental analysis. Note that any further broadening effects (e.g. instrument function) can be applied retrospectively to the archived feature via FFS (Section 4.4). The preparation and assembly of the highest quality data that comprise the *adf40* files is not considered here — see [49, 50] for details.

2.7 Helium-like Soft X-ray Resonance and Satellite Lines

One of main spectroscopically observed features in the soft x-ray region are the Li-lke satellite lines associated with the He-like resonance lines. Due to a large region of existence in temperature of the He-like ionisation stage, this feature is prominent in observations from both astrophysical and laboratory fusion plasmas (see, for example, [51] for studies on MAST). It also means that the He-like ions, within a fusion plasma, will be present across a range of radial values as the spectrometer's line of sight is trained from edge to core. There will be marked differences in local plasma conditions at different radii (e.g. electron temperature) therefore the lines do not provide a local measurement of these quantities. ADAS does not have a population model and data format for delivery of this type of feature. It is the object, here, to add this capability to ADAS such that this class of feature is available for spectral analysis (see Section 5.7 for a test case using this feature).

Early work in the astrophysical domain established the main nomenclature and method of analysis of low density plasmas [52]. Consider a He-like ion $\mathcal{A}^{+z}(1s^2\ ^1S)$ of an element $\mathcal{A}$. There is a set of four spectral lines arising from the transition from the n=2 shell of $\mathcal{A}^{+z}$. Conventionally known as w, x, y and z. w is the main resonance line $1s2p\ ^1P_1 \rightarrow 1s^2\ ^1S_0$; x is the quadrupole line $1s2p\ ^3P_2 \rightarrow 1s^2\ ^1S_0$; y is the intercombination line $1s2p\ ^3P_1 \rightarrow 1s^2\ ^1S_0$; y and lastly, z is the forbidden line $1s2s\ ^3S_1 \rightarrow 1s^2\ ^1S_0$. The z line is a strong density indicator in appropriate density regime for the charge state and the intensity of the x and y lines, relative to that of the w line are an indicator of transient state because of recombination.

The satellite lines, which lie close to these He-like lines are from the Li-like system

with a spectator electron present. An example is the transition from $1s2p(\,^1P)3d\,^2D \rightarrow 1s^2(\,^1S)3d\,^2D$, where the 3d electron is the spectator which lies close to the w line. Such lines with the spectator in the $n = 3$ shell have the greatest excursion and those with the spectator in the higher n shells accumulate close to the associated He-like line. The mechanisms for the creation of the upper population of these lines are the dielectronic process or the inner-shell collision excitation process. dielectronic recombination is a two step process. Firstly, the process known as *resonance capture* can take place (eq. 2.30):

$$\mathcal{A}^{+z}\left(1s^2\,^1S_0\right) + e \rightarrow \mathcal{A}^{+z-1}\left(1s2p\,^1P, nl\right). \tag{2.30}$$

From this state, the system can undergo the reverse process, *Auger breakup*, or a radiative transition can take place in what is called *radiative stabilisation*.

$$\mathcal{A}^{+z-1}\left(1s2p\,^1P_1, nl\right) \rightarrow \mathcal{A}^{+z-1}\left(1s^2\,^1S_0, nl\right) + h\bar{\nu}. \tag{2.31}$$

Note that the photon released in eq. 2.31 is of a similar wavelengh to the main resonance line as the inner transition is similar, yet perturbed by the presence of the nl electron, resulting in a displaced satellite line.

In this situation, using notation such that the indices ρ, μ and ν refer to ground or metastable states in singly-excited systems (z, $z-1$ and $z+1$) and σ represents excited states in doubly excited systems. For a resolved level of a distinguishable parent $A^{+z-1}_{\sigma,nlJ}$, the population is:

$$N^{+z-1}_{\sigma,nlJ} = \sum_{\mu=1}^{M^{(z-1)}} \mathcal{F}^{(exc)}_{\sigma,nlJ;\mu} N_e N^{+z-1}_{\mu} + \sum_{\rho=1}^{M^{(z)}} \mathcal{F}^{(rec)}_{\sigma,nlJ;\rho} N_e N^{+z}_{\rho} \tag{2.32}$$

and that of the level A^{+z}_{σ} may be written as

$$N^{+z}_{\sigma} = \sum_{\rho=1}^{M^{(z)}} \mathcal{F}^{(exc)}_{\sigma;\rho} N_e N^{+z}_{\rho} + \sum_{\nu=1}^{M^{(z+1)}} \mathcal{F}^{(rec)}_{\sigma;\nu} N_e N^{+z+1}_{\nu} \tag{2.33}$$

where the factors $\mathcal{F}^{(exc)}_{\sigma,nlJ;\mu} N_e N^{+z-1}_{\mu}$ and $\mathcal{F}^{(rec)}_{\sigma,nlJ;\rho} N_e N^{+z}_{\rho}$ are the contributions from inner shell excitation and dielectronic recombination for the $z-1$ times ionised ion and the factors $\mathcal{F}^{(exc)}_{\sigma;\rho} N_e N^{+z}_{\rho}$ and $\mathcal{F}^{(rec)}_{\sigma;\nu} N_e N^{+z+1}_{\nu}$ are the contributions from excitation and recombination for a z times ionised ion. The sums are over the dominant driver populations (ground and metastables) of each ionisation state, i.e. up to $M^{(z)}$ of stage

z. The emissivity function of a satellite line may be written as

$$\frac{G_{\sigma,nlJ\rightarrow\rho',nlJ'}}{N_eN^{tot}} = \frac{N_1^{+z}}{N_eN^{tot}}\left(\sum_{\mu=1}^{M^{(z-1)}}\left(A_{\sigma,nlJ\rightarrow\rho',nlJ}\mathcal{F}^{(exc)}_{\sigma,nlJ;\mu}\right)\frac{N_\mu^{+z-1}}{N_1^{+z}} + \sum_{\rho=1}^{M^{(z)}}\left(A_{\sigma,nlJ\rightarrow\rho',nlJ'}\mathcal{F}^{(rec)}_{\sigma,nlJ;\rho}\right)\frac{N_\rho^{+z}}{N_1^{+z}}\right) \tag{2.34}$$

$$= \frac{N_1^{+z}}{N_eN^{tot}}\left(\sum_{\mu=1}^{M^{(z-1)}}\mathcal{E}^{(exc)}_{\sigma,nlJ\rightarrow\rho',nlJ';\mu}R^{(z-1)}_{1;\mu}\mathcal{A}^{(z-1,z)}_{\mu,1} + \sum_{\rho=1}^{M^{(z)}}\mathcal{E}^{(rec)}_{\sigma,nlJ\rightarrow\rho',nlJ';\rho}R^{(z)}_{1;\rho}\mathcal{A}^{(z,z)}_{\rho,1}\right) \tag{2.35}$$

where $\mathcal{E}^{(exc)}_{\sigma,nlJ\rightarrow\rho',nlJ';\mu}(T_eN_e)$ is the excitation emissivity coefficient, $\mathcal{E}^{(rec)}_{\sigma,nlJ\rightarrow\rho',nlJ';\rho}(T_eN_e)$ is the dielectronic recombination emissivity coefficient and $N^{tot} = \sum_{z,\rho}N_\rho^{+z}$. The quantities $R^{(z-1)}_{1;\mu}$ and $R^{(z)}_{1;\rho}$ measure the dis-equilibrium in the ionisation balance and

$$\mathcal{A}^{(z-1,z)}_{\mu,1} = \frac{N_\mu^{+z-1}}{N_1^{+z}}|_{eq} \text{ and } \mathcal{A}^{(z,z)}_{\rho,1} = \frac{N_\rho^{+z}}{N_1^{+z}}|_{eq} \tag{2.36}$$

measure the metastable abundances in ionisation equilibrium relative to the z-times ionised ion ground. In most fusion and astrophysical plasma conditions, metastable populations of a given ionisation stage are close to quasi-static equilibrium with the ground so that $R^{(z-1)}_{1;\mu} = R^{(z-1)}_1$ independent of μ and $R^{(z)}_{1;\rho} = 1$. Thus the emissivity function depends upon three parameters $R^{(z-1)}_1$, T_e and N_e principally, although there is a weaker Z_{eff} and T_{ion} dependence at high density.

In a similar fashion, the emissivity of the main lines, between singly excited states, is defined by:

$$\frac{G_{\sigma\rightarrow\rho'}}{N_eN^{tot}} = \frac{N_1^{+z}}{N_eN^{tot}}\left(\sum_{\rho=1}^{M^{(z)}}\left(A_{\sigma\rightarrow\rho'}\mathcal{F}^{(exc)}_{\sigma;\rho}\right)\frac{N_\rho^{+z}}{N_1^{+z}} + \sum_{\nu=1}^{M^{(z+1)}}\left(A_{\sigma\rightarrow\rho'}\mathcal{F}^{(rec)}_{\sigma;\nu}\right)\frac{N_\nu^{+z+1}}{N_1^{+z}}\right) \tag{2.37}$$

$$= \frac{N_1^{+z}}{N_eN^{tot}}\left(\sum_{\rho=1}^{M^{(z)}}\mathcal{E}^{(exc)}_{\sigma\rightarrow\rho';\rho}R^{(z)}_{1;\rho}\mathcal{A}^{(z,z)}_{\rho,1} + \sum_{\nu=1}^{M^{(z+1)}}\mathcal{E}^{(rec)}_{\sigma\rightarrow\rho';\nu}R^{(z+1)}_{1;\nu}\mathcal{A}^{(z+1,z)}_{\nu,1}\right) \tag{2.38}$$

where $\mathcal{E}^{(exc)}_{\sigma\to\rho';\rho}(T_eN_e)$ is the excitation emissivity coefficient, $\mathcal{E}^{(rec)}_{\sigma\to\rho';\nu}(T_eN_e)$ is the radiative (+dielectronic) recombination emissivity coefficient. $R^{(z)}_{1;\rho}$ and $R^{(z+1)}_{1;\nu}$ measure the dis-equilibrium in the ionisation balance with a similar simplification to that above, such that $R^{(z)}_{1;\rho} = 1$ and $R^{(z+1)}_{1;\nu} = R^{(z+1)}_{1}$ independent of ν. The quantity:

$$\mathcal{A}^{(z+1,z)}_{\nu,1} = \frac{N^{+z+1}_{\nu}}{N^{+z}_{1}}|_{eq} \tag{2.39}$$

measures the ionisation equilibrium relative metastable abundances for the $z+1$-times ionised ion. Again, the emissivity function principally depends upon three parameters $R^{(z+1)}_{1}$, T_e and N_e. Thus, the theoretical local emissivity of the combined satellite lines and associated resonance line feature is functionally dependent on four parameters, $R^{(z-1)}_{1}$, $R^{(z+1)}_{1}$, T_e and N_e.

2.7.1 The Population Calculations

Conventional population modelling in generalised collisional-radiative theory addresses 'singly-excited' states built on ground and metastable parents. Very large efficiency of computation is achieved by handling the two-step dielectronic process as a single effective process populating these singly excited states [53]. To model the satellite line feature on the other hand, the dielectronic process must be separated out in the population structure — at least for the resolved satellite lines with low-lying ($n \lesssim 4$) spectator. Spectrally unresolved satellite lines with higher-lying spectators provide a second order supplementation of the parent ion lines and also the associated upper level populations of these satellite lines give a contribution to the observed satellite line intensities via cascade of the spectator. It is valid to bundle over outer quantum numbers for such populations and to adopt some of the techniques of [53] for their evaluation in finite density plasma, although care is required to avoid double counting. It should be noted that the deviation of these unresolved satellite lines is small — it may be hard, or impossible, to resolve it from the central wavelength of the resonance line. Use of a forward planned model of the system will avoid over-estimation of the intensity and width of this line from spectral fitting analysis.

For the ion A^{+z-1}, introduce principal quantum numbers n_0, n_1 and n_2. n_0 is the principal quantum shell of the ground state valence shell. The range $n_0 \le n \le n_1$ spans the spectator shells for which the individual satellite lines are distinguished in the calculations. The range $n_1 < n \le n_2$ spans spectator shells for which the satellite lines are individually unresolved, with n_2 an upper limit chosen sufficiently large for convergence in the calculation. Typically, in our calculations, we take $n_0 = 2$, $n_1 = 4$

and $n_2 \sim 15$.

2.7.1.1 Doubly-excited State Populations — Unresolved Satellites Lines

In the range $n_1 < n \leq n_2$, consider the bundled population (that is summed over substates of an nl- or n-shell) designated by $N_{\sigma,nl}$ built on an excited parent $\sigma \equiv (\gamma_\sigma J_\sigma)$ such that

$$N_{\sigma,nl} = \sum_{j,J} N_{\sigma,nljJ} \tag{2.40}$$

Then, following [53], the populations in finite density plasma are determined by the equations

$$\begin{aligned}
&- \left(N_\mathrm{e} q^\mathrm{e}_{nl-1\to nl} + N^{z_\mathrm{eff}} q^{z_\mathrm{eff}}_{nl-1\to nl}\right) N_{\sigma,nl-1} \\
&+ \left(\sum_{l'=l\pm1} N_\mathrm{e} q^\mathrm{e}_{nl\to nl'} + \sum_{l'=l\pm1} N^{z_\mathrm{eff}} q^{z_\mathrm{eff}}_{nl\to nl'} + \sum_{\sigma'=1}^{\sigma-1} \sum_{l'=l-1}^{l+1} A^\mathrm{a}_{\sigma,nl\to\sigma',\kappa l'} + \sum_{\sigma'=1}^{\sigma-1} A^\mathrm{r}_{\sigma,nl\to\sigma',nl} \right. \\
&\qquad \left. + \sum_{n'=n_1+1}^{n-1} \sum_{l'=l-1}^{l+1} A^\mathrm{r}_{\sigma,nl\to\sigma'',n'l'} + \sum_{n'=n_0}^{n_1} \sum_{l'=l-1}^{l+1} A^\mathrm{r}_{\sigma,nl\to\sigma,n'l'} \right) N_{\sigma,nl} \\
&- \left(N_\mathrm{e} q^\mathrm{e}_{nl+1\to nl} + N^{z_\mathrm{eff}} q^{z_\mathrm{eff}}_{nl+1\to nl}\right) N_{\sigma,nl+1} \\
&= N_\mathrm{e} \sum_{\rho=1}^{M^{(z)}} \sum_{l'=l-1}^{l+1} q^\mathrm{c}_{\rho,\kappa l'\to\sigma,nl} N_\rho + \sum_{\sigma''=\sigma+1}^{P^{(z)}} A^\mathrm{r}_{\sigma'',nl\to\sigma,nl} N_{\sigma'',nl} + \sum_{n''=n+1}^{n_2} \sum_{l''=l-1}^{l+1} A^\mathrm{r}_{\sigma,n''l''\to\sigma,nl} N_{\sigma,n''l''} \,.
\end{aligned} \tag{2.41}$$

Here, $M^{(z)}$ denotes the dominant ground and metastable states of the recombining ion, which are the targets for recombination, and $P^{(z)}$ denotes the complete set of active parents so, $M^{(z)} \subset P^{(z)}$. The various q^e are electron-impact excitation rate coefficients between the states labelled by the the subscript. Similarly, the labels q^{z_eff} represent ion-impact excitation rates (of effective charge z) and q^c is the capture rate. N^{z_eff} is the ion density (of effective charge z). A^a and A^r are the Auger and spontaneous emission rate coefficients respectively.

Both electron and ion dipole-allowed impact collisions are included, but only between l-levels of the same n-shell. It is noted that these have very large cross-sections (since the l-levels are nearly degenerate), that ion cross-sections are usually larger that those for electrons and in general have a density dependence. This leads to a non-linear (and therefore not simply scaleable) density behaviour of the population equations and influences our method of calculation. Inner-shell excitation from the ground and metastables of the recombined ion is ignored for this high n-part. These equations

may be solved recursively downwards through n-shells and parents. Note the double radiative sum on the right hand side and the double radiative sums on the diagonal give the outer electron radiative transition to and from the current nl-shell population. These terms were ignored in the Burgess-Bethe General Program (BBGP) doubly-excited state redistribution equations of [53] but compensated for by renormalisation to exact totals. They are generated explicitly here. It is sufficient to use hydrogenic transition probabilities between spectator nl-shells, but more precise (and resolved) values are required to levels in the range $n_0 \leq n \leq n_1$. The exact BBGP approach as detailed in [53], with the extension above, is sufficient for the unresolved dielectronic recombination contribution to the satellite line feature.

2.7.1.2 Doubly-excited State Populations — Resolved Satellite Lines

The range $n_0 \leq n \leq n_1$ provides the bulk of the dielectronic feature. It includes dielectronic contributions paralleling those of Section 2.7.1.1, but also contributions from inner shell excitation from the ground and metastables states of the A^{+z-1} ion. For $n = n_0$, with equivalent electrons in the shell, parentage is not in general well specified and so it is convenient to divide the resolved level population equations into parent-attributable ($\sigma nlJ'$) and parent-unattributable (γnJ) groups. This takes the form, for a doubly-excited parent-attributable level, of

$$
\begin{aligned}
&- \sum_{\gamma',n',J'} N_e q^e_{\gamma' n' J' \to \sigma'' nlJ} N_{\gamma',n'J'} - \sum_{\substack{\sigma',n',l',J' \\ E'<E}} N_e q^e_{\sigma' n' l' J' \to \sigma'' nlJ} N_{\sigma',n'l'J'} \\
&+ \left(\sum_{l'=l\pm1} N_e q^e_{nl\to nl'} + \sum_{l'=l\pm1} N^{z_{\rm eff}} q^{z_{\rm eff}}_{nl\to nl'} + \sum_{\sigma=1}^{\sigma''-1} \sum_{l'=l-1}^{l+1} A^a_{\sigma'',nl\to\sigma,\kappa l'} + \sum_{\sigma'=1}^{\sigma''-1} A^r_{\sigma'',nl\to\sigma',nl} \right. \\
&\qquad \left. + \sum_{n'=n_1+1}^{n-1} \sum_{l'=l-1}^{l+1} A^r_{\sigma'',nl\to\sigma'',n'l'} + \sum_{n'=n_0}^{n_1} \sum_{l'=l-1}^{l+1} A^r_{\sigma'',nl\to\sigma'',n'l'} \right) N_{\sigma'',nl} \\
&- \sum_{\substack{\sigma',n',l',J' \\ E'>E}} \left(N_e q^e_{\sigma' n' l' J' \to \sigma'' nlJ} + A^r_{\sigma',n'l'J'\to\sigma'',nlJ} \right) N_{\sigma',n'l'J'} \\
&= N_e \sum_{\sigma=1}^{M^{(z)}} \sum_{l',J'} q^c_{\sigma,\kappa l' J' \to \sigma'' nlJ} N_\sigma + \sum_{n'=n_1+1}^{n_2} \sum_{l'} A^r_{\sigma',n'l'\to\sigma'',nlJ} N_{\sigma',nl} \,. \qquad (2.42)
\end{aligned}
$$

with similar forms for parent-unattributable levels and single excited levels. There are equivalent sets of equations to equations 2.41 and 2.42 for the ion A^{+z}. Note the spectator electron cascade contribution from unresolved levels — the last term of equation 2.42. In this formulation, the helium-like lines are envelopes of the true helium-like line and the satellite lines with spectator $n > n_1$, so that the spectral

emissivity coefficient for a helium-like line $\sigma \rightarrow \sigma'$,driven by metastable ρ, is given by

$$\epsilon^{\rm eff}_{\sigma\rightarrow\sigma';\rho}(\nu) = \epsilon_{\sigma\rightarrow\sigma';\rho} + \sum_{n,l;n>n_1} \epsilon_{\sigma nl\rightarrow\sigma' nl;\rho}$$
$$= \left(A^{\rm r}_{\sigma\rightarrow\sigma'}\phi(\nu)N_\sigma + \sum_{n,l;n>n_1} A^{\rm r}_{\sigma nl\rightarrow\sigma' nl}\phi(\nu-\Delta\nu_{nl})N_{\sigma nl} \right) / N_e N_\rho \qquad (2.43)$$

For the present helium-like system, we are only concerned with the case of $\sigma' = \rho$.

In spite of the apparent simplicity of the one-, two- and three-electron systems considered here, the quality of excitation cross-sections, especially the excitations from the ground states of the ions by promotion of a $1s$ electron, has been a limiting factor on precision. In preparing the population models above we have substituted high precision data for all transitions between all singly and doubly excited states of the helium-like and lithium-like system of the form $1s^q_1 2l^q_2$, $1s^{q_1-1}2l^{q_2-1}2l'nl''$ with $n \leq 4$.

2.7.1.3 The Collisional-radiative Matrix

It is helpful to illustrate the above steps as the creation of portions of the complete collisional-radiative matrix and then transformations and condensations of these portions. Consider the set $\{P\}$ of intermediate coupling fully resolved parent states, that is states of the ion A^{+z}, comprising the ground and metastable set $\{P_0\}$ and the excited parent set $\{P^*\}$ so that $\{P_0\} \cap \{P*\} = 0$ and $\{P\} = \{P_0\} \cup \{P^*\}$. For the ion A^{+z-1}, distinguish the fully-resolved set of levels $\{S\}$ comprising low levels of unattributable parent $\{S_u\}$, assigned metastable parent by $\{S_a\}$ and assigned excited parent by $\{S^*_a\}$, so that $\{S\} = \{S_u\} \cup \{S_a\} \cup \{S^*_a\}$. Further sub-divide $\{S_a\}$ and $\{S^*_a\}$ into the subsets arising from the range $n_0 \leq n \leq n_1$, $\{S^{[01]}_a\}$ and $\{S^{[01]*}_a\}$ and the subsets from the range $n_1 < n \leq n_2$, $\{S^{[12]}_a\}$ and $\{S^{[12]*}_a\}$. Bundling applies only to the sets $\{S^{[12]}_a\}$ and $\{S^{[12]*}_a\}$ giving condensed sets of levels $\{\bar{S}^{[12]}_a\}$ and $\{\bar{S}^{[12]*}_a\}$. The collisional-radiative matrix equations then take the form

$$\begin{bmatrix} C_{u,u} & C_{u,[01]} & C_{u,[12]} & C_{u,[01]*} & C_{u,[12]*} \\ C_{[01],u} & C_{[01],[01]} & C_{[12],[01]} & C_{[01],[01]*} & C_{[01],[12]*} \\ C_{[12],u} & C_{[12],[01]} & C_{[12],[12]} & 0 & C_{[12],[12]*} \\ C_{[01]*,u} & C_{[01]*,[01]} & 0 & C_{[01]*,[01]*} & 0 \\ 0 & 0 & 0 & 0 & C_{[12]*,[12]*} \end{bmatrix} \begin{bmatrix} N_u \\ N_{[01]} \\ N_{[12]} \\ N_{[01]*} \\ N_{[12]*} \end{bmatrix} = \begin{bmatrix} r^{(r)}_u \\ r^{(r)}_{[01]} \\ r^{(r)}_{[12]} \\ r^{(d)}_{[01]*} \\ r^{(d)}_{[12]*} \end{bmatrix} \qquad (2.44)$$

Note that there is no dielectronic recombination to the $\{S^u\}$, $\{S_a^{[01]}\}$ and $\{S_a^{[12]}\}$ sets. Also, the $r^{(d)}$ coefficients are resonance capture coefficients and not final stabilised dielectronic recombination coefficients. An essential simplification is that the $\{S_a^{[12]*}\}$ set is uncoupled to the lower sets by collisional excitation. This matches the usual picture of dielectronic recombination as a quasi-static composite process, so that the stabilised rate coefficients to the singly excited levels are $r_{[12]}^{(d)} = C_{[12],[12]*}C^{-1}_{[12]*,[12]*}r^{(d)}_{[12]*}$, $r_{[01]}^{(d)} = C_{[01],[12]*}C^{-1}_{[12]*,[12]*}r^{(d)}_{[12]*}$ and $r_u^{(d)} = C_{u,[12]*}C^{-1}_{[12]*,[12]*}r^{(d)}_{[12]*}$. The $\{S_a^{[12]}\}$ set is bundled in the present treatment, being replaced by the set $\{S_a^{[1\bar{2}]}\}$. With these replacements, the matrix equations condense to

$$\begin{bmatrix} C_{u,u} & C_{u,[01]} & C_{u,[1\bar{2}]} & C_{u,[01]*} \\ C_{[01],u} & C_{[01],[01]} & C_{[01],[1\bar{2}]} & C_{[01],[01]*} \\ C_{[1\bar{2}],u} & C_{[1\bar{2}],[01]} & C_{[1\bar{2}],[1\bar{2}]} & 0 \\ C_{[01]*,u} & C_{[01]*,[01]} & 0 & C_{[01]*,[01]*} \end{bmatrix} \begin{bmatrix} N_u \\ N_{[01]} \\ N_{[12]} \\ N_{[01]*} \end{bmatrix} = \begin{bmatrix} r_u^{(r)} + r_u^{(d)} \\ r_{[01]}^{(r)} + r_{[01]}^{(d)} \\ r_{[1\bar{2}]}^{(r)} + r_{[1\bar{2}]}^{(d)} \\ r_{[01]*}^{(d)} \end{bmatrix} \tag{2.45}$$

2.7.2 Electron-impact Rate Coefficients

Both electron-impact excitation and ionisation rate coefficients are required for the present work and there are some general issues concerning their calculation and use in the present context of satellite lines and doubly-excited states. It should be noted that the calculation of this data does not form part of this work.

The *R*-matrix method has been used for excitation, a method capable of high precision because of its close-coupled character and efficient inclusion of resonances. These resonances are a superset of the doubly-excited states of satellite line population structure. The *R*-matrix equations include whole resonance series and the interference with direct excitation in which they can participate, but often omit radiative channels. On the other hand, doubly-excited population modelling can easily include most reaction channels, but in cruder approximation and it does omit interference. Radiation-damped *R*-matrix theory has been used in the cross-section calculations utilised here. It is essential to ensure proper matching of it with our population modelling and avoidance of 'double counting'. For satellites to the helium-like $n = 1$–2 lines, the situation is straightforward. Doubly-excited states of the form 1s2*lnl′* (except for 'high-*n*') only are resonances in the elastic cross-section of the $1s^2\ {}^1S_0$ state. The 1s3*lnl′* resonances, which affect the inelastic excitation cross-sections are not included in the population modelling. The *R*-matrix calculations and population modelling are 'orthogonal'. For ionisation, the concern is the $1s^22s\ {}^2S_{1/2}$ ground state of the lithium-like ion. Only direct ionisation of the 1s electron should be included in the ionisation

calculation, since the excitation/auto-ionisation is included fully in the population structure modelling.

2.7.2.1 Helium-like Excitation

The methods used here for electron impact excitation of the helium-like system have been piloted in the initial study on argon and iron [54]. The radiation-damped collision strengths are sought here and use is made of data that has been calculated using *R*-matrix method [55] in conjunction with the intermediate coupling frame transformation (ICFT) method [56] and the optical potential approach to damping [57, 58]. A complete solution, in terms of reactance or scattering (collision) matrices is obtained firstly in *LS*-coupling. In particular, use is made of multi-channel quantum defect theory (MQDT) to obtain 'unphysical' collision matrices [59]. These are then transformed, first, algebraically to *jK*-coupling and then, via the use of the term-coupling coefficients, to intermediate coupling. The ICFT method is computationally less demanding than the full Breit–Pauli approach but does not suffer the inaccuracies associated with the term-coupling of physical collision matrices. Finally, it should be noted that the use of the optical potential modifies the usual (undamped) expressions for the *R*-matrix, unphysical collision matrices and MQDT closure relations by making them complex, in general (see [57] for details and [60] for computational aspects).

AUTOSTRUCTURE [61] is used to calculate the atomic structure and, hence, to generate radial wavefunctions for the collision calculation. Details of the computations follow [54] but with energy ranges etc. adjusted for consistency along the iso-electronic sequence.

2.7.2.2 Lithium-like Excitation

The data used here results from the piloting study on Ar^{15+} and Fe^{23+} [62] . As with the He-like collision calculations, use is made of multi-channel quantum defect theory (MQDT) to obtain 'unphysical' collision matrices [59]. The outer region solutions include the long-range coupling potentials as a perturbation, still within the MQDT framework [58, 63]. The approach to the inner- and outer-shell data is to perform the calculations independently and later merge the effective collision strengths back together into a single dataset. The damping of resonances due to Auger breakup is dealt with in two distinct cases, namely, Auger breakup to states included explicitly in the calculation and Auger breakup to states not included in the close-coupling expansion. The former is dealt with within the *R*-matrix approach intrinsically and the latter uses

AUTOSTRUCTURE to calculate Auger widths for the core re-arrangement of each target level and includes them in the optical potential in the outer-region calculation.

2.7.3 Computational Details

Within ADAS, data from collisional excitation cross-section calculations are assembled into 'specific ion files' of format *adf04* for each ion. The *R*-matrix calculations, used here, produce a *type 1 adf04* file in which the interval-averaged collision strengths for all transitions between level pairs are tabulated as a function of threshold parameter together with the A-value. The ADAS code ADAS809 allows conversion of such a file to *type 3* file of Maxwell averaged collision strengths as a function of electron temperature. Both these *adf04* file types are prepared automatically following the collision calculations of Section 2.7.2 and are supplemented with equivalent electron impact ionisation data before archiving as general self-contained data resources according to standard ADAS practice.

Prior to the population modelling stage, attention must be given to the production of dielectronic data and their special assembly into Auger and recombination enhanced *adf04* files for doubly-excited population and satellite line studies. This is an alternative post-processing route for dielectronic data — to be contrasted with the *adf09* production of [53] and other prior papers of the DR Project series. The enhanced *adf04* files are merged with the high grade, but restricted, *adf04* files produced from the *R*-matrix collision calculations detailed in Section 2.7.2 to provide the final comprehensive *adf04* files which are the objective of the fundamental calculations. These steps are shown schematically by Fig. 2.6. The complete *adf04* files are the input to the derived population calculations which deliver the satellite line special feature, accessed via AFG (Section 3.1), for diagnostic application using FFS (Section 4.4).

The driver data sets for the fundamental ADAS701 (AUTOSTRUCTURE) calculations are archived in ADAS format *adf28* according to recombining ion iso-electronic sequence and are of two types — those required to produce the energy level, Auger rate and radiative rates for *J*-resolved satellites in intermediate coupling and those required to generate the data for the *BBGP nl*-bundled calculations of higher unresolved satellites. For both drivers, the output from ADAS701 is a set of temporary files (including the *oic* and *olg* files which comprise Auger & radiative rates and atomic structure data respectively). These data are collected and reorganised by a dedicated post-processor code, ADAS703, for the satellite line task. The output files from ADAS703 comprise an enhanced *adf04* file (containing the doubly-excited states) and a driver for the *BBGP*

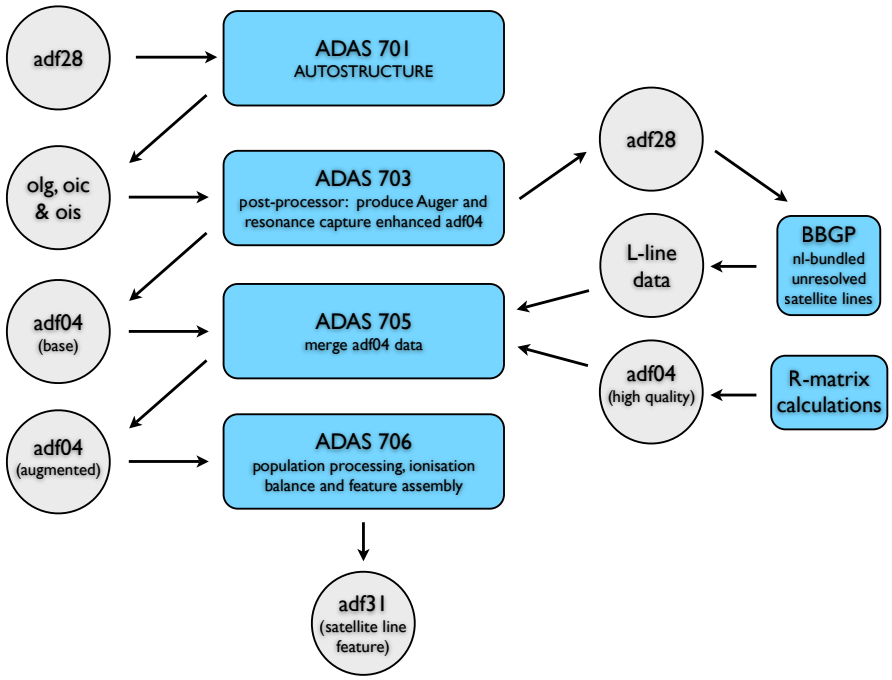


Figure 2.6: Fundamental dielectronic recombination (DR) data preparation and special feature assembly. Main program units are shown as rectangles, whilst archived data files are represented by circles.

calculation. The latter has been described in [53] and that which is produced here remains similar — except that it is augmented by A-values for outer electron radiative transition to levels of the resolved doubly-excited shells i.e. the aforementioned subset of levels, $\{S_a^{[01]*}\}$ (Section 2.7.1.3). This driver is archived also in *adf28* as above, but with postfix *bbgp*. The enhanced *adf04* file from ADAS703 must be merged with the restricted high grade collisional *adf04* file resulting from the *R*-matrix work. The merging task is handled by ADAS705. There are two further steps. Radiative recombination to singly-excited levels from an initial ground or metastable parent must be added. In the ADAS Project, the code ADAS211 calculates such rate coefficients in an effective potential distorted wave approximation creating an archive of format *adf08* analogous to the *adf09* of dielectronic recombination. It proved convenient to develop ADAS705 to merge *adf08* data into an accumulating *adf04* file also. The *adf04* file is completed by the addition of so-called *L-lines*. An *L-line* contains stabilised dielectronic recombination coefficients (to an unresolved singly-excited *nl*-bundle), as a function of temperature. The *L-line* also contains an associated wavelength, which is the mean wavelength for the stabilisation photons. Such data lines therefore give the position and magnitude of unresolved satellite contributions to the emissivity of the resonance line (and associated lines) of the parent ion. The *L-line* is evaluated at zero density, including only inner electron stabilisation, and so represents a simplification of

full redistributive modelling. The addition of the *L-lines* means that the final *adf04* file is self-contained and complete, albeit at slightly reduced precision. It can therefore be used easily in simple population modelling and associated line ratio studies, especially in an astrophysical context.

This very comprehensive development is beyond the usual satellite line studies and is general in that it would be applicable to Be-like/Li-like and Na-like/Li-like systems. These developments are recently completed in ADAS, but full exploitation, using JET spectra, are not yet possible. JET is currently undergoing a substantial upgrade program (EP2) which includes enhancement of the X-ray crystal spectrometer, diagnostic KX1. Operations resume in April 2011. In Section 5.7, analysis is restricted to non-JET data and only partial implementation of the modelling described above was used.

Chapter 3

ADAS Special Feature Application Programming Interface

This chapter unifies access to the special features described in chapter 2. It describes the development of the ADAS Feature Generator (AFG). Technically, this is known as an application programming interface (API) whose purpose is to implement a standardised interface to the disparate underlying codes. The language chosen for implementation of this API was IDL [9], but the descriptions here are general and language independent.

3.1 ADAS Feature Generator (AFG)

In order to utilise the ADAS special feature codes as a set of basis functions for the construction of complex spectral models, it was recognised that programmatically, the best way to proceed was to construct an application programming interface (API) to the underlying codes. The API provides a common access point to the ADAS special features (see Fig. 3.1). Taking such an approach means that all external programs can access ADAS special features in a structured, consistent manner for each of the special features, rather than each and every external program requiring to accommodate the idiosyncratic nature of each of the special feature codes that they need to interact with.

Exploration of the available feature modelling within ADAS indicated the key similarities and differences in attempting to produce a synthetic spectra. Table 3.1 highlights the requirements (in terms of input/format of input) for the programs central to feature generation. The table also summarises the way in which output is provided.

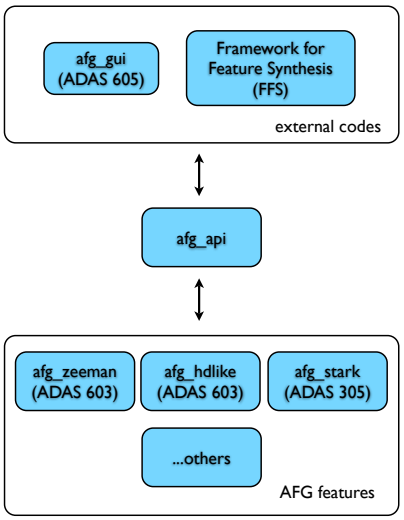


Figure 3.1: Accessing ADAS special features.

	Stark (ADAS305)	Zeeman (ADAS603)	Heavy Species (ADAS810)	Balmer/Paschen (ADAS311)	Molecule (CALCAT)	He-like (ADAS706)
formatted input file			■	■	■	■
pre-processing			■			■
observation geometry	■	■				
beam parameters	■			⋆		
magnetic field details	■	■				
electric field details	■					
λ range & no. of pixels	□					■
polarisation filter details	■					
formatted output file				■	■	■
line-resolved output	□	■		■	■	□
λ-resolved output	□		■			□

■ required by this program

□ optional for this program

⋆ Beam information is not explicitly required for generation of the Balmer series feature. ADAS311 was originally developed as a beam emission code and so, for analysis of Balmer series emission from the divertor, the effects of the beam must be disabled (i.e. beam energy = 0.0).

Table 3.1: Summary of requirements and outputs of the various ADAS special feature generation facilities.

Table 3.1 demonstrates the fact that each of the codes is quite different and the design of AFG had to be mindful of such differences in input and output between the feature codes that it seeks to unify access to. However, the table also shows that there are a great number of similarities between the codes too. Each of the feature generation routines are also similar in that they are essentially mathematical functions which depend upon a set of input parameters. In order to be useful, each special feature code requires to have methods to set and retrieve current parameter values, provide additional information about the parameters such as operational bounds, units, data type etc. and be able to evaluate the feature given the input parameter set. In fact, computationally, the problem at hand is well served by the object-oriented concept of inheritance (first introduced in *SIMULA-67* [64]). Inheritance is an abstraction in which a set of programming modules, known as 'subclasses' inherit from a common 'superclass' which defines a set of methods that are applicable to all sub-classes. The individual subclasses are, however, free to include additional methods unique to that class on top of the inherited superclass methods. Inheritance is a very useful concept as it reduces the need for code repetition, which simultaneously decreases code maintenance effort and the scope for error. Another important aspect of inheritance is what is referred to as subtype polymorphism, that is to say that all of the subclasses can be treated programmatically as being the same class of object. In reality however, the subclasses will often over-ride some of the superclass method implementations to suit their own needs, but this does not change how external programs interact with the object. In this particular scenario, the 'afg_api' superclass, from which all special feature classes (e.g. 'afg_zeeman', 'afg_stark', 'afg_hdlike', etc.) are derived, is an abstract class. An abstract class is one which is not intended to be instantiated i.e. not used alone in practice. Each of the special feature classes provide a concrete implementation of the abstract superclass method 'docalc', which is the method that performs function evaluation, 'setdesc', which sets the structure giving a description of the feature and its parameters and 'initpars' which sets default values for each of the feature parameters. The inheritance hierarchy for AFG is shown in Fig. 3.2.

The AFG description structures, accessed through the 'getdesc' method (see Fig. 3.3), have three fields, 'name', 'text' and 'parameters'. The first, 'name', is self-explanatory, 'text' contains a short textual description of what the feature is, and what underlying ADAS codes are used to generate the feature.

The last field in the description structure, 'parameters', is the most detailed as it is itself a structure, with each field comprising a further sub-structure, for each parameter, that provides information about those parameters (Fig. 3.4).

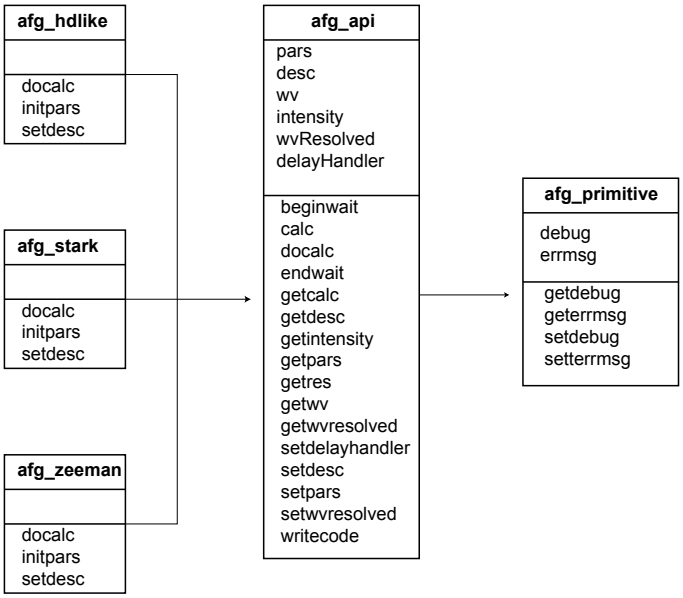


Figure 3.2: Class diagram for AFG. Each class is represented by a rectangular box, split into three compartments. The upper division is the name of the class, the middle section contains the class attributes (data members) and the lower segment holds the names of the available class methods (operations). The arrows, pointing from one class to another, indicate the inheritance hierarchy.

```
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]
```

Figure 3.3: Command line interaction with AFG, retrieving the description structure for the Zeeman feature.

```
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]
```

Figure 3.4: Examination of an AFG description structure, for the Zeeman feature, at the command line.

The parameter sub-structures are comprised of ‘desc’ — a short description of what the parameter is, ‘type’ — the data type associated with the parameter (float, integer, long, pointer) and ‘units’ — the units of parameter. The structure also has fields ‘min’ and ‘max’ defining the bounds on the parameter value and, importantly, the field ‘disptype’. This attribute provides a hint to external programs as to how they should handle input for this parameter. Note that this is related to the parameter data type, but is more specific. These ‘disptype’ selections are tabulated in Table 3.2.

‘disptype’	Description
continuous	can be varied continuously over its range
selection	takes on discrete values from a pre-defined set
field	numeric value, but not continous variation
file	name of a file
filelist	array of filenames

Table 3.2: AFG ‘disptype’ options.

Finally, there are two switches, ‘log’ — which indicates whether the feature depends on this parameter in a logarithmic fashion, or not, and ‘alterslimits’ — which denotes whether alteration of this parameter can alter the limits associated with another parameter. An example interrogation of the parameter sub-structure for the magnetic field strength (bvalue), from the Zeeman feature, at the command prompt is shown in Fig. 3.5. It should be noted that if the disptype is set to ‘selection’, then an additional attribute, ‘values’, will be present in the parameter description structure, which stores the set of possible values.

```
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
```

Figure 3.5: AFG feature parameter sub-structure for the magnetic field strength, for the Zeeman feature.

The parameter values are accessed through the method ‘getpars’ and are also returned as a structure, with the same tag names as in the description structure. Retrieval of the

parameter structure allows for easy alteration of the values and consequently setting these values for the feature evaluation (see Fig. 3.6). Alternatively, a more experienced

```
IDL> pars = zeeman->getpars()
IDL> help, pars, /str
** Structure <95adc84>, 4 tags, length=16, data
        length=12, refs=2:
   POL             INT              1
   OBSANGLE        FLOAT         90.0000
   BVALUE          FLOAT         2.50000
   FINDEX          INT              15
IDL> pars.obsangle = 88.0
IDL> pars.bvalue = 1.97
IDL> print, zeeman->setpars(pars=pars)
       1
IDL> result = zeeman->getcalc()
```

Figure 3.6: Command line interaction with AFG, altering a parameter values structure. In this case, the Zeeman feature's structure is retrieved and its values altered before passing the structure back to the AFG object for re-evaluation of the feature. Note that all AFG *set* methods return a 1, or 0 — to signify success, or failure respectively.

user (who is familiar with the parameter short names) may set parameter values at the point of evaluation. For example, a similar result to the interaction shown by Fig. 3.6, via the succinct syntax shown in Fig. 3.7. In addition to getcalc, there is the related

```
IDL>  result = zeeman->getcalc(obsangle=88.0, bvalue=1.97)
```

Figure 3.7: Command line interaction with AFG, altering parameter values directly. A similar effect is achieved to that seen by issuing the commands in Fig. 3.6, but in this case, the user is familiar with the parameter short-names and sets them directly when calling the 'getcalc' method that evaluates the feature.

method calc, which is used by getcalc but returns a flag to report success, or failure, of the underlying evaluation routine rather than returning the result itself. This method is able to take input of parameter values in the same way as getcalc. The method 'getres' is also available to retrieve the last calculated result, without re-calculation of the feature.

Some of the special features have a large number of parameters, some of which may be related to each other. There are varying levels of relationship too — perhaps they are quite loosely related in that they control one particular aspect of the feature, but some parameters may be very closely related, e.g. they are vector components of one variable. In order to produce an interface to such features, that remains organisationally clean, these relationships must be stored. This leads to the use of 'subordinates' and

'groups' in AFG. These optional attributes appear as additional fields of the top level 'desc' structure.

The subordinate structure is intended for use in the close relationship case and has three fields, 'title' — a label for the set, 'members' — the related parameters that comprise the set and, lastly, 'after' — the name of another parameter that it should appear closest to in a GUI. The AFG Stark feature (ADAS305) implements subordinates and an example command line interaction is shown in figure 3.8. From this example, we can see that three of the parameters, 'beam_dc_x', 'beam_dc_y' and 'beam_dc_z' (the x, y and z components of the beam direction) have been made members of 'beam_direction'. The full subordinate structure, for the Stark feature, is shown graphically by Fig. 3.9.

```
IDL> help, desc.subordinate, /str
** Structure <84dd394>, 4 tags, length=240, data
        length=240, refs=2:
  OBSERVATION_DIRECTION
                    STRUCT    -> <Anonymous> Array[1]
  BEAM_DIRECTION    STRUCT    -> <Anonymous> Array[1]
  BFIELD_DIRECTION
                    STRUCT    -> <Anonymous> Array[1]
  EFIELD_DIRECTION
                    STRUCT    -> <Anonymous> Array[1]
IDL> help, desc.subordinate.beam_direction, /str
** Structure <84db8c4>, 3 tags, length=60, data
        length=60, refs=2:
  TITLE             STRING    'Beam Direction'
  MEMBERS           STRING    Array[3]
  AFTER             STRING    'beam_density'
IDL> print, desc.subordinate.beam_direction.members
beam_dc_x beam_dc_y beam_dc_z
```

Figure 3.8: Examination of the AFG subordinate structure for the Stark feature from the command line.

The groups structure is intended for use for more loosely related parameters. The structure is comprised of two properties, 'title' — a label for the grouping and 'members' — array of names of the feature parameters, or indeed subordinate sets, that make up the group. Figure 3.10 shows that the Stark feature has five groupings, 'beam', 'plasma', 'bfield', 'efield' and 'observation'. As an example, the beam group has been examined; note that the beam_direction subordinate set is included as a member here along with individual parameters.

Utilization of AFG subordinates and groups are best considered by example and usage in the context of the AFG graphical user interface, ADAS605, which is discussed in Section 3.2.

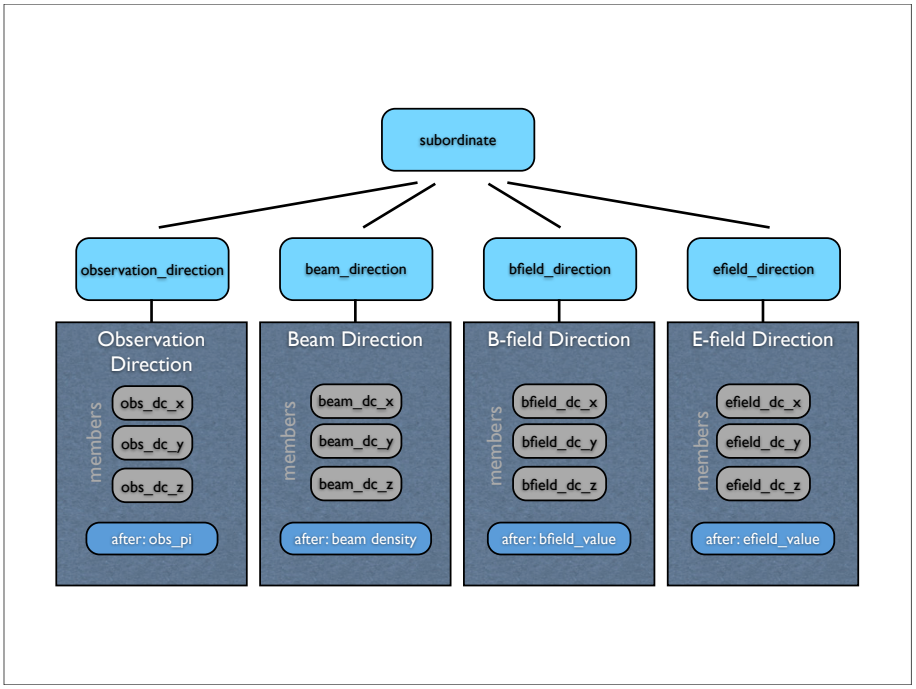


Figure 3.9: AFG 'subordinate' structure for the Stark feature (containing sub-structures *observation_direction*, *beam_direction*, *bfield_direction* and *efield_direction*) — cyan boxes. Contents of the sub-structures include: the title — white text, the members — grey boxes and the closest logical parameter — the sky-blue coloured boxes, labelled 'after'.

```
IDL> help, desc.groups, /str
** Structure <84ddc64>, 5 tags, length=252, data
         length=252, refs=2:
   BEAM            STRUCT    -> <Anonymous> Array[1]
   PLASMA          STRUCT    -> <Anonymous> Array[1]
   BFIELD          STRUCT    -> <Anonymous> Array[1]
   EFIELD          STRUCT    -> <Anonymous> Array[1]
   OBSERVATION     STRUCT    -> <Anonymous> Array[1]
IDL> help, desc.groups.beam, /str
** Structure <84dd69c>, 2 tags, length=72, data
         length=72, refs=2:
   TITLE           STRING    'Beam'
   MEMBERS         STRING    Array[5]
IDL> print, desc.groups.beam.members
beam_mass beam_energy beam_te beam_density beam_direction
```

Figure 3.10: Examination of the AFG group structure for the Stark feature from the command line.

3.2 ADAS605 — GUI to AFG

The main drive behind AFG is to ease access to the ADAS special feature routines such that they are easily incorporated into any external modelling code (such as FFS — Section 4.4). It has been shown in Section 3.1 that it is possible to do this through a series of simple commands — now common to all of the ADAS special features. However, it was thought that AFG could be made even more accessible via a graphical user interface (GUI); this code is known as ADAS605. A GUI to AFG serves several purposes. Firstly, it allows the user to explore the parameter space of the feature visually; the user can alter the parameters via tools such as sliders, drop-boxes etc. and see a plot of the resulting feature alter in real-time. Secondly, using the AFG 'writecode' feature, the GUI can act as an entry point to the novice user — after manipulation of the feature in the GUI, it is then possible to auto-generate the necessary code to create that feature, which can then be adapted for use in the user's own program. Finally, it also played a large role in the design of AFG — highlighting what information AFG would have to impart to an external program, in order to be useful.

ADAS 605 has been designed to heavily use the AFG API, such that the interface is highly dynamic i.e. its appearance is very much dependent upon the feature under consideration.

Upon selecting ADAS605 from the series 6 menu, you are presented with a simple input screen (Fig. 3.11) with a dropbox allowing selection of the feature of interest. A short description of the currently selected feature is given in the textbox below the dropbox. The text description is made available to the GUI from a getdesc call to the API, as in Fig. 3.3.

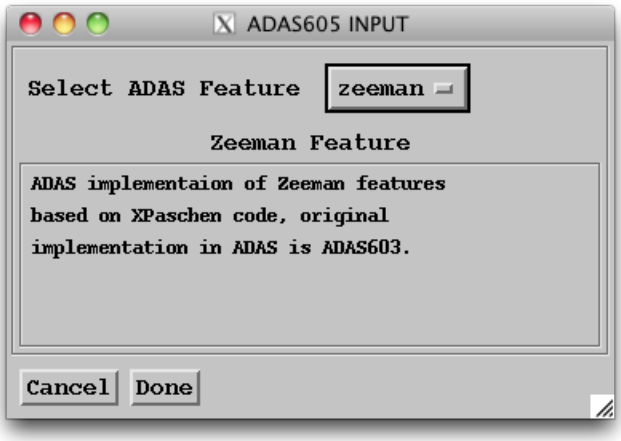
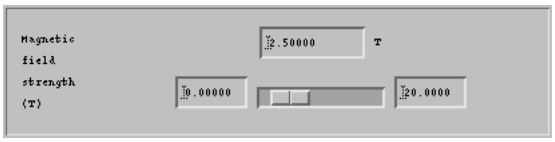
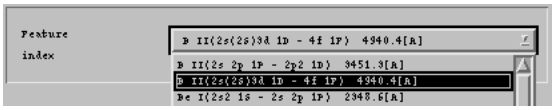


Figure 3.11: ADAS605 input screen: feature selection.

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, 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 important item of note here, is that 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 (as seen in Fig. 3.5). Firstly, 605 determines the correct type of widget to provide control of the parameter from the disptype tag (Fig. 3.12 shows a selection of widgets that ADAS605 uses for different disptypes). For example, the magnetic-field strength is of type continuous and 605 uses a slider for this type. Each of the parameter structure tags is important on building the control widgets. The 'desc' field is required to produce the label, in the example considered, the 'min' and 'max' fields place limits on the slider, 'unit' places a label next to the value and with the attribute 'log' set to 0, the slider will vary the value linearly between the two limits.



(a) continuous — slider widget



(b) selection — drop-list widget

(c) field — text field widget

(d) file / filelist* — pick-file dialog widget

Figure 3.12: Screenshots showing examples of the IDL widgets generated by ADAS605, corresponding to the various allowed 'disptype' values coming from the AFG description structures (see Table 3.2). * For (d): note that the same widget is used for both types, but that an option is set to restrict selection to a single file for 'file'.

The plot window will update (in most cases in real-time) in response to changes of feature parameters and will re-scale the plot automatically. It may be desirable to keep

a specific, fixed scale as parameters are altered and, in this case, the ‘explicit scaling’ checkbox should be checked (which will activate the X-Y min/max textboxes). The ‘use current values’ button will auto-fill these textboxes with the current X-Y min/max values.

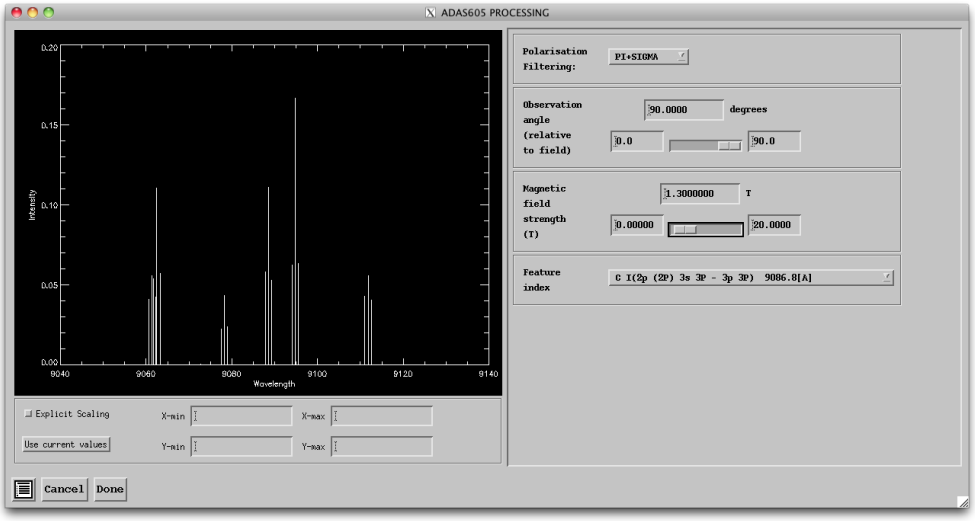


Figure 3.13: ADAS605 processing screen: allows interactive manipulation of chosen feature via custom control widgets in right hand panel, with graphical output in left panel.

As discussed in Section 3.1, some features have a multitude of parameters, some of which may have a natural grouping, which should be reflected in the way the control panel is laid out. AFG provides help to external programs like ADAS605 to organise their interface. Consider Fig. 3.14 — the AFG subordinate structure coming from the API has informed the GUI to provide a separate pop-up widget for the beam direction set (accessed by the ‘set beam direction...’ button). Additionally, the groups structure has allowed the GUI to group together the parameters related to the neutral beam in an organised way, encapsulated by rectangular section labelled ‘beam’.

The ADAS605 output screen (Fig. 3.15) follows the usual format i.e. a set of optional output types, each with the familiar ‘replace’, ‘default file’ and ‘file names’, checkbox, button and textbox respectively for specifying the output file. The output options available are ‘graphical output’ — saving the plot window as a graphic (postscript in the example Fig. 3.16), ‘X-Y output’ — the plot data in a plain text file as co-ordinate pairs (Fig. 3.17) and finally, ‘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 (Fig. 3.18). It is envisaged that production of this template source code will serve as an entry point to most users looking to utilise

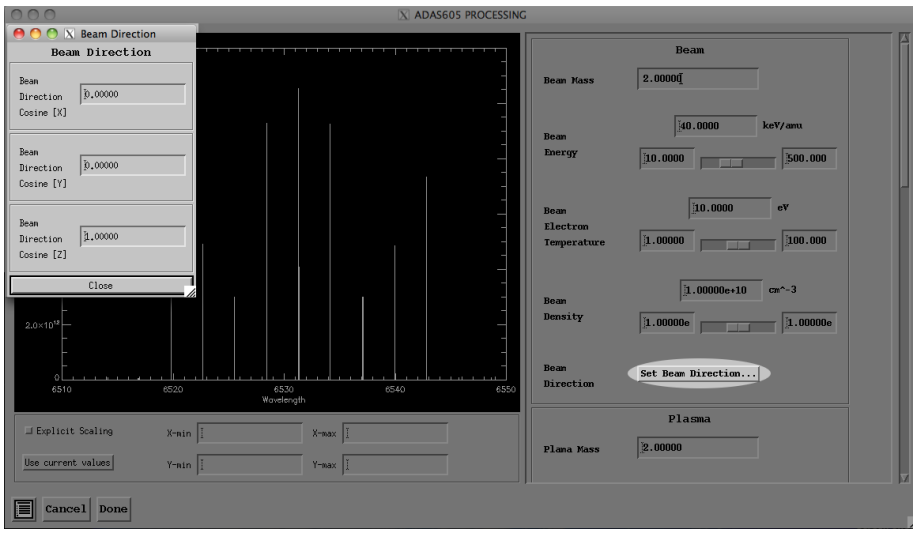


Figure 3.14: ADAS605 processing screen when the Stark feature is selected. In this case AFG groupings are in use — e.g. a selection of related parameter controls are arranged together in the panel labelled ‘Beam’. The AFG subordinate structures can also be seen at work — the ‘Beam Direction’ pop-up in the upper-left appears, upon pressing the highlighted button, exposing the component parameters that define it.

AFG in their own codes.

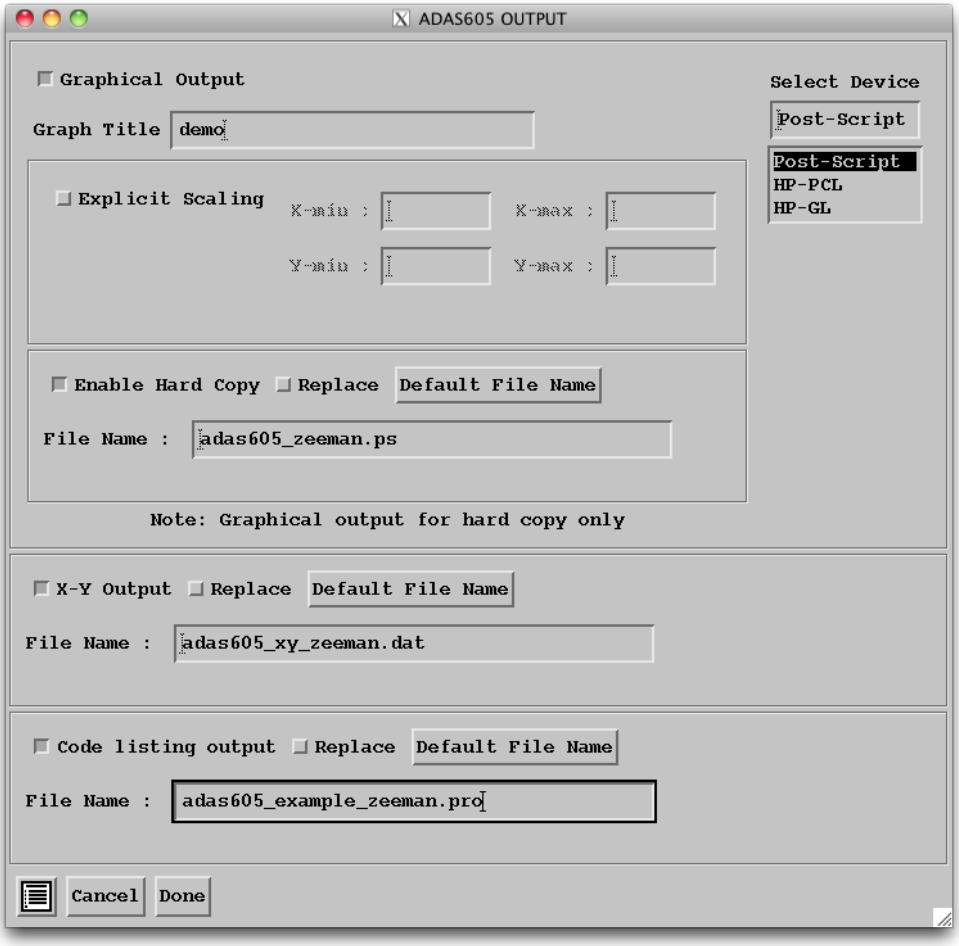


Figure 3.15: ADAS605 output screen: option to produce three types of output: graphical, X-Y plot data and finally, output of IDL source code that will recreate the feature as seen in the interactive window.

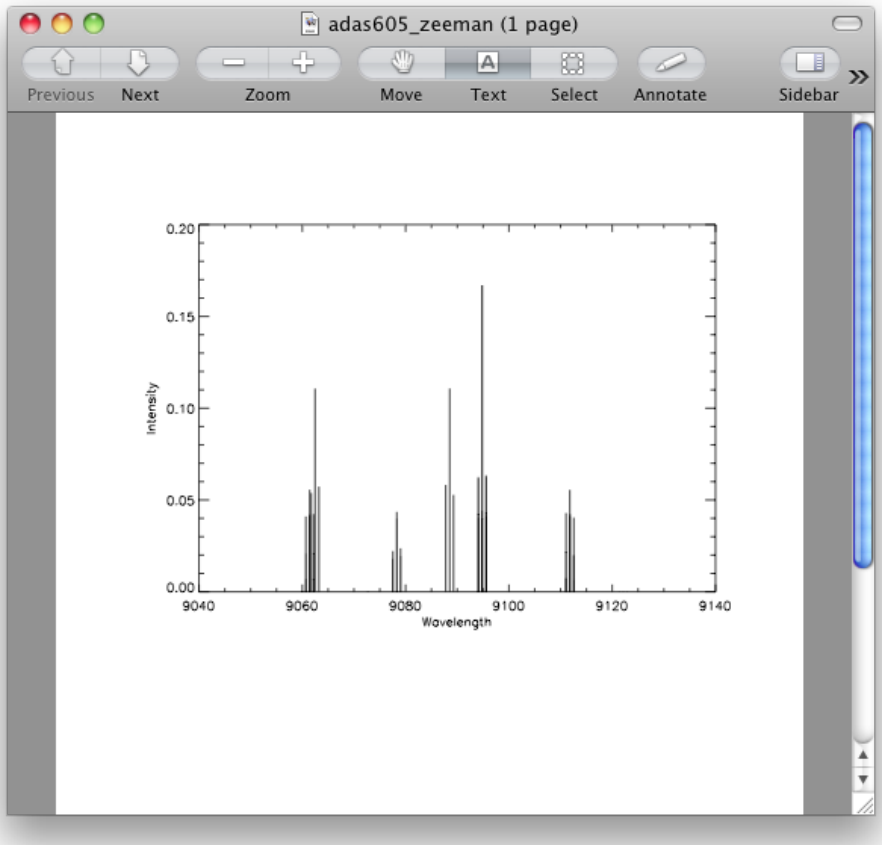


Figure 3.16: ADAS605 ‘graphical output’.

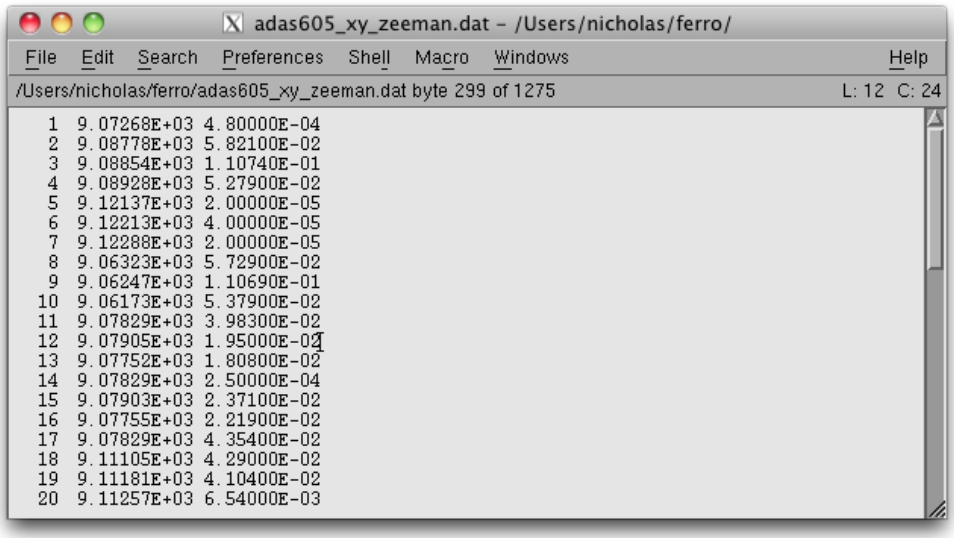


Figure 3.17: ADAS605 ‘X-Y output’.

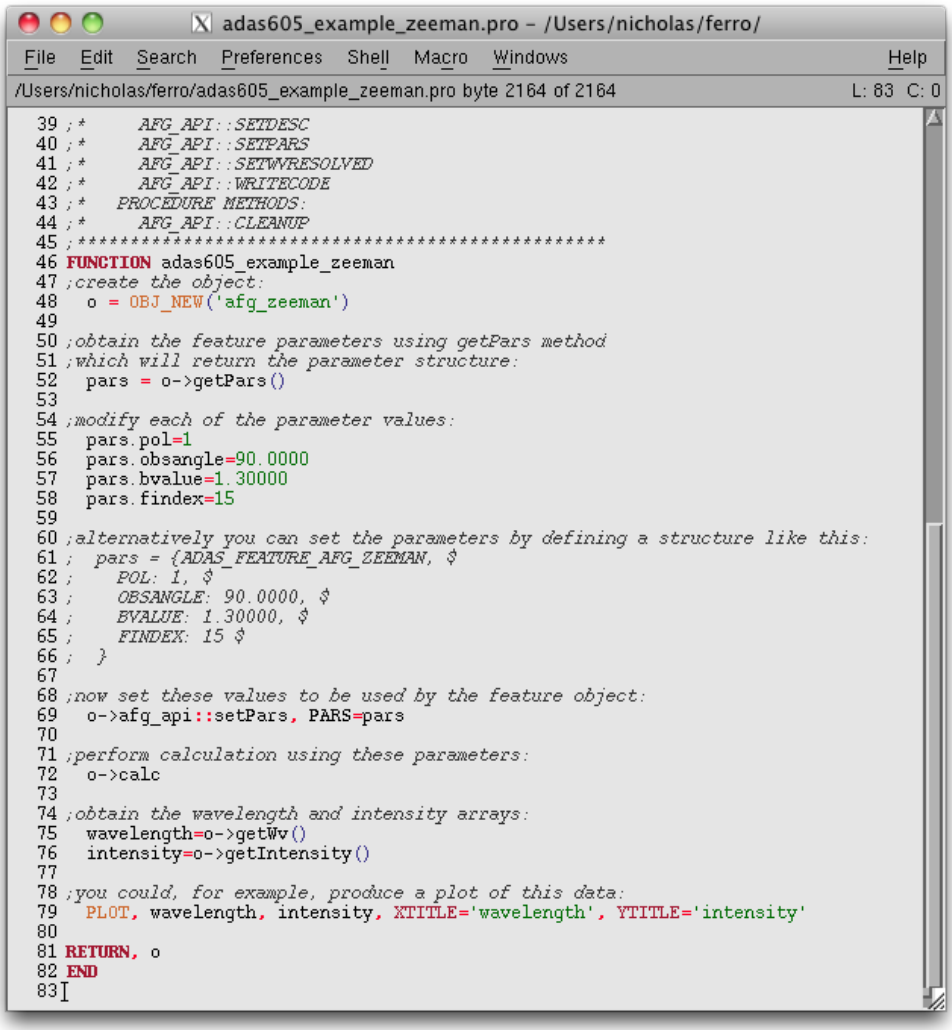


Figure 3.18: ADAS605 ‘code listing output’.

Chapter 4

Combinations of Functions for Spectral Fitting

The notation used in this section is such that functions are defined as $f\{a,b\}(x)$ where f is the symbol representing the function; a and b are enclosed in a set of {} and, as such, are considered parameters of the function f; finally x is the independant variable that the function f depends upon (in terms of spectra x is pixel / wavelength). The distinction between the parameters (a & b) and x has been made since, although in a mathematical sense they are all variables on which f depends, a and b remain constant as the function is evaluated for each value of x to build up a spectral profile for a given set of values of the parameters enclosed in {}. This means that $f\{a,b\}$ represents a function, rather than a single value of the function f evaluated for a given pair of a and b. In turn, this means that it is valid to write the convolution of two functions as $f\{a,b\} * g\{c,d\}$, with a value of that convolved function, at a given x: $[f\{a,b\} * g\{c,d\}](x)$. Additionally, the notation allows for the parameter set (a, b, etc.) to be functions themselves. Finally, it should be noted that 'non-fitting' parameters, that is to say properties of functions that will not vary during a fit (e.g. *AFG* parameters that do not have continuous *disptype* — see Section 3.1, or tolerance settings for the generating codes) are not included in the function definitions here.

4.1 Introduction

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, such as those provided by ADAS (via AFG, as detailed in 3.1).

At this point, consideration 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; this is reviewed in Section 4.12. Further to the documentation of model element calculation / partial derivative formulations, attention is given to combinations of functions likely to be commonly encountered in spectral analysis. Each of the functions defined here (including intermediates such as the broadening functions (eq. 4.5) are implemented, programmatically in the FFS system.

4.2 Functions Considered

4.2.1 Un-broadened Line

The most basic element that must be included as part of the generalised modelling system is a 'Dirac-Delta-like' function, attributing emissivity of an electronic transition to a single point in wavelength space; in this respect, the element shall be described as *un-broadened*. It should be noted that such an element is not of great use in isolation — at the very least a spectral line will exhibit *natural broadening* (excited states have a finite lifetime and the Heisenberg uncertainty principle suggests that there is an associated width associated with the energy/time uncertainty in the transition from upper state j to lower state i: $\Delta\nu_{j\to i} = \frac{A_{j\to i}}{4\pi}$). For atomic transitions, this effect is not significant and other broadening mechanisms (such as thermal Doppler and instrumental effects) dominate. However, the point remains — this element requires some form of broadening operator element to be useful in practical applications.

4.2.1.1 Definition

$$\varepsilon\{\lambda_0, \phi\}(\lambda) = \phi\delta(\lambda - \lambda_0) \tag{4.1}$$

where δ is the delta function (A.23 & A.24), λ_0 is the position of the line in wavelength space and ϕ is the line intensity.

The convolution of two such functions is:

$$\begin{aligned}\left[\varepsilon\{x_1, \phi_1\} * \varepsilon\{x_2, \phi_2\}\right](x) &= \int_{-\infty}^{+\infty} \varepsilon\{x_1, \phi_1\}(x')\varepsilon\{x_2, \phi_2\}(x - x')dx' \\ &= \phi_1\phi_2 \int_{-\infty}^{+\infty} \delta(x' - x_1)\,\delta(x - x' - x_2)\,dx'.\end{aligned} \tag{4.2}$$

Now, utilising the sifting property of the delta function (eq. A.25),

$$\begin{aligned}\left[\varepsilon\{x_1, \phi_1\} * \varepsilon\{x_2, \phi_2\}\right](x) &= \phi_1\phi_2\delta(x - x_1 - x_2) \\ &= \varepsilon\{(x_1 + x_2), \phi_1\phi_2\}\end{aligned} \tag{4.3}$$

4.2.2 Gaussian

One of the primary broadening mechanisms requiring representation in the modelling system, is *Doppler broadening*. The thermal motion of the emitting atoms, relative to the observer results in red or blue shifting of the emitted photons. The emitter velocity distribution results in a spectral profile that is Gaussian in shape. The Gaussian line shape is also commonly used to represent the instrument function associated with the spectrometer's finite resolution.

4.2.2.1 Definition

The normalised Gaussian function (centered on zero) is defined as:

$$G\{w_g\}(x) = \frac{C}{\sqrt{\pi}w_g} \exp\left(-\frac{C^2x^2}{w_g^2}\right) \tag{4.4}$$

where w_g is the full width at half maximum and the constant $C = 2\sqrt{\ln 2}$ (see Section A.4).

A Gaussian broadening function can then be described by:

$$B_g\left\{f\{\dots\}, w_g\right\}(x) = \left[G\{w_g\} * f\{\dots\}\right](x) = \int_{-\infty}^{+\infty} G\left\{w_g\right\}(x - x') f\{\dots\}(x')dx' \tag{4.5}$$

where $f\{\dots\}$ represents a generic function, with an unknown parameter set. Note that it is important to use a normalised Gaussian (i.e. of unit area) as the kernel in the convolution operation for the broadening. The area of this integral is the product of the area of the two operand functions (eq. A.6), therefore this choice of kernel preserves the area (intensity) of the original function.

A Gaussian line is the result of the Gaussian broadening function (eq. 4.5) applied to the line function (eq. 4.1) i.e. the convolution of the Gaussian function (eq. 4.4) with the line function (eq. 4.1):

$$\begin{aligned} I_g\{\lambda_0, \phi, w_g\}(\lambda) &= B_g\left\{\varepsilon\{\lambda_0, \phi\}, w_g\right\}(\lambda) \\ &= \left[G\left\{w_g\right\} * \varepsilon\{\lambda_0, \phi\}\right](\lambda) \\ &= \int_{-\infty}^{+\infty} G\left\{w_g\right\}(\lambda - \lambda')\varepsilon\{\lambda_0, \phi\}(\lambda')d\lambda' \\ &= \phi \int_{-\infty}^{+\infty} G\left\{w_g\right\}(\lambda - \lambda')\delta(\lambda' - \lambda_0)d\lambda' \\ &= \phi G\left\{w_g\right\}(\lambda - \lambda_0) \\ &= \frac{C\phi}{\sqrt{\pi} w_g} \exp\left(-\frac{C^2(\lambda - \lambda_0)^2}{w_g^2}\right). \end{aligned} \tag{4.6}$$

4.2.2.2 Partial Derivatives

The partial derivatives of the Gaussian line function, with respect to its parameters λ_0, w_g and ϕ are given by:

$$\frac{\partial}{\partial \lambda_0} I_g\left\{\lambda_0, \phi, w_g\right\}(\lambda) = \frac{2C^2(\lambda - \lambda_0)}{w_g^2} I_g\left\{\lambda_0, \phi, w_g\right\}(\lambda), \tag{4.7}$$

$$\frac{\partial}{\partial w_g} I_g\left\{\lambda_0, \phi, w_g\right\}(\lambda) = \frac{1}{w_g}\left(2C^2\frac{(\lambda - \lambda_0)^2}{w_g^2} - 1\right) I_g\left\{\lambda_0, \phi, w_g\right\}(\lambda), \tag{4.8}$$

$$\frac{\partial}{\partial \phi} I_g\left\{\lambda_0, \phi, w_g\right\}(\lambda) = \frac{1}{\phi} I_g\left\{\lambda_0, \phi, w_g\right\}(\lambda). \tag{4.9}$$

4.2.3 Doppler

Instead of abstracting to the (Gaussian) mathematical line shape, it may be preferable to have a Doppler element, which is defined by physical parameters (i.e. temperature), rather than line profile parameters (i.e. full-width at half maximum).

Assuming a Maxwell speed distribution along a line of sight, $f(v)$, then $f(v)dv$ is the fraction of particles with speed in range $v \rightarrow v + dv$ such that:

$$f(v)dv = \sqrt{\frac{m}{2\pi kT}} exp\left(-\frac{mv^2}{2kT}\right) dv. \tag{4.10}$$

For non-relativistic thermal particles, the doppler shift is given by $\lambda = \lambda_0 \left(1 + \frac{v}{c}\right)$ and so, by substitution:

$$f(\lambda)d\lambda = \sqrt{\frac{mc^2}{2\pi kT\lambda_0^2}} exp\left(-\frac{mc^2(\lambda - \lambda_0)^2}{2kT\lambda_0^2}\right) d\lambda. \tag{4.11}$$

This is a normal distribution (Gaussian) with standard deviation:

$$\sigma = \lambda_0 \sqrt{\frac{kT}{mc^2}}. \tag{4.12}$$

The full width at half maximum (FWHM), w_g, is related to the standard deviation σ as:

$$w_g = 2\sigma\sqrt{2\ln 2}. \tag{4.13}$$

So, the FWHM for the Doppler element, w_d, is given by:

$$w_d = \frac{2\lambda_0}{c} \sqrt{2\ln 2 \left(\frac{kT}{m}\right)} \tag{4.14}$$

where m is the mass of the emitter.

4.2.3.1 Definition

The Doppler element is then defined in terms of the Gaussian element, but with a width parameterised in terms of the the thermal temperature, as prescribed by eq. 4.14:

$$D\{\lambda_0, \phi, T\}(\lambda) = I_g \{\lambda_0, \phi, w_d(\lambda_0, T)\} (\lambda) \tag{4.15}$$

note that there is a hidden variable (the mass, m) here — it is not included as it is considered by FFS to be a *property* of the function, rather than a fitting *parameter*.

4.2.3.2 Partial Derivatives

$$\frac{\partial}{\partial \lambda_0} D\{\lambda_0, \phi, T\}(\lambda) = \left(\frac{2C^2(\lambda - \lambda_0)}{[w_d(\lambda_0, T)]^2}\left(1 + \frac{1}{\lambda_0}\right) - \frac{1}{\lambda_0}\right) D\{\lambda_0, \phi, T\}(\lambda), \tag{4.16}$$

$$\frac{\partial}{\partial T} D\{\lambda_0, \phi, T\}(\lambda) = \frac{1}{2T}\left(2C^2 \frac{(\lambda - \lambda_0)^2}{[w_d(\lambda_0, T)]^2} - 1\right) D\{\lambda_0, \phi, T\}(\lambda), \tag{4.17}$$

$$\frac{\partial}{\partial \phi} D\{\lambda_0, \phi, T\}(\lambda) = \frac{1}{\phi} D\{\lambda_0, \phi, T\}(\lambda). \tag{4.18}$$

4.2.4 Lorentzian

As discussed in Section 2.4, another commonly observed broadening mechanism is *pressure broadening* — the Coulomb interaction of the emitter with neighbouring particles results in Stark splitting of the states. The resultant line shape is approximately Lorentzian.

4.2.4.1 Definition

The normalised Lorentzian function (centered on zero) is defined as:

$$L\{w_l\}(x) = \left(\frac{1}{\pi}\right) \frac{\frac{w_l}{2}}{x^2 + \left(\frac{w_l}{2}\right)^2} \tag{4.19}$$

where w_l is the full width at half maximum.

A Lorentzian broadening function can then be described by:

$$\begin{aligned} B_l\{f\{\dots\}, w_l\}(x) &= [L\{w_l\} * f\{\dots\}](x) \\ &= \int_{-\infty}^{+\infty} L\{w_l\}(x - x') f\{\dots\}(x) dx'. \end{aligned} \tag{4.20}$$

Similarly to the Gaussian broadener (eq. 4.5), a kernel of unit area is used to preserve the operand area (intensity) in the result.

A Lorentzian line is the result of the Lorentzian broadening function (eq. 4.5) applied to the line function (eq. 4.1) i.e. the convolution of the Lorentzian function (eq. 4.19)

with the line function (eq. 4.1):

$$\begin{aligned} I_l\{\lambda_0,\phi,w_l\}(\lambda) &= B_l\{\varepsilon\{\lambda_0,\phi\},w_l\}(\lambda) \\ &= \left[L\{w_l\} * \varepsilon\{\lambda_0,\phi\}\right](\lambda) \\ &= \int_{-\infty}^{+\infty} L\{w_l\}(\lambda-\lambda')\varepsilon\{\lambda_0,\phi\}(\lambda)d\lambda' \\ &= \phi\int_{-\infty}^{+\infty} L\{w_l\}(\lambda-\lambda')\delta(\lambda'-\lambda_0)d\lambda' \\ &= \phi L\{w_l\}(\lambda-\lambda_0) \\ &= \left(\frac{\phi}{\pi}\right)\frac{\frac{w_l}{2}}{(\lambda-\lambda_0)^2+\left(\frac{w_l}{2}\right)^2}. \end{aligned} \tag{4.21}$$

4.2.4.2 Partial Derivatives

The partial derivatives of the Lorentzian line function, with respect to its parameters λ_0, w_l and ϕ are given by:

$$\frac{\partial}{\partial\lambda_0}I_l\{\lambda_0,\phi,w_l\}(\lambda) = \left(\frac{2(\lambda-\lambda_0)}{(\lambda-\lambda_0)^2+\left(\frac{w_l}{2}\right)^2}\right)I_l\{\lambda_0,w_l,A_l\}(\lambda), \tag{4.22}$$

$$\frac{\partial}{\partial w_l}I_l\{\lambda_0,\phi,w_l\}(\lambda) = \left(\frac{1}{w_l}\right)\frac{(\lambda-\lambda_0)^2-\frac{w_l{}^2}{2}}{\left((\lambda-\lambda_0)^2+\frac{w_l{}^2}{2}\right)}I_l\{\lambda_0,w_l,A_l\}(\lambda), \tag{4.23}$$

$$\frac{\partial}{\partial\phi}I_l\{\lambda_0,\phi,w_l\}(\lambda) = \frac{1}{\phi}I_l\{\lambda_0,w_l,A_l\}(\lambda). \tag{4.24}$$

4.2.5 Voigt

It is possible that a spectral line will be subject to several broadening effects — pressure broadening could result in a Lorentzian distribution of the line intensity, but the line could also exhibit a Gaussian component due to its thermal temperature. Regardless of individual broadening mechanisms, lines will have a characteristic width — the instrument function. Consideration must be given to combining these effects and firstly, the combined result of Gaussian and Lorentzian broadening is considered.

4.2.5.1 Definition

The Voigt function is the convolution of the Gaussian and Lorentzian functions (eqs 4.4 and 4.19 respectively):

$$\begin{aligned}
V\left\{w_g, w_l\right\}(x) &= \left[L\{w_l\} * G\{w_g\}\right](x) \\
&= \int_{-\infty}^{+\infty} L\{w_l\}(x-x')G\{w_g\}(x')dx' \\
&= \int_{-\infty}^{+\infty} \left(\frac{1}{\pi}\right) \frac{\frac{w_l}{2}}{(x-x')^2 + \left(\frac{w_l}{2}\right)^2} \left(\frac{C}{\sqrt{\pi w_g}}\right) \\
&\qquad \exp\left(-\frac{C^2(x')^2)}{w_g^2}\right) dx' \\
&= \frac{1}{\pi^{\frac{3}{2}}} \int_{-\infty}^{+\infty} \frac{\frac{w_l}{2}}{\left(x - \frac{w_g}{C}t\right)^2 + \left(\frac{w_l}{2}\right)^2} \exp\left(-t^2\right) dt \\
&= \frac{1}{\pi^{\frac{3}{2}}} \int_{-\infty}^{+\infty} \frac{\left(\frac{C}{w_g}\right)^2 \left(\frac{w_l}{2}\right)}{\left(\frac{C}{w_g}x - t\right)^2 + \left(\frac{C}{w_g}\right)^2 \left(\frac{w_l}{2}\right)^2} \exp\left(-t^2\right) dt, \qquad (4.25)
\end{aligned}$$

$$\text{where} \quad t = \frac{C}{w_g}x' \implies \frac{dt}{dx'} = \frac{C}{w_g}.$$

Note that:

$$V\left\{w_g, w_l\right\}(x) = \left[L\{w_l\} * G\{w_g\}\right](x) \equiv B_l\{G\{w_g\}, w_l\}(x) \equiv B_g\{L\{w_g\}, w_l\}(x). \qquad (4.26)$$

Equation 4.25 can be re-written in terms of the complex error function (see A.8.1). Specifically, it is the real part of this function, K (eqn A.40) that is useful here, with the parameters of K, a and b given by:

$$a = \frac{Cx}{w_g} \quad \text{and} \quad b = \frac{Cw_l}{2w_g}.$$

K is then substituted into equation 4.25, such that we obtain the result:

$$V\{w_g, w_l\}(x) = \frac{C}{\sqrt{\pi} w_g} K\left(\frac{Cx}{w_g}, \frac{Cw_l}{2w_g}\right). \qquad (4.27)$$

A Voigt broadening function can then be described by:

$$\begin{aligned} B_v\left\{f\{\dots\}, w_g, w_l\right\}(x) &= \left[V\{w_g, w_l\} * f\{\dots\}\right](x) \\ &= \int_{-\infty}^{+\infty} V\left\{w_g, w_l\right\}(x - x')f(x)dx'. \end{aligned} \tag{4.28}$$

Similarly to the Gaussian and Lorentzian broadening functions (eq. 4.5 and eq. 4.20), a kernel of unit area is used to preserve the operand area (intensity) in the result.

Application of the Voigt broadening function (eq. 4.28) to the line function (eq. 4.1) (equivalent to convolution of (eq. 4.27) and (eq. 4.1)) results in a Voigt line profile:

$$\begin{aligned} I_v\{\lambda_0, \phi, w_g, w_l, \phi\}(\lambda) &= B_v\{\varepsilon\{\lambda_0, \phi\}, w_g, w_l\} \\ &= \left[V\left\{w_g, w_l\right\} * \varepsilon\left\{\lambda_0, \phi\right\}\right](\lambda) \\ &= \int_{-\infty}^{+\infty} V\left\{w_g, w_l\right\}(\lambda - \lambda')\varepsilon\left\{\lambda_0, \phi\right\}(\lambda')d\lambda' \\ &= \phi \int_{-\infty}^{+\infty} V\left\{w_g, w_l\right\}(\lambda - \lambda')\delta(\lambda' - \lambda_0)d\lambda' \\ &= \phi V\left\{w_g, w_l\right\}(\lambda - \lambda_0) \\ &= \frac{C\phi}{\sqrt{\pi}w_g} K\left(\frac{C(\lambda - \lambda_0)}{w_g}, \frac{Cw_l}{2w_g}\right). \end{aligned} \tag{4.29}$$

The parameters, a and b, of K are in this case:

$$a = \frac{C(\lambda - \lambda_0)}{w_g} \quad \text{and} \quad b = \frac{Cw_l}{2w_g}.$$

4.2.5.2 Partial Derivatives

The derivatives of this function can also be expressed in terms of complex error function (see Section A.8.1) and its partial derivatives with respect to the parameters a and b (see Section A.8.2), along with the partial derivatives of a and b themselves, with respect to the parameters of interest λ_0, w_g and w_l (eq. 4.30).

First, note that:

$$\begin{aligned} \frac{\partial a}{\partial w_g} &= -\frac{C(\lambda - \lambda_0)}{w_g^2}, & \frac{\partial a}{\partial \lambda_0} &= -\frac{C}{w_g}, \\ \frac{\partial b}{\partial w_g} &= -\frac{Cw_l}{2w_g^2}, & \frac{\partial b}{\partial w_l} &= \frac{C}{2w_g}. \end{aligned} \tag{4.30}$$

Using these results (eq. 4.30) together with those from Section A.8.2, we can define

the partial derivatives as follows:

$$\begin{aligned}\frac{\partial}{\partial\lambda_0}I_v\{\lambda_0,\phi,w_g,w_l,\phi\}(\lambda) &= \frac{C\phi}{\sqrt{\pi}w_g}\left(\frac{\partial}{\partial a}(K(a,b))\frac{\partial a}{\partial\lambda_0}\right)\\ &= \frac{C\phi}{\sqrt{\pi}w_g}\left(2(bL(a,b)-aK(a,b)))\left(-\frac{C}{w_g}\right)\right)\\ &= -\frac{2C^2\phi}{\sqrt{\pi}w_g^2}\left(bL(a,b)-aK(a,b)\right), \end{aligned} \tag{4.31}$$

$$\begin{aligned}\frac{\partial}{\partial w_g}I_v\{\lambda_0,\phi,w_g,w_l,\phi\}(\lambda) &= \frac{\partial}{\partial w_g}\left(\frac{C\phi}{\sqrt{\pi}w_g}\right)K(a,b)\\ &\quad+\frac{C\phi}{\sqrt{\pi}w_g}\left(\frac{\partial}{\partial a}(K(a,b))\frac{\partial a}{\partial w_g}+\frac{\partial}{\partial b}(K(a,b))\frac{\partial b}{\partial w_g}\right)\\ &= \left(-\frac{C\phi}{\sqrt{\pi}w_g^2}\right)K(a,b)\\ &\quad+\frac{C\phi}{\sqrt{\pi}w_g}\left(2(bL(a,b)-aK(a,b)))\left(-\frac{C(\lambda-\lambda_0)}{w_g^2}\right)\right.\\ &\qquad\left.+\left(aL(a,b)+bK(a,b)-\frac{1}{\sqrt{\pi}}\right)\left(-\frac{Cw_l}{2w_g^2}\right)\right)\\ &= \left(-\frac{C\phi}{\sqrt{\pi}w_g^2}\right)K(a,b)\\ &\quad-\frac{2C^2\phi}{\sqrt{\pi}w_g^3}\left((bL(a,b)-aK(a,b))(\lambda-\lambda_0)\right.\\ &\qquad\left.+\left(L(a,b)a+bK(a,b)-\frac{1}{\sqrt{\pi}}\right)\left(\frac{w_l}{2}\right)\right), \end{aligned} \tag{4.32}$$

$$\begin{aligned}\frac{\partial}{\partial w_l}I_v\{\lambda_0,\phi,w_g,w_l,\phi\}(\lambda) &= \frac{C\phi}{\sqrt{\pi}w_g}\left(\frac{\partial}{\partial b}(K(a,b))\frac{\partial b}{\partial w_l}\right)\\ &= \frac{C\phi}{\sqrt{\pi}w_g}2\left(aL(a,b)+bK(a,b)-\frac{1}{\sqrt{\pi}}\right)\left(\frac{C}{2w_g}\right)\\ &= \frac{C^2\phi}{\sqrt{\pi}w_g^2}\left(aL(a,b)+bK(a,b)-\frac{1}{\sqrt{\pi}}\right) \end{aligned} \tag{4.33}$$

and

$$\begin{aligned}\frac{\partial}{\partial\phi}I_v\{\lambda_0,\phi,w_g,w_l,\phi\}(\lambda) &= \frac{\partial}{\partial\phi}\left(\frac{C\phi}{\sqrt{\pi}w_g}\right)K(a,b)\\ &= \frac{C}{\sqrt{\pi}w_g}K(a,b)\\ &= \frac{1}{\phi}I_v\{\lambda_0,\phi,w_g,w_l,\phi\}(\lambda).\end{aligned} \tag{4.34}$$

4.2.6 Linear Background

Typically, spectroscopic data will exhibit some sort of background emission, in addition to discrete spectral lines (and/or special features). In some case, this background could, itself, be diagnostically useful (e.g. analysis of Bremstrahhlung emission) and so, the background should be modelled using a special feature element. However, in other cases, the background may not be of any real interest or a suitable model may not exist. In this situation, it may be desirable to use a cruder representation for the background emission e.g. a simple line function. The definition of such a function and its partial derivatives is trivial, but included for completeness.

$$\mathrm{y}\{m,c\}(x) = m\,(x - x_0) + c \tag{4.35}$$

4.2.6.1 Partial Derivatives

$$\frac{\partial}{\partial m}y(x) = x - x_0 \tag{4.36}$$

$$\frac{\partial}{\partial c}y(x) = 1 \tag{4.37}$$

4.2.7 Addition Operator

In constructing a completely flexible, modular modelling system, it is necessary to include some functions that have the sole purpose of connecting the other components together. In FFS, these are known as *operator elements*. It should be noted that some of these so-called operator element functions have already been defined above, whilst deriving some of the functions — namely the broadening elements (eq. 4.5, 4.20, 4.28).

One of the most important operator elements to include in the system is a simple addition operator; to allow for superposition of various primitive line features and/or

special features.

4.2.7.1 Definition

$$\boxplus \{f_1\{\dots\} \dots f_n\{\dots\}\}(x) = \sum_{i=1}^{n} f_i\{\dots\}(x) \tag{4.38}$$

4.2.7.2 Partial Derivatives

For parameters p of one, or more, of the functions f_i:

$$\frac{\partial}{\partial p} \boxplus \{f_1\{\dots\} \dots f_n\{\dots\}\}(x) = \sum_{i=1}^{N} \frac{\partial}{\partial p} f_i\{\dots\}(x) \tag{4.39}$$

4.2.8 Scale Factor Operator

It is likely that a special feature modelling code will provide a synthetic form which will provide the relative intensities of the components, rather than absolute intensities. It is also possible to define a pseudo special feature by specifying set of connections between feature parameters using the coupling system. In either case, such data must be scaled for confrontation with experimental data. It is possible to define a generalised multiplication operator that would multiply the output of the two operand elements, but for the most commonly used scenario (simple scale factor multiplication) this would require creation of another model element, for the sole purpose of holding the scaling parameter. It is more useful to define this operator such that it only takes a single operand function and that the scaling parameter is integral.

4.2.8.1 Definition

$$\boxtimes \{f\{\dots\}, a\}(x) = af\{\dots\}(x) \tag{4.40}$$

4.2.8.2 Partial Derivatives

$$\frac{\partial}{\partial a} \boxtimes \{f\{\dots\}, a\}(x) = f\{\dots\}(x) \tag{4.41}$$

For all other parameters p of the function f:

$$\frac{\partial}{\partial p} \boxtimes \{f\{\dots\}, a\}(x) = a\frac{\partial}{\partial p} f\{\dots\}(x) \tag{4.42}$$

4.2.9 Shift Operator

Many of the special feature codes will produce a spectrum which is *wavelength resolved* i.e. a broadened feature; a true line shape rather than point impulse spikes. It is possible that although the code is accurately predicting the feature in terms of intensity, that the positioning is less accurate. The observed special feature emission could also be subject Doppler shifting from the root position. Whatever the cause, FFS must provide an operator for shifting features when there is disparity with the experimental data to be fitted — the shift itself, of course, could be diagnostic. The shift operator must also be able to handle shifts in terms of number of pixels on a CCD as well as those specified by wavelength.

4.2.9.1 Definition

$$\mathcal{S}\{f\{\dots\}, s\}(x) = f\{\dots\}(x - s) \tag{4.43}$$

4.2.9.2 Partial Derivatives

$$\frac{\partial}{\partial s}\mathcal{S}\{f\{\dots\}, s\}(x) = -\frac{\partial}{\partial(x - s)} f\{\dots\}(x - s) \tag{4.44}$$

For all other parameters p of the function f:

$$\frac{\partial}{\partial p}\mathcal{S}\{f\{\dots\}, s\}(x) = \frac{\partial}{\partial p} f\{\dots\}(x - s) \tag{4.45}$$

4.2.10 AFG

One of the most significant modules included in FFS is the AFG interface element. This allows seamless integration of the ADAS special features in FFS. The model element interacts with the AFG API for access. This means that only a single element class is required for all of the models within ADAS; it is not necessary to implement an individual element class for any future AFG provided features.

4.2.10.1 Definition

$$f\{\text{func\_name}, \dots\}(x) = \text{AFG}_{\text{func\_name}}\{\dots\}(x) \tag{4.46}$$

4.2.10.2 Partial Derivatives

$$\frac{\partial}{\partial p} f\{\text{func\_name}, \dots\}(x) = \frac{\partial}{\partial p} \text{AFG}_{\text{func\_name}}\{\dots\}(x) \tag{4.47}$$

4.3 Practical Examples

4.3.1 Convolution of two Normalized, Un-shifted Gaussian Functions

$$\begin{aligned}
&\Big[G\{w_{g_1}\} * G\{w_{g_2}\}\Big](x) \\
&\quad = \int_{-\infty}^{+\infty} G\{w_{g_1}\}(x - x') G\{w_{g_2}\}(x') dx' \\
&\quad = \frac{C^2}{\pi w_{g_1} w_{g_2}} \int_{-\infty}^{+\infty} \exp\left(\frac{-C^2(x - x')^2}{w_{g_1}^2}\right) \exp\left(\frac{-C^2(x')^2}{w_{g_2}^2}\right) dx' \\
&\quad = \frac{C^2}{\pi w_{g_1} w_{g_2}} \int_{-\infty}^{+\infty} \exp\left(\frac{-C^2(x^2 - 2xx' - (x')^2)}{w_{g_1}^2}\right) \\
&\qquad\qquad \exp\left(\frac{-C^2(x')^2}{w_{g_2}^2}\right) dx' \\
&\quad = \frac{C^2}{\pi w_{g_1} w_{g_2}} \exp\left(\frac{-C^2 x^2}{w_{g_1}^2}\right) \int_{-\infty}^{+\infty} \exp\left(-C^2(x')^2 \left(\frac{1}{w_{g_1}^2} + \frac{1}{w_{g_2}^2}\right)\right. \\
&\qquad\qquad \left. + \left(\frac{2C^2 x}{w_{g_1}^2}\right) x'\right) dx'
\end{aligned} \tag{4.48}$$

Equation 4.48 is a standard integral:

$$\int_{-\infty}^{+\infty} \exp\left(-p^2 x^2 \pm qx\right) dx = \exp\left(\frac{q^2}{4p^2}\right) \frac{\sqrt{\pi}}{p} \quad [p > 0] \tag{4.49}$$

$$\text{with} \quad p = C\sqrt{\frac{1}{w_{g_1}^2} + \frac{1}{w_{g_2}^2}} \quad \text{and} \quad q = \frac{2C^2 x^2}{w_{g_1}^2}$$

Now it is possible to re-write equation 4.48 as:

$$\begin{aligned}\Big[G\{w_{g_1}\} * G\{w_{g_2}\}\Big](x) &= \frac{C}{\sqrt{\pi} w_{g_1} w_{g_2} \sqrt{\frac{1}{w_{g_1}^2} + \frac{1}{w_{g_2}^2}}} \exp\left(\frac{-C^2x^2}{w_{g_1}^2}\right) \exp\left(\frac{-C^2x^2}{w_{g_1}^4\left(\frac{1}{w_{g_1}^2} + \frac{1}{w_{g_2}^2}\right)}\right) \\ &= \frac{C}{\sqrt{\pi}\sqrt{w_{g_1}^2 + w_{g_2}^2}} \exp\left(\left(\frac{-C^2x^2}{w_{g_1}^2}\right)\left(1 - \frac{1}{w_{g_1}^2\left(\frac{1}{w_{g_1}^2} + \frac{1}{w_{g_2}^2}\right)}\right)\right) \\ &= \frac{C}{\sqrt{\pi}\sqrt{w_{g_1}^2 + w_{g_2}^2}} \exp\left(\left(\frac{-C^2x^2}{w_{g_1}^2}\right)\left(\frac{w_{g_1}^2}{w_{g_1}^2 + w_{g_2}^2}\right)\right) \\ &= \frac{C}{\sqrt{\pi}\sqrt{w_{g_1}^2 + w_{g_2}^2}} \exp\left(\frac{-C^2x^2}{w_{g_1}^2 + w_{g_2}^2}\right) \end{aligned} \tag{4.50}$$

such that we can write:

$$\Big[G\{w_{g_1}\} * G\{w_{g_2}\}\Big](x) = G\left\{\sqrt{w_{g_1}^2 + w_{g_2}^2}\right\}(x) \tag{4.51}$$

Therefore, the convolution of two normalized, un-shifted Gaussian functions results in another Gaussian profile, with a FWHM that is given by $w_{\text{new}} = \sqrt{w_{g_1}^2 + w_{g_2}^2}$. This is an important result for optimisation of this model representation in Section 4.7.

4.3.2 Convolution of N-Gaussian Profile with Gaussian

We can define an instrumental profile function in terms of a sum of Gaussian functions, each with their own area (φ_i), width (w_{g_i}) and finite shift (τ_i):

$$\begin{aligned} I_{\text{I.F.}}\{\tau_1, \varphi_1, w_1, \ldots, \tau_n, \varphi_n, w_n\} &= \sum_{i=1}^{n} B_g\left\{\varepsilon\{\tau_i, \varphi_i\}, w_{g_i}\right\}(x) \\ &= \sum_{i=1}^{n}\Big[G\{w_{g_i}\} * \varepsilon\{\tau_i, \varphi_i\}\Big](x) \end{aligned} \tag{4.52}$$

Consider the convolution of such a profile with a Gaussian line:

$$I_g\{\lambda_0, \phi, w_g\} * I_{\text{I.F.}}\{\{\tau_i\}, \{\varphi_i\}, \{w_{g_i}\}\}(\lambda)$$

from the distributive property of convolution (eqn A.4),

$$= \sum_{i=1}^{n}\Big[\Big(G\{w_g\} * \varepsilon\{\lambda_0, \phi\}\Big) * \Big(G\{w_{g_i}\} * \varepsilon\{\tau_i, \varphi_i\}\Big)\Big](\lambda)$$

and using the the associative property of convolution (eqn A.3),

$$= \sum_{i=1}^{n} \left[\left(G\{w_g\} * G\{w_{g_i}\} \right) * \left(\varepsilon\{\lambda_0, \phi\} * \varepsilon\{\tau_i, \varphi_i\} \right) \right] (\lambda)$$

substituting the result of equations 4.3 and 4.50,

$$= \sum_{i=1}^{n} \left[G\left\{ \sqrt{w_g^2 + w_{g_i}^2} \right\} * \varepsilon\{(\lambda_0 + \tau_i), (\phi\varphi_i)\} \right] (\lambda)$$

from the result of eq. 4.6,

$$= \sum_{i=1}^{n} I_g \left\{ (\lambda_0 + \tau_i), (\phi\varphi_i), \sqrt{w_g^2 + w_{g_i}^2} \right\} (\lambda)$$

$$= \phi \sum_{i=1}^{n} \frac{C\varphi_i}{\sqrt{\pi}\sqrt{w_g^2 + w_{g_i}^2}} \exp\left(\frac{-C^2(\lambda - \lambda_0 - \tau_i)^2}{w_g^2 + w_{g_i}^2} \right) \tag{4.53}$$

4.4 Framework for Feature Synthesis

Theoretical representations of experimental spectra can be considered to be comprised of a set of model elements; from simpler mathematical line shapes to more complex special feature representations. One can also define a set of 'operator elements' which can apply a function to the results of other elements, in some cases the function will bring together combinations of element results to represent the spectra in its entirety. Section 4.2 has detailed the mathematical representation of the various element types. Programmatically, each of the elements can be represented by an object that provides a method for performing calculation of that component of the spectra. This is the approach taken by the package developed for this work — the Framework for Feature synthesis (FFS). The computational organisation of the package can be seen in Fig. 4.1.

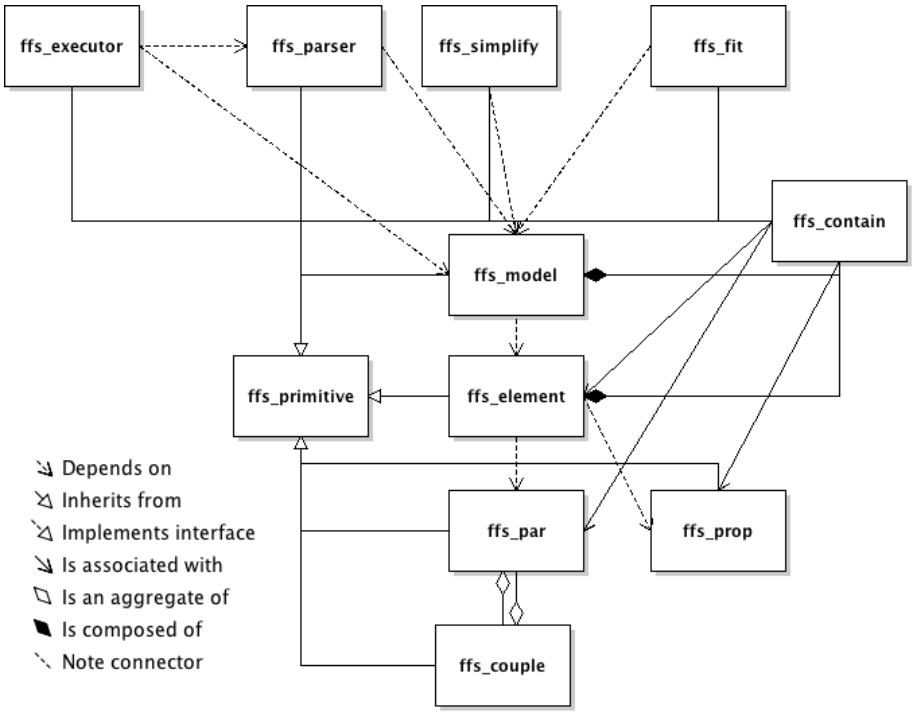


Figure 4.1: Class diagram for FFS.

The 'ffs_model' class is central to FFS, acting as the main manager of the spectral model — providing control of the component features (or model elements) and respective parameters. The model class can set parameter values, limits, coupling (see Section 4.6) and will enact the evaluation of the model spectra and partial derivatives. The framework is constructed such that 'ffs_model' objects use an 'ffs_contain' object to manage a set of 'ffs_element' objects, which, themselves, use ffs_contain objects to manage a set of 'ffs_par' and 'ffs_prop' objects. This basic view of the hierarchy, for a

simple example model, is displayed in Fig. 4.2.

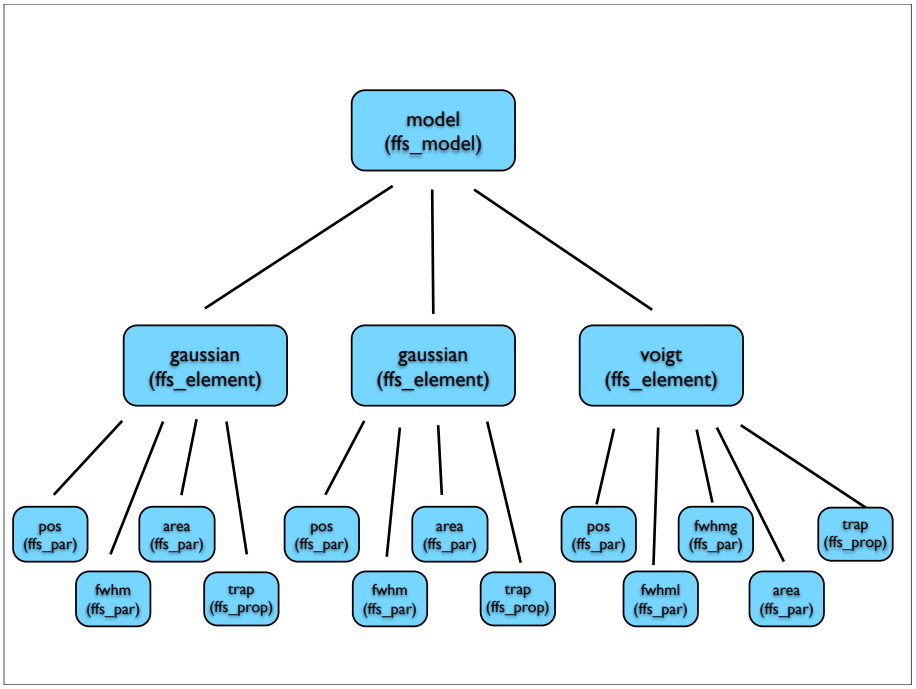


Figure 4.2: The model-element-par hierarchy for a simple model in FFS, consisting of two Gaussian lines and a voigt line shape. The Gaussian shapes have three parameters: position, full-width at half maximum and area. The voigt has four: position, lorentzian component of width, Gaussian component of width and area. The three elements, in this case also have a property 'trap' — see main text for discussion on *ffs_prop* objects.

To clarify, the use of ffs_par and ffs_prop: a distinction has been made between what are considered to be fitting parameters (those which may be varied during a fit to experimental spectra) and those which are considered to be properties / settings of the feature codes, but considered as static quantities with respect to fitting. This is related to the identification of parameter types discussed in Section 3.1. Note that this should not be confused with the ability to set (possible) fitting parameters to a fixed value during a fit.

In terms of implementation, ffs_element is an abstract class, from which FFS component features should inherit. Figure, 4.3 shows 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.

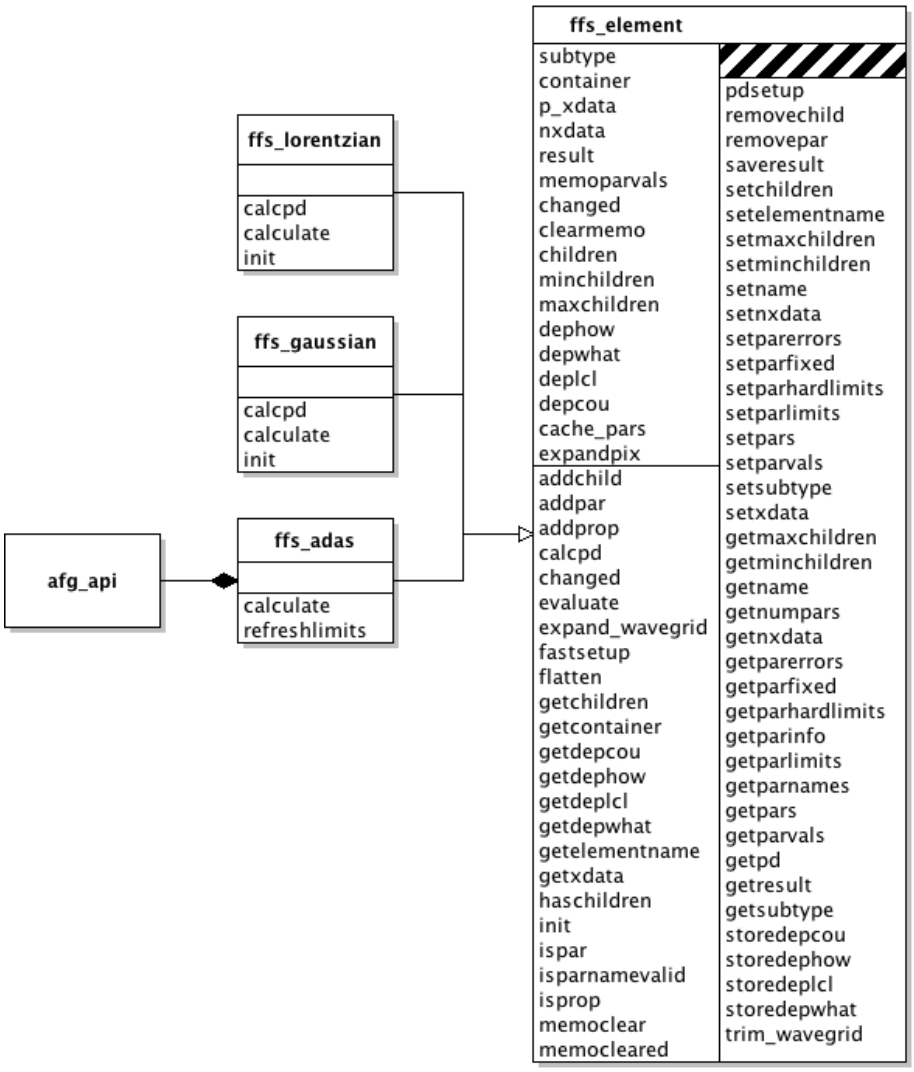


Figure 4.3: FFS element class and some example subclasses. Note that the class data entries are intentionally blank for the subclasses — they only have inherited data members. Also note that a summary of the methods of the core FFS classes is presented in Appendix D.

An element of particular note is 'ffs_adas', which interfaces with AFG (discussed in sec 3.1). Since AFG standardises access to the ADAS special features, all current and future inclusions are immediately available for use in FFS via this class.

Figure 4.2 displayed a model in which there is only a single layer of elements in the tree structure — elements that are independent of each other. As mentioned previously, 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.

Manually setting such parent-child relationships for the model elements would soon become cumbersome for a model of any degree of complexity, so it was quickly established that it would be important to find another way to define such relations. This, together with the need for a quick easy method to set parameter values and limits for fitting, as well as a method of defining cross-element parameter coupling, led to the creation of a new scripting language to define such information for the model.

4.5 Model Definition Language

Section 4.4 discussed the formation of a system to manage complicated spectra by means of modular constructs referred to as elements. In the case of an operator element, it requires input of the result of other elements. This, of course, means that such elements must be provided with a list of operand elements to act upon. In fact, in order to specify the construct for an arbitrarily complex model spectra, it was necessary to set out a model definition language (MDL). The syntax of this language was chosen to follow the format of the 'LISt Processor' language (LISP) syntax [65]. This means that a 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.

The expressions defining elements take the form of that shown in Fig. 4.4 where

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

Figure 4.4: MDL — element definition syntax.

'elementclass' is the name of the class type of the element, 'operands' (as noted above) is optional and can in fact be a list of element definition expressions, 'elementname', as expected, specifies a reference name for the element being defined. The 'optinput' parameter, as the name suggests, is optional and simply provides a method of passing an additional parameter to the specified element class at the point of creation.

To illustrate the use of this syntax consider a simple example model spectrum (named 'example') shown in Fig. 4.5 comprised of a Gaussian broadening function (eq. 4.5) to represent an instrument (apparatus) function, with two lines of interest, one represented by a Lorentzian shape (eq. 4.21) and the other by a Gaussian shape (4.6) with a linear background. Note that 'addition' operator elements (eq. 4.38) have been used where necessary.

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

Figure 4.5: MDL — model definition syntax.

Following the mathematical notation defined at the start of this chapter, the example FFS model defined in Fig. 4.5 is equivalent to:

$$\left[\boxplus\{B_g\{\boxplus\{I_g\{\lambda_{g0},\phi_g,w_g\},I_l\{\lambda_{l0},\phi_l,w_l\}\},w_{B_g}\},y\{m,c\}\}\right](\lambda). \tag{4.54}$$

4.6 Parameter Coupling

One of the issues encountered when performing spectral fitting is 'overfitting', which results in fits that may well provide a set of modelled values that are very close to the experimental data points, i.e. producing a very low residual, but in fact the model is really providing an excellent fit to the statistical 'noise' of this particular data set, rather than the underlying function. This can be the result of having too many free parameters in the model compared with the number of data points (see Section 4.8 for more details). It is also likely that as the number of parameters increase, there will be higher levels of covariance between the parameters. This reduces confidence in accurately

estimating the values of any of these parameters independently. To overcome these problems, it is possible to increase the number of recorded data points, or decrease the number of free parameters. Often, it is difficult, or indeed impossible to control the former. This leaves the possibility of reducing the number of free parameters. One could simply set some of the parameters of the model to be fixed — using data obtained from some other experimental measurement, for example. Alternatively, the fixed parameter can be considered as an assumption of the model used in fitting the data. However, it is also possible to impart some theoretical knowledge onto the numerical model and couple parameters together to help constrain a fit. Effectively, the coupled parameters appear fixed from the point of view of the fitting algorithm (i.e. removed from the set of free parameters), but in terms of model calculation, they are varying as a function of the (still free) parameter, to which they are coupled.

FFS provides a system for handling complex coupling between parameters. Coupling is again specified by the model definition language (see Section 4.5). The format of the coupling expressions is demonstrated by Fig. 4.6, where '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)
```

Figure 4.6: MDL — main coupling syntax.

The coupling expressions, 'cexpression' are prefix statements defining how the parameter is coupled (Fig. 4.7). The operators are arithmetic (+, -, *, /, ^) and the operands are numeric values, model parameters (specified in the same way as the parameter being coupled i.e. elementnamex.parnamex), or indeed further nested expressions — allowing for definition of more complex coupling functions.

```
(operator operands)
```

Figure 4.7: MDL — coupling expression syntax

The coupling of model parameters results in an issue for partial derivative calculation — analytic expressions for partial derivatives of the model may be known, with respect to some or all of the parameters that the free parameter has been coupled to. However, the performance associated with the use of the analytic partial derivatives is lost, unless the coupling expressions themselves are differentiable, giving the inter-parameter dependence derivative i.e. the terms required for applying the chain rule. FFS allows for exactly this — the coupling object can programmatically perform analytic partial

differentiation of the textual coupling expressions of the form shown in Fig. 4.6. It is important to note that calculation of partial derivatives of complex coupling functions, with respect to the coupled parameters is being done analytically using custom, FFS algorithms.

The manner in which this is done requires several steps. Firstly, the parsing object, ffs_parser, takes the MDL coupling statements and pre-parses these into a new form for the coupling object to deal with. This is necessary because coupling objects must be associated at the parameter level and, as such, are 'unaware' of other elements (and associated parameters) in the hierarchy. The model parser does, of course, work at model level and therefore the ffs_parser is capable of translating MDL statements into a simple intermediate syntactical form and supplies this to the coupling object, along with the associated list of operand parameter object references for those included in the coupling expression. This intermediate syntax is (intentionally) not greatly removed from the MDL coupling syntax defined in figures 4.6 and 4.7 — in fact the only change is that the text labels for the parameters are replaced with '$$i$$' where i is an integer ($\{1 \ldots n\}$) used to identify each of the n parameters in the expression.

The coupling object must then analyse the newly formed coupling statement — extracting the various fragments of data: the operator, parameters, constants and nested expressions. This step happens as soon as the coupling expression is set for a given coupling object and the results cached for fast access when the partial derivatives are requested. When the request is made, each of the (now tabulated) nested expression must be taken in turn (most highly nested, outward) and the partial derivative of each operand taken. If the operand is a constant, then the result is immediately known to be zero. However, should an operand be a parameter, a check must be done to establish whether this parameter is also coupled to another parameter (potentially the parameter for which the partial derivative has been requested is somewhere along the coupling chain). In this case this entire procedure must be carried out for the coupled parameter (and so on, through the coupling chain) to obtain the result. If the parameter is not coupled, then there are only two trivial outcomes: the parameter is the dependency being sought — the result is 1, or the parameter is not and the result is 0. If the operand is an expression, then the result has already been pre-calculated and can be stored here. Regardless of operand type, the value of the operand is also cached. This sequence is detailed in figure 4.8.

With the operand values and partial derivatives stored, the calculation has been vastly simplified. For each nested expression the operator is applied to the stored list of operand partial derivatives and the list of stored values used where appropriate

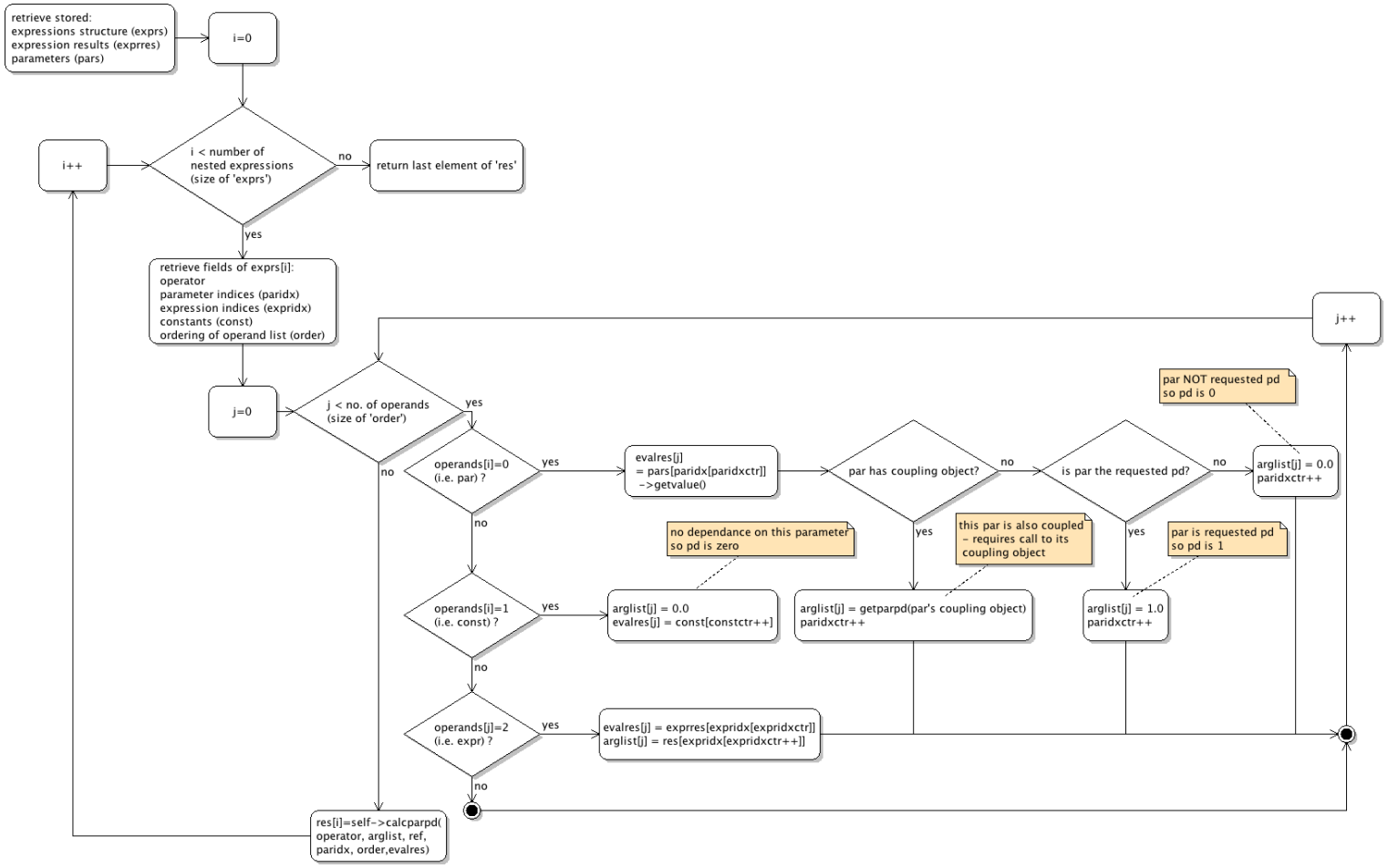


Figure 4.8: Flowchart showing the algorithm for retrieving analytic partial derivatives through coupled parameters.

(application of the product rule for the multiplication operator, for example).

Consider an example of parameter coupling across two elements in a model — elem1 has a parameter (par1) which depends on two parameters of elem2 (par2 and par3). The function defining the dependency is such that par1 is to be coupled to the sum of par3 and three times par2 i.e. in traditional infix notation: elem1.par1 = 3.0*elem2.par2 + elem2.par3. The equivalent MDL describing this is shown in Fig. 4.9.

```
(couple elem1.par1 (+ (* elem2.par2 3.0) elem2.par3))
```

Figure 4.9: Example of an MDL coupling statement. The infix equivalent of this expression is: elem1.par1 = 3.0*elem2.par2 + elem2.par3.

After translation by the model parser, the expression stored by the parameter1 coupling object is as seen in Fig. 4.10. Note that (as stated previously) the coupling object is also supplied with the corresponding parameter object references.

```
(+ (* $1 3.0) $2)
```

Figure 4.10: The coupling statement shown in 4.9 after pre-parse for use by coupling object.

As described above, the coupling object extracts the information from this statement and stores it as a useful structure for inter-parameter partial derivative calculation. The debug mode of the program allows us to follow the program logic output of the code running in this mode. This is shown in figures 4.11 and 4.12, as it parses the inner-nested and then outer expressions (these are 'exprs' in Fig. 4.8), respectively.

```
expression parsed: * $1 3.0
operator: *
pars (index):          0
constants:         3.0000000
other expressions (index):        -1
order of evaluation (0:par, 1:const, 2:expr):          0        1
error status (0:none, 1:operator, 2:par, 3:const,
    4:num_expr):     0
```

Figure 4.11: Debug output from *ffs_couple* as it parses the inner expression of the coupling statement displayed in Fig. 4.10. The coupling object has extracted various tokens from the original expression relevant to value and partial derivative retrieval (see main text for details).

In the order in which they appear in the debug output (Fig. 4.11 and Fig. 4.12), with the labels used in figure 4.8 in brackets, the output displays: the nested expression

itself, operator (operator), pars (paridx), constants (const), other expressions (expridx), order of evaluation (order) and error status. Most of the entries are self-explanatory. However, to clarify a few of the items, 'paridx' is an array of zero-based indices for the parameter list supplied by the main model parser, 'const' is a list of constants used in the expression and 'expridx' is a one-based index referencing any cached nested expressions. In each case, if none of the operands of the current expression are of that particular type then this is indicated by the value -1. The quantity labelled 'order' labels each of the operands for that expression: 0 indicates a parameter, 1 denotes a constant and 2 implies that the operand is an expression. If necessary, the error status helps indicate to the user which part of the the coupling string appears to be causing a parsing error.

```
expression parsed: + (* $1 3.0) $2
operator: +
pars (index):         1
constants:        -1
other expressions (index):              1
order of evaluation (0:par, 1:const, 2:expr):         2         0
error status (0:none, 1:operator, 2:par, 3:const,
    4:num_expr):     0
```

Figure 4.12: Debug output from *ffs_couple* (similar to that shown in Fig. 4.11 — except that this is related to the outer expression of the coupling statement displayed in Fig. 4.10 rather than the inner). A main point to note is that the results of Fig. 4.11 are referenced here. See main text for details.

4.7 Optimisation of the Model

It is possible that some models will possess combinations of elements that can be readily reduced to a more optimum representation i.e. it is possible that 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 utilising analytic expressions for the parameter partial derivatives, or both. Consideration of an example best demonstrates the requirement for a 'simplification' system and the advantages that it brings. Let us return to the example model shown to demonstrate the MDL syntax (Fig. 4.5). This model incorporates a Gaussian broadening function 'brdg' that could be considered to represent the instrumental broadening function for a given spectrometer. The two child elements, the Gaussian 'g' and the Lorentzian 'l' could represent some spectral lines of interest. It makes sense to define the model in this way from an experimental point of view; the instrumental broadening acting on the underlying line shapes. However, this model is not efficient for spectral fitting — the broadening function must perform convolution of a normalised (to preserve area) Gaussian function with that of the operand element (eq. 4.5). In this example, the convolution performed would be the Gaussian kernel with the result of its child of the addition element (which is the sum of a Gaussian and Lorentzian element). Since convolution exhibits a distributive property (eq. A.4) this is equivalent to the sum of the convolutions of the broadening Gaussian with the Gaussian and Lorentzian lines (see eq. 4.25). The results of these two convolutions are known — it has been demonstrated that the convolution of a Gaussian with another results in another related Gaussian (eq. 4.51). Similarly, convolution of a Gaussian function with a Lorentzian function results in the Voigt function (see eq. 4.25) and, further to that, a computationally efficient form of that function is available (eq. 4.29). Fast computation of the partial derivatives are also possible from this definition of the Voigt function. Clearly it is highly desirable to have access to these functional forms. The ffs_simplify class is the component of FFS that allows this type of optimisation to be exploited. It does so by traversing the model element hierarchy (Fig. 4.2) seeking combinations of elements for which an optimisation is known (such as the Gaussian-Lorentzian example given here). This process takes place recursively. Initially the optimal form of the root element of the model tree is requested, but this result is dependant upon the optimal form of its child elements and those on theirs and so on. Once the leaf nodes pairs have been optimised, the new form will 'bubble up' through the tree until the entire tree has been optimised. This means that at the end of the process, FFS will have a second, more efficient model, built from a new set of elements from those originally created by

the user. Since the new model will be constructed using a new set of elements, this has the consequence that the parameter set for this model is now also different. This is not desirable from the user's point of view — they are interested in the original parameter set from the input model. However, via use of the coupling system, defined in Section 4.6, ffs_simplify couples the new parameters to those from the original model. In fact, from the user (and fitting program) perspective, FFS' use of an optimised model is completely opaque.

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

Figure 4.13: The 'simplification' of a user specified model. On the left is the originally defined model, on the right is an equivalent, optimised, version.

In order to perform this task, the ffs_simplify routine uses a reference table of rules defining more efficient representations for a set of operator element-element pairs (the set of rules can be found in Appendix C). This is implemented by storing structures with fields 'parent' and 'child', which denote the type of element that the operator element and operand element are respectively. In addition to this, the structures have 'replacement', which is the type of element that the pairing is to be replaced by and then finally there is the 'coupling' field, which defines how the new element parameters should relate to the original model representation. The 'coupling' entry is an array of string pairs — one for each new parameter; the first element is the name of a new parameter and the second is an FFS coupling expression (see Section 4.6) relating the new parameter to those of the original element pair. The simplification rule list structure for the broaden_gauss – gauss pairing (as in the optimisation example in Fig. 4.13) is shown in Fig. 4.14.

The full procedure for optimisation of the model (including the use of the rule-list for known pairs) is described by Fig. 4.15. Note that there are also some special cases to be accounted for and that these are handled independently from the rule-list approach. Currently, the system deals with two special cases — addition element branches in the model tree and broadening elements, with an addition element child.

```
{parent:'ffs_broaden_gauss', $
child:'ffs_gaussian', $
replacement:'ffs_gaussian', $
coupling:ptr_new([ $
        ['pos', '(child.pos)'], $
        ['fwhm', '(^ (+ (^ parent.fwhm 2) (^ child.fwhm 2)) 0.5)'], $
        ['area', '(child.area)'] $
        ]) $
}
```

Figure 4.14: Rule list structure example.

The ‘expand_add’ method, referred to in the diagram, is a recursive routine that extracts all child elements of all addition elements in that branch and inserts them into the model tree, at the depth level of the top-level addition element. This is shown in Fig. 4.16. The ‘expand_broaden’ method, handles a broadener element operating on an addition child element. In this case, it is necessary to extract the child elements of the addition element and create duplicate broadener elements — one for each of the elements within the addition — such that on the next pass of the optimiser any known pairings between the broadener operator and the grandchild elements become apparent. The algorithm used to handle optimisation of broadener element branches is shown in Fig. 4.17.

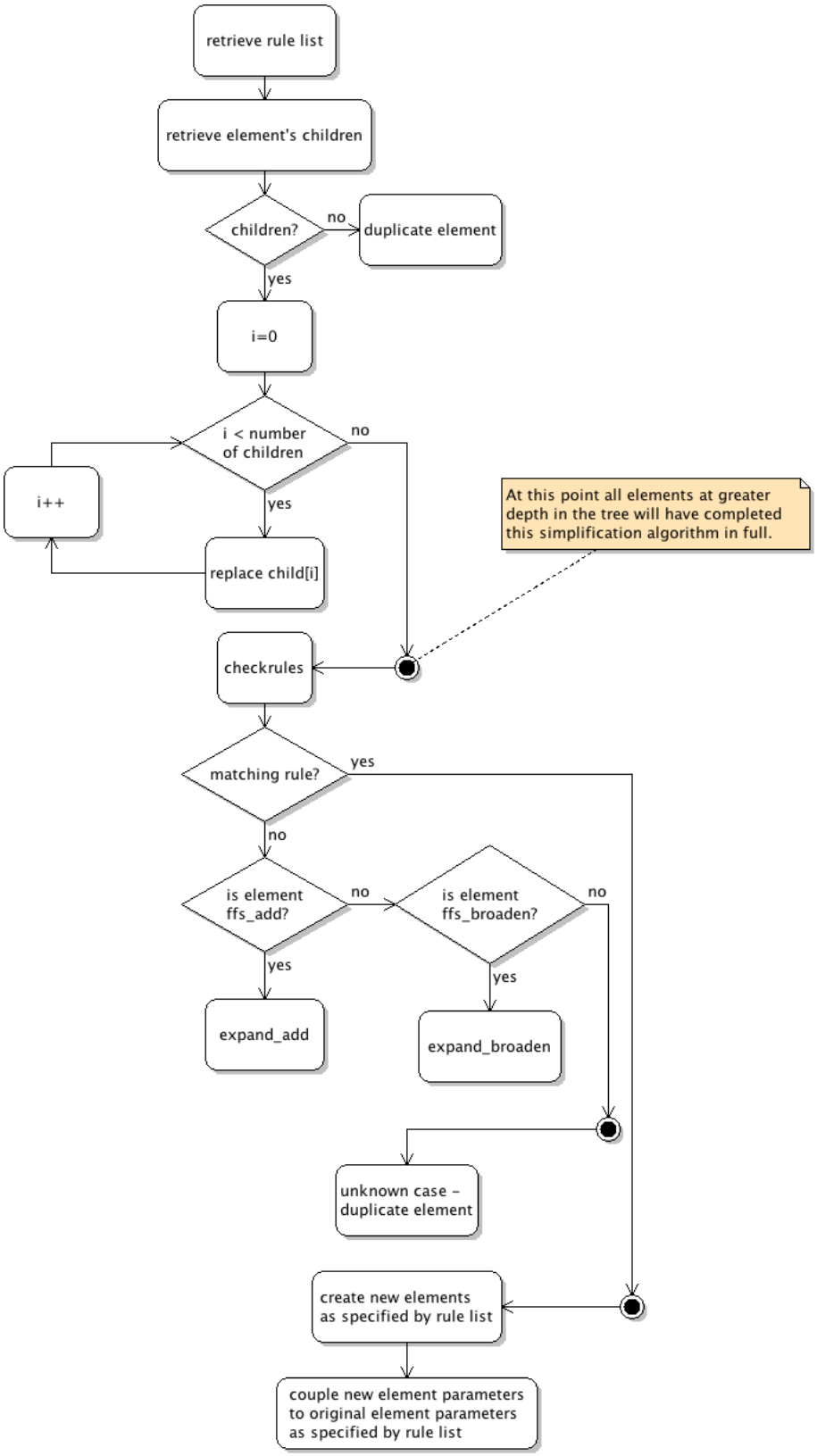


Figure 4.15: Optimisation procedure. Note that the operation *replace child* follows the algorithm defined by this flowchart i.e. this is a recursive method call.

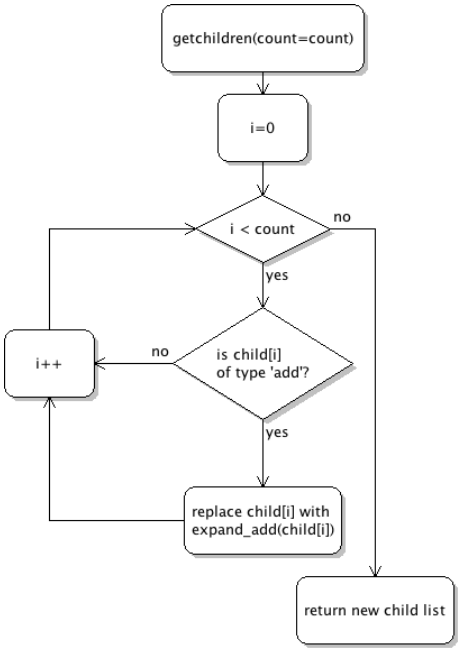


Figure 4.16: Dealing with an addition element branch.

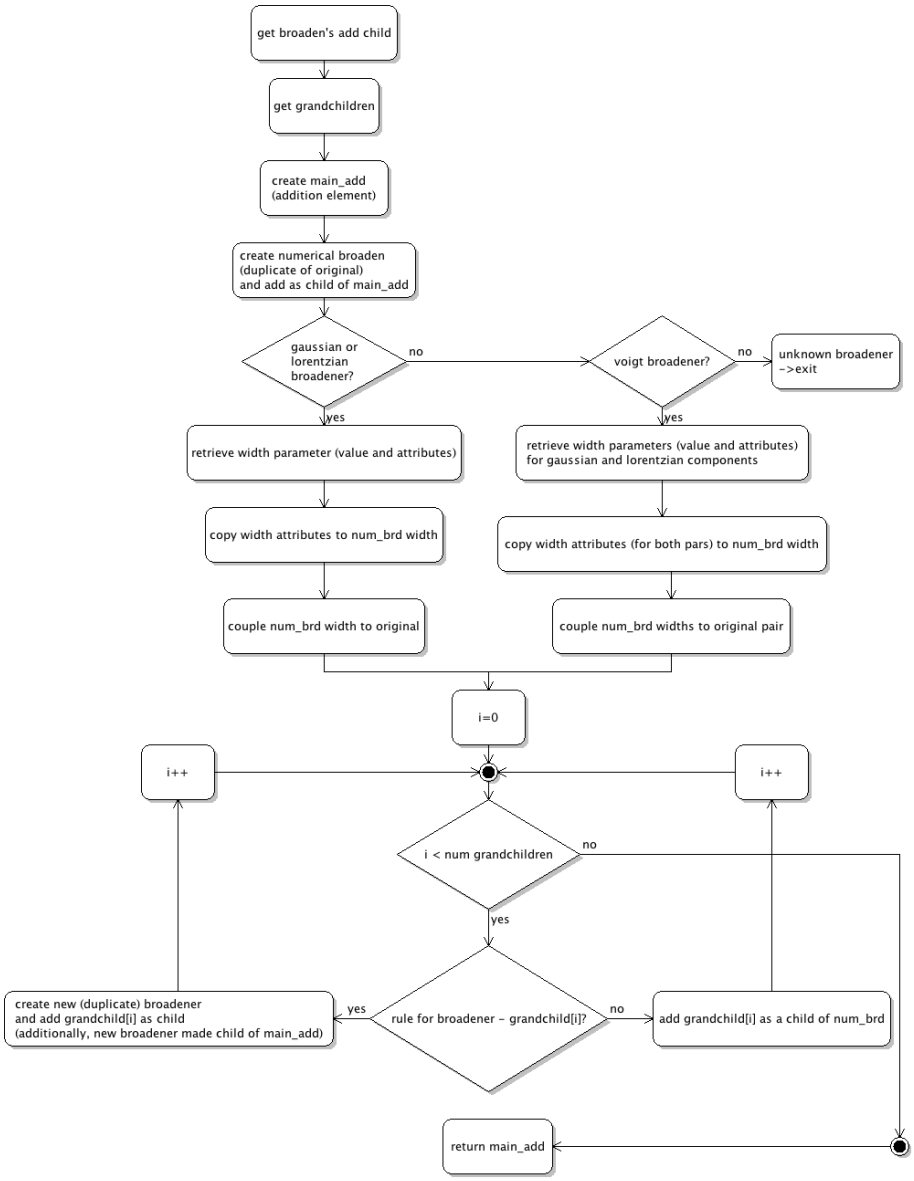


Figure 4.17: Dealing with a broadener element branch.

4.8 Non-linear Least Squares Fitting

The statistical basis of χ^2 fitting is detailed by Bevington [66] and further details, including the computational implementation of various methods used to perform least squares fitting of experimental data are well described and discussed by 'Numerical Recipes: The Art of Scientific Computing' [67]. However, a short summary is provided here for completeness.

To determine physical parameters from a recorded experimental spectrum (wavelength/pixel, x and intensity, y), it is possible to define a parameterised model representation of the data (f) and fit this to the observed spectra via modification of the model parameter set (p) until an optimal solution is obtained. Least-squares fitting is named such as it involves iteratively minimising the merit function, χ^2, which is the sum of the squared residuals between data values (y_i) and modelled values ($f\{p\}(x_i)$), weighted by the square of the inverse of the standard deviation (σ_i) of each data point. So, for N data points:

$$\chi^2\{p\} = \sum_{i=0}^{N-1} \left(\frac{y_i - f\{p\}(x_i)}{\sigma_i} \right)^2 . \tag{4.55}$$

There are a variety of methods to achieve minimisation of such a function. The method of steepest descent is one of the simplest approaches, with the parameter increments simply calculated by taking a step along the gradient (in the negative direction) of the χ^2 surface at the current point:

$$\delta p_i = -h_i \frac{\partial \chi^2}{\partial p_i}, \tag{4.56}$$

where h_i is a constant defining the size of the step taken. This method is useful in that it will work reliably from reasonably far away from the true minimum, i.e., from relatively poor initial estimates. Unfortunately, however, it is not obvious what value should be selected for h_i; too large and the step could over-shoot, but selecting a small step could result in slow convergence. The speed of convergence is also hampered if the hypersurface consists of long, narrow 'valleys' as the algorithm would spend many iterations traversing the valley rather than the more direct route along the floor.

An alternative route to minimisation is prescribed by the 'Gauss Newton' method. Firstly, consider the Taylor expansion of the function that is to be minimised, χ^2, noting that the series is truncated at second order, essentially making the assumption that the χ^2 surface is parabolic near the minimum:

$$\chi^2 \approx \chi^2\{p_{\text{cur}}\} + \sum_{i=0}^{n-1} \frac{\partial \chi^2\{p_{\text{cur}}\}}{\partial p_i} \delta p_i + \frac{1}{2} \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} \frac{\partial^2 \chi^2\{p_{\text{cur}}\}}{\partial p_j \partial p_i} \delta p_i \delta p_j + O\left(\delta p^3\right). \tag{4.57}$$

The gradient of this surface is a vector given by:

$$\nabla\chi^2 = \sum_{i=0}^{n-1} \frac{\partial\chi^2}{\partial p_i}\hat{p}_i, \tag{4.58}$$

the components of which (using the approximation of 4.57) are described by:

$$\left(\nabla\chi^2\right)_i \approx \frac{\partial\chi^2\{p_{\text{cur}}\}}{\partial p_i} + \sum_{j=0}^{n-1} \frac{\partial^2\chi^2\{p_{\text{cur}}\}}{\partial p_j \partial p_i}\delta p_i \tag{4.59}$$

such that:

$$\nabla\chi^2 \approx \nabla\chi^2\{p_{\text{cur}}\} + \nabla^2\chi^2\{p_{\text{cur}}\}\delta p. \tag{4.60}$$

Returning to the expression for χ^2 (eq. 4.55), we can evaluate the first and second derivatives terms in eq. 4.59 (or, equivalently, in eq. 4.60):

$$\left(\nabla\chi^2\right)_i = \frac{\partial\chi^2\{p_{\text{cur}}\}}{\partial p_i} = -2\sum_{i=0}^{N-1}\left(\frac{y_i - f\{p_{\text{cur}}\}(x_i)}{\sigma_i^2}\right)\left(\frac{\partial f\{p_{\text{cur}}\}(x_i)}{\partial p_i}\right) \tag{4.61}$$

and:

$$\left(\nabla^2\chi^2\right)_{ij} = \frac{\partial^2\chi^2\{p_{\text{cur}}\}}{\partial p_j \partial p_i} = 2\sum_{i=0}^{N-1}\frac{1}{\sigma_i^2}\left[\left(\frac{\partial f\{p_{\text{cur}}\}(x_i)}{\partial p_i}\right)\left(\frac{\partial f\{p_{\text{cur}}\}(x_i)}{\partial p_i}\right) - (y_i - f\{p_{\text{cur}}\}(x_i))\left(\frac{\partial^2 f\{p_{\text{cur}}\}}{\partial p_j \partial p_i}\right)\right]. \tag{4.62}$$

$\nabla^2\chi^2$ is known as the Hessian matrix and, unfortunately, is computationally expensive to calculate — it requires calculation of both first and second derivatives. In practice, an approximation to $\nabla^2\chi^2$,

$$\alpha = \frac{1}{2}\nabla^T\nabla\chi^2, \tag{4.63}$$

is used (i.e. dropping the second order terms in expression 4.62, also note that the factor $\frac{1}{2}$ is conventionally added to eliminate the factor 2). This matrix is often referred to as the *curvature matrix* as it gives a measure of the curvature of the χ^2 surface. The approximation is in keeping with the one implicitly made when truncating the Taylor series; that the function is being assumed to be almost linear at very small deviation from the solution. It should be noted that the term dropped also contains the residual; the method is sensitive to the the initial conditions of the fit, i.e. , the residuals must be relatively small. The main advantage of the method, however, is that it provides rapid convergence to the solution.

Coupled with our definition of the curvature matrix α, is the definition of the vector:

$$\beta = -\frac{1}{2}\nabla\chi^2\{p_{\text{cur}}\} \tag{4.64}$$

By substitution of these quantities and setting $\nabla\chi^2 = 0$ to find the minimum and solving eq. 4.60 for the parameter adjustment vector, δp:

$$\delta p = \alpha^{-1}\beta. \tag{4.65}$$

The fitting algorithm implemented for use in this work, is a version of that developed by Marquardt; the so called 'Levenberg-Marquardt algorithm' [18]. 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. There are two main features of the algorithm that achieve this. The first is that consideration of the dimensionality of eq. 4.56 shows that h_i has units of p_i^2. The reciprocal of the diagonal elements of the curvature matrix, i.e. $\frac{1}{\alpha_{ii}}$ share these units, so this at least gives some information of as to the size of the constant involved. The approximate nature of this is compensated for by a numerical fudge factor λ, such that eq. 4.56 becomes:

$$\delta p_i = (\lambda\alpha_{ii})^{-1}\beta_i. \tag{4.66}$$

The Marquardt algorithm then blends the two methods by re-defining the curvature matrix α, by multiplying the diagonal elements by the factor $1 + \lambda$:

$$\alpha'_{ij} = \begin{cases} \alpha_{ij}(1+\lambda) & i = j \\ \alpha_{ij} & i \neq j \end{cases}, \tag{4.67}$$

such that the parameter increments are now given by:

$$\delta p = \alpha'^{-1}\beta. \tag{4.68}$$

This means that the damping factor λ can be adjusted to be large, making the matrix α' diagonally dominant such that it moves towards eq. 4.56, i.e. the method of steepest descent. However, λ can also be adjusted to a small value, moving the solution to that of the Gauss-Newton method (eq. 4.65).

The inverse of the curvature matrix, is known as the covariance matrix:

$$C = \alpha^{-1}. \tag{4.69}$$

The matrix is so called because the elements $C_{ij}(i \neq j)$ give the covariance between the two parameters p_i and p_j. The diagonal elements, C_{ii} are the variances of the parameters p_i, so the standard deviation of the parameters is given by:

$$\sigma_i = \sqrt{C_{ii}}. \tag{4.70}$$

Rather than the covariances, it is often more useful to consider the derived quantity, correlation[1], when considering interdependence of the parameters. This has elements defined by:

$$C^N_{ij} = \frac{C_{ij}}{\sqrt{C_{ii}C_{jj}}}. \tag{4.71}$$

Finally, note that when evaluating the quality of the fit, it is more useful to consider a χ^2 statistic that is normalised to the number of degrees of freedom, that is:

$$\chi^2_N = \frac{\chi^2}{N - n_p} \tag{4.72}$$

where N is the number of data points and n_p is the number of free parameters in the fit. $\chi^2_N = 1$ indicates a high quality fit. However, it should be noted that $\chi^2_N < 1$ suggests an *unexpectedly good fit*. This can occur due to over-estimation of the errors on the experimental data values (i.e. that the values of σ_i are too large). Alternatively, this can be an indication of *over-fitting* — that is to say that the model contains too many free parameters compared with the number of data points.

4.9 A Custom Fitting Code

FFS is a computational framework for provision of complex spectral model specification. The spectral fitting process itself, however, is (by design) 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 [68]. This code is an IDL port of a well known Fortran fitting routine, MINPACK-1 [69], part of the the Netlib library of routines. This package has, at its core, the Levenberg-Marquardt algorithm detailed above.

[1] Also referred to as *normalised covariance*.

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. Similarly, coupling is handled internally (to arbitrary complexity and so, surpasses the capability of the MPFIT specification). Finally, upon confrontation with experimental data, the core algorithm did not appear to be entirely robust. Under many circumstances, fitting would fail, often reporting that the curvature matrix was singular. Investigation as to the cause of the apparent frailty was undertaken and the cause found to be numerical instability in the calculations for the suggested parameter corrections (eq. 4.68). The instances where this occurred involved parameter sets where individual parameter values (and partial derivatives) were highly disparate in absolute terms. This means that in performing the Householder operations for the QR-decomposition of the matrix, numerical inaccuracies can quickly occur. Technically, it can be said that the curvature matrix, in these cases, is *ill-conditioned* to numerical operations. The *condition number*, κ, of a matrix A is calculated via $\kappa(A) = ||A||||A^{-1}||$; a condition number close to 1 indicates a *well-conditioned* matrix. The condition numbers in the failing scenarios were of the order $\sim 10^5$ or, in some case, even $\sim 10^6$. This behaviour is well known and, often, the approach taken to tackle this issue, is to normalise the experimental data (in intensity and/or wavelength dimensions). The reason for this, is that typically mathematical line shapes such as Gaussians are being used to fit the data and the 'problem' parameter is that which controls the intensity (area) of the line shape which may be of the order $\sim 10^{11}$; the other parameters, wavelength and line-width could be ~ 100 and ~ 1 respectively. However, this approach may be simple to implement, but for a complex fitting system such as FFS, where model could exhibit a great range of physical parameters (of disparate absolute values) this is not appropriate. Instead, it is necessary to instead scale (normalize) the parameters themselves. Unfortunately, the parameter scaling must occur within the fitting routine; it is necessary to scale the parameters (as well as the corresponding parameter limits and model partial derivatives) when assembling the curvature matrix to find the (scaled) parameter improvements, but these values must be 'un-scaled', at each iteration, to enable model re-evaluation for the 'goodness-of-fit' (χ^2) test. This requirement, combined with the fact that FFS already handled the additional features (such as parameter limits and coupling) provided by MPFIT, led to the creation of a custom fitting code *ffs_fit* (also based on the Levenberg-Marquardt algorithm) which

handles the parameter scaling. This may initially sound trivial, but there are some important details to consider here. Ideally, one would make it such that all parameters are scaled such that they are altered over the range [0,1] (the scaled limits), with the initial scaled value taken as the fraction of the initial true value over the true range. However, in practice, the model parameters may not have an upper bound, lower bound, or indeed no specified limits at all. Further to this, computationally it is undesirable to perform all of this scaling/un-scaling at each fitting iteration, so care must be taken to do so as efficiently as possible. The FFS code therefore performs an initial scale setup procedure which caches the various data required to allow rapid scaling/unscaling of parameter values, limits and partial derivatives. The data cached includes the initial (true) parameter value (p_i), the scaled limits, the scaling factor (f) which is the effective range and the scaling constant (c) which is the effective lower limit. The initial scaled value is set to: $p_s = \frac{p_i - c}{f}$. As stated above, setup is straight forward if the parameter has specified lower and upper limits (bounds); the scaled limits are taken as [0,1], $f = l - u$ and $c = l$ such that the scaled value is the fractional proportion of the range that the initial true value is. In all other cases, the initial scaled value will be 0.5. If the parameter only has an upper limit set, the scaled limits are set to $(-\infty, 1]$, $f = 2(u - p_i)$ and $c = -\frac{1}{2}f + p_i = 2p_i - u$. In the case of a single lower limit, the scaled limits become $[0, +\infty)$, $f = 2(p_i - l)$ and $c = -\frac{1}{2}f + p_i = l$. If the parameter has no limits, then the scaled limits remain $(-\infty, +\infty)$, $f = 2p_i$ and $c = 0$. The various model partial derivatives can now be quickly scaled, without traversing the decision tree present in setup, via: $\frac{\partial M}{\partial p_s} = f\frac{\partial M}{\partial p}$. Similarly the parameter values and parameter errors are readily un-scaled: $p = fp_s + c$ and $\sigma_p = f\sigma_{p_s}$, respectively.

4.10 Batch Fitting

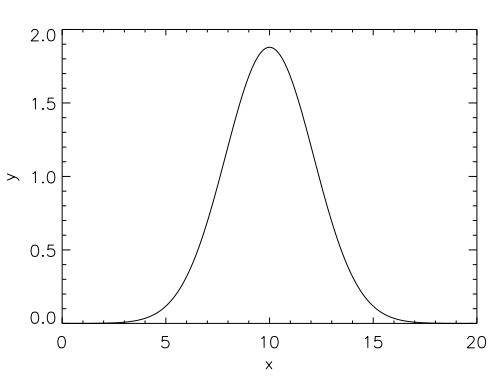
It is often the case that an experiment will record multiple spectra — usually a series of spectra in time, space or both. It is desirable to fit all of the spectra in a systematic, automated scheme to identify trends in the derived parameters across the series. To this end, scripts exist as part of FFS which cycle through the frames of spectra performing fits using a single model definition. Batch fitting in this way therefore calls for compromise — it is possible that as conditions change across the series, the model used may no longer be suitable and so, the resultant fits are poor i.e. the value of the χ^2 merit function are unexpectedly high (i.e. greater than a set tolerance). Improvement may be achieved by relaxation of parameter limits in the model to allow more flexibility, but this should not be done indiscriminately. The poor fits, however, may be the result of

another problem — the fitting routine may have discovered a local minimum in the χ^2 surface and become 'stuck'. This is an artifact of the Levenberg-Marquardt algorithm which, while not overly sensitive to initial conditions, is not a global fitting algorithm. There is no simple solution to this problem and so, typically one selects an alternate set of initial parameter values in an effort to find the true minimum. To do so in an automatic fashion, the batch routine fits all of the spectral frames for a given track and then, for any poor fits, attempts to use the final, determined parameter values for the closest frames, for which the fits were deemed acceptable, as the initial value set to re-fit the failed frames. The rationale behind this selection is that the neighbouring frames are the most likely to have similar spectral signatures to those that require re-fits and so this parameter set is more likely to be in the vicinity of that which will truly minimise the χ^2 merit function. In any case, regardless of the validity of the rationale, the new initial parameter values provide an alternate starting point in the function parameter space to try and avoid the local minimum and find a path to the true solution. The number of re-fit attempts is arbitrary and is decided upon the basis of trade-off against computation time.

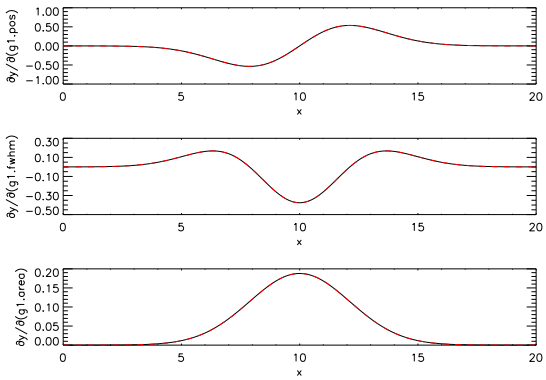
4.11 Validation of Results

In order to have confidence in complex systems such as FFS, it is necessary to validate the results of the code. For instance, the partial derivatives calculated via the analytic expressions, outlined in Section 4.2, must produce similar results to those obtained via numerically derived derivatives (within a tolerance accounting for the inherent inaccuracy of numerical methods). Here, the results for the most commonly used lineshapes (Gaussian, Lorentzian and Voigt) are considered (Fig. 4.18). In all cases it can be seen that the code returns equivalent results, when the mode of operation is changed from numeric, to analytic, as required.

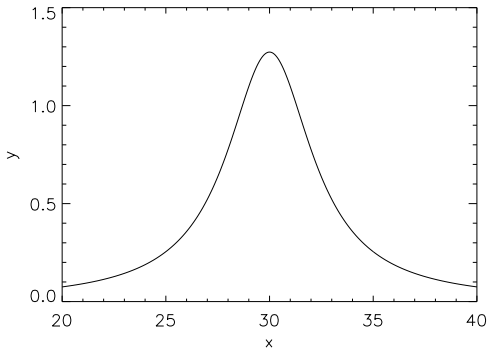
Another area requiring validation is the simplification system — it is important that the new model representation produces a modelled spectrum that is, within reason, the same as that resulting from the original definition. Again, only a selection of example comparisons can be shown here; specifically the broadening elements, when applied to a selection of operands. The operands are varied in that some are wavelength resolved (*gridded* in FFS terminology), whilst others are like point impulse functions specified at a single wavelength (i.e. un-broadened lines), further to this the number of data points for the operand are altered. Each of theses cases must be examined, as the broadening elements use different numerical methods for each (for efficiency



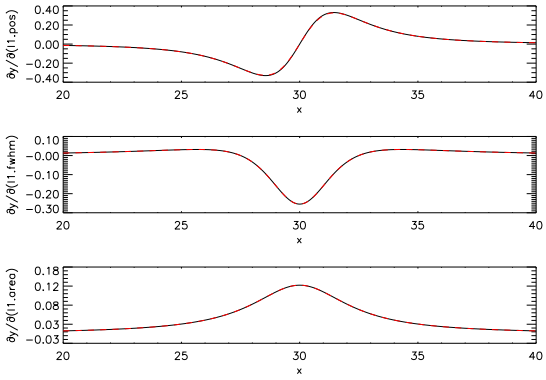
(a) Gaussian function.



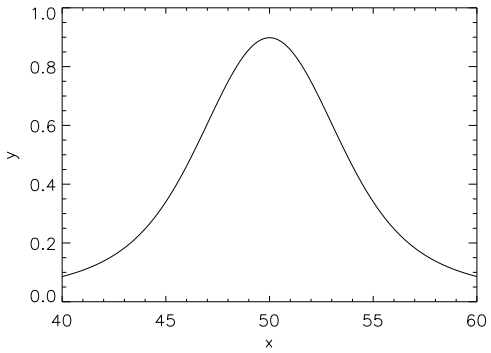
(b) Gaussian partial derivatives.



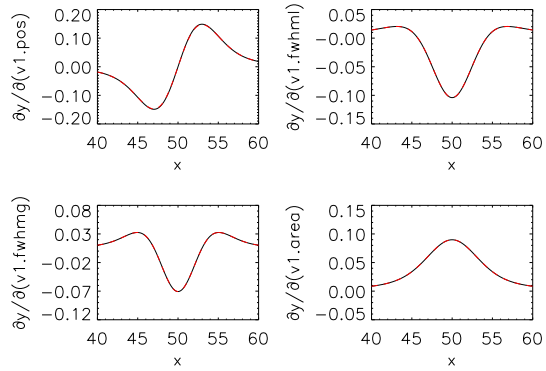
(c) Lorentzian function.



(d) Lorentzian partial derivatives.



(e) Voigt function.



(f) Voigt partial derivatives.

Figure 4.18: The left-hand column (a, c & e) show plots of example functions (Gaussian, Lorentzian and Voigt). The right-hand column (b, d & f) displays plots of the partial derivatives of these functions — when calculated numerically (method of finite difference), shown by the red, dashed line and the result when using the analytic solution (the solid black line). Note that the element names are *g1, l1* and *v1* respectively, with the plots labelled accordingly.

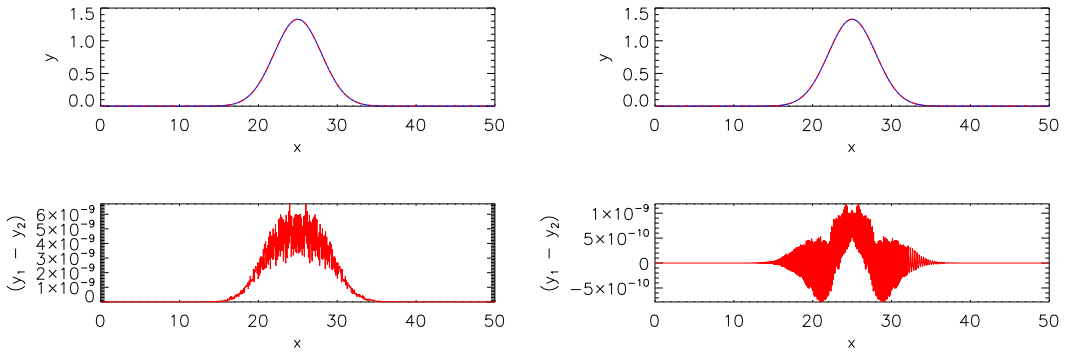
of calculation). Firstly, we take the simplest example of the broadener applied to a single un-broadened line — these are the convolutions as shown by eq. 4.6, 4.21 and 4.29, which re-distribute the line intensity over a finite wavelength range — resulting in a Gaussian, Lorentzian, or Voigt, respectively. The numerical broadeners discretise the convolution integral into a finite sum of the product of the (normalised) broadening kernel with the function to be broadened. However, in this situation (where the operand function is a point impulse), discretised sum or otherwise, the impulse function sifts out a single non-zero product term i.e. the numerical and analytical results are identical. To verify this, plots were made of the results when using the original model definitions and compared with those resulting from the model specified by the FFS simplifier (see Table 4.1). There were no differences for these examples, as expected.

	Original	Simplified
(a)	(broaden_gauss (line l1) bg)	(gaussian new_gaussian)
(b)	(broaden_lorentz (line l1) bl)	(lorentzian new_lorentzian)
(c)	(broaden_voigt (line l1) bv)	(voigtian new_voigt)

Table 4.1: The 'simplification' of the various broadener elements acting upon an un-broadened line element. On the left is the originally defined model, on the right is an equivalent, optimised, version. Note that the internal coupling expressions, connecting the parameters of the original and simplified models have been omitted here.

As discussed above, the convolution integral is approximated by a discretised sum. Therefore, unlike the *un-gridded* cases, one should not expect exact correspondence between the numerically broadened model and FFS optimised model output. In order to avoid re-calculation of the operand of the broadening element, or interpolation of calculated values, the approximation made is a simple one — the composite midpoint rule. For a set of wavelength values $\{x_0, x_1, \ldots, x_n\}$, the sub-interval width (step) for a point, x_i, ($x_1 \leq x_i \leq x_{n-1}$ is taken as $\frac{1}{2}(x_{n+1} - x_{n-1})$. At the end points, x_0 and x_n, the intervals are $x_1 - x_0$ and $x_n - x_{n-1}$ respectively. At this point it is important to note, that as well as providing more efficient model calculation (and therefore, rapid fitting), that the use of the FFS simplifier bypasses the need for the approximations made here. In fact, to describe this trial as a 'test of the FFS simplification system', is a misnomer — the FFS simplified model should provide a more accurate result than the original and provide a benchmark for the quality of the numerical broadening. In any case, the results should be fundamentally similar and this is supported by the output of the code for another trial shown in Fig. 4.19. In this illustration, the model (evaluated with, and without, the simplification enabled) is a Gaussian broadener applied to a another Gaussian function (MDL shown in Fig. 4.20). As expected, there is a marginal

difference in the two outputs (for the reasons discussed above) but, importantly, the absolute value of the difference between y_1 and y_2 is very small compared with the function values. Further to this, when evaluated for a larger number of data points (see Fig. 4.19) the difference is reduced — this supports the reasoning that numerical accuracy is the source of the discrepancy as the step size has been effectively reduced i.e. moving towards a continuous representation of the integral.



(a) 1000 data point calculation.

(b) 5000 data point calculation.

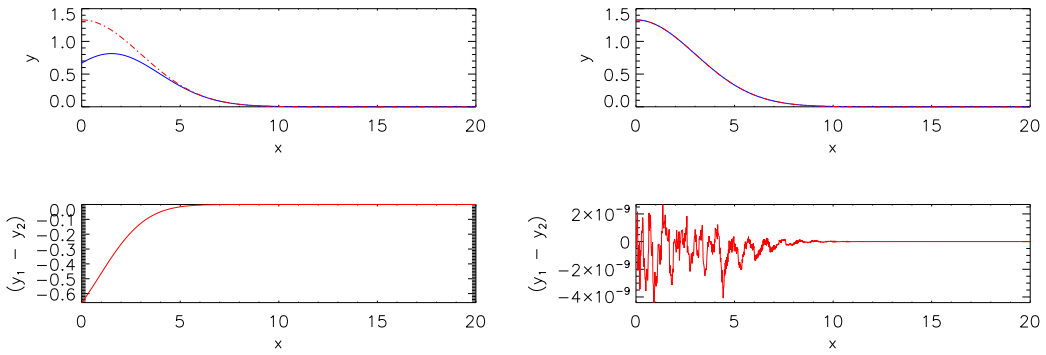
Figure 4.19: The upper pair of graphs show overlaid plots of the function as calculated via original model representation, i.e. a Gaussian broadener acting upon Gaussian, (the blue line) and via optimisation, i.e. a Gaussian, (the red, dashed line). The lower plots show the differences of the two pairs of functions — there is a small discrepancy due to the limitation of the numerical approximation, which improves with the number of data points. (a) and (b) are calculated for 1000 and 5000 abscissae respectively.

```
(broaden_gauss          |   (gaussian new_gauss)
    (gaussian g)        |   (couple new_gauss.pos = g.pos)
 bg)                    |   (couple new_gauss.fwhm (^ (+ (^ bg.fwhm 2)
                        |       (^ g.fwhm 2)) 0.5))
                        |   (couple new_gauss.area = g.area)
```

Figure 4.20: The ‘simplification’ of the Gaussian broadener acting on a Gaussian. On the left is the originally defined MDL. On the right is an equivalent, optimised representation, complete with MDL coupling statements.

There is another, more significant, error that can occur when using the numerical broadener that must be dealt with — edge truncation. When using the FFS simplifier, the model is reformed via analytical convolution of the functions and the resultant function is then evaluated over the desired wavelength range. However, when the numerical broadening elements are used, FFS must approximate the resultant function using the discrete values of the component functions which have been evaluated over the specified wavelength grid. This is an important distinction because it means that

the broadening kernel can only be applied to the data points supplied by the child element branch. This presents an issue if the child function is truncated at the edge of the wavelength range — contributions to the integral outside the bounds will be omitted, with the consequence that the numerical solution will be diminished at the edges of the grid. To overcome this, the broadener must expand the wavelength region for the child branch and evaluate the model in these regions to provide the required *top-up* at the edge. The broadener can then truncate the new result, back to the original region of interest. The extent to which the wavelength bounds should be extended is not well defined; compromise must be made between addition of a larger number of data points and the increase in accuracy and a lower number of data points, with fewer calculations and therefore faster computation. The minimum expansion region is, however, related to the width of the broadener and so, provides some indication of scale — FFS defaults to twice this width. Returning to the example model defined in Fig. 4.20, with the centre of the Gaussian line, g, moved to the lower wavelength bound, such that a large proportion of the non-zero values of the function exist outside the region evaluated. The model was calculated with and without the expansion of the operand wavelength grid (in this case, the operand is just the Gaussian g). Figure 4.21 demonstrates the marked difference that is observed when this correction is included.



(a) (b)

Figure 4.21: (a) and (b) show, in the upper graph, overlaid plots of the function as calculated via original model representation (the blue line) and via optimisation (the red, dashed line). The lower plots shows the difference of the two functions. Sub-figure (a) was calculated without extending the wavelength region that the child elements were calculated over, whilst (b) does exactly this — to reduce the discrepancy compared with the analytical result.

Similarly, it is important to again verify analytic versus numerical partial derivative calculation in this case. This also, simultaneously, verifies that the coupling system is working as expected; the parameters of the simplified model are connected via

coupling to the originally specified model definition.

4.12 Analytic / Numerical fitting Speed Comparison

One of the main advantages of using analytic formulae for partial derivatives is performance; use of finite difference methods requires multiple function evaluations which can potentially be computationally expensive. Similarly, the use of the simplification feature is designed to use the most optimum functional form and retain analytical forms for partial derivatives, wherever possible, on the same performance-related grounds.

This is an important consideration when attempting to perform inter-shot analysis of spectra i.e. parameter estimates from spectral fits from one pulse influencing the decisions made on control parameters for the next pulse. This time window can be approximately 10 minutes and so, speed of computation can be important if there is a large amount of data. The number of spectra to be analysed varies depending on the confinement time and the temporal and spatial resolution of the instrument involved, but typically one could easily expect to be fitting approximately 200 frames of data, for 10 spatial positions, i.e. thousands of spectra. Each of the individual spectra must be fitted with a theoretical representation and in doing so the χ^2 based algorithms evaluate the model, at least once, every iteration to check for improvement in fit. The algorithm must also calculate partial derivatives with respect to all model parameters. If analytical solutions are not known, then finite differences are used and depending on whether one-sided, or two-sided differences are used, then this adds another one or two function evaluations to each iteration.

Table 4.2 shows time taken when requesting calculation of the model partial derivatives, one hundred times, for a selection of models. The first column describes the model in use, but the usual MDL has been reduced for brevity — names for each element have been dropped and the element types have been abbreviated. For the purposes of this table: bg — broaden_gauss, bl — broaden_lorentz, bv — broaden_voigt, gs — Gaussian, lz — lorentzian, vt — voigt and l — un-broadened line. The columns that follow reflect the various modes that FFS can have active — from left to right they are the times taken when using: numerical derivatives, analytical derivatives, a simplified model with numerical derivatives and finally simplified model with analytical derivatives.

The first thing to note from the results is that, in most cases, there is a dramatic decrease in calculation time when the model simplifier is used e.g. from the first row, numerical

Model	Num.	Ana.	Num. (sim.)	Ana. (sim.)
(bg (bg (+ gs lz vt)))	849.67	850.41	6.93	1.04
(bg (bl (+ gs lz vt)))	533.61	532.91	7.53	0.85
(bg (bv (+ gs lz vt)))	1702.32	1697.88	8.45	1.21
(bg (+ gs lz vt))	290.18	289.79	4.98	0.69
(bg l)	0.37	0.33	0.48	0.14
(bl (bg (+ gs lz vt)))	744.29	743.54	7.54	0.85
(bl (bl (+ gs lz vt)))	425.80	425.94	6.35	0.79
(bl (bv (+ gs lz vt)))	1591.30	1591.58	8.19	1.02
(bl (+ gs lz vt))	189.37	188.82	5.80	0.62
(bv (bg (+ gs lz vt)))	1917.16	1915.20	8.40	1.21
(bv (bl (+ gs lz vt)))	1573.48	1572.71	8.22	1.01
(bv (bv (+ gs lz vt)))	2923.82	2920.48	9.15	1.36
(bv (+ gs lz vt))	1228.51	1224.72	7.51	0.84
(bg (bg (+ gs vt))) *	371.25	369.85	2.43	0.74
(bg (+ gs vt)) *	194.23	193.91	2.22	0.48
gs	0.34	0.06	0.41	0.11
(+ gs vt) *	1.57	0.21	1.79	0.30
lz	0.24	0.07	0.31	0.11
vt	0.64	0.09	0.76	0.16

Table 4.2: Partial derivative calculation performance test. Entries are time taken (in seconds) to evaluate all partial derivatives 100 times (for each respective model). With respect to the column titles — 'Num.' is FFS running in numeric calculation mode (finite differences), while 'Ana.' signifies 'analytic' mode was active. The 'sim.' in the third and fourth columns signifies that the simplifier was in use. Entries marked with a "*" have coupling between the positions of the lines.

calculation of the partial derivatives using the simplifier takes just 0.82% of the time taken without its use. Secondly, the benefits of known analytic solutions for partial derivatives is clearly hidden from us, unless the simplifier is used; again, looking at the the first entry — there is very little difference in the calculation time with FFS in numerical mode versus the analytic option (in fact, we see a slight increase in calculation time), but with once the simplifier is in use — we can see that the simplified model with analytics is taking just 15.03% of the time taken to calculate via finite differences. Comparing the two extrema, use of analytics combined with the optimisation system, partial derivative calculation time has been reduced by 99.88%.

Similar performance gains are seen for all models which involve broadening elements acting upon known child elements (e.g. Gaussian). This is not, of course going to be seen for all element combinations (those that the simplifier is unaware of) and in these circumstances there may even be an increase in calculation time due to the overhead of the simplifier machinery (i.e. creation of a secondary model and parameter value linkage via coupling). The table contains entries for models containing single elements which, by definition, cannot be reduced to a simpler form — which demonstrate FFS taking longer to calculate the partial derivatives when the optimiser is active. In the case of the model comprised of a single Gaussian element, when using numerically evaluated partial derivatives, the optimiser overheads can be seen to be adding 0.07s onto the time taken for one hundred model partial derivative requests. This is a 21.63% increase in calculation time which, in isolation, seems somewhat poor. However, consideration of the overall picture must come into play here. Firstly, FFS is more likely to be in use for more complicated models than a simple Gaussian fit. For models of higher complexity, it is likely that performance enhancement for some parts of the model will outweigh detrimental overheads on parts where no improvement has been made. Secondly, even in the case of the single Gaussian model, improvement in partial derivative calculation from use of analytic solution far exceeds the deficit from simplifier overheads. By default, use of analytics and simplifier are both enabled and calculation takes just 32.35% of the time taken without either. Finally, there is the simple fact that use of the optimiser is entirely optional and the user can disable its use if preferred.

Chapter 5

Experimental Analysis

In this chapter, the objective is to report on the practical implementation and testing of the theoretical feature representation discussed in the previous chapter. Use will be made of spectral data coming from both astrophysical and magnetically confined fusion sources. The chapter will be progressive in that it will begin with the simplest of individual line spectra and move to more advanced cases where there is a relationship between the governing parameters by connection through coupling, or indeed using a full special feature. In this sense, the development will follow ADAS itself. The first code in the ADAS package, called ADAS602, was a principal fitting code used in astrophysics for many years. It was based on a maximum likelihood method and has been used extensively for the examination of spectra from the Coronal Diagnostic Spectrometer (CDS) on the SOHO spacecraft. These fittings with ADAS602 were, in fact, used to produce the quiet sun spectral atlas for CDS [70]. Subsequent ADAS development in spectral fitting led to the use of the code XPASCHEN. This code, which became ADAS603, was developed at Forschungszentrum Jülich [10, 11] for fitting Zeeman split features (due to the magnetic field — see Section 2.2). It will be important for these code capabilities to be reproduced by the present development. The full power of AFG and FFS is, of course, realised by much more complex and inter-related special features (the FFS models used in the examples that follow are documented in Appendix B). For testing these capabilities, this thesis is dependent upon data coming from the magnetically confined fusion domain, particularly from the JET tokamak. JET has a number of distinct plasma regions which result in markedly different spectral features. It is an objective to explore in this chapter a few of these, including emission created at the extreme periphery of the plasma, emission from the hot core of the plasma, emission from the beam penetrated plasma and emission from heavy species near the last closed flux surface. This set of special features spans a

great range of wavelengths and spectrometers. So, the data will come from visible spectrographs, from UV and EUV survey spectrometers and soft X-ray instruments. Our starting point, however, is a test, noisy synthetic spectrum.

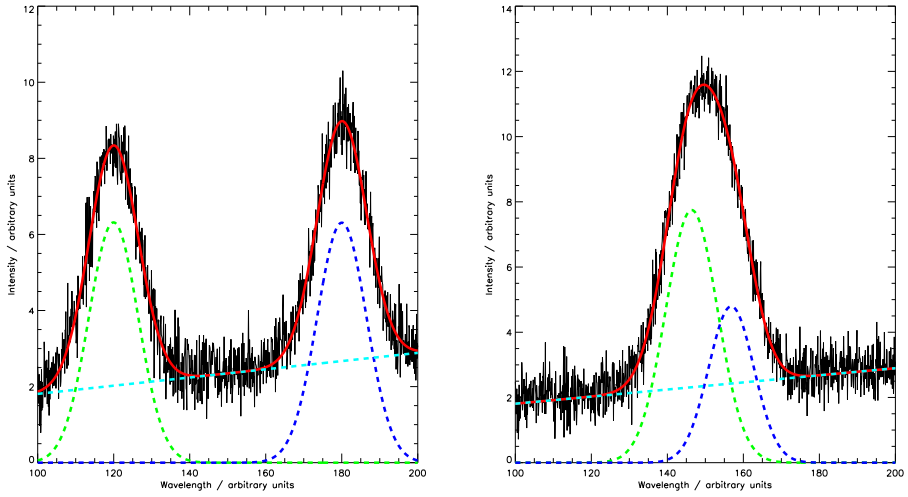
5.1 Initial Validation

Consider an example of two, nearby, spectral lines sitting on a sloping background and subject to substantial noise. The lines are brought together, with a fixed noise level. In the first trial, all FFS model parameters are free. In subsequent trials, constraints are placed on the parameters, via coupling. The correlation matrices from the results of the fits are examined and compared. Figure 5.1 shows the data and fitted models for two cases. Tables 5.1 and 5.2 show the correlation matrices from the resultant fits. In Table 1, note the diagonal entries correspond to full, self-correlation of unity. With the first placing of the lines (trial 1) there is low cross-correlation between the two lines, ~ 0.1, on the other hand there is a more substantial correlation of magnitude ~ 0.7 between the linear background y-intercept, gradient, left-most line area and full-width at half maximum parameters, as expected.

With the placing in trial 2, where the lines are nearly overlapping, note the strong cross-correlation both within and between the parameters of the two lines. Physically, the fit is suggesting that the two line model is becoming less favourable compared to a single line model. In this trial, with the only constraint that the line positions remain within the data wavelength interval, it is possible that, depending on initial conditions, the algorithm can find local minima in which one of the lines is effectively 'removed' from the fit as its position is moved to the edge of the wavelength grid and reduced to zero area.

5.2 An Illustration from SOHO-CDS

Moving on from synthetic data, this next study is of spectral data from the Normal Incidence Spectrometer (NIS) of the Coronal Diagnostic Spectrometer (CDS) on the Solar and Heliospheric Observatory (SOHO) spacecraft [71]. NIS has two spectrometers, NIS-1 and NIS-2, which span spectral ranges 308 – 381 Å and 513 – 633 Å. In the preparation of the quiet sun spectral atlas [70] a wavelength calibration was established for each of the detectors based on a reference set of strong, unblended lines with low variance in the fitting parameters (a star rating system was



(a) trial 1 (b) trial 2

Figure 5.1: The two plots here show the data to be fitted along with an overlay of the FFS model for two cases. Both have two Gaussian lines sitting on a linear background. The data is represented by a solid black line and the fitted model result is in red. The FFS model elements are indicated by various dashed, coloured lines.

	g1.pos	g1.fwhm	g1.area	g2.pos	g2.fwhm	g2.area	bg.c	bg.m
g1.pos	1.00	-0.13	-0.15	0.08	0.15	0.18	0.26	-0.29
g1.fwhm	-0.13	1.00	0.66	-0.13	-0.09	-0.11	-0.64	0.50
g1.area	-0.15	0.66	1.00	-0.15	-0.10	-0.13	-0.74	0.58
g2.pos	0.08	-0.13	-0.15	1.00	0.14	0.17	0.24	-0.26
g2.fwhm	0.15	-0.09	-0.10	0.14	1.00	0.67	0.29	-0.51
g2.area	0.18	-0.11	-0.13	0.17	0.67	1.00	0.35	-0.61
bg.c	0.26	-0.64	-0.74	0.24	0.29	0.35	1.00	-0.90
bg.m	-0.29	0.50	0.58	-0.26	-0.51	-0.61	-0.90	1.00

Table 5.1: Correlation matrix resulting from the fit of trial 1. The labels follow the prescription of Section 4.5. ‘g1’ and ‘g2’ refer to the two Gaussian lines. ‘bg’ refers to the linear background where parameter ‘c’ is the y-intercept at the lower wavelength bound and ‘m’ is the gradient.

	g1.pos	g1.fwhm	g1.area	g2.pos	g2.fwhm	g2.area	bg.c	bg.m
g1.pos	1.00	0.97	1.00	0.98	-0.91	-1.00	-0.17	0.18
g1.fwhm	0.97	1.00	0.97	0.93	-0.83	-0.96	-0.22	0.18
g1.area	1.00	0.97	1.00	0.99	-0.93	-1.00	-0.19	0.18
g2.pos	0.98	0.93	0.99	1.00	-0.95	-0.99	-0.16	0.17
g2.fwhm	-0.91	-0.83	-0.93	-0.95	1.00	0.93	0.12	-0.21
g2.area	-1.00	-0.96	-1.00	-0.99	0.93	1.00	0.17	-0.19
bg.c	-0.17	-0.22	-0.19	-0.16	0.12	0.17	1.00	-0.77
bg.m	0.18	0.18	0.18	0.17	-0.21	-0.19	-0.77	1.00

Table 5.2: Correlation matrix resulting from fit of trial 2. Labelling is as Table 5.1.

used for the lines). The spectral line shape used in the ADAS602 fitting program was Gaussian. Around 1999, in the course of a periodic re-alignment of the spacecraft platform, communication with SOHO was lost. SOHO is located at the L1 Lagrange point and calculations suggested that the high-gain antenna, over a period of a few months, would move back towards Earth's direction and allow re-establishment of communications. This was somewhat a race against time as the spacecraft would move out of the Lagrange stability region within about four months. A weak signal was ultimately picked up by the Spanish ground station of the NASA ground network. Full communication was gradually restored and the spacecraft brought 'back to life'. The CDS spectrometer returned to operation, but had experienced extremes of temperature and it was found that the spectral profiles had become substantially distorted. A line, which was originally a pure Gaussian, was now asymmetric. Spectral fitting at that time suggested that the new profile could be represented by a composite line shape with a Gaussian and Lorentzian component, with the significance of each controlled by additional fitting parameters. The function used is defined by equation 5.1, where λ_0 is the line centre, I_0 is the line height, w is the line width and α_1 and α_2 are the parameters which control the prominence of the Gaussian and Lorentzian components for the left-hand and right-hand sides of the line shape respectively [72].

$$I\{I_0, \lambda_0, w, \alpha_1, \alpha_2\} = \begin{cases} I_0 \left((1-\alpha_1) \exp\left(-\frac{(\lambda-\lambda_0)^2}{w^2}\right) + \alpha_1 \frac{1}{1+\left(\frac{\lambda-\lambda_0}{2\sqrt{\ln 2}w}\right)^2} \right) & \lambda < \lambda_0 \\ I_0 \left((1-\alpha_2) \exp\left(-\frac{(\lambda-\lambda_0)^2}{w^2}\right) + \alpha_2 \frac{1}{1+\left(\frac{\lambda-\lambda_0}{2\sqrt{\ln 2}w}\right)^2} \right) & \lambda \geq \lambda_0 \end{cases} \tag{5.1}$$

The test of FFS, in this case, comprises two parts. Firstly, a fitting of a 'pre-loss' spectral interval from NIS-2 for direct comparison with ADAS602 is made. Secondly, an FFS model, using a Gaussian and Lorentzian pair, is used to represent one of

the reference lines in the spectral survey, but with 'post-loss' data. Once this was complete, this fit provided a new spectral primitive that could be replicated, using the FFS coupling system, to represent each of the lines in a blended OIV ($2s^2 2p\ ^2P$ – $2s2p^2\ ^2P$) multiplet feature at ~ 553Å.

	FFS value	FFS error	ADAS602 value	ADAS602 error
l1.pos	553.410	0.060	553.410	0.098
l1.intensity	5.435	1.107	5.436	1.591
l2.pos	554.146	0.097	554.147	0.088
l2.intensity	9.716	3.620	9.713	3.203
l3.pos	554.591	0.047	554.591	0.033
l3.intensity	26.352	4.054	26.356	1.035
l4.pos	555.330	0.059	555.330	0.031
l4.intensity	5.844	1.143	5.845	1.607
bg.fwhm	0.551	0.055	0.551	0.248
backg.c	1.251	0.664	1.251	6.798
backg.m	0.030	0.214	0.004	90.093

Table 5.3: The fit parameters resulting from the fit to the (pre-loss) NIS-2 OIV multiplet ($2s^2 2p\ ^2P$ – $2s2p^2\ ^2P$). Parameter names are specified as described in Section 4.5, where 'l1', 'l2', 'l3' and 'l4' are the four lines, 'bg' is the Gaussian broadening element and 'backg' is the linear background. Note that, in this case, the line intensities values quoted are those from dummy parameters that sum the two components that are used to represent the lines (see main text for details) — however, the original labels for the simplified model are retained here for clarity. The first two columns give the parameter values and standard error as determined by FFS (from the diagonal elements of the covariance matrix) while the third and fourth columns show similar quantities, but as determined by ADAS602.

In the first case, it can be seen that FFS replicates the functionality of the ADAS602 and obtains a fit that is almost identical, as expected. This can be seen from Fig. 5.2, which compares the fitted models and experimental data, and from the results Table 5.3. For clarity, the fitted, relative intensities, as calculated by both programs, have been normalised to that of the left most line in the multiplet and tabulated in Table 5.5.

Due to the distortion on the recorded spectra, it is more problematic to fit the post-loss data. As discussed, ADAS602 uses a custom form to fit the new line shapes. The approach taken here is slightly different in that a connected pair of functional forms are used to fit each of the multiplet components. Specifically, a Gaussian and Lorentzian, coupled using FFS' MDL (see 4.5) are used and the example MDL is shown in Fig. 5.3. In order to determine suitable coupling between the Gaussian and Lorentzian primitives, to create the asymmetric line shape suitable for post-loss spectra, one of the high quality (4-star) reference lines was taken in isolation and

	FFS value	FFS error	ADAS602 value	ADAS602 error
l1.pos	553.521	0.090	553.520	0.064
l1.intensity	3.131	0.918	3.382	0.898
l2.pos	554.242	0.142	554.246	0.054
l2.intensity	5.665	3.131	6.149	1.802
l3.pos	554.672	0.069	554.671	0.021
l3.intensity	15.089	3.417	15.787	0.624
l4.pos	555.379	0.104	555.393	0.025
l4.intensity	3.450	1.099	3.473	1.130
bg.fwhm	0.535	0.086	0.518	0.182
backg.c	0.717	0.735	0.976	4.185
backg.m	0.009	0.247	-0.005	29.875

Table 5.4: The fit parameters resulting from the fit to the (post-loss) NIS-2 OIV multiplet ($2s^2 2p\ ^2P$ – $2s2p^2\ ^2P$). Parameter names are specified as described in Section 4.5, where 'l1', 'l2', 'l3' and 'l4' are the four lines, 'bg' is the Gaussian broadening element and 'backg' is the linear background. The first two columns give the parameter values and standard error as determined by FFS, while the third and fourth columns show similar quantities, but as determined by ADAS602.

	l1/l1	l2/l1	l3/l1	l4/l1
FFS	1.000	1.788	4.849	1.075
ADAS602	1.000	1.787	4.849	1.075
ADAS208 ($\mathcal{PEC}$)	1.000	2.023	5.001	1.046

Table 5.5: Comparison of line intensities as determined by fitting the pre-loss data with FFS and ADAS602 with the theoretically calculated photon emissivity coefficients ($\mathcal{PEC}$) values from ADAS208. In all cases the values have been normalised to the intensity of the the left most line comprising the multiplet, for clarity.

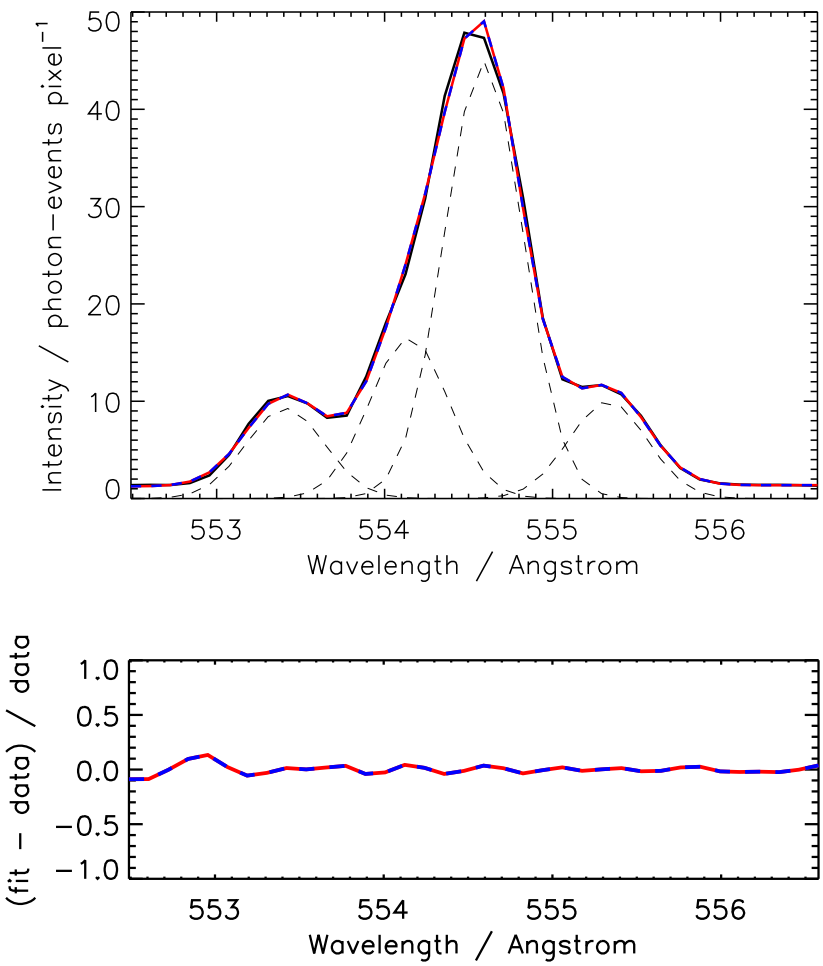


Figure 5.2: OIV multiplet as recorded by NIS-2 before the loss of SOHO, with fitted models from FFS and ADAS602 overlaid. Both fitting programs are using a similar model: four Gaussian line shapes, with a common width parameter, plus a linear background. The experimental data is in black, the FFS fit is the red, dashed line and the fit from ADAS602 shown in a blue, dashed line. A similar colour scheme is retained for the residual shown in the lower plot. Note that larger values of the scaled residual only occur at regions of low intensity.

fitted, with free, dummy parameters connecting the offset and intensity ratio of the two components. The line chosen in this case, was He I ($1s^2\ ^1S_0 - 2s2p\ ^1P_1$) at 584.334Å. These parameters (along with the fitted Lorentz width) could then be set fixed and used, in effect, to form a new feature primitive, for each of the multiplet components, in place of the Gaussian used in the pre-loss data. Figure 5.3 details the MDL used to do this and the resulting fit, using this model in Fig. 5.4

```
(dummy d)
(setval d.wingfactor 0.5)
(fixed d.wingfactor)
(setval d.wingshift 0.5)
(fixed d.wingshift)

(couple l1wing.intensity = d.wingfactor*l1.intensity)
(couple l1wing.pos = l1.pos + d.wingshift)
```

Figure 5.3: MDL statements to form the custom line shape for 'post-loss' SOHO-CDS data. FFS dummy parameters are used to control the displacement of the centre of the Lorentzian from the centre of the Gaussian (d.wingshift) and the relative intensity of the Lorentzian to that of the Gaussian (d.wingfactor). Note that these are set to a fixed value, derived from another fit (see main text for details). The coupling statements shown then make the connection between the main (Gaussian) line intensity (l1.intensity) and the (Lorentzian) wing distortion intensity (l1wing.intensity) and similarly for the displacement in line centres (l1.pos and l1wing.pos). Further statements of this type are then used to couple the rest of the lines in a similar way.

	l1/l1	l2/l1	l3/l1	l4/l1
FFS	1.000	1.809	4.819	1.102
ADAS602	1.000	1.818	4.668	1.027
ADAS208 ($\mathcal{PEC}$)	1.000	2.023	5.001	1.046

Table 5.6: Comparison of line intensities as determined by fitting the post-loss data with FFS and ADAS602 with the theoretically calculated photon emissivity coefficients ($\mathcal{PEC}$) values from ADAS208. In all cases the values have been normalised to the intensity of the the left most line comprising the multiplet, for clarity.

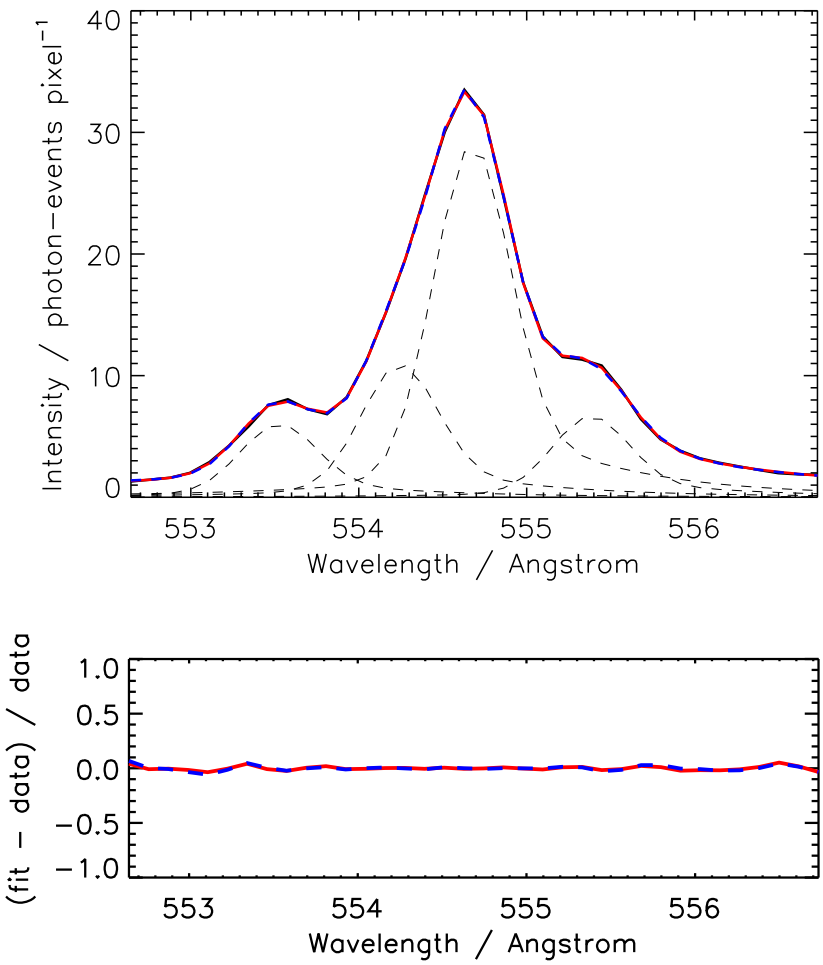


Figure 5.4: OIV multiplet as recorded by NIS-2 after the loss of SOHO, with fitted models from FFS and ADAS602 overlaid. In this case, the models differ slightly. ADAS602 is using four shapes resulting from the function as defined by eq. 5.1, while FFS is using four Gaussians, each with an associated Lorentzian with coupled relative intensity and positional offset (these parameters were determined from a previous fit of reference line He I at 584 Å). The experimental data is in black, the FFS fit is the red, dashed line and the fit from ADAS602 shown in a blue, dashed line. A similar colour scheme is retained for the residual shown in the lower plot.

5.3 Divertor Detachment Experiment at JET

This example stems from an interesting series of experiments on approaching the divertor density limit on JET [43]. A schematic of the divertor region is shown in Fig. 5.5 showing the magnetic flux surfaces of the plasma, the positions of Langmuir probes (KY4D) on the strike plate and the lines of sight of the spectrometer, KT3A, which observes the divertor region from above. Langmuir probes provide an established method for direct measurement of local electron temperature and density in attached plasmas. Unlike a passive measurement technique, such as spectroscopy, probes have the disadvantage that they can introduce impurities into the plasma, through physical sputtering from the probe surface due to ion bombardment; this, of course, also erodes the probe. The charged electrode of the probe means that it can also locally distort the electrical properties of the plasma. If the machine is operated with a detached divertor plasma, as is the case in this experiment, there is a more significant issue to consider — the probe will no longer be in contact with the plasma. Operation in this mode will be particularly necessary for higher power devices such as ITER [28], where the heat load, with a fully attached plasma, on the divertor target plates will be significant. In order to alleviate the load on the divertor surfaces, neutral hydrogen gas (or indeed impurity seeded gas) can be introduced to the divertor. Charge exchange can occur between the neutral gas atoms and the ions and atoms reaching the divertor from the main plasma, which then undergo radiative cooling, making them less energetic and so less efficient at sputtering. In this scenario, we look towards spectroscopic techniques (such as the vertical LOS offered by KT3) to diagnose the divertor temperature and density.

The experiments involved a real-time feeback control system for gas puffing, with the intention of operating in a stable detached regime. The feedback arrangement used data from the far infra-red (FRI) interferometry diagnostic, KG1 [73] which provides a line-integrated electron density measurement. Spectroscopic observation, from KT3A, was of a spectral interval encompassing a series of Deuterium Balmer lines. The KT3 lines of sight pass through an entire vertical section of plasma, but neutral hydrogen emission is localised to a zone of existence, determined by its temperature and ionisation potential. This zone is close to the divertor target. As discussed in the introduction, the Balmer series of lines can be treated as an integral special feature, from which density, temperature and transient ionisation state can be deduced. In the present example, FFS will operate using a simpler model comprising Voigt line shapes, where the Lorentzian components of the widths of the lines are coupled to density. In addition to these, Gaussian line shapes will be used to represent a further seven

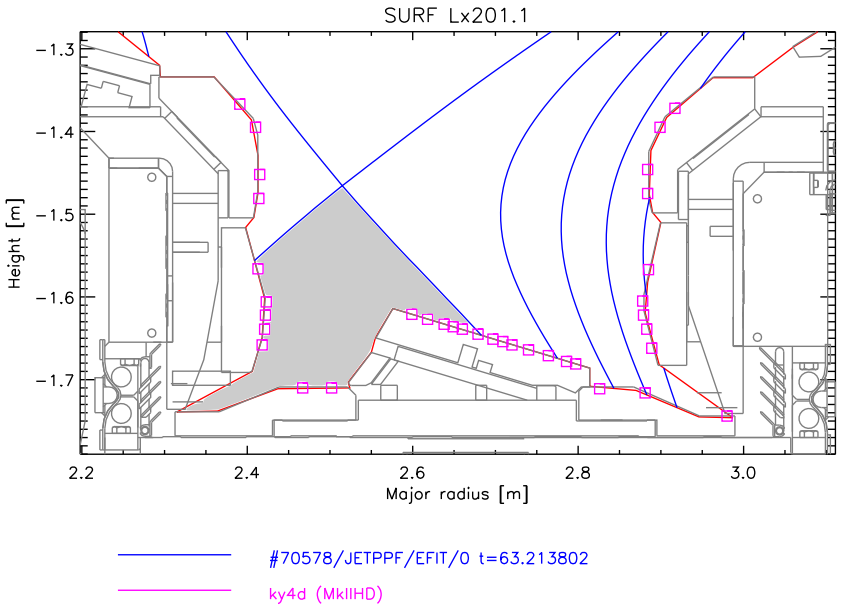


Figure 5.5: Poloidal cross-section of JET. The magenta squares mark the positions of the Langmuir probes (KY4D). The set across the divertor strike plate provide a comparison to the spectroscopic data of this study. Note the position of the 'private flux region' enclosed by the last closed flux surface and the base of the divertor (indicated by the grey-shaded area). The probes in this region do not receive direct plasma flux. Figure produced by the JET utility program, *SURF*.

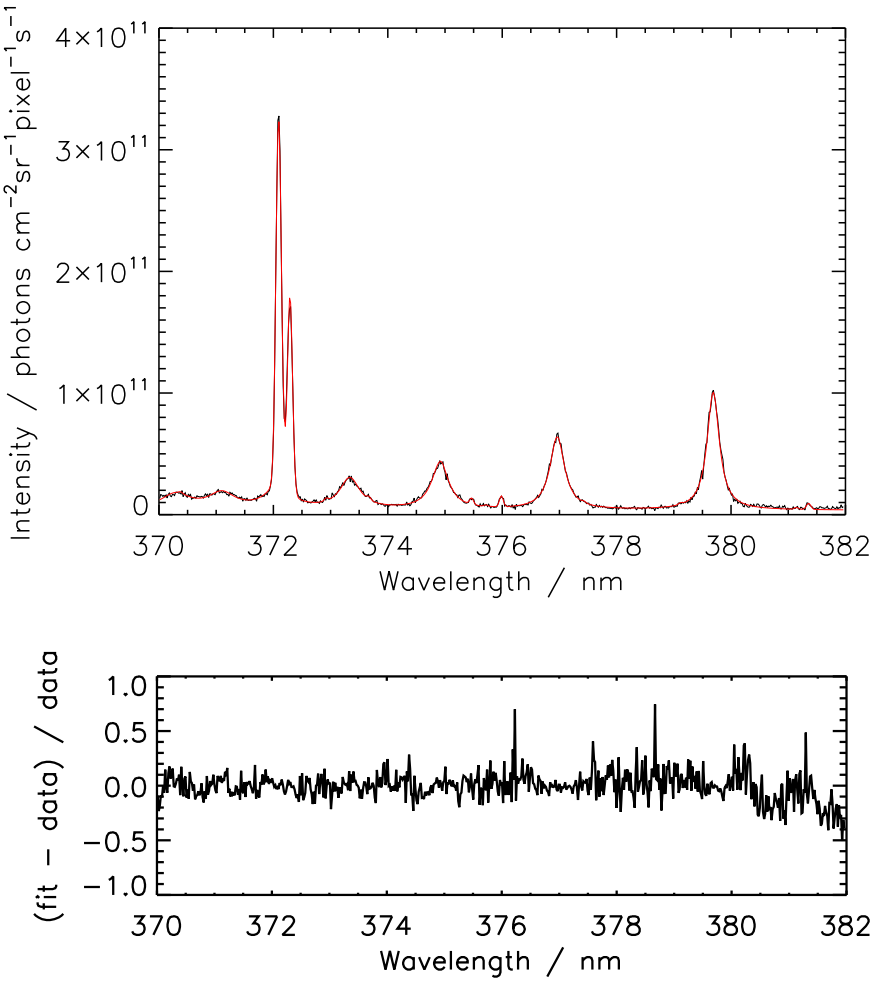


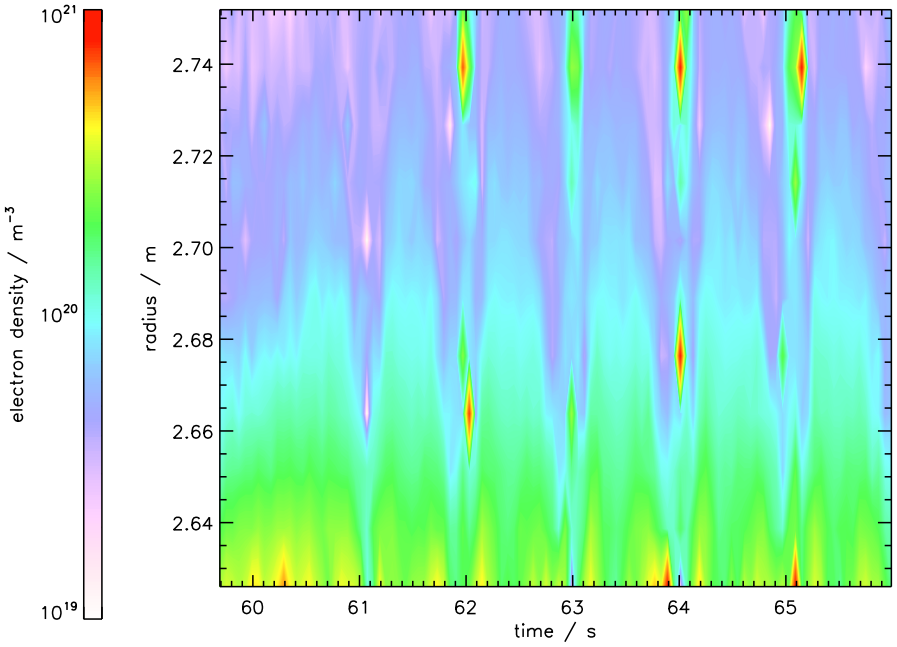
Figure 5.6: Spectrum as recorded by instrument KT3A at JET, pulse #70578. The higher members of the Balmer series, commencing with D(n=10→2), are present, merging into the free-bound continuum. Note the strong Beryllium multiplet at 372nm. The fitted model is shown in red, with the fit residual shown in the lower plot.

impurity lines present in the spectrum and a simple, linear background element is also included. This builds upon the work of Meigs *et al* [74, 32]. Multiple observations were taken throughout the discharge, of which one frame, complete with the fitted model, is shown in Fig. 5.6. The KT3A system has multiple lines of sight through the divertor, so the complete data set allows reconstruction of the density profile across the divertor at each time.

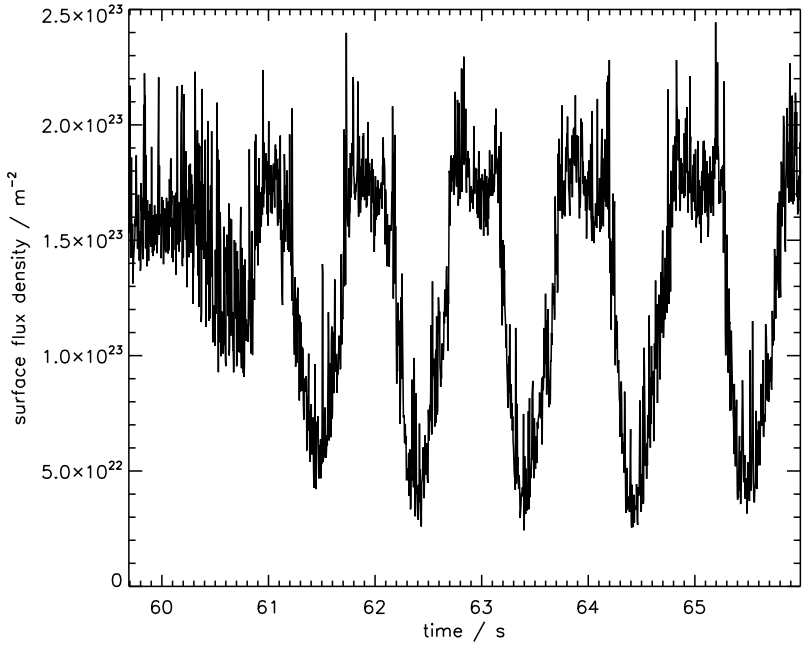
In order to reduce the number of free parameters in the fit, several couplings are made between the model parameters. The Gaussian components of the Balmer lines are coupled — the emission is fairly localised and so, thermal broadening should be similar and, in any case, a large proportion of this width component will be instrumental. The

positions of the Balmer lines, can be readily coupled relative to each other. Similar couplings have been made between the Gaussian widths and positions of the three strong Be III lines in the model spectrum. Relative positional couplings have also been made between four weak oxygen lines in the spectrum. In this case, the coupling has been made with a view to fitting many spectra (for each time and radial position) in a batch operation. It is not practical to create individual models for each recorded frame, instead we will re-use a single FFS model definition for each fit. It is likely that some of the weaker features will not appear in many frames and fitting, with what is now an inappropriate model, will prove problematic unless such constraints are made from the outset. Despite the effort to reduce the parameter set in this way, this remains a 38-parameter model.

Using the described model, all of the observations from KT3A, for JET pulse number 70578 were fitted. A contour plot of the derived density parameter is shown in Fig. 5.7, along with the particle flux density time trace from a Langmuir probe near the divertor strike-point. The data from the probe shows an oscillating pattern — the divertor plasma periodically switching from detached operation back to an attached divertor leg. The same oscillation is seen in the spectroscopically derived density measurement. This was in fact accidental behaviour resulting from incorrect gain setting on the real-time control system controlling the gas puffing into the divertor. However, this actually serves as an excellent point of comparison between density measurement from the divertor probes and those obtained spectroscopically — a comparison can be made in the time interval at the points of detachment (or re-attachment). Recorded data from 11 of the divertor probes from diagnostic KY4D can been used to construct a density profile across the divertor at several times during the pulse. The density profile obtained from the spectral fits can then be compared. The probes record at a much higher temporal resolution than the spectrometer and so the measurements are, in fact, averages over the exposure time of the recorded frames. Figures 5.8 shows this for frames around $t = 63.23$ s (where the plasma is beginning a detachment cycle).

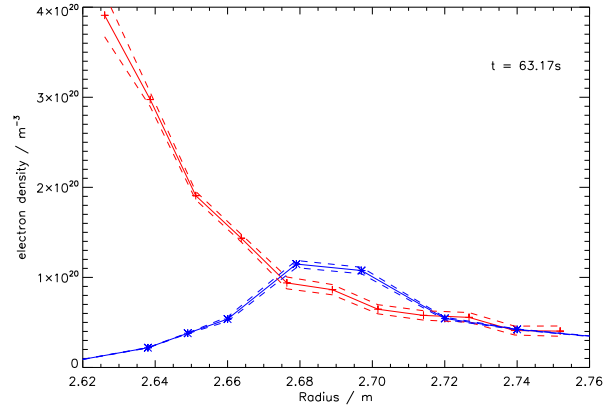


(a) FFS derived electron density

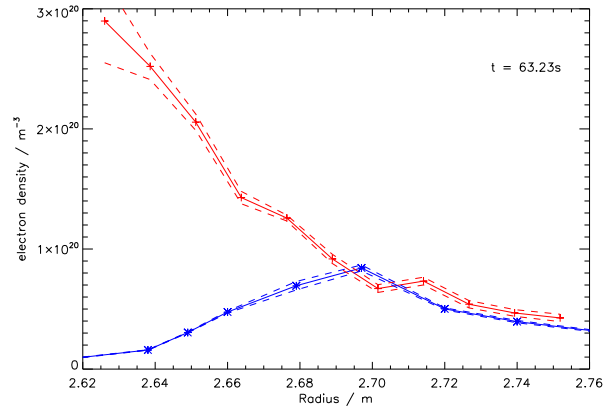


(b) Particle flux arriving at Langmuir probe S17C at JET

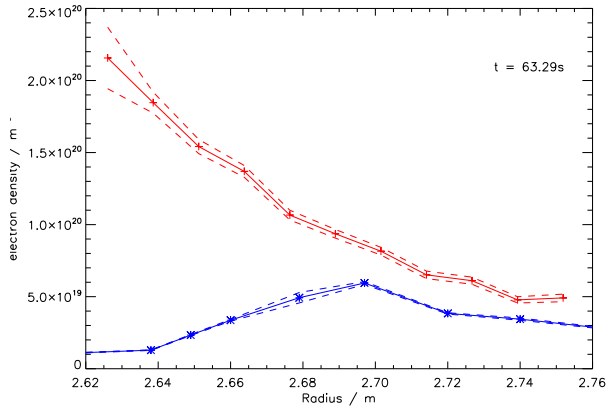
Figure 5.7: (a) A colour-shaded representation of the electron density, as a function of position and time, as extracted from a set of KT3 measurements by FFS. Note the broad periodicity in time arising from attachment / detachment of the divertor plasma from the strike plate. (b) A graph of ion fluxes as measured by one of the probes (S17C) on the strike plate at R=2.679m. The anti-phase relationship supports the attachment / detachment scenario, as discussed in the main text. In (a), the bright (red) features occurring in periods of attachment, suggesting very high density are spurious. The signals at these points are very weak and it is not possible to infer density from the fits. The display program does not reject these failed fits.



(a) t=63.17s



(b) t=63.23s



(c) t=63.29s

Figure 5.8: Electron density measurements versus radial position from FFS fits to spectroscopic data (KT3A, in red) and Langmuir probes (KY4D, in blue) at three times during the pulse. The fall off below 2.68m of the probe signals indicates the private flux region (see Fig. 5.5). The spectroscopic signals remain valid as the the emitting volume moves up from the divertor floor. The large error bar on the spectroscopic data at 2.63 m, reflects known increased noise levels for this track. It can be seen from 5.7 that moving from (a) – (c), the plasma progresses from attached to fully detached. By time (c), the probes are seen to be ineffective — the signal falls substantially.

5.4 Zeeman Split Features

At the relatively cool temperatures in the JET divertor and with magnetic fields circa $\sim$ 3 T, the Zeeman splitting of impurity spectral lines is evident and diagnostic. If resolvable, the separation of the components gives a measure of the local magnetic field strength. A familiar example is the CI ($2s^2 2p3s\ ^3P$ – $2s^2 2p3p\ ^3P$) $\sim$ 909 nm for which a simulation was shown in fig 2.1. It is noted that the nominally complex feature is quite condensed, with strong sub-feature overlap. At issue, therefore, is whether information on field strength can still be extracted. For such extraction, the availability of a theoretical special feature model, incorporating the sub-feature connections is essential. As discussed in section 2.2, ADAS has some history of handling the Zeeman / Paschen-Back split features, stemming from work by Hey [10, 11]. ADAS has archived Zeeman / Paschen-Back special features in the code xPaschen (the underlying code for ADAS603). AFG provides access to these special features. An FFS-AFG fit using the xPaschen primitive is shown in Fig. 5.9.

This model was then used by FFS in batch operation (Section 4.10) to fit all of the spectra exhibiting this Zeeman split carbon feature for shot #78658 at JET. The results of the fits allow for construction of a magnetic field strength profile in time and space for the shot (as shown in Fig. 5.10). The magnetic equilibrium code *Flush* was then used to construct a similar contour, over the same radii/time range. There is broad correspondence between the two data sets, rather than complete agreement. However, it should be noted that the spectra from which the magnetic field values are inferred are from lines of sight which are not completely vertical, and that the carbon emission will be from a (range of) finite height(s) above the divertor floor. This means that there is some uncertainty as to the radius that should be attributed to the derived magnetic field values. Also pertinent to this issue is the fact that the height at which the radial profile has been calculated by *Flush* was arbitrarily set to $z = -1.6$ m. The smooth nature of the *Flush* plot is the result of the interpolative nature of the routine — a reconstruction of the magnetic field structure is being performed using data points external to the vacuum vessel (diagnostic field coils).

Gafert attempted the more difficult task of determining the relative levels of carbon emission from the inner and outer scrape-off layer at JET [75] (a similar study has been carried out at the Axially Symmetric Divertor EXperiment Upgrade (*ASDEX-U*) tokamak [76]). This meant using a horizontal, mid-plane view, which would display two overlapped features corresponding to emission from impurity atoms near the in-board wall (high magnetic field) and out-board wall (low magnetic field). The analysis sought to use two xPaschen features to infer the intensity ratio of the two zones

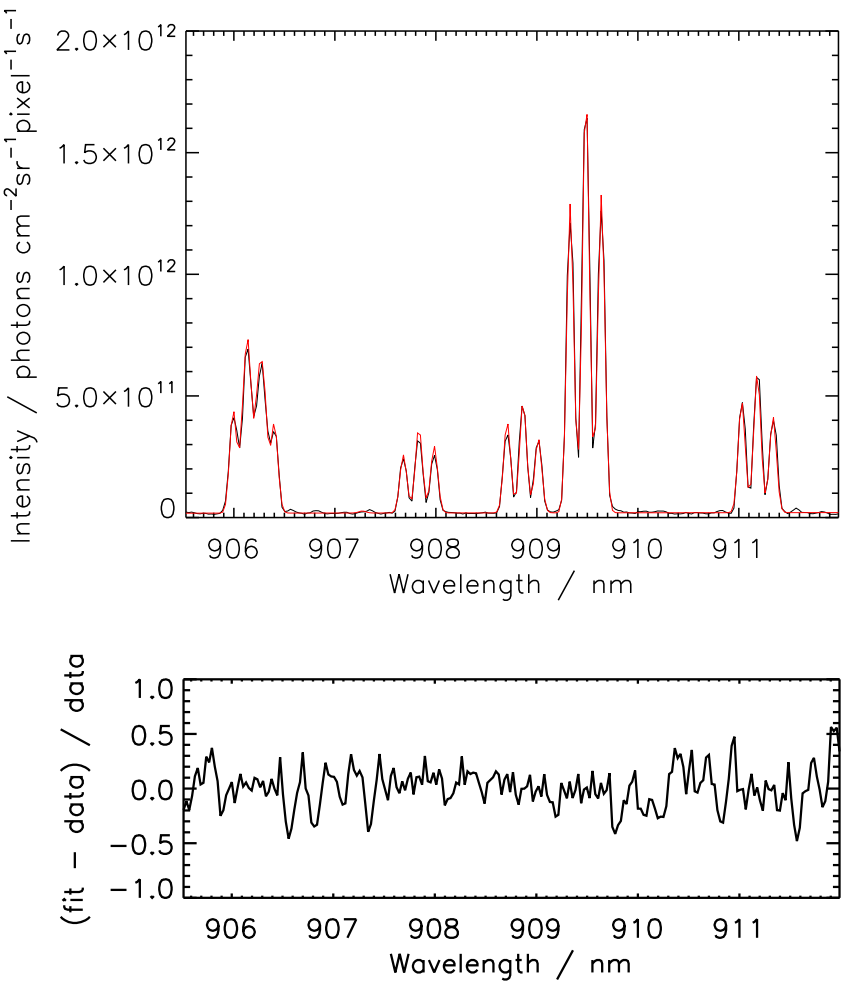
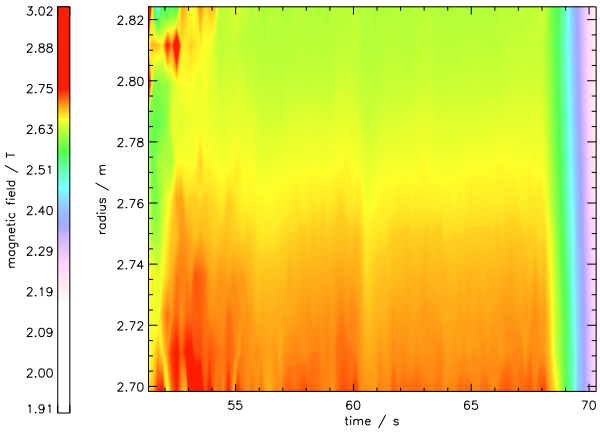
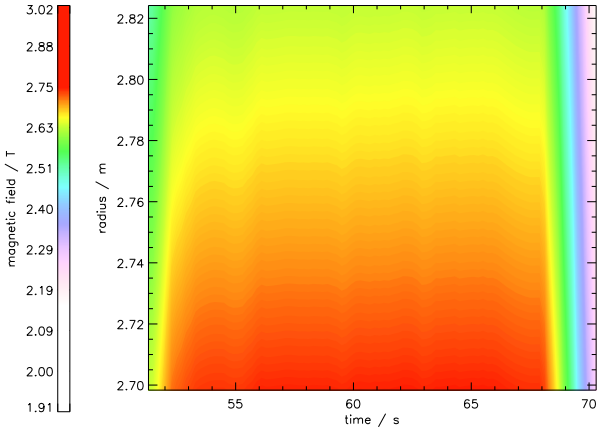


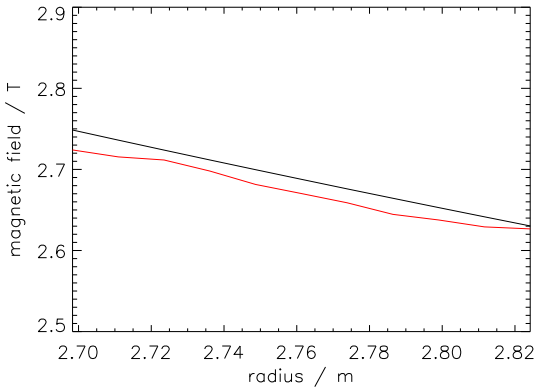
Figure 5.9: C I ($2s^2 2p3s\ ^3P - 2s^2 2p3p\ ^3P$) emission at ~ 909 nm as recorded by KT3C at JET, pulse #78658 (R = 2.749 m, t = 61.3 s). The fitted xPaschen / ADAS603 model is shown in red, with the fit residual shown in the lower plot. The magnetic field, determined from the fit, is $2.665 \pm 1.89 \times 10^{-3}$ T.



(a) Magnetic field obtained from FFS fit.



(b) Magnetic field as determined by magnetic reconstruction code *Flush*.



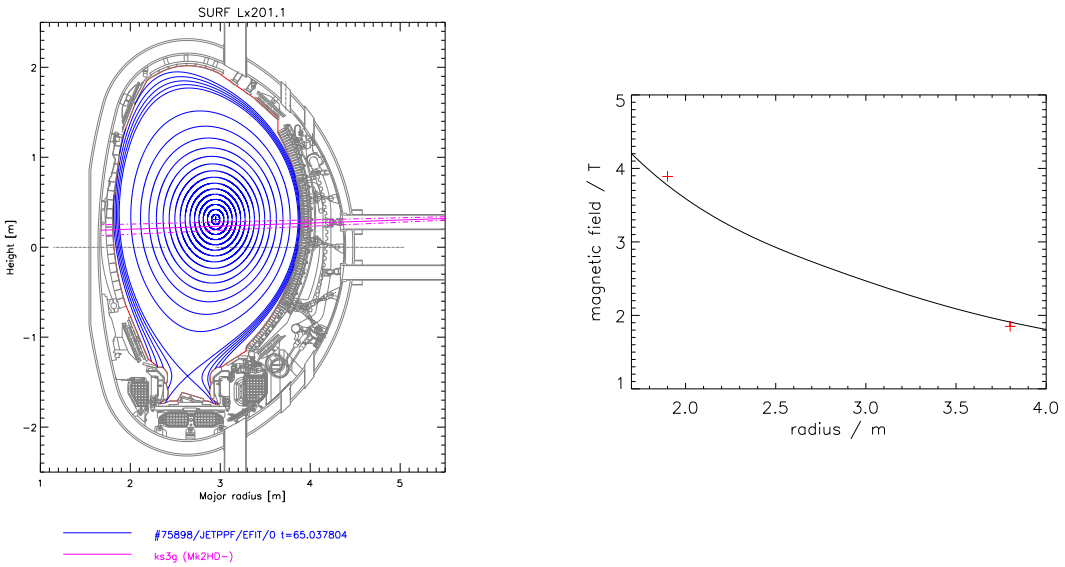
(c) Radial magnetic field profile (t = 60s).

Figure 5.10: (a) A colour-shaded representation of the magnetic field, as a function of position and time, as determined from batch fitting a series of KT3C spectra using FFS. (b) A similar contour plot, over similar space and time, as calculated by *Flush*. The two plots, while not identical, show a broad agreement across time and space. (c) A time-slice (at t = 60s) of the 2D magnetic field data. *Flush* results are shown in black, whilst the red line shows the data from FFS.

of emission. It was noted that there strong cross-correlation of line widths and the separation of the Zeeman components. In practice, the magnetic field parameter was removed from the fit — instead these were treated as input data, with the values coming from the magnetic equilibrium data (EFIT). Further to this, only a single electron temperature parameter was used (rather than two; for the inner and outer emission sources) with a fixed ratio between inner and outer components of the spectra. That is to say that the Gaussian broadening applied to the xPaschen output is controlled by a single parameter.

For this work, an attempt is made to fit the same C III ($1s^2 2s3s\,^3S$ – $1s^2 2s3p\,^3P$) multiplet, observed at JET, pulse #75898, using spectrometer KS8, which is using a viewport previously used for the (now redundant) KS3 horizontal line of sight (see Fig. 5.11). The FFS model used for the fit actually makes use of four FFS-AFG xPaschen elements (Section 3.1); separate model elements are used for the π and σ components for both the high and the low field side. This allows for optical effects that may suppress the polarised components to varying degrees. The ratio of the multiplication factors (Section 4.2.8) applied to the π and σ components is, however, retained across the inner and outer pairs via coupling. Finally, the in-board / out-board elements are allowed separate wavelength shifts (Section 4.2.9) to account for the differing flow velocities at these locations.

The FFS fit (shown in Fig. 5.12) provides estimates of the magnetic field strengths of 3.89 ± 0.02 T at the in-board wall ($R \sim 1.9$ m) and 1.85 ± 0.02 T at the out-board wall ($R \sim 3.8$ m). If we compare these results to the magnetic field strength, as a function of major radius, as calculated by *Flush* (Fig. 5.11(b)), we see that these estimates appear reasonable.



(a) (b)

Figure 5.11: (a) Shows the KS8 line of sight across the JET mid-plane — figure produced by, JET utility program, *SURF*. (b) Displays the magnetic field strength as a function of major radius, at height $z = 0.2$ m i.e. indicative of the magnetic field variation along the line of sight. The red crosses indicate the magnetic field values as calculated by the FFS fit. Note that the associated values of radii are approximate — without modelling the impurity transport, one can only assume that the emission is very close to the walls of the machine.

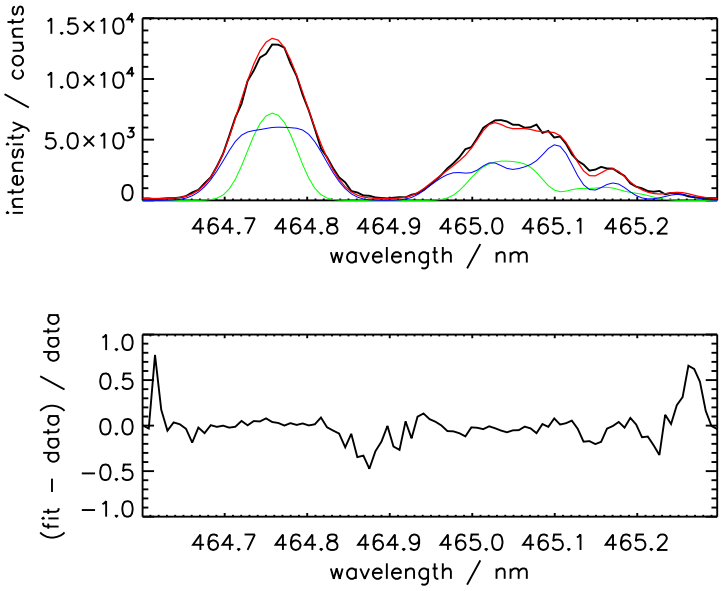


Figure 5.12: C III ($1s^2 2s3s\,^3S$ – $1s^2 2s3p\,^3P$) multiplet at ~ 465 nm as recorded by KS8 at JET, pulse #75898. The fitted FFS model is shown in red, with the low-field and high-field side Zeeman component features indicated in green and blue, respectively. The fit residual shown in the lower plot.

5.5 Diatomic Molecular Spectra in the JET Divertor

A previous study at JET identified molecular spectral emission resulting from CD, C_2 and BeD by Duxbury *et al* [42]. An AFG module has been produced to allow manipulation of the *CALCAT* synthetic feature [45] for use in a pedagogical sense and for confrontation with experiment via FFS. At this point, we return to the experimental analysis undertaken in the aforementioned study [42], which was performed only in a qualitative sense, and attempt a more quantitative analysis, using FFS-AFG. Firstly, the data recorded by the survey spectrometer KS3A is considered for analysis. KS3A, at the time of this experiment, provided wavelength coverage of 414 nm – 736 nm, over 1010 pixels — therefore, the data is of relatively low resolution. The model utilised two *CALCAT* features to represent the BeD ($A^2\Pi - X^2\Sigma$) molecular emission (Duxbury *et al* suggests that it is possible to see overlapping features, at different temperature, along the line of sight). Figure 5.13 shows the results of the fit to KS3A data for JET pulse #35687. It should be noted that with a complex feature like this, where fitting parameters alter a finer sub-structure of the feature and data of low resolution is used, then confidence in the determined parameters is low. In this case, a spread of so-called rotational and vibrational temperatures will provide similarly good fits.

Attention is, therefore, turned to higher resolution data coming from KS3B, which provides coverage of $\sim$ 8 nm over 750 pixels. As this wavelength interval is far smaller than that of KS3A, which displayed the molecular feature in its entirety, we must focus on a sub-region of the spectral signature. It was found that the band head assists the fitting algorithm localise the emission. The *P* and *R* branches of the feature show a large degree of repetition and the model uses a 'shift element' (Section 4.2.9) to account for discrepancy in wavelength between the theoretical model and experiment — the fitting algorithm can easily find local minima.

Experiments at the *PISCES-B* divertor simulator [77] indicate that molecules resulting from chemical erosion processes will exhibit vibrational bands that are not in Boltzmann proportions. It has also been suggested that the impact energies of the fuel ions in a divertor are low — such that chemical, rather than physical, sputtering is the dominant formation mechanism [78]. Therefore we should expect that the emission considered here requires a model with additional free parameters to control the proportions of the the molecular vibrational bands. These parameters can be considered as deviates from Boltzmann.

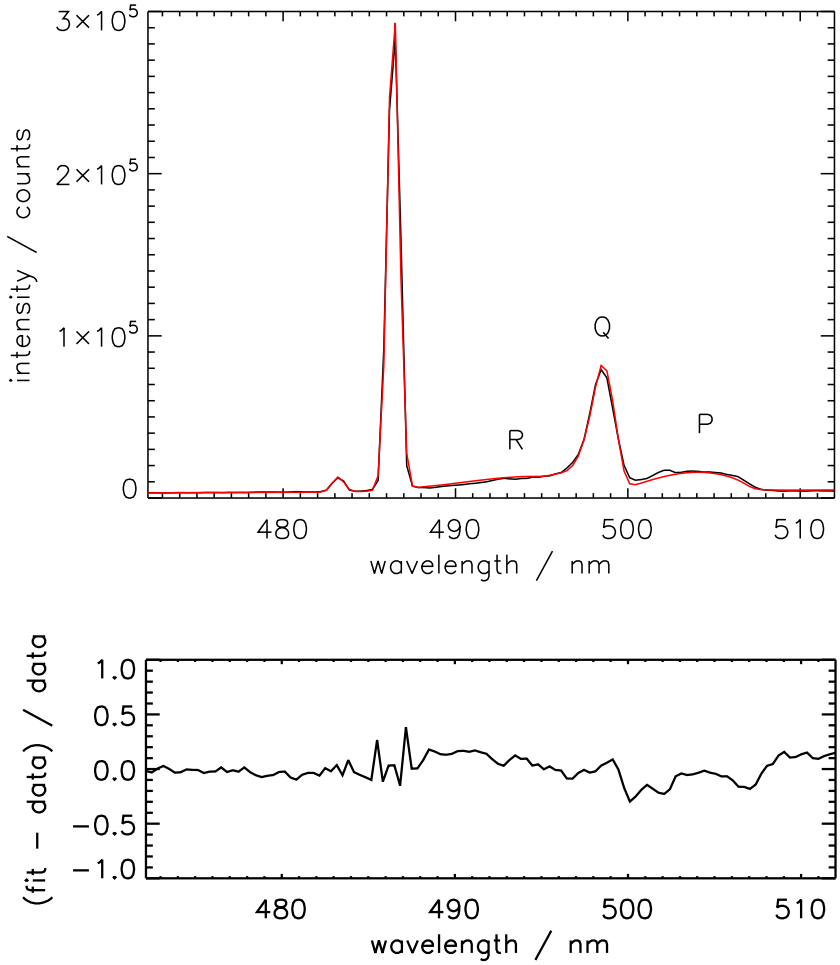


Figure 5.13: Observed beryllium deuteride (BeD) spectrum from instrument KS3A during JET pulse #35687. The transition shown is $A^2\Pi$ – $X^2\Sigma$. Some atomic lines are also present (e.g. deuterium line D_β (n=4→2) at ~ 486 nm). A fitted FFS model is overlaid in red. The model, via *AFG*, is utilising the *CALCAT* molecular modelling code to represent the BeD molecule, whilst the atomic lines are represented by simple Gaussian line shapes. A linear background element is also included. The vibrational and rotational temperatures determined from the fit are: T_r = 3962 K and T_v = 5509 K. There is broad correspondence between the simulated spectrum and the data — agreement is seen for the Q and R branches, but there is a marked discrepancy in the P branch.

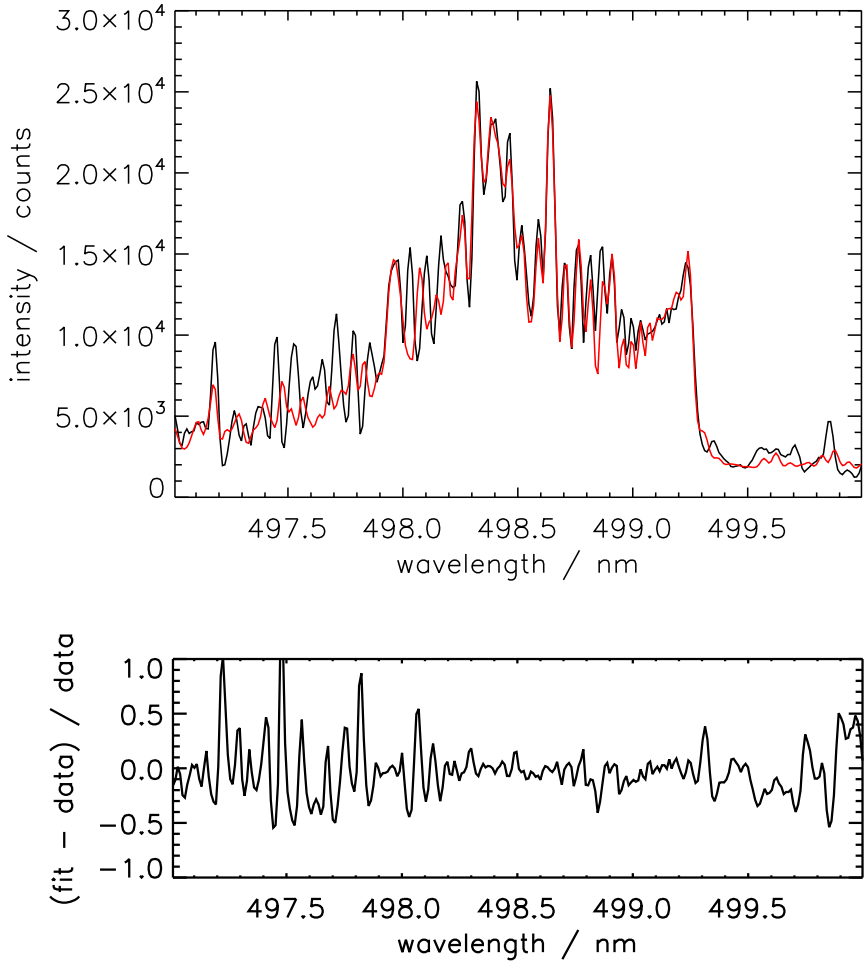


Figure 5.14: Similar beryllium deuteride (BeD) feature as displayed in fig 5.13 but, at higher resolution, from instrument KS3B during JET pulse #35689. A fitted FFS model is overlaid in red. The model consists of two *AFG-CALCAT* elements, with a linear background. Connected Gaussian lines have also been included, which appear to be the result of transitions in Fe I, the source of which is unknown. The vibrational and rotational temperatures determined from the fit are: T_{v_1} = 5847 K and T_{r_1} = 2106 K, T_{v_2} = 8392 K and T_{r_2} = 2746 K. There is broad correspondence between the simulated spectrum and the data — but it cannot be described as a *good fit*.

5.6 VUV Divertor Impurity Spectra

Thus far, the analysis scenarios considered have either not required an atomic population model or, in the case of the Balmer series feature (Section 5.3), not relevant in extracting the parameter of interest (density). Now we consider a Violet / Ultra Violet (V/UV) spectrum from diagnostic KT7 at JET. This apparatus is a 'double SPRED' spectrometer with two fixed gratings (2105 g/mm and 450 g/mm) and has a vertical LOS through the core plasma, to the divertor region and is primarily used to analyse the emission from impurity species present here. In this example, a spectrum from the 2105 g/mm grating during JPN #35421 at $t = 18.02$ s showing various CIII and CIV lines is examined. As discussed in Section 2.3, such lines, for a given ionisation stage, are connected; the ratio of their intensities defined by the underlying atomic populations. Further to this, there is of course the connection between the various ionisation stages of the ion. The ratios of these lines are then very much diagnostic — they can infer the density and temperature environment of the emitters. ADAS can provide the required photon emissivities required to model these spectra (ADAS208). An AFG module has been written that allows interpolation of the archived data ($\mathcal{PEC}$) for each of the ionisation stages required, for the input parameters (T_e, n_e). The FFS model uses two ADF15 data files (one for the C III and the other for the C IV emission present) via the FFS-AFG feature (Section 3.1). To each of these, a multiplier element (Section 4.2.8) is used to scale the $\mathcal{PEC}$s), before a Gaussian broadening function is applied. A Voigt function is used to fit the He II *nuisance* line at 304 Å and, finally, a linear background element has been included. The results of the fit of this model to the data is shown in Fig. 5.16. The electron temperature extracted from the C III line components of the model is 5.53 ± 0.18 eV and from the C IV lines, 15.72 ± 0.60 eV. It was not possible to deduce the electron densities from the observed data — these fit parameters exhibited very large error bars. Examination of the dependance of the $\mathcal{PEC}$s on temperature and density, over a reasonable range for the experiment, however, shows that this is not unexpected - there is very weak dependence of the $\mathcal{PEC}$s with density over the expected density range — a large spread of densities are equally likely from a fitting perspective. An example plot of the $\mathcal{PEC}$ surface for the C IV ($1s^2 2s$ – $1s^2 3p$) line at 312.4 Å is displayed in Fig. 5.15.

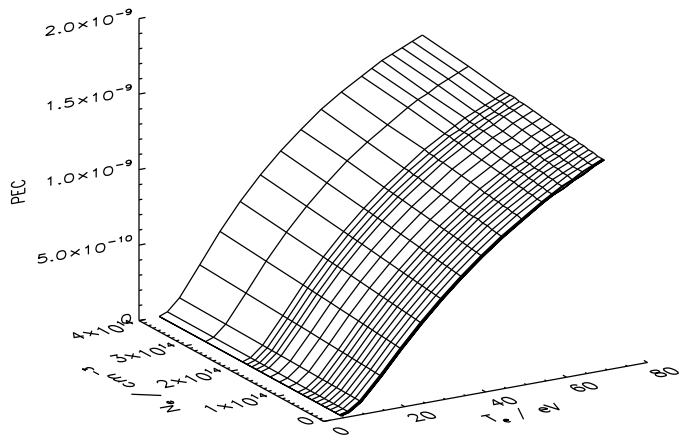


Figure 5.15: Surface plot of $\mathcal{PEC}$ for C IV ($1s^2 2s$ – $1s^2 3p$) at 312.4 Å for the range of density and temperature in the archive files used for the fit (Fig. 5.16). Note that there is a strong dependance on electron temperature (T_e), but the density (n_e) dependence is not nearly as significant.

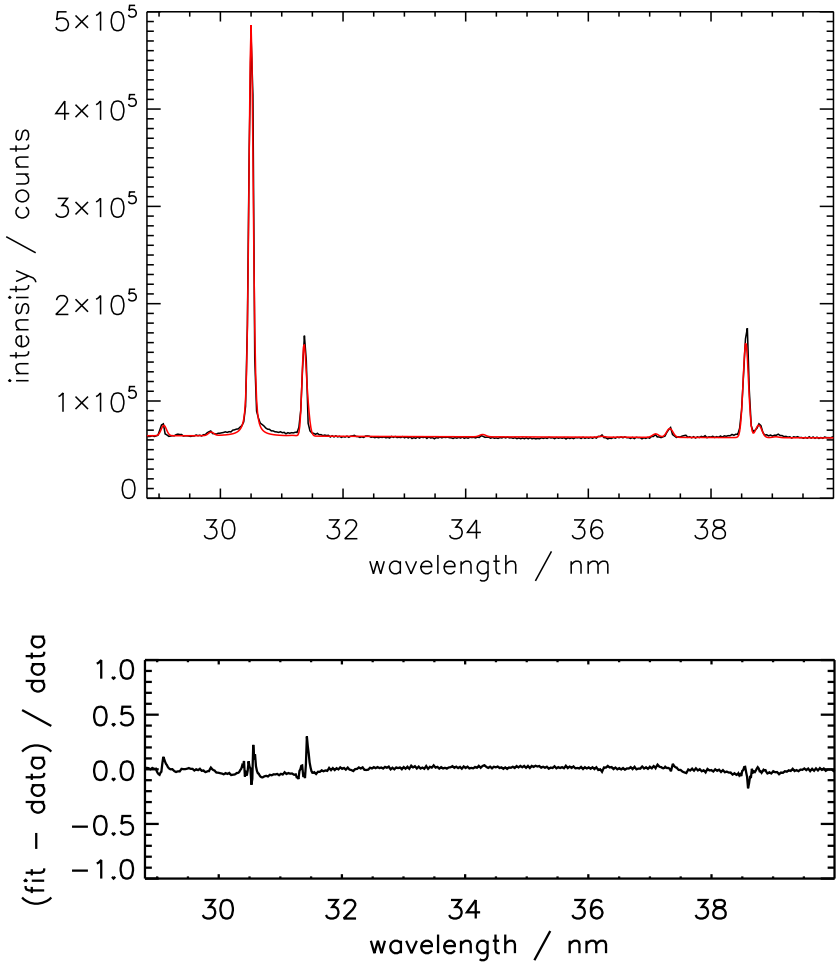


Figure 5.16: Spectrum recorded by JET diagnostic KT7/2, JET pulse #35421 (t = 18.02 s). The experimental data is shown by the black solid line, whilst the FFS fit is shown in red. The lower plot shows the residual from the fit. The electron temperature extracted from the fit to the C III line components of the model is 5.53 ± 0.18 eV and from the C IV lines, 15.72 ± 0.60 eV.

5.7 He-like Argon Spectra from TEXTOR

This example considers soft x-ray spectra emitted by He-like Argon, as recorded at the **T**okamak **EX**periment for **T**echnology **O**riented **R**esearch (*TEXTOR*) [79]. Specifically, experimental data from shot 81156 is examined here [80]. The apparatus involved is a bent-crystal Johann-Bragg spectrometer [81], with a horizontal, radial line of sight, which provides high resolution data across a narrow wavelength region of 3.94 – 4.05 Å. Following the notation of Gabriel [52], the familiar w, x, y and z lines of the He-like system (see Table 5.7 for details) can be identified in the experimental data. The $n = 2$ (a, k, q, r) satellite lines (arising from the related Li-like system) are resolved. The s, t and j lines of the Li-like ion are also visible, although they blend with the x, y and z lines respectively. Details of these $n = 2$ satellite lines are given in Table 5.8. Additionally, there is an envelope of $n = 3$ satellites observed in the wing of the main w resonance line. There is an accumulation of $n = 4$ satellites which are heavily blended with the w line too. Lines with spectators lying in higher n will not deviate greatly from the w line centre and are not readily resolved (but will 'enhance' the intensity of the w line).

Label	Transition	A^r / s^{-1}	Wavelength / Å
w	$1s2p\ ^1P_1 - 1s^2\ ^1S_0$	1.09×10^{14}	3.9491
x	$1s2p\ ^3P_2 - 1s^2\ ^1S_0$	3.16×10^{8}	3.9661
y	$1s2p\ ^3P_1 - 1s^2\ ^1S_0$	1.46×10^{12}	3.9696
z	$1s2s\ ^3S_1 - 1s^2\ ^1S_0$	4.45×10^{6}	3.9945

Table 5.7: Details of the spectral lines of interest arising from the He-like stage of Argon (Ar^{16+}). The labelling follows Gabriel's notation [52]. The corresponding atomic transition, associated radiative rate (A^r / s^{-1}) and the wavelength of the emission are listed for each.

Label	Transition	A^r / s^{-1}	A^a / s^{-1}	Wavelength / Å
a	$1s2p^2\ ^2P_{\frac{3}{2}} - 1s^22p\ ^2P_{\frac{3}{2}}$	1.36×10^{14}	9.50×10^{12}	3.9857
j	$1s2p^2\ ^2D_{\frac{5}{2}} - 1s^22p\ ^2P_{\frac{3}{2}}$	4.94×10^{13}	1.54×10^{14}	3.9938
k	$1s2p^2\ ^2D_{\frac{3}{2}} - 1s^22p\ ^2P_{\frac{1}{2}}$	5.69×10^{13}	1.47×10^{14}	3.9898
q	$1s2s2p\ ^2P_{\frac{3}{2}} - 1s^22s\ ^2S_{\frac{1}{2}}$	9.73×10^{13}	3.23×10^{12}	3.9813
r	$1s2s2p\ ^2P_{\frac{1}{2}} - 1s^22s\ ^2S_{\frac{1}{2}}$	8.44×10^{13}	1.59×10^{13}	3.9834
s	$1s2s2p\ ^2P_{\frac{3}{2}} - 1s^22s\ ^2S_{\frac{1}{2}}$	8.07×10^{12}	9.41×10^{13}	3.9678
t	$1s2s2p\ ^2P_{\frac{1}{2}} - 1s^22s\ ^2S_{\frac{1}{2}}$	2.43×10^{13}	8.18×10^{13}	3.9687

Table 5.8: Similar to Table 5.7, except for satellite lines (with $n = 2$ spectator electrons) from the Li-like stage of Argon (Ar^{15+}).The Auger rates of the doubly excited states (A^a) are given in addition to the data included in Table 5.7.

The *FFS* model used to fit this data comprises of a a He-like satellite line feature (as described in Section 2.7) together with a simple linear background. The fitted spectrum is displayed in Fig. 5.17. The fitted model shows broad agreement with the experimental data and deduces parameter values of $T_e = 442 \pm 0.14$ eV and $n_e = 9.36 \times 10^{13} \pm 5.38 \times 10^{12}$ cm^{-3}. The temperature determined is somewhat low compared to that suggested by Marchuk (1.01 – 1.05 keV) [80], however, it is noted in this work that charge exchange recombination contributes to the intensities of the lines in this spectra – an effect not included in the model used here. Additionally, there is an issue regarding errors in the energies of the levels coming from the atomic structure codes. Due to the fact that this feature is observed in such a narrow wavelength region (and therefore a small range of energy) even small errors in the atomic energy levels will result in a corresponding error in the transition energies between levels. This can lead to spectral lines having significantly incorrect wavelength positions. For the current model, the level energies were adjusted to the best available data, recommended by NIST, compiled by Saloman [82]. However, the data is not a complete set; it does not contain all of the necessary energy levels and so, some of those used in the modelling remain un-adjusted. Therefore, it is possible that some of the lines in the simulated spectra could be incorrectly placed — distorting the results of the fit.

As discussed in Section 2.7, the model used here used only a partial implementation of the full model as described, due to time constraints. However, the modelling work is now complete and it is anticipated that this will lead to further analysis, of new JET data, once operations resume, in late 2011.

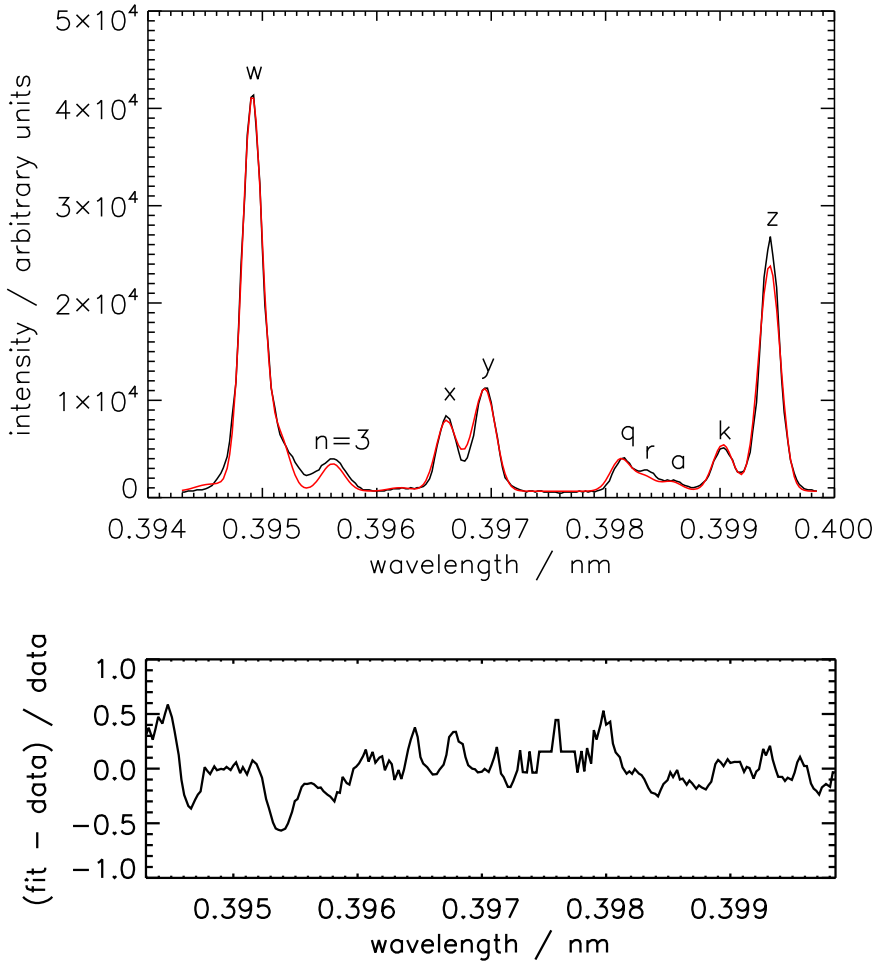


Figure 5.17: He-like Argon x-ray spectrum from *TEXTOR* shot #81156. The experimental data is shown by the black solid line, whilst the FFS fit is shown in red. The lower plot shows the residual from the fit.

Chapter 6

Conclusions and Future Work

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 (Section 3.1). The accompanying graphical exploration program (Section 3.2) delivers a visual demonstration of special feature parameter response as a prelude to the automated fitting of such features to experimental data (which is handled by the second of the aforementioned tools, the Framework for Feature Synthesis (FFS). The FFS package (sec. 4.4) 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. Models are easily specified by a custom model definition language (Section 4.5). Consideration has been given to several aspects of spectral fitting and the required functionality incorporated into the system e.g. complex coupling relations between model parameters can be set (also specified via the model definition language) — Section 4.6. This facility can reduce the free parameter set of the model and so, help constrain the fitting problem. 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.

Another practical aspect of spectral fitting that has been investigated, is computational performance; the analytic results of common combinations of the basis functions have been documented and efficient forms of the functions and/or their partial derivatives identified (Section 4.2)). 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 (Section 4.7). Importantly, the coupling system can also perform analytic differentiation of

the coupling relations — such that the computational benefit of using known partial derivative expressions is not lost when such parameters become intermediates within a coupling chain defined for a free parameter. A brief practical study of the success of the model optimisation, in terms of performance, is carried out in Section 4.12. The attention given to performance permits inter-shot analysis of spectra i.e. fitting and analysis of spectra within the time period between experimental pulses at tokamaks such as JET. This means that FFS can provide useful data that can, in turn, be used to direct the setup of subsequent pulses in the experimental session — an important practical consideration in a scenario where access for experiments is time-restricted due to high demand.

In addition to the flexible modelling framework, a custom fitting code has been developed to exploit such models for analysis of data (Section 4.9) and further scripts are in place that allow automated fitting of spectra — providing spatial or temporal profiles of fit parameters (Section 4.10). Finally, the framework has been taken through to full application for a range of spectra resulting from a number of different plasma scenarios and therefore confronted with spectral signatures that require models of various degrees of complexity (Chapter 5). The first experimental scenario considered (SOHO-CDS spectra — see Section 5.2) demonstrated that FFS is able to replicate the functionality of an existing dedicated fitting code ADAS602. This included providing an alternative method of dealing with an irregular instrument function problem, without any additional coding — the MDL was flexible enough to provide a solution using the coupling system. FFS was then used to examine a quite different set of spectra; visible-region spectra sourced from the JET divertor (Section 5.3). In this study, the coupling system again proved useful for coupling deuterium Balmer-series line widths to the parameter of interest — electron density. This improves upon previous analysis [74] in which physical parameters are inferred, retrospectively, from fits that have determined line shape parameters. Instead, FFS allows for direct inference of the physical parameter and is determined by the best fit to the entire connected feature; rather than individual lines. The divertor detachment study also involved use of the batch fitting scripts and demonstrated how FFS can be used to build the aforementioned spatial and temporal profiles of parameters. The chapter also displays the results of the use of special feature models in FFS (accessed via AFG) for Zeeman features, diatomic molecular emission, V/UV impurity spectra and He-like satellite lines in the x-ray region (Sections, 5.4, 5.5, 5.6 and 5.7 respectively). The initial fit of a single Zeeman feature demonstrates the ability of FFS to replace another dedicated fitting code ADAS603. The example study then moves past the capabilities of ADAS603 and uses two ADAS Zeeman special features simultaneously model two

overlapping features observed along the line of sight. Again, the number of free parameters is reduced by use of the coupling system which, in this case, maintains a similar ratio of the π and σ polarised components across the two features. The diatomic molecular example attempts to re-examine the data considered in previous work [42]. Whereas this study used the CALCAT program to establish a more qualitative piece of analysis, effort is made here to obtain more quantitative results using FFS in conjunction with an AFG-CALCAT feature — with some success. The V/UV impurity spectra and satellite line analysis are both examples of the range of features that are immediately available for fitting with FFS due to the access to ADF15 archive files (photon emissivity coefficients) via AFG. It is important to note that although there is a difference in complexity of the models required in these cases, the end product (the ADF15 file) is a standardised format and so, can be used in a similar way.

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 astrophysical or fusion spectra — a new special feature model is easily integrated by means of a lightweight AFG wrapper to an external modelling code.

This is not to say that the development of the modelling and fitting framework is complete. To reach its full potential, there are several key areas that require attention, some of which are more practical concerns (i.e. software related) while others stem directly from experimental requirements.

Firstly, there is the issue of bringing the various FFS software elements together to form a complete user-friendly package, most likely GUI driven. Secondly, there were some issues with the model definition language that were highlighted during confrontation with experiment; mostly irritations rather than genuine flaws. Of these, one of the most apparent was that if a model contains a number of model elements of the same type, a user would spend a lot of time typing very similar commands to set the values, limits and so on, for each — inclusion of array-like syntax would alleviate this problem. Another MDL issue was identified during the analysis of 'post-loss' SOHO data (Section 5.2). In this case it was desirable to define a new line shape which was a composite of the supplied basis functions. Currently, this requires excessive repetition of the definition for the composite shape and manually written coupling statements. It would be far better to allow for something akin to Lisp macros; that is to say, allow for on the fly element definition, without the need for a new FFS-AFG module. However, attention is drawn to the fact that this is an extension to the MDL and does not require redesign. Finally, there is a problem, not yet handled by FFS, driven by the experimental data; line-of-sight issues. There are two main concerns in

this regard. Firstly, there is the fact that, typically, any given spectrum is the result of emission from the plasma atoms along that line of sight i.e. non-local. In some cases, such as the high-n Balmer series recombination feature considered in Section 5.3, it is reasonable to assume that the emission is occurring over a small length along the line of sight. Further to this, again using the current FFS capabilities, it is possible to model something like the high/low field Zeeman-split signature considered in Section 5.4, where the emission is coming from two distinct regions, by use of a duplicate (overlapping) feature for the second region. In general, however, it is not suitable to make such assumptions and/or the use of multiple 'cells' to model the full line of sight becomes cumbersome and ultimately unworkable. Ideally, one would allow for a further functional form that would define how a given model representation would be integrated over the line of sight. The second issue related to line-of sight, is the ability to perform parallel fitting of spectral frames from multiple lines-of-sight. One could envisage a construct that allows for connections to be made across several tracks (i.e. data from different radial positions) via an additional function that models the plasma transport across this region and everything is fit simultaneously. Substantial additional infrastructure is required to facilitate this type of functionality into FFS, but the result would realise the full potential of the current system.

In addition to the direct improvements that can be made to the FFS software, there is a great deal that can be explored with the system in its current form. Certainly, there is further work that will be enabled by use of the enhanced satellite line feature model (2.7), taking it beyond the preliminary demonstration shown in Section 5.7. However, there is also further breadth in the range of special feature analysis possible with FFS, using the features provided by AFG. Modelling of emission from heavy species (Section 2.6) is already relevant for machines like *ASDEX-U* (all tungsten PFCs) and will shortly be important for *JET* which will feature a tungsten divertor when operations resume in 2011. This upgrade has, of course, been made with a view towards *ITER* which will also exhibit tungsten components, so there is some mileage in such studies. FFS-AFG will also be relevant in providing special feature modelling of the motional Stark effect seen from the beams. As discussed in Section 2.2, an enhanced version of the ADAS305 code will be available via AFG in due course. There is also interest in creating a new special feature model for Balmer/Paschen series emission. A preliminary model, combining ADAS population modelling with advanced Stark broadening routines already exists. A complete solution would also incorporate a model to handle merging between resolved spectral lines and the continuum. This type of model is relevant for *JET* divertor spectra and would be an advancement compared with the demonstrative study shown in Section 5.3 which used

Voigt line shapes. This feature could be used in support of analysis for recombination spectra studies at other fusion devices, e.g. *MAST*, where a current upgrade program will include the installation of a new *Super-X* divertor configuration.

It is hoped that the flexibility and breadth of coverage of the modelling package developed here will make it an effective, favoured tool for spectral analysis of astrophysical and fusion plasmas.

Bibliography

[1] J. Wesson. 'The Science of JET'. Technical Report JETR(99)13, EFDA-JET (1999)

[2] V. Kashyap and J. J. Drake. 'PINTofALE: Package for the interactive analysis of line emission'. Bull. Astron. Soc. India, **28** (2000) 475–476. ADS: http://adsabs.harvard.edu/abs/2000BASI...28..475K

[3] E. Landi, G. D. Zanna, P. R. Young, K. P. Dere, H. E. Mason and M. Landini. 'Atomic Database for Emission Lines. VII. New Data for X-Rays and Other Improvements'. Astrophys. J. Suppl. Ser., **162**(1) (2006) 261–280. doi:10.1086/498148

[4] R. K. Smith, N. S. Brickhouse, D. A. Liedahl and J. C. Raymond. 'Standard Formats for Atomic Data: the APED'. In 'ASP Conference Series', volume 247, 159. Astronomical Society of the Pacific (2001)

[5] J. S. Kaastra, R. Mewe and H. Nieuwenhuijzen. 'SPEX: A New Code for Spectral Analysis of X and UV Spectra'. In 'Proceedings of the Eleventh Colloquium on UV and X-Ray Spectroscopy of Astrophysical and Laboratory Plasmas held on May 29– June 2, 1995, Nagoya, Japan.', 411–416. Univ. Acad. Press. (1996)

[6] A. D. Whiteford, M. G. von Hellermann, L. D. Horton and K.-D. Zastrow. 'CXSFIT — User Manual'. Technical Report ADAS-R(07)01, ADAS (2007). Available from: http://www.adas.ac.uk/notes/adas_r07-01.pdf

[7] M. G. von Hellermann, R. Barnsley, W. Biel, E. Delabie, N. Hawkes, R. Jaspers, D. Johnson, F. Klinkhamer, O. Lischtschenko, O. Marchuk, B. Schunke, M. Singh, B. Snijders, H. P. Summers, D. Thomas, S. Tugarinov and P. Vasu. 'Active beam spectroscopy for ITER'. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, **623**(2) (2010) 720–725. doi:10.1016/j.nima.2010.04.011

[8] H. P. Summers (2007). Atomic Data and Analysis Structure User Manual. Available from: http://www.adas.ac.uk

[9] D. W. Fanning. *IDL Programming Techniques*. Fanning Software Consulting (2000). ISBN 978-0966238327

[10] J. D. Hey, Y. T. Lie, D. Rusbüldt and E. Hintz. 'Doppler Broadening and Magnetic Field Effects on Some Ion Impurity Spectra Emitted in the Boundary Layer of a Tokamak Plasma'. Contrib. Plasma Phys., **34**(6) (1994) 725–747. doi:10.1002/ctpp.2150340605

[11] J. D. Hey, C. C. Chu and P. Mertens. 'Zeeman Spectroscopy as a Tool for Studying Atomic Processes in Edge Plasmas.' Contrib. Plasma Phys., **42** (2002) 635–644. doi:10.1002/1521-3986(200211)42:6/7<635::AID-CTPP635>3.0.CO;2-M

[12] F. R. Meysman, J. J. Middelburg, P. M. J. Herman and C. H. R. Heip. 'Reactive transport in surface sediments. I. Model complexity and software quality'. Comput. Geosci., **29**(3) (2003) 291–300. doi:10.1016/S0098-3004(03)00006-2

[13] F. R. Meysman, J. J. Middelburg, P. M. J. Herman and C. H. R. Heip. 'Reactive transport in surface sediments. II. Media: an object-oriented problem-solving environment for early diagenesis'. Comput. Geosci., **29**(3) (2003) 301–318. doi:10.1016/S0098-3004(03)00007-4

[14] J. Järvi. 'Object-oriented model for partially separable functions in parameter estimation'. Acta Cybernetica, **14**(2) (1999) 285–302. Available from: http://parasol.cs.tamu.edu/~jarvi/papers/acta99.ps

[15] G. Booch, R. A. Maksimchuk, M. W. Engel, B. J. Young, J. Conallen and K. A. Houston. *Object-oriented Analysis and Design with Applications (third edition)*. Addison Wesley (2007). ISBN 978-0201895513

[16] K. A. Arnaud. 'XSPEC: The First Ten Years'. In 'Astronomical Data Analysis Software and Systems V', volume 101 of *Astron. Soc. Pac. Conf. Ser.*, 17 (1996). Available from: http://adsabs.harvard.edu/abs/1996ASPC..101...17A

[17] B. Dorman and K. A. Arnaud. 'Redesign and Reimplementation of XSPEC'. In F. R. Harnden Jr., F. A. Primini, & H. E. Payne (Editor), 'Astronomical Data Analysis Software and Systems X', volume 238 of *Astron. Soc. Pac. Conf. Ser.*,

415–+ (2001). ADS: http://adsabs.harvard.edu/abs/2001ASPC..238..415D

[18] D. W. Marquardt. 'An Algorithm for Least-Squares Estimation of Nonlinear Parameters'. J. Soc. Indust. Appl. Math., **11**(2) (1963) 431–441

[19] F. James and M. Roos. 'Minuit - a system for function minimization and analysis of the parameter errors and correlations'. Comput. Phys. Commun., **10**(6) (1975) 343–367. ISSN 0010-4655. doi:10.1016/0010-4655(75)90039-9

[20] F. James. 'MINUIT Function Minimization and Error Analysis Reference Manual v. 94.1'. CERN Program Library Long Writeup D506, CERN (1998). Available from: http://tevewwg.fnal.gov/tools/minuit.ps

[21] S. Kirkpatrick, J. C. D. Gelatt and M. P. Vecchi. 'Optimization by Simulated Annealing'. Science, **220**(4598) (1983) 671–680. doi:10.1126/science.220.4598.671

[22] J. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, 1st mit press edition (1992). ISBN 978-0262581110

[23] D. H. Besset. *Object-oriented Implementation of Numerical Methods: An Introduction with Java and Smalltalk*. Academic Press (2001). ISBN 978-1558606791

[24] B. W. Char, K. O. Geddes, W. M. Gentleman and G. H. Gonnet. 'The design of maple: A compact, portable and powerful computer algebra system'. In 'Computer Algebra', volume 162 of *Lecture Notes in Computer Science*, 101–115. Springer (1983). doi:10.1007/3-540-12868-9_95

[25] B. W. Char, K. O. Geddes and G. H. Gonnet. 'The maple symbolic computation system'. SIGSAM Bull., **17**(3–4) (1983) 31–42. doi:10.1145/1089338.1089344

[26] S. Wolfram. *The MATHEMATICA Book*. Cambridge University Press (1999). ISBN 978-0521643146

[27] P. J. Mohr, B. N. Taylor and D. B. Newell. 'CODATA recommended values of the fundamental physical constants: 2006'. Rev. Mod. Phys., **80**(2) (2008) 633–730. doi:10.1103/RevModPhys.80.633

[28] R. Aymar, P. Barabaschi and Y. Shimomura. 'The ITER design'. Plasma Phys. Control. Fusion, **44**(5) (2002) 519–565. doi:10.1088/0741-3335/44/5/304

[29] W. H. Heintz. 'Determination of the Runge-Lenz Vector'. Am. J. Phys., **42**(12) (1974) 1078–1082. ISSN 0002-9505. doi:10.1119/1.1987941

[30] H. A. Bethe and E. E. Salpeter. *Quantum Mechanics of One- and Two-Electron Atoms*. Dover Publications Inc. (2009). ISBN 978-0486466675 (paperback)

[31] L. D. Landau and E. M. Lifshitz. *Quantum Mechanics (Non-Relativistic Theory: Volume 3)*. Butterworth Heinemann (1981). ISBN 978-0750635394

[32] A. Meigs, W. Fundamenski, C. Jupen, A. Larsen, S. Loch, M. O'Mullane and H. Summers. 'Density and Temperature Measurements in Detached Recombining JET Divertors'. In '27th EPS Conference on Controlled Fusion and Plasma Physics', volume 24B, 1264–1267 (2000). Available from: https://fusion.gat.com/conferences/meetings/eps00/pdf/p3_121.pdf

[33] M. Koubiti, S. Loch, H. Capes, L. Godbert-Mouret, Y. Marandet, A. Meigs, R. Stamm and H. Summers. 'Smooth line merging into the continuum and Stark broadening of deuterium Balmer lines for plasma diagnostics'. J. Quant. Spectrosc. Radiat. Transfer, **81**(1–4) (2003) 265–273. doi:10.1016/S0022-4073(03)00079-7

[34] D. R. Inglis and E. Teller. 'Ionic Depression of Series Limits in One-Electron Spectra'. Astrophys. J., **90** (1939) 439–448. ADS: http://adsabs.harvard.edu/full/1939ApJ....90..439I

[35] H. R. Griem. *Spectral Line Broadening by Plasmas*. Academic Press Inc. (1974). ISBN 978-0123028501

[36] H. R. Griem. *Principles of Plasma Spectroscopy*. Cambridge University Press (1997). ISBN 978-0521455046

[37] E. Oks. *Stark Broadening of Hydrogen and Hydrogenlike Spectral Lines in Plasmas*. Alpha Science (2006). ISBN 1-84265-252-4

[38] B. Talin, A. Calisti, L. Godbert, R. Stamm, R. W. Lee and L. Klein. 'Frequency-fluctuation model for line-shape calculations in plasma spectroscopy'. Phys. Rev. A, **51**(3) (1995) 1918–1928. doi:10.1103/PhysRevA.51.1918

[39] B. Talin, A. Calisti, S. Ferri, M. Koubiti, T. Meftah, C. Moss, L. Mouret, R. Stamm, S. Alexiou, R. W. Lee and L. Klein. 'Ground work supporting the

codes based upon the frequency fluctuation model'. J. Quant. Spectrosc. Radiat. Transfer, **58**(4–6) (1997) 953–964. doi:10.1016/S0022-4073(97)00101-5

[40] K. Behringer. 'Measurement of CH4/CD4 fluxes and of chemical carbon erosion from CH/CD band emission'. J. Nucl. Mater., **176–177** (1990) 606–610. doi:10.1016/0022-3115(90)90114-3

[41] S. Brezinsek, A. Pospieszczyk, M. Stamp, A. Meigs, A. Kirschner, A. Huber and P. Mertens. 'Identification of molecular carbon sources in the JET divertor by means of emission spectroscopy'. J. Nucl. Mater., **337–339** (2005) 1058–1063. doi:10.1016/j.jnucmat.2004.10.114

[42] G. Duxbury, M. F. Stamp and H. P. Summers. 'Observations and modelling of diatomic molecular spectra from JET'. Plasma Phys. Control. Fusion, **40**(3) (1998) 361–370. doi:10.1088/0741-3335/40/3/002

[43] S. Brezinsek, A. Meigs, S. Jachmich, M. Stamp, J. Rapp, R. Felton, R. Pitts, V. Philipps, A. Huber, R. Pugno, G. Sergienko and A. Pospieszczyk. 'The impact of divertor detachment on carbon sources in JET L-mode discharges'. J. Nucl. Mater., **390–391** (2009) 267–273. doi:10.1016/j.jnucmat.2009.01.100

[44] G. Duxbury, M. G. OMullane, H. P. Summers, A. D. Whiteford, A. G. Meigs, M. F. Stamp, K. H. Behringer, S. Brezinsek and JET EFDA contributors. 'Detectability of diatomic tritides on the JET tokamak'. In 'Proceedings of the 31st EPS Conference', volume 28G, P5.175 (2004). London, UK. 28th June – 2nd July. Available from: http://epsppd.epfl.ch/London/pdf/P5_175.pdf

[45] H. M. Pickett. 'The fitting and prediction of vibration-rotation spectra with spin interactions'. J. Mol. Spectrosc., **148**(2) (1991) 371–377. doi:10.1016/0022-2852(91)90393-O

[46] R. Neu, R. Dux, A. Kallenbach, T. Pütterich, M. Balden, J. Fuchs, A. Herrmann, C. Maggi, M. O'Mullane, R. Pugno, I. Radivojevic, V. Rohde, A. Sips, W. Suttrop, A. Whiteford and the ASDEX Upgrade team. 'Tungsten: an option for divertor and main chamber plasma facing components in future fusion devices'. Nucl. Fusion, **45**(3) (2005) 209–218. doi:10.1088/0029-5515/45/3/007

[47] R. Neu, R. Dux, A. Geier, O. Gruber, A. Kallenbach, K. Krieger, H. Maier, R. Pugno, V. Rohde and S. Schweizer. 'Tungsten as plasma-facing material in

ASDEX Upgrade'. Fusion Engineering and Design, **65**(3) (2003) 367–374. ISSN 0920-3796. doi:10.1016/S0920-3796(02)00381-2

[48] T. Pütterich, R. Neu, R. Dux, A. D. Whiteford, M. G. O'Mullane and the ASDEX Upgrade Team. 'Modelling of measured tungsten spectra from ASDEX Upgrade and predictions for ITER'. Plasma Physics and Controlled Fusion, **50**(8) (2008) 085016. doi:10.1088/0741-3335/50/8/085016

[49] A. D. Whiteford. *On the spectral emission of impurity species for diagnostic application to magnetically confined fusion plasmas*. Ph.D. thesis, University of Strathclyde (2004). Available from: http://www.adas.ac.uk/theses/whiteford_thesis.pdf

[50] A. R. Foster. *On the Behaviour and Radiating Properties of Heavy Elements in Fusion Plasmas*. Ph.D. thesis, University of Strathclyde (2008). Available from: http://www.adas.ac.uk/theses/foster_thesis.pdf

[51] M. J. Nelson, R. Barnsley, F. Keenan, H. Meyer, C. A. Bunting, P. G. Carolan, N. J. Conway, G. Cunningham, I. Lehane and M. R. Tournianski. 'A high-resolution soft x-ray spectrometer on the MAST tokamak'. Rev. Sci. Instrum., **75**(10) (2004) 3734–3736. doi:10.1063/1.1781373

[52] A. H. Gabriel. 'Dielectronic satellite spectra for highly-charged helium-like ionlines'. Mon. Not. R. Astr. Soc., **160** (1972) 99–119. ADS: http://adsabs.harvard.edu/abs/1972MNRAS.160...99G

[53] N. R. Badnell, M. G. O'Mullane, H. P. Summers, Z. Altun, M. A. Bautista, J. Colgan, T. W. Gorczyca, D. M. Mitnik, M. S. Pindzola and O. Zatsarinny. 'Dielectronic recombination data for dynamic finite-density plasmas I. Goals and methodology'. Astron. Astrophys., **406**(3) (2003) 1151–1165. doi:10.1051/0004-6361:20030816

[54] A. D. Whiteford, N. R. Badnell, C. P. Ballance, M. G. O'Mullane, H. P. Summers and A. L. Thomas. 'A radiation-damped R-matrix approach to the electron-impact excitation of helium-like ions for diagnostic application to fusion and astrophysical plasmas'. J. Phys. B, **34**(15) (2001) 3179–3191. doi:10.1088/0953-4075/34/15/320

[55] P. Burke and K. A. Berrington. *Atomic and Molecular Processes: An R-Matrix Approach*. Institure of Physics Publishing (1993). ISBN 978-0750301992

[56] D. C. Griffin, N. R. Badnell and M. S. Pindzola. 'R-matrix electron-impact excitation cross sections in intermediate coupling: an MQDT transformation approach'. J. Phys. B, **31**(16) (1998) 3713–3727. doi:10.1088/0953-4075/31/16/022

[57] F. Robicheaux, T. W. Gorczyca, M. S. Pindzola and N. R. Badnell. 'Inclusion of radiation damping in the close-coupling equations for electron-atom scattering'. Phys. Rev. A, **52**(2) (1995) 1319–1333. doi:10.1103/PhysRevA.52.1319

[58] T. W. Gorczyca and N. R. Badnell. 'Radiation damping in highly charged ions: an R-matrix approach'. J. Phys. B, **29**(7) (1996) L283–L290. doi:10.1088/0953-4075/29/7/007

[59] T. W. Gorczyca and N. R. Badnell. 'Quantum defect theory with deeply closed channels'. J. Phys. B, **33**(13) (2000) 2511–2523. doi:10.1088/0953-4075/33/13/311

[60] N. R. Badnell and D. C. Griffin. 'Electron-impact excitation of Fe 20+, including n=4 levels'. J. Phys. B, **34**(4) (2001) 681–697. doi:10.1088/0953-4075/34/4/316

[61] N. R. Badnell. 'On the effects of the two-body non-fine-structure operators of the Breit-Pauli Hamiltonian'. J. Phys. B, **30**(1) (1997) 1–11. doi:10.1088/0953-4075/30/1/005

[62] A. D. Whiteford, N. R. Badnell, C. P. Ballance, S. D. Loch, M. G. O'Mullane and H. P. Summers. 'Excitation of Ar 15+ and Fe 23+ for diagnostic application to fusion and astrophysical plasmas'. J. Phys. B, **35**(17) (2002) 3729–3740. doi:10.1088/0953-4075/35/17/309

[63] N. R. Badnell and M. J. Seaton. 'Quantum defect theory with long-range multipole potentials'. J. Phys. B, **32**(15) (1999) 3955–3964. doi:10.1088/0953-4075/32/15/323

[64] K. Nygaard and O.-J.Dahl. 'The development of the SIMULA languages'. In R. L. Wexelblat (Editor), 'History of programming languages I', 439–480. ACM, New York, NY, USA (1981). ISBN 0-12-745040-8. doi:10.1145/800025.1198392

[65] J. McCarthy. 'Recursive functions of symbolic expressions and their computation by machine, part I'. Commun. Assoc. Comp. Mach., **3**(4) (1960) 184–195. doi:10.1145/367177.367199

[66] P. R. Bevington and D. K. Robinson. *Data Analysis and Error Reduction for the Physical Sciences*. McGraw-Hill Higher Education (2002). ISBN 978-0071199261 (paperback)

[67] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing: 3rd Edition*. Cambridge University Press (2007). ISBN 978-0521880688

[68] C. B. Markwardt. 'Non-linear Least-squares Fitting in IDL with MPFIT'. In D. A. Bohlender, D. Durand and P. Dowler (Editors), 'Astronomical Society of the Pacific Conference Series', volume 411 of *Astron. Soc. Pac. Conf. Ser.*, 251 (2009). ADS: http://adsabs.harvard.edu/abs/2009ASPC..411..251M

[69] J. J. Moré, B. S. Garbow and K. E. Hillstrom. 'User Guide for MINPACK-1'. Technical report, Argonne National Laboratory (1980). Available from: http://www.mcs.anl.gov/~{}more/ANL8074a.pdf

[70] D. H. Brooks, G. A. Fischbacher, A. Fludra, R. A. Harrison, D. E. Innes, E. Landi, M. Landini, J. L. A. C. Lanzafame, S. D. Loch, R. W. P. McWhirter, H. P. Summers and W. T. Thompson. 'The quiet Sun extreme ultraviolet spectrum observed in normal incidence by the SOHO coronal diagnostic spectrometer'. Astron. Astrophys., **347** (1999) 277–312. ADS: http://adsabs.harvard.edu/abs/1999A%26A...347..277B

[71] B. Fleck and Z. Svestka. *The First Results from SOHO*. Springer (1998). ISBN 978-0792348825 (1998 hardback)

[72] W. T. Thompson. 'Post-recovery broadened line profiles'. CDS Software Note 53, NASA Goddard Flight Center (1999). Available from: http://solar.bnsc.rl.ac.uk/swnotes/cds_swnote_53.pdf

[73] A. Boboc, M. Gelfusa, A. Murari, P. Gaudio and J.-E. Contributors. 'Recent developments of the JET far-infrared interferometer-polarimeter diagnostic'. Rev. Sci. Instrum., **81**(10) (2010) 10D538. doi:10.1063/1.3478146

[74] A. G. Meigs, G. M. McCracken, C. Maggi, R. D. Monk, L. D. Horton, M. von Hellermann, M. F. Stamp and P. Breger. 'Spectroscopic Electron Density Measurements and Evidence of Recombination in High Density JET Divertor Discharges'. In 'Proceedings of the 1998 ICPP & 25th EPS Conference on Plasma Physics and Controlled Fusion', volume 22C, 373–376 (1998). Available from: http://epsppd.epfl.ch/Praha/WEB/98ICPP_W/B161PR.PDF

[75] J. Gafert, W. Fundamenski, M. Stamp, J. D. Strachan and JET EFDA Contributors. 'Distribution of Carbon Impurity Sources Between Low and High Field Side Measured via Zeeman-Spectroscopy in JET'. In 'Proceedings of the 28th EPS Conference on Contr. Fusion and Plasma Phys.', volume 25A, 1637–1640 (2001). Funchal, Portugal. 18th June – 22nd June. Available from: http://www.iop.org/Jet/fulltext/EFDC010254.PDF

[76] R. Pugno, A. Kallenbach, D. Bolshukhin, R. Dux, J. Gafert, R. Neu, V. Rohde, K. Schmidtmann, W. Ullrich and U. Wenzel. 'Spectroscopic investigation on the impurity influxes of carbon and silicon in the ASDEX upgrade experiment'. J. Nucl. Mater., **290–293** (2001) 308–311. ISSN 0022-3115. doi:10.1016/S0022-3115(00)00633-4

[77] D. Nishijima, R. P. Doerner, M. J. Baldwin, G. D. Temmerman and E. M. Hollmann. 'Properties of BeD molecules in edge plasma relevant conditions'. Plasma Phys. Control. Fusion, **50**(12) (2008) 125007. doi:10.1088/0741-3335/50/12/125007

[78] A. Kirschner, D. Borodin, S. Droste, V. Philipps, U. Samm, G. Federici, A. Kukushkin and A. Loarte. 'Modelling of tritium retention and target lifetime of the ITER divertor using the ERO code'. J. Nucl. Mater., **363–365** (2007) 91–95. doi:10.1016/j.jnucmat.2007.01.002

[79] O. Neubauer, G. Czymek, B. Giesen, P. W. Hüttemann, M. Sauer, W. Schalt and J. Schruff. 'Design features of the tokamak TEXTOR'. Fusion Sci. Technol., **47** (2005) 76–86. Available from: http://juwel.fz-juelich.de:8080/dspace/bitstream/2128/2735/1/63286.pdf

[80] O. Marchuk, G. Bertschinger, H.-J. Kunze, N. R. Badnell and S. Fritzsche. 'Cascades between doubly excited levels in helium-like argon'. J. Phys. B: At. Mol. Opt. Phys., **37**(9) (2004) 1951. doi:10.1088/0953-4075/37/9/014

[81] J. Weinheimer, I. Ahmad, O. Herzog, H.-J. Kunze, G. Bertschinger, W. Biel, G. Borchert and M. Bitter. 'High-resolution x-ray crystal spectrometer/polarimeter at torus experiment for technology oriented research-94'. Rev. Sci. Instrum., **72**(6) (2001) 2566–2574. doi:10.1063/1.1370558

[82] E. B. Saloman. 'Energy Levels and Observed Spectral Lines of Ionized Argon, Ar II through Ar XVIII'. J. Phys. Chem. Ref. Data, **39**(3) (2010) 033101. doi:10.1063/1.3337661

[83] P. Heinzel. ‘Derivatives of the Voigt functions’. Bulletin of the Astronomical Institutes of Czechoslovakia, **29** (1978) 159–162. ADS: http://adsabs.harvard.edu/abs/1978BAICz..29..159H

[84] R. J. Wells. ‘Rapid approximation to the Voigt/Faddeeva function and its derivatives’. J. Quant. Spectrosc. Radiat. Transfer, **62**(1) (1999) 29–48. doi:10.1016/S0022-4073(97)00231-8

Appendix A

Mathematical Notes

A.1 Convolution — Definition and Basic Properties

The definition of the convolution of two functions, f and g is given as:

$$[f * g](x) = \int_{-\infty}^{+\infty} f(x')g(x - x')dx' \tag{A.1}$$

Convolution has several mathematical properties which can be easily proven using the integral definition. Among these the following are useful:

$$a * b = b * a \qquad \text{(commutative)} \tag{A.2}$$

$$(a * b) * c = a * (b * c) \qquad \text{(associative)} \tag{A.3}$$

$$a * (b + c) = (a * b) + (a * c) \qquad \text{(distributive)} \tag{A.4}$$

$$\frac{\partial}{\partial x}(a * b) = \frac{\partial a}{\partial x} * b = a * \frac{\partial b}{\partial x} \qquad \text{(derivative)} \tag{A.5}$$

A.2 Area of Convolved Functions

The area under a convolution integral is simply the product of the areas of the functions undergoing convolution:

$$\begin{aligned} A_{f*g} &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x')g(x - x')dx'dx \\ &= \int_{-\infty}^{+\infty} f(x') \left[\int_{-\infty}^{+\infty} g(x - x')dx \right] dx' \end{aligned}$$

let $y = x - x'$

$$
\begin{aligned}
&= \int_{-\infty}^{+\infty} f(x') \left[\int_{-\infty}^{+\infty} g(y) dy \right] dx' \\
&= A_g \int_{-\infty}^{+\infty} f(x') dx' \\
&= A_g A_f
\end{aligned} \tag{A.6}
$$

A.3 Raw Moments of Convolved Functions

The (normalized) n^{th} raw moment of a function f is denoted by:

$$
< x^n >_f = \frac{\int x^n f(x) dx}{\int f(x) dx} \tag{A.7}
$$

A.3.1 First Raw Moment

The first raw moment (centroid) of the convolution of the functions f and g i.e. $f * g$ is given by:

$$
\begin{aligned}
< x >_{f*g} &= \frac{\int x[f * g](x) dx}{\int [f * g](x) dx} \\
&= \frac{1}{A_g A_f} \int x \left[\int f(x') g(x - x') dx' \right] dx \\
&= \frac{1}{A_g A_f} \int f(x') \left[\int x g(x - x') dx \right] dx'
\end{aligned}
$$

let $t = x - x' \implies \frac{dt}{dx} = 1$

$$
\begin{aligned}
&= \frac{1}{A_g A_f} \int f(x') \left[\int (t + x') g(t) dt \right] dx' \\
&= \frac{1}{A_g A_f} \int f(x') \left[\int t g(t) dt + x' \int g(t) dt \right] dx' \\
&= \frac{1}{A_g A_f} \int f(x') \left[A_g < x >_g + x' A_g \right] dx' \\
&= \frac{1}{A_g A_f} \left[< x >_g A_g A_f + < x >_f A_f A_g \right] \\
&= < x >_g + < x >_f
\end{aligned} \tag{A.8}
$$

A.3.2 Second Raw Moment

The second raw moment of $f * g$ is:

$$
\begin{aligned}
< x^2 >_{f*g} &= \frac{\int x^2 [f * g](x) dx}{\int [f * g](x) dx} \\
&= \frac{1}{A_g A_f} \int x^2 \left[\int f(x') g(x - x') dx' \right] dx \\
&= \frac{1}{A_g A_f} \int f(x') \left[\int x^2 g(x - x') dx \right] dx' \\
\text{let } t = x - x' \implies \tfrac{dt}{dx} &= 1 \\
&= \frac{1}{A_g A_f} \int f(x') \left[\int (t + x')^2 g(t) dt \right] dx' \\
&= \frac{1}{A_g A_f} \int f(x') \\
&\quad \left[\int t^2 g(t) dt + 2x' \int t' g(t) dt + (x')^2 \int g(t) dt \right] dx' \\
&= \frac{1}{A_g A_f} \int f(x') \left[A_g < x^2 >_g + 2x' A_g < x >_g + (x')^2 A_g \right] dx' \\
&= \frac{1}{A_g A_f} [< x^2 >_g A_g A_f + \\
&\quad 2 < x >_f < x >_g A_g A_f + < x^2 >_f A_g A_f] \\
&= < x^2 >_g + 2 < x >_f < x >_g + < x^2 >_f
\end{aligned}
\tag{A.9}
$$

A.4 Gaussian Line Widths

The normal distribution function is given by:

$$
f(x) = \frac{1}{\sigma \sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2} \right) \tag{A.10}
$$

We can define a characteristic 'half width at half maximum' by considering the value of the variate, x when the function value is half of its maximum value. The constant

factor can be ignored and so,

$$\exp\left(\frac{-x^2}{2\sigma^2}\right) = \frac{1}{2} \tag{A.11}$$

$$\frac{-x^2}{2\sigma^2} = \ln\frac{1}{2} \tag{A.12}$$

$$x^2 = 2\sigma^2 \ln 2 \tag{A.13}$$

$$x = \pm\sqrt{2\ln 2}\sigma \tag{A.14}$$

such that the half width at half maximum is:

$$x_{\mathrm{HWHM}} = \sqrt{2\ln 2}\sigma \tag{A.15}$$

It immediately follows that the full width at half maximum is:

$$x_{\mathrm{FWHM}} = 2\sqrt{2\ln 2}\sigma \tag{A.16}$$

It is also possible to define this function in terms of a '$\frac{1}{e}$–width', which we can be determined in a similar way:

$$\exp\left(\frac{-x^2}{2\sigma^2}\right) = \exp(-1) \tag{A.17}$$

$$x^2 = 2\sigma^2 \tag{A.18}$$

$$x = \pm\sqrt{2}\sigma \tag{A.19}$$

So, the $\frac{1}{e}$–half-width and $\frac{1}{e}$–width are:

$$x_{\frac{1}{e}\text{-half}} = \sqrt{2}\sigma \tag{A.20}$$

$$x_{\frac{1}{e}} = 2\sqrt{2}\sigma \tag{A.21}$$

In this work, we will typically use the full width at half maximum when referring to the width of a Gaussian line, so we will use equation A.16 to define the normal distribution function as:

$$\begin{aligned} f(x) &= \frac{2\sqrt{\ln 2}}{\sqrt{\pi}w_g}\exp\left(-\frac{4\ln 2x^2}{w_g^2}\right) \\ &= \frac{C}{\sqrt{\pi}w_g}\exp\left(-\frac{C^2x^2}{w_g^2}\right) \end{aligned} \tag{A.22}$$

where $C = 2\sqrt{\ln 2}$ and w_g is the full width at half maximum

A.5 Delta Function

The delta function is defined by the following:

$$\delta(x) = \begin{cases} 0 & x \neq 0 \\ \infty & x = 0 \end{cases} \tag{A.23}$$

$$\int_{-\infty}^{\infty} \delta(x)dx = 1 \tag{A.24}$$

This means that the delta function exhibits the so called 'sifting' property:

$$\int_{-\infty}^{\infty} f(x)\delta\left(x - x_0\right)dx = f(x_0) \tag{A.25}$$

A.6 Error Function

A.6.1 Definition

The error function is defined as:

$$\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \tag{A.26}$$

A.7 Complementary Error Function

A.7.1 Definition

The complementary error function is defined as:

$$
\begin{aligned}
\operatorname{erfc}(z) &= 1 - \operatorname{erf}(z) \\
&= 1 - \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \\
&= 1 - \frac{2}{\sqrt{\pi}} \left(\int_0^\infty e^{-t^2} dt - \int_z^\infty e^{-t^2} dt \right) \\
&= 1 - \operatorname{erf}(\infty) + \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-t^2} dt \\
&= \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-t^2} dt
\end{aligned} \tag{A.27}
$$

By change of variable and subsequently integrating by parts, we can acquire series representation of erfc:

$$
u = t^2 \quad ; \quad \frac{1}{2t} du = dt \quad ; \quad t = z \implies u = z^2 \quad ; \quad t \to \infty \implies u \to \infty \tag{A.28}
$$

$$
\begin{aligned}
\operatorname{erfc}(z) &= \frac{1}{\sqrt{\pi}} \int_{z^2}^{\infty} u^{-\frac{1}{2}} e^{-u} du \\
&= \frac{1}{\sqrt{\pi}} \left(\left[-u^{\frac{1}{2}} e^{-u} \right]_{z^2}^{\infty} - \frac{1}{2} \int_{z^2}^{\infty} u^{-\frac{3}{2}} e^{-u} du \right) \\
&= \frac{1}{\sqrt{\pi}} \left(z^{-1} e^{-z^2} - \frac{1}{2} \left(\left[-u^{-\frac{3}{2}} e^{-u} \right]_{z^2}^{\infty} - \frac{3}{2} \int_{z^2}^{\infty} u^{-\frac{5}{2}} e^{-u} du \right) \right) \\
&= \frac{1}{\sqrt{\pi}} \left(z^{-1} e^{-z^2} - \frac{1}{2} \left(z^{-3} e^{-z^2} \right. \right. \\
&\qquad \left. \left. - \frac{3}{2} \left(\left[-u^{-\frac{5}{2}} e^{-u} \right]_{z^2}^{\infty} - \frac{5}{2} \int_{z^2}^{\infty} u^{-\frac{7}{2}} e^{-u} du \right) \right) \right) \\
&= \frac{1}{\sqrt{\pi}} \left(z^{-1} e^{-z^2} - \frac{1}{2} \left(z^{-3} e^{-z^2} \right. \right. \\
&\qquad \left. \left. - \frac{3}{2} \left(z^{-5} e^{-z^2} - \frac{5}{2} \left(\left[-u^{-\frac{7}{2}} e^{-u} \right]_{z^2}^{\infty} - \int_{z^2}^{\infty} u^{-\frac{9}{2}} e^{-u} du \right) \right) \right) \right) \\
&= \frac{1}{\sqrt{\pi}} \left(z^{-1} e^{-z^2} - \frac{1}{2} \left(z^{-3} e^{-z^2} \right. \right. \\
&\qquad \left. \left. - \frac{3}{2} \left(z^{-5} e^{-z^2} - \frac{5}{2} \left(-z^{-7} e^{-z^2} - \dots \right) \right) \right) \right) \\
&= \frac{1}{\sqrt{\pi}} e^{-z^2} \left(\left(\frac{1}{z} \right) - \frac{1}{2} \left(\frac{1}{z^3} \right) \right. \\
&\qquad \left. + \frac{1 \times 3}{2^2} \left(\frac{1}{z^5} \right) - \frac{1 \times 3 \times 5}{2^3} \left(\frac{1}{z^7} \right) - \dots \right) \\
&= \frac{1}{\sqrt{\pi}} e^{-z^2} \left(\frac{1}{z} + \sum_{n=1}^{\infty} (-1)^n \left(\frac{1}{z^{2n+1}} \right) \left(\frac{1}{2^n} \right) \prod_{i=1}^{n} 2i - 1 \right)
\end{aligned}
\tag{A.29}
$$

A.7.2 Derivative

For clarity, first make the following assignment:

$$
A(z) = \sum_{n=1}^{\infty} (-1)^n \left(\frac{1}{z^{2n+1}} \right) \left(\frac{1}{2^n} \right) \prod_{i=1}^{n} 2i - 1 \tag{A.30}
$$

such that

$$
\operatorname{erfc}(z) = \frac{1}{\sqrt{\pi}} e^{-z^2} \left(\frac{1}{z} + A(z) \right) \tag{A.31}
$$

Now taking the derivative of erfc:

$$\begin{aligned}\frac{\partial}{\partial z}\text{erfc}(z) &= \frac{1}{\sqrt{\pi}}\left(-2e^{-z^2} - z^{-2}e^{-z^2} + e^{-z^2}B(z) + e^{-z^2}C(z)\right) \\ &= \frac{1}{\sqrt{\pi}}e^{-z^2}\left(-2 - z^{-2} + B(z) + C(z)\right)\end{aligned} \tag{A.32}$$

where the following substitutions have used:

$$\begin{aligned}B(z) &= -2zA(z) \\ &= \sum_{n=1}^{\infty}(-1)^{n+1}\left(\frac{2^{1-n}}{z^{2n}}\right)\prod_{i=1}^{n}2i-1\end{aligned} \tag{A.33}$$

and

$$\begin{aligned}C(z) &= \frac{\partial}{\partial z}A(z) \\ &= -(2n+1)\frac{A(z)}{z} \\ &= \sum_{n=1}^{\infty}(-1)^{n+1}(2n+1)\left(\frac{2^{-n}}{z^{2n+2}}\right)\prod_{i=1}^{n}2i-1\end{aligned} \tag{A.34}$$

The first term of the summation in eqn A.33 can be extracted and so, re-written in terms of C as follows:

$$B = \frac{1}{z^2} + \sum_{n=2}^{\infty}(-1)^{n+1}\left(\frac{2^{1-n}}{z^{2n}}\right)\prod_{i=1}^{n}2i-1$$

let $j = n-1$

$$= \frac{1}{z^2} + \sum_{j=1}^{\infty}(-1)^{j+2}\left(\frac{2^{-j}}{z^{2(j+1)}}\right)\prod_{i=1}^{j+1}2i-1$$

now extracting the last term of the product,

$$\begin{aligned}&= \frac{1}{z^2} + \sum_{j=1}^{\infty}(-1)^{j+2}(2j+1)\left(\frac{2^{-j}}{z^{2j+2}}\right)\prod_{i=1}^{j}2i-1 \\ &= \frac{1}{z^2} - C\end{aligned} \tag{A.35}$$

Substitution of this result into eqn A.32 yields:

$$\frac{\partial}{\partial z}\left(\text{erfc}(z)\right) = -\frac{2}{\sqrt{\pi}}e^{-z^2} \tag{A.36}$$

A.8 Complex Error Function

A.8.1 Definition

$$w(z) = e^{-z^2}\text{erfc}(-iz) \tag{A.37}$$

$$= e^{-z^2}\left(1 + \frac{2i}{\sqrt{\pi}}\int_0^z e^{t^2}dt\right) \tag{A.38}$$

The complex error function can be represented as:

$$w(z) = K(a,b) + iL(a,b) \tag{A.39}$$

The real part of the complex error function is given as:

$$K(a,b) = \frac{b}{\pi}\int_{-\infty}^{+\infty}\frac{e^{-t^2}}{(a-t)^2 + b^2}dt \tag{A.40}$$

and the imaginary part:

$$L(a,b) = \frac{1}{\pi}\int_{-\infty}^{+\infty}\frac{e^{-t^2}(a-t)}{(a-t)^2 + b^2}dt \tag{A.41}$$

A.8.2 Derivative

An expanded version of the derivation of Heinzel [83] is presented here. For computational details relevant to this work (i.e. fast calculation of the voigt function, see Wells [84]).

A complex function can be differentiated with respect to the complex variable z in the same fashion as a real function by a real independent variable, if a unique derivative exists, regardless of the path taken in the complex plane as $z \to z + \delta z$. That is to say that if the Cauchy Riemann equations hold, then the complex function is 'complex differentiable'.

If we consider alternative forms of the functions $K(a,b)$ and $L(a,b)$ respectively (obtained by taking the Fourier transforms of the functions, some manipulation, followed by invertion):

$$K(a,b) = \frac{2}{\sqrt{\pi}}\int_0^{\infty} e^{-t^2-2bt}\cos(2at)dt \tag{A.42}$$

and

$$L(a,b) = \frac{2}{\sqrt{\pi}} \int_0^\infty e^{-t^2-2bt} \sin(2at) dt. \tag{A.43}$$

Now, taking the partial derivatives of these functions, with respect to a and b:

$$\frac{\partial}{\partial a} K(a,b) = -\frac{2}{\sqrt{\pi}} \int_0^\infty 2te^{-t^2-2bt} \sin(2at) dt, \tag{A.44}$$

$$\frac{\partial}{\partial b} K(a,b) = -\frac{2}{\sqrt{\pi}} \int_0^\infty 2te^{-t^2-2bt} \cos(2at) dt, \tag{A.45}$$

$$\frac{\partial}{\partial a} L(a,b) = \frac{2}{\sqrt{\pi}} \int_0^\infty 2te^{-t^2-2bt} \cos(2at) dt, \tag{A.46}$$

$$\frac{\partial}{\partial b} L(a,b) = -\frac{2}{\sqrt{\pi}} \int_0^\infty 2te^{-t^2-2bt} \sin(2at) dt. \tag{A.47}$$

So, this shows that:

$$\frac{\partial K(a,b)}{\partial a} = \frac{\partial L(a,b)}{\partial b} \qquad \text{and} \qquad \frac{\partial L(a,b)}{\partial a} = -\frac{\partial K(a,b)}{\partial b} \tag{A.48}$$

i.e. that the Cauchy Riemann equations are satisfied.

We can now safely take the derivative of the function $w(z)$ with respect to the complex variate z (using the result A.36):

$$\begin{aligned} \frac{d}{dz} w(z) &= \operatorname{erfc}(-iz) \frac{d}{dz} e^{-z^2} + e^{-z^2} \frac{d}{dz} (\operatorname{erfc}(-iz)) \\ &= -2ze^{-z^2} \operatorname{erfc}(-iz) + \frac{2i}{\sqrt{\pi}} e^{-z^2} e^{z^2} \\ &= -2zw(z) + \frac{2i}{\sqrt{\pi}}. \end{aligned} \tag{A.49}$$

Substitution of $z = a + ib$ and $w(z) = K(a,b) + iL(a,b)$:

$$\begin{aligned} \frac{d}{dz} w(z) &= -2(a+ib)(K(a,b) + iL(a,b)) + \frac{2i}{\sqrt{\pi}} \\ &= -2\left(aK(a,b) + iaL(a,b) + ibK(a,b) - bL(a,b) - \frac{i}{\sqrt{\pi}}\right) \\ &= 2(bL(a,b) - aK(a,b)) - 2\left(aL(a,b) + bK(a,b) - \frac{1}{\sqrt{\pi}}\right) i. \end{aligned} \tag{A.50}$$

It is possible to also obtain the following expression for $\frac{d}{dz}w(z)$:

$$\begin{aligned}
\frac{d}{dz}w(z) &= \frac{d}{dz}\left(K(a,b) + iL(a,b)\right) \\
&= \frac{d}{dz}K(a,b) + i\frac{d}{dz}L(a,b) \\
&= \left(\frac{\partial}{\partial a}(K(a,b))\frac{\partial a}{\partial z} + \frac{\partial}{\partial b}(K(a,b))\frac{\partial b}{\partial z}\right) \\
&\quad + i\left(\frac{\partial}{\partial a}(L(a,b))\frac{\partial a}{\partial z} + \frac{\partial}{\partial b}(L(a,b))\frac{\partial b}{\partial z}\right) \\
&= \left(\frac{\partial K(a,b)}{\partial a} + \frac{\partial L(a,b)}{\partial b}\right) + i\left(\frac{\partial L(a,b)}{\partial a} - \frac{\partial K(a,b)}{\partial b}\right). \qquad \text{(A.51)}
\end{aligned}$$

Comparing real and imaginary parts of results A.50 and A.51:

$$\mathrm{Re}\left\{\frac{dw(z)}{dz}\right\} = \frac{\partial K(a,b)}{\partial a} + \frac{\partial L(a,b)}{\partial b} = 2b(L(a,b) - aK(a,b)), \qquad \text{(A.52)}$$

$$\mathrm{Im}\left\{\frac{dw(z)}{dz}\right\} = \frac{\partial L(a,b)}{\partial a} - \frac{\partial K(a,b)}{\partial b} = -2\left(aL(a,b) + bK(a,b) - \frac{1}{\sqrt{\pi}}\right). \qquad \text{(A.53)}$$

Appendix B

FFS MDL Listings

The following section comprises the various FFS model definitions used in performing the fits detailed in Section 5.

Listing B.1: Model used to fit the *pre-loss* SOHO data in Section 5.2

```
(model oiiimultiplet

    (+
        (broaden_gauss
            (+  (line l1)
                (line l2)
                (line l3)
                (line l4)
            )
        bg)
        (background-linear backg)
    )
)

(setval backg.c 0.01)
(setval backg.m 0.01)

(setval bg.fwhm 0.3)
(setlimits bg.fwhm 0.1 1.0)

(setval l1.pos 553.307)
(setlimits l1.pos 553.0 554.0)
(setval l1.intensity 3.0)
(setlimits l1.intensity 0.01 100.0)

(setval l2.pos 554.125)
(setlimits l2.pos 554.0 555.0)
(setval l2.intensity 2.5)
(setlimits l2.intensity 0.01 100.0)

(setval l3.pos 554.475)
(setlimits l3.pos 554.0 555.0)
(setval l3.intensity 5.0)
```

```
(setlimits l3.intensity  0.01 100.0)

(setval l4.pos 555.292)
(setlimits l4.pos 555.0 556.0)
(setval l4.intensity 2.5)
(setlimits l4.intensity 0.01 100.0)
```

Listing B.2: Model used to fit the *post-loss* SOHO data in Section 5.2

```
(model oiiimultiplet
    (+
        (broaden_gauss
            (+  (line l1)
                (line l2)
                (line l3)
                (line l4)
            )
        bg)
        (broaden_lorentz
            (+
                (line wingl1)
                (line wingl2)
                (line wingl3)
                (line wingl4)
            )
        bl)
        (background-linear backg)
    )
)

(dummy d)
(setval d.wingfactor 0.51002621)
(fixed d.wingfactor)

(setval d.wingshift  0.41592492)
(fixed d.wingshift)

(couple d.combinedintensity1 = l1.intensity + wingl1.intensity)
(fixed d.combinedintensity1)
(couple d.combinedintensity2 = l2.intensity + wingl2.intensity)
(fixed d.combinedintensity2)
(couple d.combinedintensity3 = l3.intensity + wingl3.intensity)
(fixed d.combinedintensity3)
(couple d.combinedintensity4 = l4.intensity + wingl4.intensity)
(fixed d.combinedintensity4)

(setval bl.fwhm 1.4657422)
(setlimits bl.fwhm 0.1 2.0)
(fixed bl.fwhm)

(setval backg.c 0.01)
(setval backg.m 0.01)

(setval bg.fwhm 0.3)
(setlimits bg.fwhm 0.1 1.0)

(setval l1.pos 553.307)
```

```
(setlimits l1.pos 553.0 554.0)
(setval l1.intensity 10.0)
(setlimits l1.intensity 0.01 100.0)
(couple wingl1.pos = l1.pos + d.wingshift)
(couple wingl1.intensity (* l1.intensity d.wingfactor))

(setval l2.pos 554.125)
(setlimits l2.pos 554.0 555.0)
(setval l2.intensity 20.0)
(setlimits l2.intensity 0.01 100.0)
(couple wingl2.pos = l2.pos + d.wingshift)
(couple wingl2.intensity (* l2.intensity d.wingfactor))

(setval l3.pos 554.475)
(setlimits l3.pos 554.0 555.0)
(setval l3.intensity 20.0)
(setlimits l3.intensity  0.01 100.0)
(couple wingl3.pos = l3.pos + d.wingshift)
(couple wingl3.intensity (* l3.intensity d.wingfactor))

(setval l4.pos 555.192)
(setlimits l4.pos 555.0 555.5)
(setval l4.intensity 10.0)
(setlimits l4.intensity 0.01 100.0)
(couple wingl4.pos = l4.pos + d.wingshift)
(couple wingl4.intensity (* l4.intensity d.wingfactor))
```

Listing B.3: Model for deuterium recombination spectra (Balmer series) observed in the JET divertor (Section 5.3)

```
(model balmer_kt3a_70578

    (+
        (voigtian d10t2)
        (voigtian d11t2)
        (voigtian d12t2)
        (voigtian d13t2)
        (voigtian d14t2)
        (voigtian d15t2)
        (voigtian d16t2)
        (gaussian g1)
        (gaussian g2)
        (gaussian g3)
        (gaussian g4)
        (gaussian g5)
        (gaussian g6)
        (gaussian g7)
        (gaussian oiiia)
        (gaussian oiiib)
        (gaussian oiiic)
        (gaussian be3a)
        (gaussian be3b)
        (gaussian be3c)
        (gaussian g8)
        (background-linear backg)
    )
```

```
)

(dummy d)

(setval backg.m -1.0e9)
(setlimits backg.m -6.0e9 -1.0e7)

(setval backg.c 1.0e10)
(setlimits backg.c 1.0e9 4.0e10)

(setval d.dens 2.0e20)
(log d.dens)
(setlimits d.dens 1.0e19 8.0d20)

(couple d10t2.fwhml (* (/ (* 0.00139 (^ (* d10t2.pos 1e-9) 2)
    (^ d.dens (/ 2.0 3.0)) (- (^ 10.0 2) (^ 2.0 2))) 1.8836516e09) 1e9))
(couple d11t2.fwhml (* (/ (* 0.00139 (^ (* d11t2.pos 1e-9) 2)
    (^ d.dens (/ 2.0 3.0)) (- (^ 11.0 2) (^ 2.0 2))) 1.8836516e09) 1e9))
(couple d12t2.fwhml (* (/ (* 0.00139 (^ (* d12t2.pos 1e-9) 2)
    (^ d.dens (/ 2.0 3.0)) (- (^ 12.0 2) (^ 2.0 2))) 1.8836516e09) 1e9))
(couple d13t2.fwhml (* (/ (* 0.00139 (^ (* d13t2.pos 1e-9) 2)
    (^ d.dens (/ 2.0 3.0)) (- (^ 13.0 2) (^ 2.0 2))) 1.8836516e09) 1e9))
(couple d14t2.fwhml (* (/ (* 0.00139 (^ (* d14t2.pos 1e-9) 2)
    (^ d.dens (/ 2.0 3.0)) (- (^ 14.0 2) (^ 2.0 2))) 1.8836516e09) 1e9))
(couple d15t2.fwhml (* (/ (* 0.00139 (^ (* d15t2.pos 1e-9) 2)
    (^ d.dens (/ 2.0 3.0)) (- (^ 15.0 2) (^ 2.0 2))) 1.8836516e09) 1e9))
(couple d16t2.fwhml (* (/ (* 0.00139 (^ (* d16t2.pos 1e-9) 2)
    (^ d.dens (/ 2.0 3.0)) (- (^ 16.0 2) (^ 2.0 2))) 1.8836516e09) 1e9))

(setval d10t2.pos 379.7) 10->2 379.826
(setval d11t2.pos 377.099) 11->2 377.099
(setval d12t2.pos 375.051) 12->2 375.051
(setval d13t2.pos 373.472) 13->2 373.472
(setval d14t2.pos 372.229) 14->2 372.229
(setval d15t2.pos 371.232) 15->2 371.232
(setval d16t2.pos 370.4) 16->2 370.4

(couple d11t2.pos (* (^ (/ 11.0 10.0) 2) (/ (- (^ 10.0 2) (^ 2.0 2))
    (- (^ 11.0 2) (^ 2.0 2))) d10t2.pos))
(couple d12t2.pos (* (^ (/ 12.0 10.0) 2) (/ (- (^ 10.0 2) (^ 2.0 2))
    (- (^ 12.0 2) (^ 2.0 2))) d10t2.pos))
(couple d13t2.pos (* (^ (/ 13.0 10.0) 2) (/ (- (^ 10.0 2) (^ 2.0 2))
    (- (^ 13.0 2) (^ 2.0 2))) d10t2.pos))
(couple d14t2.pos (* (^ (/ 14.0 10.0) 2) (/ (- (^ 10.0 2) (^ 2.0 2))
    (- (^ 14.0 2) (^ 2.0 2))) d10t2.pos))
(couple d15t2.pos (* (^ (/ 15.0 10.0) 2) (/ (- (^ 10.0 2) (^ 2.0 2))
    (- (^ 15.0 2) (^ 2.0 2))) d10t2.pos))
(couple d16t2.pos (* (^ (/ 16.0 10.0) 2) (/ (- (^ 10.0 2) (^ 2.0 2))
    (- (^ 16.0 2) (^ 2.0 2))) d10t2.pos))

(setlimits d10t2.pos 379.0 381.0)
(setlimits d11t2.pos 376.0 378.0)
(setlimits d12t2.pos 374.0 376.0)
(setlimits d13t2.pos 373.0 374.0)
(setlimits d14t2.pos 371.5 372.5)
(setlimits d15t2.pos 371.0 371.5)
(setlimits d16t2.pos 370.1 371.0)
```

```
(setval d10t2.fwhmg 0.08) 10->2
(setval d11t2.fwhmg 0.08) 11->2
(setval d12t2.fwhmg 0.08) 12->2
(setval d13t2.fwhmg 0.08) 13->2
(setval d14t2.fwhmg 0.08) 14->2
(setval d15t2.fwhmg 0.08) 15->2
(setval d16t2.fwhmg 0.08) 16->2

(setval d10t2.fwhml 0.1) 10->2
(setval d11t2.fwhml 0.1) 11->2
(setval d12t2.fwhml 0.1) 12->2
(setval d13t2.fwhml 0.1) 13->2
(setval d14t2.fwhml 0.1) 14->2
(setval d15t2.fwhml 0.1) 15->2
(setval d16t2.fwhml 0.1) 16->2

(setval d10t2.area 2.0e11) 10->2
(setval d11t2.area 1.0e11) 11->2
(setval d12t2.area 8.0e10) 12->2
(setval d13t2.area 6.0e10) 13->2
(setval d14t2.area 4.0e10) 14->2
(setval d15t2.area 4.0e10) 15->2
(setval d16t2.area 2.0e10) 16->2

(setmin d10t2.area 1.0e9) 10->2
(setmin d11t2.area 1.0e9) 11->2
(setmin d12t2.area 1.0e9) 12->2
(setmin d13t2.area 1.0e9) 13->2
(setmin d14t2.area 1.0e9) 14->2
(setmin d15t2.area 1.0e9) 15->2
(setmin d16t2.area 1.0e9) 16->2

(setlimits d10t2.fwhmg 0.02 2.0) 10->2
(setlimits d11t2.fwhmg 0.02 2.0) 11->2
(setlimits d12t2.fwhmg 0.02 2.0) 12->2
(setlimits d13t2.fwhmg 0.02 2.0) 13->2
(setlimits d14t2.fwhmg 0.02 2.0) 14->2
(setlimits d15t2.fwhmg 0.02 2.0) 15->2
(setlimits d16t2.fwhmg 0.02 2.0) 16->2

(setlimits d10t2.fwhml 0.01 2.0) 10->2
(setlimits d11t2.fwhml 0.01 2.0) 11->2
(setlimits d12t2.fwhml 0.01 2.0) 12->2
(setlimits d13t2.fwhml 0.01 2.0) 13->2
(setlimits d14t2.fwhml 0.01 2.0) 14->2
(setlimits d15t2.fwhml 0.01 2.0) 15->2
(setlimits d16t2.fwhml 0.01 2.0) 16->2

(couple d11t2.fwhmg = d10t2.fwhmg)
(couple d12t2.fwhmg = d10t2.fwhmg)
(couple d13t2.fwhmg = d10t2.fwhmg)
(couple d14t2.fwhmg = d10t2.fwhmg)
(couple d15t2.fwhmg = d10t2.fwhmg)
(couple d16t2.fwhmg = d10t2.fwhmg)

(setval g1.pos 381.3)                    Be I 3813.454
```

```
(setlimits g1.pos 381.0 381.5)
(setval g1.area 1.0e8)
(setlimits g1.area 1.0e7 1.0e9)
(setval g1.fwhm 0.10)

(setval g2.pos 379.13)                  O III    3791.26
(setlimits g2.pos 378.0 380.0)
(setval g2.area 2.0e9)
(setlimits g2.area 1.0e8 1.0e10)
(couple g2.fwhm = oiiia.fwhm)

(setval g3.pos 377.4015)                  O III    3774.026
(setlimits g3.pos 377.0 378.0)
(setval g3.area 1.0e9)
(setlimits g3.area 1.0e8 7.0e9)
(couple g3.fwhm = oiiia.fwhm)

(setval g4.pos 374.9075)                  Be II    3749.3
(setlimits g4.pos 374.0 376.0)
(setval g4.area 2.0e9)
(setlimits g4.area 1.0e8 1.0e10)
(couple g4.fwhm = oiiia.fwhm)

(setval g5.pos 373.6792)                  O IV      3736.85
(setlimits g5.pos 373.0 374.0)
(setval g5.area 9.0e8)
(setlimits g5.area 1.0e8 1.0e10)
(couple g5.fwhm = oiiia.fwhm)

(setval g6.pos 372.9161)                  OIII     3729.225
(setlimits g6.pos 372.0 375.0)
(setval g6.area 9.0e8)
(setlimits g6.area 1.0e8 1.0e10)
(couple g6.fwhm = oiiia.fwhm)

(setval g7.pos 370.3291)                  O III    3703.37
(setlimits g7.pos 370.0 371.0)
(setval g7.area 9.0e8)
(setlimits g7.area 1.0e8 1.0e10)
(couple g7.fwhm = oiiia.fwhm)

(setval g8.pos 381.3657)
(setlimits g8.pos 381.0 382.0)
(setval g8.area 2.0e9)
(setlimits g8.area 1.0e8 1.0e10)
(setval g8.fwhm 0.1)
(setlimits g8.fwhm 0.02 0.5)

(setval oiiia.pos 375.987)              O III 3 759.87
(setlimits oiiia.pos 375.0 377.0)
(setval oiiia.area 1.0e9)
(setval oiiia.fwhm 0.1)

(couple oiiib.pos (- oiiia.pos 0.266))    O III 3 757.21
(setval oiiib.area 1.0e9)
(setval oiiib.fwhm 0.1)
```

```
(couple oiiic.pos (- oiiia.pos 0.520))   O III 3 754.67
(setval oiiic.area 1.0e9)
(setval oiiic.fwhm 0.1)

(couple g1.fwhm = oiiia.fwhm)
(couple oiiib.fwhm = oiiia.fwhm)
(couple oiiic.fwhm = oiiia.fwhm)

(setval be3a.pos 372.092)        Be III 3720.92
(setval be3a.area 1.0e11)
(setval be3a.fwhm 0.05)

(setval be3b.pos 372.14)         Be III 3721.4
(setval be3b.area 5.0e10)
(setval be3b.fwhm 0.05)

(setval be3c.pos 372.298)        Be III 3722.98
(setval be3c.area 5.0e10)
(setval be3c.fwhm 0.05)

(setlimits oiiia.area 1.0e8 1.0e10)
(setlimits oiiib.area 1.0e8 1.0e10)
(setlimits oiiic.area 1.0e8 1.0e10)

(setlimits g1.fwhm 0.02 0.3)
(setlimits oiiia.fwhm 0.02 0.3)
(setlimits oiiib.fwhm 0.02 0.3)
(setlimits oiiic.fwhm 0.02 0.3)

(setlimits be3a.area 1.0e7 8.0e11)
(setlimits be3b.area 1.0e7 8.0e11)
(setlimits be3c.area 1.0e7 8.0e11)

(setlimits oiiia.pos 375.5 376.5)

(setlimits be3a.pos 371.8 372.2)

(setlimits be3a.fwhm 0.02 0.2)
(couple be3b.fwhm = be3a.fwhm)
(couple be3c.fwhm = be3a.fwhm)
(couple be3b.pos = be3a.pos + 0.048)
(couple be3c.pos = be3a.pos + 0.206)
```

Listing B.4: Model for C I Zeeman split feature observed in JET divertor (Section 5.4)

```
(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)
```

Listing B.5: Model for overlapping high/low field C III Zeeman feature as seen at JET (Section 5.4)

```
(model zeeman
    (+
        (shift-lambda
            (broaden_gauss
                (+
                    (*  (adas-zeeman ciiilowpi)ciiilowpimult)
                    (*  (adas-zeeman ciiilowsigma)ciiilowsigmamult)
                low_add)
            bg1)
        sh1)
        (shift-lambda
            (broaden_gauss
                (+
                  (*  (adas-zeeman ciiihighpi) ciiihighpimult)
                  (*  (adas-zeeman ciiihighsigma) ciiihighsigmamult)
```

```
                high_add)
            bg2)
        sh2)
        (background-linear backg)
    add)
)

(setval sh1.lambda 0.01)
(setlimits sh1.lambda 0.0 0.05)
(setval sh2.lambda 0.01)
(setlimits sh2.lambda 0.0 0.05)

(setval backg.c 50.0)
(setlimits backg.c 0.0 200.0)
(setval backg.m 0.0001)
(setlimits backg.m 0.0 100.0)

(setval bg1.fwhm 0.02)
(setlimits bg1.fwhm 0.01 0.04)
(setval bg2.fwhm 0.02)
(setlimits bg2.fwhm 0.01 0.04)

##################################
    zeeman features
##################################
(setval ciiilowpi.findex 28)
(setval ciiilowpi.obsangle 90.0)
(fixed ciiilowpi.obsangle)
(setval ciiilowpi.bvalue 1.9)
(setlimits ciiilowpi.bvalue 1.7 2.0)
(setval ciiilowpimult.factor 1000.0)
(setlimits ciiilowpimult.factor 1.0 3000.0)
(setval ciiilowpi.pol 2)

(setval ciiilowsigma.findex 28)
(setval ciiilowsigma.obsangle 90.0)
(fixed ciiilowsigma.obsangle)
(couple ciiilowsigma.bvalue = ciiilowpi.bvalue)
(setval ciiilowsigmamult.factor 1000.0)
(setlimits ciiilowsigmamult.factor 1.0 3000.0)
(setval ciiilowsigma.pol 3)

(setval ciiihighpi.findex 28)
(couple ciiihighpi.obsangle = ciiilowpi.obsangle)
(setval ciiihighpi.bvalue 3.8)
(setlimits ciiihighpi.bvalue 3.0 4.5)
(setval ciiihighpimult.factor 1000.0)
(setlimits ciiihighpimult.factor 1.0 3000.0)
(setval ciiihighpi.pol 2)

(setval ciiihighsigma.findex 28)
(couple ciiihighsigma.obsangle = ciiilowsigma.obsangle)
(couple ciiihighsigma.bvalue = ciiihighpi.bvalue)
(couple ciiihighsigmamult.factor
    (* ciiihighpimult.factor
        (/ ciiilowsigmamult.factor ciiilowpimult.factor)))
(setlimits ciiihighsigmamult.factor 1.0 3000.0)
```

```
(setval ciiihighsigma.pol 3)
```

Listing B.6: Model for BeD diatomic molecular emission from JET divertor (Section 5.5)

```
(model molecule

    (+
        (broaden_gauss
                (*(shift-lambda (adas-picket bed)sh) mult)
        bg)
        (broaden_gauss
            (*(shift-lambda (adas-picket bed2)sh2) mult2)

        bg2)
       (background-linear backg)
       (gaussian fei49705)
       (gaussian fei49731)
       (gaussian fei49786)
       (gaussian fei49825)
       (gaussian fei49833)
       (gaussian fei49838)
       (gaussian fei49853)
       (gaussian fei49889)
       (gaussian fei49922)
    )

)
(dummy fei)
(setval fei.fwhm 0.03)

(setval fei49705.pos 497.14958)   497.04958
(setlimits fei49705.pos 497.1 497.2)
(couple fei49705.fwhm = fei.fwhm)
(setval fei49705.area 55.0)


(setval fei49731.pos 497.41016)   497.31016
(setlimits fei49731.pos 497.38 497.45)
(couple fei49731.fwhm = fei.fwhm)
(setval fei49731.area 162.0)

(setval fei49786.pos 497.96030)   497.86030
(setlimits fei49786.pos 497.9 498.0)
(couple fei49786.fwhm = fei.fwhm)
(setval fei49786.area 105.0)

(setval fei49825.pos 498.34996)   498.24996
(setlimits fei49825.pos 498.3 498.4)
(couple fei49825.fwhm = fei.fwhm)
(setval fei49825.area 615.0)

(setval fei49833.pos 498.42504)   498.32504
(setlimits fei49833.pos 498.4 498.5)
(couple fei49833.fwhm = fei.fwhm)
(setval fei49833.area 325.0)
```

```
(setval fei49838.pos 498.48256)   498.38256
(setlimits fei49838.pos 498.44 498.5)
(couple fei49838.fwhm = fei.fwhm)
(setval fei49838.area 445.0)

(setval fei49853.pos 498.62526)    498.52526
(setlimits fei49853.pos 498.6 498.7)
(couple fei49853.fwhm = fei.fwhm)
(setval fei49853.area 360.0)

(setval fei49889.pos 498.99498)    498.89498
(setlimits fei49889.pos 498.9 499.05)
(couple fei49889.fwhm = fei.fwhm)
(setval fei49889.area 144.0)

(setval fei49922.pos 499.22680)    499.12680
(setlimits fei49922.pos 499.2 499.25)
(couple fei49922.fwhm = fei.fwhm)
(setval fei49922.area 140.0)

(setval backg.m -10.0)
(setval backg.c 1506.6)
(setlimits backg.c 1000.0 2500.0)

(setval bg.fwhm 0.03)
(setlimits bg.fwhm 0.01 0.1)
(fixed bg.fwhm)
(couple bg2.fwhm = bg.fwhm)

(setval sh.lambda 0.18) 1.87
(setlimits sh.lambda 0.0 0.3)

(setval mult.factor 5000.0)
(setlimits mult.factor 1000.0 15000.0)

(setval bed.rtemp 2200.0)
(setlimits bed.rtemp 1000.0 10000.0)
(setval bed.vtemp 8000.0)
(setlimits bed.vtemp 1000.0 12000.0)
(setval bed.a 0.1)
(setlimits bed.a 0.01 0.99999999)
(setval bed.b 0.1)
(setlimits bed.b 0.01 0.99999999)
(setval bed.c 0.1)
(setlimits bed.c 0.01 0.99999999)
(setval bed.d 0.1)
(setlimits bed.d 0.01 0.99999999)

(setval sh2.lambda 0.24)
(setlimits sh2.lambda 0.0 0.3)

(setval mult2.factor 5000.0)
(setlimits mult2.factor 1000.0 15000.0)

(setval bed2.rtemp 3000.0)
(setlimits bed2.rtemp 1000.0 9000.0)
```

```
(setval bed2.vtemp 3000.0)
(setlimits bed2.vtemp 1000.0 9000.0)

(setval bed2.a 0.1)
(setlimits bed2.a 0.01 0.99999999)
(setval bed2.b 0.1)
(setlimits bed2.b 0.01 0.99999999)
(setval bed2.c 0.1)
(setlimits bed2.c 0.01 0.99999999)
(setval bed2.d 0.1)
(setlimits bed2.d 0.01 0.99999999)
```

Listing B.7: Model for C III / C IV emission in the V / UV region as observed at JET (Section 5.6)

```
(model
    (+
        (shift-lambda
            (broaden_gauss (* (+(adas-adf15archive carbon2)) mult2) bg2)
        shift2)
        (shift-lambda
            (broaden_gauss (* (+(adas-adf15archive carbon3)) mult3) bg3)
        shift3)
        (background-linear backg)
        (voigtian heii304)
        (gaussian oii312)
    )
)

(setval oii312.pos 31.386)
(setlimits oii312.pos 31.0 31.5)
(setval oii312.fwhm 0.1)
(setmin oii312.fwhm 0.05)
(setval oii312.area 1.0e4)
(setlimits oii312.area 1.0e3 1.0e5)

(setval heii304.pos 30.5)
(setval heii304.fwhmg 0.05)
(setlimits heii304.fwhmg 0.03 0.8)
(setval heii304.fwhml 0.05)
(setlimits heii304.fwhml 0.03 0.8)
(setval heii304.area 5.0e4)
(setlimits heii304.area 1.0e4 1.0e5)

(setval shift2.lambda 0.15)
(setlimits shift2.lambda 0.1 0.25)

(setval shift3.lambda 0.15)
(setlimits shift3.lambda 0.1 0.2)

(setval backg.c 6.0e4)
(setlimits backg.c 5.0e4 1.2e5)
(setval backg.m 0.0)
(setlimits backg.m -200 0.0)

(setval carbon2.te 10.0)
```

```
(setlimits carbon2.te 5.0 20.0)
(setval carbon2.dens 5.0e13)
(setlimits carbon2.dens 1.0e13 2.2e14)
(setval carbon2.filenames /home/cnich/afg/examples/c2_pec.pass)

(setval carbon3.te 20.0)
(setlimits carbon3.te 5.0 40.0)
(setval carbon3.dens 5.0e13)
(setlimits carbon3.dens 1.0e13 2.2e14)
(setval carbon3.filenames /home/cnich/afg/examples/c3_pec.pass)

(setval bg2.fwhm 0.07)
(setlimits bg2.fwhm 0.03 0.3)

(setval bg3.fwhm 0.07)
(setlimits bg3.fwhm 0.03 0.3)

(setval mult2.factor 2.0e13)
(setlimits mult2.factor 1.0e13 5.0e14)

(setval mult3.factor 8.0e12)
(setlimits mult3.factor 5.0e12 3.0e13)
```

Appendix C

FFS Simplification Rules

This appendix displays an extract of the current *ffs_simplify* source code, showing the IDL structures that define the *rule list* used when optimising a model definition.

Listing C.1: IDL source showing the FFS simplification rules structure.

```
;=============================================
; FFS_BROADEN_GAUSSIAN
;=============================================
    {parent:'ffs_broaden_gauss', $
     child:'ffs_gaussian', $
     replacement:'ffs_gaussian', $
     coupling:ptr_new([ $
                    ['pos', '(child.pos)'], $
                    ['fwhm', '(^ (+ (^ parent.fwhm 2) (^ child.fwhm 2)) 0.5)'], $
                    ['area', '(child.area)'] $
                ]) $
    }, $
    {parent:'ffs_broaden_gauss', $
     child:'ffs_lorentzian', $
     replacement:'ffs_voigtian', $
     coupling:ptr_new([ $
                    ['pos', '(child.pos)'], $
                    ['fwhmg', '(parent.fwhm)'], $
                    ['fwhml', '(child.fwhm)'], $
                    ['area', '(child.area)'] $
                ]) $
    }, $
    {parent:'ffs_broaden_gauss', $
     child:'ffs_broaden_lorentz', $
     replacement:'ffs_broaden_voigt', $
     coupling:ptr_new([ $
                    ['fwhmg', '(parent.fwhm)'], $
                    ['fwhml', '(child.fwhm)'] $
                ]) $
    }, $
    {parent:'ffs_broaden_gauss', $
     child:'ffs_voigtian', $
     replacement:'ffs_voigtian', $
```

```
     coupling:ptr_new([ $
                ['pos', '(child.pos)'], $
                ['fwhmg', '(^ (+ (^ parent.fwhm 2) (^ child.fwhmg 2)) 0.5)'], $
                ['fwhml', '(child.fwhml)'], $
                ['area', '(child.area)'] $
            ]) $
    }, $
    {parent:'ffs_broaden_gauss', $
     child:'ffs_line', $
     replacement:'ffs_gaussian', $
     coupling:ptr_new([ $
                    ['pos', '(child.pos)'], $
                    ['fwhm', '(parent.fwhm)'], $
                    ['area', '(child.intensity)'] $
                ]) $
    }, $
;=============================================
; FFS_BROADEN_LORENTZ
;=============================================
    {parent:'ffs_broaden_lorentz', $
     child:'ffs_lorentzian', $
     replacement:'ffs_lorentzian', $
     coupling:ptr_new([ $
                    ['pos', '(child.pos)'], $
                    ['fwhm', '(+ parent.fwhm child.fwhm)'], $
                    ['area', '(child.area)'] $
                ]) $
    }, $
    {parent:'ffs_broaden_lorentz', $
     child:'ffs_gaussian', $
     replacement:'ffs_voigtian', $
     coupling:ptr_new([ $
                    ['pos', '(child.pos)'], $
                    ['fwhmg', '(child.fwhm)'], $
                    ['fwhml', '(parent.fwhm)'], $
                    ['area', '(child.area)'] $
                ]) $
    }, $
    {parent:'ffs_broaden_lorentz', $
     child:'ffs_voigtian', $
     replacement:'ffs_voigtian', $
     coupling:ptr_new([ $
                    ['pos', '(child.pos)'], $
                    ['fwhmg', '(child.fwhmg)'], $
                    ['fwhml', '(+ parent.fwhm child.fwhml)'], $
                    ['area', '(child.area)'] $
                ]) $
    }, $
    {parent:'ffs_broaden_lorentz', $
     child:'ffs_line', $
     replacement:'ffs_lorentzian', $
     coupling:ptr_new([ $
                    ['pos', '(child.pos)'], $
                    ['fwhm', '(parent.fwhm)'], $
                    ['area', '(child.intensity)'] $
                ]) $
    }, $
```

```
;=============================================
; FFS_BROADEN_VOIGTIAN
;=============================================
    {parent:'ffs_broaden_voigt', $
     child:'ffs_voigtian', $
     replacement:'ffs_voigtian', $
     coupling:ptr_new([ $
                    ['pos', '(child.pos)'], $
                    ['fwhml', '(+ parent.fwhml child.fwhml)'], $
                    ['fwhmg', '(^ (+ (^ parent.fwhmg 2) (^ child.fwhmg 2)) 0.5)'], $
                    ['area', '(child.area)'] $
                ]) $
    }, $
    {parent:'ffs_broaden_voigt', $
     child:'ffs_gaussian', $
     replacement:'ffs_voigtian', $
     coupling:ptr_new([ $
                    ['pos', '(child.pos)'], $
                    ['fwhml', '(parent.fwhml)'], $
                    ['fwhmg', '(^ (+ (^ parent.fwhmg 2) (^ child.fwhm 2)) 0.5)'], $
                    ['area', '(child.area)'] $
                ]) $
    }, $
    {parent:'ffs_broaden_voigt', $
     child:'ffs_lorentzian', $
     replacement:'ffs_voigtian', $
     coupling:ptr_new([ $
                    ['pos', '(child.pos)'], $
                    ['fwhml', '(+ parent.fwhml child.fwhm)'], $
                    ['fwhmg', '(parent.fwhmg)'], $
                    ['area', '(child.area)'] $
                ]) $
    }, $
    {parent:'ffs_broaden_voigt', $
     child:'ffs_line', $
     replacement:'ffs_voigtian', $
     coupling:ptr_new([ $
                    ['pos', '(child.pos)'], $
                    ['fwhml', '(parent.fwhml)'], $
                    ['fwhmg', '(parent.fwhmg)'], $
                    ['area', '(child.intensity)'] $
                ]) $
    }, $
;=============================================
; FFS_MULTIPLY
;=============================================
    {parent:'ffs_multiply', $
     child:'ffs_gaussian', $
     replacement:'ffs_gaussian', $
     coupling:ptr_new([ $
                    ['pos', '(child.pos)'], $
                    ['fwhm', '(child.fwhm)'], $
                    ['area', '(* child.area parent.factor)'] $
                ]) $
    }, $
    {parent:'ffs_multiply', $
     child:'ffs_lorentzian', $
```

```
     replacement:'ffs_lorentzian', $
     coupling:ptr_new([ $
                    ['pos', '(child.pos)'], $
                    ['fwhm', '(child.fwhm)'], $
                    ['area', '(* child.area parent.factor)'] $
                ]) $
    }, $
    {parent:'ffs_multiply', $
     child:'ffs_voigtian', $
     replacement:'ffs_voigtian', $
     coupling:ptr_new([ $
                ['pos', '(child.pos)'], $
                ['fwhmg', 'child.fwhmg)'], $
                ['fwhml', '(child.fwhml)'], $
                ['area', '(* child.area parent.factor)'] $
            ]) $
    } $
] $
)
```

Appendix D

FFS Core Routine Methods

The following listings are extracts from the core routines' documentation headers. These are provided to show a summary of the methods associated with these classes.

Listing D.1: Public methods for class *ffs_primitive*

```
PUBLIC ROUTINES:
[seterrmsg]
  PURPOSE:
    Sets state variable errmsg with input error message string
  INPUTS:
    errmsg   - String. The error message to set.
    obj      - Keyword. Indicates object return type to be used.
  OUTPUTS:
    Output is different, depending upong keywords set in method
    call, but will be a typical error value for that type of variable (this
    is for convienent use of the routine in the return statement of the
    originating method).
    If the 'obj' keyword is set:
      Object reference. Null object reference.
    Default:
        Integer. returns 0.
  SIDE EFFECTS:
    None.

[geterrmsg]
  PURPOSE:
   Returns the currently set error message string.
  INPUTS:
    None
  OUTPUTS:
    String - returns the currently set error message string.
  SIDE EFFECTS:
    None

[setdebug]
  PURPOSE:
    Sets debug flag.
  INPUTS:
```

```
    Integer - 1 or 0.
  OUTPUTS:
    Returns 1 if successful.
    Returns 0 if unsuccessful.
  SIDE EFFECTS:
    None

[getdebug]
  PURPOSE:
    Returns value of debug flag.
  INPUTS:
    None.
  OUTPUTS:
    Integer - 1 or 0.
  SIDE EFFECTS:
    None
```

Listing D.2: Public methods for class *ffs_model*

```
PUBLIC ROUTINES:
In addtion to the methods listed below, this object inherits methods from
ffs_primitive - refer to this class' documentation for more details.
.....................................................................
[mpfunct]
  PURPOSE:
    function for mpfit to use for evaluating the function
    since it can't access the methods of the model object.
  INPUTS:
    x           -   the independent variable.
    p           -   parameter values.
    modelobj    -   object reference of the model to be used for calculation.
  OUTPUTS:
    Returns the modelled intensity values.
  SIDE EFFECTS:
    None.
.....................................................................
[iselement]
  PURPOSE:
    Checks if input array elements are ffs_element object references.
  INPUTS:
    Object reference to check.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
.....................................................................
[ispar]
  PURPOSE:
    Checks if input array elements are ffs_par object references.
  INPUTS:
    Object reference to check.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
.....................................................................
[getxdata]
  PURPOSE:
    Retrieves the object's xdata.
```

```
  INPUTS:
    None.
  OUTPUTS:
    double; array comprising the x-axis data points.
  SIDE EFFECTS:
    None.
.....................................................................
[setxdata]
  PURPOSE:
    Sets the wavelength grid that this model is intended to be placed upon
    (This is not the same as 'wavelength' field in calculate structure).
  INPUTS:
    double; array comprising the x-axis data points.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[setparfixed]
  PURPOSE:
    Sets specified parameters in specified elements to be fixed.
  INPUTS:
    integer(boolean); set to fix parameter.
    Keywords to be passed to getpars to specify pars: elementname, position
    parname, parpos, count, free, fixed, fullname.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getparfixed]
  PURPOSE:
    Retrieves the array of pars via method getPars then uses their getFixed
    method to retrieve the value of 'fixed' for each par.
  INPUTS:
    Keywords to be passed to getpars to specify pars: elementname, position
    parname, parpos, count, fullname.
  OUTPUTS:
    integer; array of 'fixed' flag values (0 or 1).
  SIDE EFFECTS:
    None.
.....................................................................
[setparerrors]
  PURPOSE:
    Sets the error attribute for the specified parameters.
  INPUTS:
    double; the error associated with the parameter value.
    Keywords to be passed to getpars to specify pars: elementname, position,
    parname, parpos, count, free, fixed, fullname.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getparerrors]
  PURPOSE:
    Retrieves the array of pars via method getPars then uses their geterror
```

```
    method to retrieve the value of 'error' for each par.
  INPUTS:
    Keywords to be passed to getpars to specify pars: elementname, position
    parname, parpos, count, fullname=fullname
  OUTPUTS:
    Returns the errors for the specified pars.
  SIDE EFFECTS:
    None.
.....................................................................
[setparvals]
  PURPOSE:
    Retrieves the array of parameters via method getPars then uses their
    setValue methods to set appropriate value in the parameter value array
    passed in 'parVals'
  INPUTS:
    if for fitting ffs_pars (/fitting) - then double; array of par values.
    if for non-fitting ffs_props (/static) - then any type can be set as value.
    Keywords to be passed to getpars to specify pars: elementname, position,
    parname, parpos, static, fitting, fixed, free, fullname.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getparvals]
  PURPOSE:
    Retrieves the array of parameters via method getPars then uses their
    getValue methods.
  INPUTS:
    Keywords to be passed to getpars to specify pars: elementname, position
    parname, parpos, static, fitting, fixed, free, fullname.
  OUTPUTS:
    If for fitting ffs_pars (/fitting) - then double; array of par values.
    If for non-fitting ffs_props (/static) - then pointer array to various
    types.
    Keyword 'count' supplies the number of par values returned.
  SIDE EFFECTS:
    None.
.....................................................................
[setpars]
  PURPOSE:
    Clears existing parameters and adds the new parameters passed in, for
    specified elements.
  INPUTS:
    object reference; array of ffs_par objects.
    Keywords 'elementname' and 'position' specify elements to setpars for;
    default is for all elements.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getpars]
  PURPOSE:
    Retrieves the specified element's parameter objects.
  INPUTS:
    Keywords to specify which pars are required:
```

```
        elementname -   string; array of elementnames.
        position    -   integer; position of element in container.
        parname     -   string; array of par names.
        parpos      -   integer; position in par container.
        count       -   integer; (outward); the number of elements returned.
        static      -   integer(boolean); set to return only ffs_props in the element.
                        i.e. properties of the element that are non-fitting
                        parameters.
        fitting     -   integer (boolean); set to only return the ffs_pars in
                        the element i.e. the fitting parameters.
    Setting no keywords will default to returning all parameters.
  OUTPUTS:
   object reference; array of ffs_par objects
  SIDE EFFECTS:
    None.
.....................................................................
[getnumpars]
  PURPOSE:
    Returns the number of pars in the specified element (can filter by
    /static, /fitting).
  INPUTS:
    Keywords to be passed to getpars to specify pars: elementname, position,
    static, fitting.
  OUTPUTS:
    integer; the number of pars in the element.
  SIDE EFFECTS:
    None.
.....................................................................
[getparindex]
  PURPOSE:
    Return the index of the parameter, specified by elementname and parname
    (both are required), in the flattened list of all of the model's
    parameters.
  INPUTS:
    elementname     -   the element to which the specified parameter belongs.
    parname         -   the parameter from which to obtain the index.
  OUTPUTS:
    integer; the index of the parameter in a flattened list of the model
    parameters.
  SIDE EFFECTS:
    None.
.....................................................................
[getparnames]
  PURPOSE:
    Retrieves the names of the parameters for those specified by:
    containing element's name or position and optionally by par position.
    It is also possible to filter fitting pars and static properties.
  INPUTS:
    Keywords to specify pars:
        elementname -   string; array of elementnames.
        position    -   integer; position of element in container.
        parpos      -   integer; position in par container.
        static      -   integer (boolean); set to return only ffs_props in the element.
                        i.e. properties of the element that are non-fitting
                        parameters.
        fitting     -   integer (boolean); set to only return the ffs_pars in
                        the element i.e. the fitting parameters.
```

```
    Setting no keywords will default to returning all par names.
  OUTPUTS:
    string; array of parnames.
  SIDE EFFECTS:
    None.
.....................................................................
[getnumelements]
  PURPOSE:
    Returns the number of pars in the element (can filter by /static, /fitting).
  INPUTS:
    Keywords to be passed to getpars to specify pars: static, fitting.
  OUTPUTS:
    integer; the number of pars in the element.
  SIDE EFFECTS:
    None.
.....................................................................
[getelements]
  PURPOSE:
    returns an array of the elements stored within the container.
    setting the postion or name keywords allows retrieval of specific elements.
    by default, without supplying any keywords, the method will return ALL
    elements. Outward keyword count supplies the number of parameters in the
    container.
  INPUTS:
    Keywords to specify elements:
        all         -   integer(switch); enable to return all element
        position    -   integer; position of element in container.
        elementname -   string; array of elementnames.
        parname     -   string; name of a parameter that the sought element
                        should have.
        count       -   integer; (outward); the number of elements returned.
    Setting no keywords will default to returning all parameters.
  OUTPUTS:
    object reference; array of the specified element object references.
  SIDE EFFECTS:

.....................................................................
[getelementnames]
  PURPOSE:
    Retrieves the array of specified elements via method getelements then
    uses their getname methods.
  INPUTS:
    position    -   integer; keyword to be passed to getelements to specify
                    elements.
    count       -   integer; (outward); the number of elements returned.
  OUTPUTS:
    string; array of element names
  SIDE EFFECTS:

.....................................................................
[getelementtypes]
  PURPOSE:
    Retrieves the class type of the element specified
  INPUTS:
    Retrieves the array of specified elements via method getelements then
    utilises 'obj_class' to get the type of object.
  OUTPUTS:
```

```
    string; the element object class type.
  SIDE EFFECTS:
    None.
.....................................................................
[addelement]
  PURPOSE:
    Adds an ffs_element object to the model, either by object reference or
    by element name - if the latter, then a new 'ffs_element' with that name
    is created and added to the element.
  INPUTS:
    in          -   object reference; an existing ffs_element.
    elementname -   string; the name of the new element to be added in.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[setmodelname]
  PURPOSE:
    Sets the name of the model. Used to identify model at higher level in FFS.
    Note that this knowingly duplicates setname functionality.
  INPUTS:
    String; the name of the model.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getmodelname]
  PURPOSE:
    Retrieves the model's name. Note that this knowingly duplicates getname
    functionality.
  INPUTS:
    None.
  OUTPUTS:
    String; the name of the model.
  SIDE EFFECTS:
    None.
.....................................................................
[setname]
  PURPOSE:
    Sets the name of the model. Used to identify model at higher level in FFS.
    Note that this knowingly duplicates setmodelname functionality.
  INPUTS:
    String; the name of the model.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getname]
  PURPOSE:
    Retrieves the model's name. Note that this knowingly duplicates
    getmodelname functionality.
  INPUTS:
    None.
  OUTPUTS:
```

```
    String; the name of the model.
  SIDE EFFECTS:
    None.
.....................................................................
[evaluate]
  PURPOSE:
    Evaluates the model.
  INPUTS:
    memo    -   enables buffering of previous evaluation result and parameter
                values - should improve performance during fitting.
                If there has not been a change in parameter values, then this
                eliminates the need for re-calculation.
  OUTPUTS:
    A structure containing the following fields:
        wavelength  -   wavelength grid for intensity values.
        intensity   -   intensity values.
        gridded     -   whether the intensity values are mapped to a wavelength
                        grid or not: 1 or 0.
  SIDE EFFECTS:
    None.
.....................................................................
[getresult]
  PURPOSE:
    Returns the saved model evaluation result structure.
  INPUTS:
    None.
  OUTPUTS:
    A structure containing the following fields:
        wavelength  -   wavelength grid for intensity values.
        intensity   -   intensity values.
        gridded     -   whether the intensity values are mapped to a wavelength
                        grid or not: 1 or 0.
  SIDE EFFECTS:
    None.
.....................................................................
[findpar]
  PURPOSE:
    Gives which element a par belongs to
  INPUTS:
    A par object reference.
  OUTPUTS:
    The name of the element which the par belongs to.
    Returns -1 if the par is not found in the model.
  SIDE EFFECTS:
    None.
.....................................................................
[fastsetup]
  PURPOSE:
    Sets cacheing state variables up so routines can use
    the /fast option
  INPUTS:
    None
  OUTPUTS:
    Returns 1 if succesful.
  SIDE EFFECTS:
    Use of this facility assumes the model will not change
    between calls (i.e. setup of coupling, limits etc. etc.)
```

Listing D.3: Public methods for class *ffs_element*

```
PUBLIC ROUTINES:
In addtion to the methods listed below, this object inherits methods from ffs_primitive
- refer to this class' documentation for more details.
.....................................................................
[ispar]
  PURPOSE:
    Checks if input array elements are ffs_par object references.
  INPUTS:
    Object reference to check.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[isprop]
  PURPOSE:
    Checks if input array elements are ffs_prop object references.
  INPUTS:
    Object reference; the references to check.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[setxdata]
  PURPOSE:
    Sets the wavelength grid that this feature is intended to be placed upon
    (This is not the same as 'wavelength' field in calculate structure).
  INPUTS:
    double; array comprising the x-axis data points.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getxdata]
  PURPOSE:
    Retrieves the object's xdata.
  INPUTS:
    None.
  OUTPUTS:
    double; array comprising the x-axis data points.
  SIDE EFFECTS:
    None.
.....................................................................
[setname]
  PURPOSE:
    Sets the element's name; used to identify element at higher level in FFS.
    Note that this knowingly duplicates setelementname functionality.
  INPUTS:
    String; the name of the element, should be unique within a model.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
```

```
  SIDE EFFECTS:
    None.
.....................................................................
[getname]
  PURPOSE:
    Retrieves the element's name.
  INPUTS:
    None.
  OUTPUTS:
    String; the name of the element
  SIDE EFFECTS:
    None.
.....................................................................
[setelementname]
  PURPOSE:
    Sets the element's name; used to identify element at higher level in FFS.
    Note that this knowingly duplicates setname functionality.
  INPUTS:
    String; the name of the element, should be unique within a model.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[setpars]
  PURPOSE:
    Clears the parcontainer and adds the new parameters passed in.
  INPUTS:
    object reference; array of ffs_par objects.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getpars]
  PURPOSE:
    Retrieves the element's parameter objects.
  INPUTS:
    Keywords to specify which pars are required:
        all         -   integer (boolean); set to retrieve all pars for the
                        element (default).
        parname     -   string; array of par names.
        position    -   integer; position in par container
        count       -   integer; (outward); the number of elements returned.
        static      -   boolean; set to return only ffs_props in the element.
                        i.e. properties of the element that are non-fitting
                        parameters.
        fitting     -   integer (boolean); set to only return the ffs_pars in
                        the element i.e. the fitting parameters.
  OUTPUTS:
    object reference; array of ffs_par objects
  SIDE EFFECTS:
    None.
.....................................................................
[setparfixed]
  PURPOSE:
    Sets specified parameters in the element to be fixed.
```

```
  INPUTS:
    integer(boolean); set to fix parameter.
    Keywords to be passed to getpars to specify pars: parname, position.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getparfixed]
  PURPOSE:
    Retrieves the array of pars via method getPars then uses their getFixed
    method to retrieve the value of 'fixed' for each par.
  INPUTS:
    Keywords to be passed to getpars to specify pars: parname, position,
  OUTPUTS:
    integer; array of 'fixed' flag values (0 or 1).
  SIDE EFFECTS:
    None.
.....................................................................
[setparlimits]
  PURPOSE:
    Sets the (soft) limits for specified parameters in the element.
  INPUTS:
    double, two element array; specifies the lower/upper limits
    for the parameter value (the range in which it is free to
    vary during a fit).'limited' detailed above must be
    specified together with these limits to enforce or
    disable them.
    Keywords to be passed to getpars to specify pars: parname, position.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getparlimits]
  PURPOSE:
    Retrieves the specified parameter's (soft) limits.
  INPUTS:
    Keywords to be passed to getpars to specify pars: parname, position.
  OUTPUTS:
    double, two element array; specifies the lower/upper limits
    for the parameter value (the range in which it is free to
    vary during a fit).'
  SIDE EFFECTS:
    None.
.....................................................................
[setparhardlimits]
  PURPOSE:
    Sets the hard limits for specified parameters in the element.
  INPUTS:
    double, two element array; specifies the lower/upper hard
    limits for the parameter value. These values would normally
    be set to prevent values being selected that will cause
    evaluation of an FFS element to fail. These are, in a sense,
    limits placed upon the other soft limits detailed above.
    Keywords to be passed to getpars to specify pars: parname, position.
  OUTPUTS:
```

```
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getparhardlimits]
  PURPOSE:
    Sets the specified parameter's hard limits.
  INPUTS:
    Keywords to be passed to getpars to specify pars: parname, position.
  OUTPUTS:
    double, two element array; specifies the lower/upper hard
    limits for the parameter value. These values would normally
    be set to prevent values being selected that will cause
    evaluation of an FFS element to fail. These are, in a sense,
    limits placed upon the other soft limits detailed above.
  SIDE EFFECTS:
    None.
.....................................................................
[setparvals]
  PURPOSE:
    Retrieves the array of parameters via method getPars then uses their
    setValue methods to set appropriate value in the parameter value array
    passed in 'parVals'
  INPUTS:
    if for fitting ffs_pars (/fitting) - then double; array of par values.
    if for non-fitting ffs_props (/static) - then any type can be set as value.
    Keywords to be passed to getpars to specify pars: name/parname, position,
    static, fitting.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getparvals]
  PURPOSE:
    Retrieves the array of parameters via method getPars then uses their
    getValue methods.
  INPUTS:
    Keywords to be passed to getpars to specify pars: name/parname, position,
    static, fitting.
  OUTPUTS:
    If for fitting ffs_pars (/fitting) - then double; array of par values.
    If for non-fitting ffs_props (/static) - then pointer array to various types.
    Keyword 'count' supplies the number of par values returned.
  SIDE EFFECTS:
    None.
.....................................................................
[getparnames]
  PURPOSE:
    Retrieves the array of parameters via method getPars then uses their
    getname methods.
  INPUTS:
    Keywords to be passed to getpars to specify pars: position, static, fitting.
  OUTPUTS:
    Returns a string array of the parameter names of the specified positions or all
    by default.
  SIDE EFFECTS:
```

```
    None.
.....................................................................
[getnumpars]
  PURPOSE:
    Returns the number of pars in the element (can filter by /static, /fitting).
  INPUTS:
    Keywords to be passed to getpars to specify pars: static, fitting.
  OUTPUTS:
    integer; the number of pars in the element.
  SIDE EFFECTS:
    None.
.....................................................................
[addpar]
  PURPOSE:
    If a parameter is passed in, validity of the object verified, then
    added to the element. If instead keyword 'parname' is set, then a
    new 'ffs_par' with that name is instantiated and added to the element.
  INPUTS:
    par     -   object reference; an existing ffs_par.
    parname -   string; the name of a parameter to add.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[addprop]
  PURPOSE:
    If a ffs_prop is passed in, validity of the object verified, then
    added to the element. If instead keyword 'name' is set, then a
    new 'ffs_prop' with that name is instantiated and added to the element.
  INPUTS:
    prop    -   object reference; an existing ffs_prop.
    name    -   string; the name of a property to add.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[setchildren]
  PURPOSE:
    Sets the child element objects for this element.
  INPUTS:
    obj     -   object reference; the child object(s) to set.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getchildren]
  PURPOSE:
    Retrieves the child element objects for this element.
  INPUTS:
    None.
  OUTPUTS:
    object reference; the child objects for this element.
    count (keyword): Number of child objects
  SIDE EFFECTS:
```

```
    None.
.....................................................................
[evaluate]
  PURPOSE:
    Evaluates this element - generates the line profile. The result is stored
    to the object as well as returned through this method.
  INPUTS:
    memo -  enables buffering of previous evaluation result and parameter
            values - should improve performance during fitting.
            If there has not been a change in parameter values, then this
            eliminates the need for re-calculation.
  OUTPUTS:
     A structure containing the following fields:
        wavelength  -   wavelength grid for intensity values.
        intensity   -   intensity values.
        gridded     -   whether the intensity values are mapped to a wavelength
                        grid or not: 1 or 0.
  SIDE EFFECTS:
    None.
.....................................................................
[setsubtype]
  PURPOSE:
    Sets the element's subtype; used for elements which can have
    multiple behaviours (e.g. ADAS, broaden)
  INPUTS:
    String; the subtype of the element.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getsubtype]
  PURPOSE:
    Gets the element's subtype; used for elements which can have
    multiple behaviours (e.g. ADAS, broaden)
  INPUTS:
    None
  OUTPUTS:
    Scalar String specifying the subtype, blank string if not set.
  SIDE EFFECTS:
    None.
.....................................................................
[fastsetup]
  PURPOSE:
    Sets cacheing state variables up so routines can use
    the /fast option
  INPUTS:
    None
  OUTPUTS:
    Returns 1 if succesful.
  SIDE EFFECTS:
    Use of this facility assumes the model will not change
    between calls (i.e. setup of coupling, limits etc. etc.)
.....................................................................
```

Listing D.4: Public methods for class *ffs_par*

```
PUBLIC ROUTINES:
In addtion to the methods listed below, this object inherits methods from ffs_primitive
- refer to this class' documentation for more details.
.....................................................................
[setmpfitstr]
  PURPOSE:
    Sets all of the necessary object state data via a structure
    of the form required by MPFIT.
  INPUTS:
    None.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getmpfitstr]
  PURPOSE:
    Retrieves all of the necessary object state data and forms the structure
    required by MPFIT.
  INPUTS:
    None.
  OUTPUTS:
    Returns an MPFIT parameter structure.
  SIDE EFFECTS:
    None.
.....................................................................
[setname]
  PURPOSE:
    Sets the parameter's name; used to identify par at higher level in FFS.
    Note that this knowingly duplicates setparname functionality.
  INPUTS:
    String; the name of the parameter, should be unique within a given element.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getname]
  PURPOSE:
    Retrieves the parameter's name.
  INPUTS:
    None.
  OUTPUTS:
    String; the name of the parameter.
  SIDE EFFECTS:
    None.
.....................................................................
[setparname]
  PURPOSE:
    Sets the parameter's name; used to identify par at higher level in FFS.
    Note that this knowingly duplicates setname functionality.
  INPUTS:
    String; the name of the parameter, should be unique within a given element.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
```

```
  SIDE EFFECTS:
    None.
.....................................................................
[setvalue]
  PURPOSE:
    Sets the parameter's value.
  INPUTS:
    double; the value of the parameter.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getvalue]
  PURPOSE:
    Retrieves the parameter's value.
  INPUTS:
    None.
  OUTPUTS:
    double; the value of the parameter.
  SIDE EFFECTS:
    None.
.....................................................................
[setfixed]
  PURPOSE:
    Specifies if the parameter value is allowed to vary during a fit.
  INPUTS:
    integer; 1 or 0 (true or false)
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getfixed]
  PURPOSE:
    Retrieves the value of 'fixed'.
  INPUTS:
    None.
  OUTPUTS:
    integer; 1 or 0 (true or false)
  SIDE EFFECTS:
    None.
.....................................................................
[setlimited]
  PURPOSE:
    Sets whether parameter has lower/upper bounds.
  INPUTS:
    integer, two element array with possible values 0 or 1;
    to signify if the parameter is bound on lower/upper limit values
    or not. e.g [0,1] indicates no bound on the low side and a bound
    on the upper side.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getlimited]
```

```
  PURPOSE:
    Retrieves the value of 'limited'
  INPUTS:
    None.
  OUTPUTS:
    integer, two element array with possible values 0 or 1;
    to signify if the parameter is bound on lower/upper limit values
    or not. e.g [0,1] indicates no bound on the low side and a bound
    on the upper side.
  SIDE EFFECTS:
    None.
.....................................................................
[setmin]
  PURPOSE:
    Sets the parameter's lower (soft) limit.
  INPUTS:
    double; the value of the lower bound.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getmin]
  PURPOSE:
    Retrieves the value of the lower bound.
  INPUTS:
    None.
  OUTPUTS:
    double; the value of the lower bound.
  SIDE EFFECTS:
    None.
.....................................................................
[setmax]
  PURPOSE:
    Sets the parameter's upper (soft) limit.
  INPUTS:
    double; the value of the upper bound.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getmax]
  PURPOSE:
    Retrieves the value of the upper bound.
  INPUTS:
    None.
  OUTPUTS:
    double; the value of the upper bound.
  SIDE EFFECTS:
    None.
.....................................................................
[sethardlimits]
  PURPOSE:
    Sets the parameter's hard limits.
  INPUTS:
    double, two element array; specifies the lower/upper hard
```

```
    limits for the parameter value. These values would normally
    be set to prevent values being selected that will cause
    evaluation of an FFS element to fail. These are, in a sense,
    limits placed upon the other soft limits detailed above.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[gethardlimits]
  PURPOSE:
    Sets the parameter's hard limits.
  INPUTS:
    None.
  OUTPUTS:
    double, two element array; specifies the lower/upper hard
    limits for the parameter value. These values would normally
    be set to prevent values being selected that will cause
    evaluation of an FFS element to fail. These are, in a sense,
    limits placed upon the other soft limits detailed above.
  SIDE EFFECTS:
    None.
.....................................................................
[sethardmin]
  PURPOSE:
    Sets the parameter's lower hard limit.
  INPUTS:
    double; the value for the lower hard limit.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[gethardmin]
  PURPOSE:
    Retrieves the parameter's lower hard limit.
  INPUTS:
    None.
  OUTPUTS:
    double; the value for the lower hard limit.
  SIDE EFFECTS:
    None.
.....................................................................
[sethardmax]
  PURPOSE:
    Sets the parameter's upper hard limit.
  INPUTS:
    double; the value for the upper hard limit.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[gethardmax]
  PURPOSE:
    Retrieves the parameter's upper hard limit.
  INPUTS:
```

```
    None.
  OUTPUTS:
    double; the value for the upper hard limit.
  SIDE EFFECTS:
    None.
.....................................................................
[setlimits]
  PURPOSE:
    Sets the parameter's (soft) limits.
  INPUTS:
    double, two element array; specifies the lower/upper limits
    for the parameter value (the range in which the user wishes
    the par to vary during a fit).'limited' detailed above must be
    specified together with these limits to enforce or
    disable them.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getlimits]
  PURPOSE:
    Retrieves the parameter's (soft) limits.
  INPUTS:
    None.
  OUTPUTS:
    double, two element array; specifies the lower/upper (soft) limits
    for the parameter value (the range in which it is free to
    vary during a fit). Note that this should not be confused with
    'getefflimits' (see below).
  SIDE EFFECTS:
    None.
.....................................................................
[getefflimits]
  PURPOSE:
    Retrieves the parameter's (effective) limits.
  INPUTS:
    None.
  OUTPUTS:
    double, two element array; specifies the lower/upper limits
    for the parameter value (the range in which it is free to
    vary during a fit). This will be the same as the soft limits
    when limited is '[1,1]'. If either soft limit is disabled, i.e.
    limited is '[0,1]', '[1,0]', or '[0,0]' hard limit values will be
    substituted.
  SIDE EFFECTS:
    None.
.....................................................................
[setstep]
  PURPOSE:
    Sets the step size used for this parameter in numerical fitting.
  INPUTS:
    double; the iteration step size by which to alter the
    parameter value. Notes: Setting zero step size means that
    the step size is selected  automatically. This value is
    overridden when relstep (see below) is set.
  OUTPUTS:
```

```
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getstep]
  PURPOSE:
    Retrieves the step size used for this parameter in numerical fitting.
  INPUTS:
    None.
  OUTPUTS:
    double; the iteration step size by which to alter the
    parameter value. Note: zero step size means that the step size is selected
    automatically. This value is overridden when relstep (see below) is set.
  SIDE EFFECTS:
    None.
.....................................................................
[setrelstep]
  PURPOSE:
    Sets the step size used for this parameter in numerical fitting,
    as a fraction of the parameter value.
  INPUTS:
    float; the iteration step size by which to alter the
    parameter value, as a fraction of the value. Notes: This
    setting overrides 'step' if also present. A default step is
    selected if the par value is zero.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getrelstep]
  PURPOSE:
    Retrieves the step size (as a fraction of the parameter value) used for
    this parameter in numerical fitting,
  INPUTS:
    None.
  OUTPUTS:
    float; the iteration step size by which to alter the
    parameter value, as a fraction of the value. Note: A default step is
    selected if the par value is zero.
  SIDE EFFECTS:
    None.
.....................................................................
[setmpside]
  PURPOSE:
    Specific to mpfit; the sidedness of the finite
    difference when computing numerical derivatives.
  INPUTS:
    integer;  this field
    can take four values:
     0 - one-sided derivative computed automatically
     1 - one-sided derivative (f(x+h) - f(x)  )/h
    -1 - one-sided derivative (f(x)   - f(x-h))/h
     2 - two-sided derivative (f(x+h) - f(x-h))/(2*h)
    Where H is the STEP parameter described above.  The
    "automatic" one-sided derivative method will chose a
    direction for the finite difference which does not
```

```
    violate any constraints.  The other methods do not
    perform this check.  The two-sided method is in
    principle more precise, but requires twice as many
    function evaluations.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getmpside]
  PURPOSE:
    Retrieves the value of mpside.
  INPUTS:
    None.
  OUTPUTS:
    integer;  this field
    can take four values:
     0 - one-sided derivative computed automatically
     1 - one-sided derivative (f(x+h) - f(x)  )/h
    -1 - one-sided derivative (f(x)   - f(x-h))/h
     2 - two-sided derivative (f(x+h) - f(x-h))/(2*h)
    Where H is the STEP parameter described above.  The
    "automatic" one-sided derivative method will chose a
    direction for the finite difference which does not
    violate any constraints.  The other methods do not
    perform this check.  The two-sided method is in
    principle more precise, but requires twice as many
    function evaluations.
  SIDE EFFECTS:
    None.
.....................................................................
[setcoupled]
  PURPOSE:
    Set the parameter to utilise a coupling object to determine it's value.
  INPUTS:
    object reference; a coupling object.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getcoupled]
  PURPOSE:
    Retrieves the coupling object reference.
  INPUTS:
    None.
  OUTPUTS:
    object reference; a coupling object (null object signifies no coupling
    or error with coupling object).
  SIDE EFFECTS:
    None.
.....................................................................
[uncouple]
  PURPOSE:
    Destroys coupling object - uncouples this parameter.
  INPUTS:
    None.
```

```
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[setmpmaxstep]
  PURPOSE:
    Sets the maximum change allowed in parameter value in one fitting
    iteration.
  INPUTS:
    double; the maximum change to be made in the parameter value.  During the
    fitting process, the parameter will never be changed by more than this
    value in one iteration. A value of 0 indicates no maximum.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getmpmaxstep]
  PURPOSE:
    Retrieves the maximum change allowed in parameter value in one fitting
    iteration.
  INPUTS:
    None.
  OUTPUTS:
    double; the maximum change to be made in the parameter value.  During the
    fitting process, the parameter will never be changed by more than this
    value in one iteration. A value of 0 indicates no maximum.
  SIDE EFFECTS:
    None.
.....................................................................
[setmpprint]
  PURPOSE:
    Specific to mpfit; sets whether this parameter value will be printed by
    the default mpfit 'iterproc' that is executed each iteration.
  INPUTS:
    integer, 0 or 1; if set to 1, then the default ITERPROC (see mpfit) will
    print the parameter value. If set to 0, the parameter value will not be
    printed. This tag can be used to selectively print only a few parameter
    values out of many.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getmpprint]
  PURPOSE:
    Retrieves the value of mpprint.
  INPUTS:
    None.
  OUTPUTS:
    integer, 0 or 1; if set to 1, then the default ITERPROC (see mpfit) will
    print the parameter value. If set to 0, the parameter value will not be
    printed. This tag can be used to selectively print only a few parameter
    values out of many.
  SIDE EFFECTS:
    None.
```

```
.....................................................................
[setproperty]
  PURPOSE:
    Provides a common method for setting all object properties.
    The appropriate set method is called depending upon the keyword used.
  INPUTS:
    There are keywords for each of the properties to set (name, parname,
    value, error, fixed, limited, hardlimits, hardmin, hardmax, limits,
    step, relstep, mpside, mpmaxstep, coupled, mpprint). The appropriate
    value should be supplied via these (see the individual set method for
    details).
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getproperty]
  PURPOSE:
    Provides a common method for setting all object properties.
    The appropriate set method is called depending upon the keyword used.
  INPUTS:
    Set the appropriate keyword from the following list to retrieve the
    property:
        name, parname, value, error, fixed, limited, hardlimits, hardmin,
        hardmax, limits, step, relstep, mpside, mpmaxstep, coupled, mpprint
  OUTPUTS:
    The value of the requested property (see individual get methods for details).
  SIDE EFFECTS:
    None.
.....................................................................
```

Listing D.5: Public methods for class *ffs_prop*

```
PUBLIC METHODS:
In addtion to the methods listed below, this object inherits methods from ffs_primitive
 - refer to this class' documentation for more details.

[setname]
  PURPOSE:
    Sets property name.
  INPUTS:
    String - the name to give to the property.
  OUTPUTS:
    Integer - Returns 1 if successful.
              Returns 0 if unsuccessful.
  SIDE EFFECTS:
    None
[getname]
  PURPOSE:
    Returns the property's name.
  INPUTS:
    None.
  OUTPUTS:
    String - property name.
  SIDE EFFECTS:
    None
[setvalue]
```

```
    Sets property value.
  INPUTS:
    Any data type.
  OUTPUTS:
    Integer - Returns 1 if successful.
              Returns 0 if unsuccessful.
  SIDE EFFECTS:
    None
[getvalue]
  PURPOSE:
    Returns the property's value.
  INPUTS:
    None.
  OUTPUTS:
    Unknown - returns whatever data the value pointer refers to.
```

Listing D.6: Public methods for class *ffs_contain*

```
PUBLIC ROUTINES:
In addtion to the methods listed below, this object inherits methods from
ffs_primitive - refer to this class' documentation for more details.
.....................................................................
[get]
  PURPOSE:
    Retrieves specified objects from the container.
  INPUTS:
    all         -   integer(switch); enable to return all objects
    isa         -   string; filters the array returned by the all keyword,
                    returning only the objects that inherit from the
                    specfied class.
    position    -   integer; position of object in container.
    name        -   string; FFS name of sought object.
    count       -   integer; (outward); the number of objects returned.
  OUTPUTS:
    object reference; array of the specified object references.
  SIDE EFFECTS:
    None.
.....................................................................
[getnames]
  PURPOSE:
    Checks if input array elements are ffs_element object references.
  INPUTS:
    None.
  OUTPUTS:
    string; array of object names
  SIDE EFFECTS:
    None.
.....................................................................
[nametoref]
  PURPOSE:
    Retrieves object reference for specified FFS name.
  INPUTS:
    name        -   string; FFS name of sought object.
  OUTPUTS:
    object reference; of the named object.
  SIDE EFFECTS:
     None.
```

```
.....................................................................
[add]
  PURPOSE:
    Adds object reference to the container.
  INPUTS:
    objects     -   object reference; of the object to be added.
    position    -   integer; position within the container, at which the
                    object should be added.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
.....................................................................
[remove]
  PURPOSE:
    Removes specified object reference from the container.
  INPUTS:
    objref      -   object reference; of the object to be removed.
    name        -   string; FFS name of object sought for removal.
    position    -   integer; position of object in container.
    all         -   integer(switch); enable to remove all objects
    destroy     -   integer(switch); enable to destroy specified objects
                    in addition to simply removing their reference from the
                    container.
  OUTPUTS:
    Returns a 1 or a 0 to signify success and failure respectively.
  SIDE EFFECTS:
    Will destroy specified objects if "destroy" keyword set.
.....................................................................
```

Listing D.7: Public methods for class *ffs_parser*

```
PUBLIC ROUTINES:
setdefinition
  PURPOSE:
    Sets the model from a string.
  INPUTS:
    String or stringarray with valid model
  OUTPUTS:
    Returns 1 if model parsed and validated
    Returns 0 if error occurred, use geterrmsg
    method for details of the error.
  SIDE EFFECTS:
    None

setfile
  PURPOSE:
    Reads a model from a specified file.
  INPUTS:
    Filename to read from.
  OUTPUTS:
    Returns 1 if model is successfully parsed etc.
    Returns 0 if error occurred, use geterrmsg
    method for details of the error.
  SIDE EFFECTS:
    None but note that if the file changes another
    call to setfile must be done in order for the
    model parser to realise.
```

```
apply
  PURPOSE:
    Applies information in parser to a model object.
  INPUTS:
    An ffs_model object to apply to.
  OUTPUTS:
    Returns 1 if model is successfully parsed etc.
    Returns 0 if error occurred, use geterrmsg
    method for details of the error.
  SIDE EFFECTS:
    No direct side effects but creates elements
    and calls methods of ffs_model which may
    have side effects.

Routines below this point are public but it is unlikely that
they will be required under normal usage of the parser. The
below routines make it possible to interface ffs_parser with
an element manager distinct from ffs_model.

setelements
  PURPOSE:
    Gives the parser the object references for
    each individual element.
  INPUTS:
    elements - objarr of object references
    in the same order as given out by the
    required_elements method
  OUTPUTS:
    Always returns 1
  SIDE EFFECTS:
    None

info
  PURPOSE:
    Returns the elements required to generate
    the model.
  INPUTS:
    None
  OUTPUTS:
    Returns a structure:
            .title:         Overall model title
            .elements:      Array of required elements
            .optional:      Array of optional parameters
                            to these elements at
                            creation time.
            .name:          User-supplied name of the
                            element
  SIDE EFFECTS:
    None

geterrmsg
  PURPOSE:
    Gets error state.
  INPUTS:
    None.
  OUTPUTS:
```

```
    Returns string of error explanation of last
    error, typically set when a function returned
    a zero.
  SIDE EFFECTS:
          None
  INHERITED FROM:
    ffs_primitive

getmodel
  PURPOSE:
    Returns model string
  INPUTS:
    Keyword /strip: Returns model as one string
    with all whitespace reduced to one space.
    Otherwise, model will be returned as it was
    given to the object either as a strarr or as
    it appeared in the file.
  OUTPUTS:
    The string representation of the model.
  SIDE EFFECTS:
    None

getfile
  PURPOSE:
    Gives the filename which was read.
  INPUTS:
    None
  OUTPUTS:
    Filename which was read.
  SIDE EFFECTS:
    None

setvalues
  PURPOSE:
    Sets the values of parameters if declerations
    of their values were set within the original
    model string.
  INPUTS:
    None (gets input from modelstring which was
    passed in via setfile or setdefinition methods)
  OUTPUTS:
    Returns 1 if successful.
    Returns 0 if unsuccessful.
  SIDE EFFECTS:
    Changes element parameter values.

setchildren
  PURPOSE:
    Sets the children of the various elements.
  INPUTS:
    None (gets input from modelstring which was
    passed in via setfile or setdefinition methods)
  OUTPUTS:
    Returns 1 if successful.
    Returns 0 if unsuccessful.
  SIDE EFFECTS:
    Changes the children of each element
```

```
setlimits
  PURPOSE:
    Sets the limits of parameters if declerations
    of their limits were set within the original
    model string.
  INPUTS:
    None (gets input from modelstring which was
    passed in via setfile or setdefinition methods)
  OUTPUTS:
    Returns 1 if successful.
    Returns 0 if unsuccessful.
  SIDE EFFECTS:
    Changes element parameter limits.

setcouple
  PURPOSE:
    Sets the coupling of parameters if declerations
    of their coupling were set within the original
    model string.
  INPUTS:
    None (gets input from modelstring which was
    passed in via setfile or setdefinition methods)
  OUTPUTS:
    Returns 1 if successful.
    Returns 0 if unsuccessful.
  SIDE EFFECTS:
    Creates coupling object and passes it's
    reference to the targetted parameter.

setfixedfree
  PURPOSE:
    Sets the various parameters to fixed or
    free if declerations of this was set
    within the original model string.
  INPUTS:
    None (gets input from modelstring which was
    passed in via setfile or setdefinition methods)
  OUTPUTS:
    Returns 1 if successful.
    Returns 0 if unsuccessful.
  SIDE EFFECTS:
    Changes parameter switch via setfixed() method.

setloglinear
  PURPOSE:
    Sets the various parameters to have a
    log or linear dependence from the point
    of view of later fitting.
  INPUTS:
    None (gets input from modelstring which was
    passed in via setfile or setdefinition methods)
  OUTPUTS:
    Returns 1 if successful.
    Returns 0 if unsuccessful.
  SIDE EFFECTS:
    Changes parameter switch via setfixed() method.
```

```
docmds
  PURPOSE:
    Wrapper to setlimittvalues, sets, secouple,
    setfixedfree and seinear. Also tloglindirectly
    sets a flag to say er or not anwhethything
    was found.
  INPUTS:
    None (gets input from modelstring which was
    passed in via setfile or setdefinition methods)
  OUTPUTS:
    Returns 1 if successful.
    Returns 0 if unsuccessful.
  SIDE EFFECTS:
    Changed parameters via subordinate methods
    Resets the number of expressions found.

getexprfound
  PURPOSE:
    Gets whether or not an expression has been found
    used to find if the last call to docmds actually
    did anything or not.
  INPUTS:
    None
  OUTPUTS:
    Returns > 0 if expressions were found.
    Returns 0 if no expressions were found.
  SIDE EFFECTS:
    None but note this is not an alternative
    to checking the error state of docmds.
    docmds will return 0 if a command
    failed, whereas this variable will be
    0 if no command was found at all.

getdefinition
  PURPOSE:
    Gets a string from the model.
  INPUTS:
    A model reference.
    Tabstop (optional keyword) - The number of
            spaces to use for indentation
  OUTPUTS:
    Returns a string or stringarray with a
    valid model string.
  SIDE EFFECTS:
    None

writedefinition
  PURPOSE:
    Writes a model file from a model.
  INPUTS:
    A model reference.
    A filename
    Tabstop (optional keyword) - The number of
            spaces to use for indentation
  OUTPUTS:
    Returns 1 if successful.
```

```
    Returns 0 if unsuccessful.
  SIDE EFFECTS:
    Writes a file
```

Listing D.8: Public methods for class *ffs_couple*

```
 PUBLIC ROUTINES:
  In addtion to the methods listed below, this object inherits methods from
  ffs_primitive - refer to this class' documentation for more details.

[getvalue]
  PURPOSE:
    Evaluates the coupling expression.
  INPUTS:
    None
  OUTPUTS:
    Returns the value obtained from evaluating the coupling expression.
  SIDE EFFECTS:
    None
[setexpr]
  PURPOSE:
    Sets the coupling expression.
  INPUTS:
    A string defining the coupling string, in the prescribed format
    (see 'EXPLANATION' above)
  OUTPUTS:
    Returns 1 if successful.
    Returns 0 if unsuccessful.
  SIDE EFFECTS:
    None
```

Listing D.9: Public methods for class *ffs_simplify*

```
PUBLIC METHODS:
In addtion to the methods listed below, this object inherits methods from
ffs_primitive - refer to this class' documentation for more details.
.....................................................................
[apply]
  PURPOSE:
    main method to invoke simplification on a model reference.
  INPUTS:
    ref         -   object reference for the ffs_model to be 'simplified'.
  OUTPUTS:
    A reference to the new simplified ffs_model.
  SIDE EFFECTS:
    Creates a new ffs_model. This will involve creation of (amongst others)
    ffs_model, ffs_element, ffs_par ffs_contain, ffs_prop and ffs_couple
    objects.
.....................................................................

PRIVATE METHODS:
Although IDL does not support private routines, these are labelled as such
As they are not envisaged that users will typical call these methods
directly.
.....................................................................
[getrules]
```

```
  PURPOSE:
    Returns the rule list.
  INPUTS:
    None.
  OUTPUTS:
    Array of structures that represent the individual rules.
    The structures have the following fields:
        parent      -   string | parent object class,
        child       -   string | child object class,
        replacement -   string | replacement object class,
        coupling    -   pointer| to an 2 x 'number of replacement pars'
                               string array that details coupling
                               between the original parent & child pars
                               and the new replacement.
  SIDE EFFECTS:
    None.
.....................................................................
[setrules]
  PURPOSE:
    Sets up the rule list to be used for element combination replacement.
  INPUTS:
    None.
  OUTPUTS:
    returns         -   integer | returns a 1 or a 0 to signify success and
                        failure respectively.
  SIDE EFFECTS:
.....................................................................
[cleanuprules]
  PURPOSE:
    Frees up pointers created in defining rule list (Including the rule list
    pointer itself).
  INPUTS:
    None.
  OUTPUTS:
    returns         -   integer | a 1 or a 0 to signify success and failure
                        respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[checkrules]
  PURPOSE:
    Checks the rule list for a particular parent/child combination.
  INPUTS:
    ref             -   object reference | parent element,
    child           -   object reference | child element.
  OUTPUTS:
    returns         -   integer | the index of the rule list identified as
                        corresponding to the input parent/child combination.
                        If no rule is found, a value of '-1' is returned.
  SIDE EFFECTS:
.....................................................................
[expand_add]
  PURPOSE:
    Takes an add element reference and checks through its evaluation
    hierarchy for further add elements and extracts their child elements
    forming a new list of children i.e. removes unnecessary add elements.
  INPUTS:
```

```
    ref             -   object reference | ffs_add element to expand.
  OUTPUTS:
    returns         -   object reference | array of the new children of the
                        expanded ffs_add.
    count           -   integer (keyword) | provides number of references
                        being returned.
  SIDE EFFECTS:
    None.
.....................................................................
[expand_broaden]
  PURPOSE:
    To deal with the case of an 'ffs_broaden_*' element operating on an
    'ffs_add' element.
  INPUTS:
    ref             -   object reference | ffs_broaden_* element to expand.
  OUTPUTS:
    returns         -   object reference | new representation of the
                        original 'ref'. obj_new() is returned to signify
                        failure i.e. invalid object reference.
  SIDE EFFECTS:
    Creates new 'ffs_broaden_*' and 'ffs_add' objects.
.....................................................................
[duplicate]
  PURPOSE:
    To create a copy of the requested ffs_element. Including parameter
    values and coupling.
  INPUTS:
    ref             -   object reference | ref of the element to be
                        duplicated.
    no_children     -   keyword | if set then the child elements of ref will
                        not be duplicated and set.
  OUTPUTS:
    returns         -   object reference | the duplicated element. obj_new()
                        is returned to signify failure i.e. invalid object
                        reference.
  SIDE EFFECTS:
    Creates an object of the same class as 'ref'.
.....................................................................
[bumpcplexpr]
  PURPOSE:
    Increments the markers in a given coupling expression.
  INPUTS:
    expr            -   string | the couple expression to modify.
    increment       -   integer | number by which to increase the markers.
    exclude1st      -   keyword | if set, does not increment the first
                        marker.
  OUTPUTS:
    returns         -   integer | a 1 or a 0 to signify success and failure
                        respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[couple]
  PURPOSE:
    Creates a coupling object and sets it for the input destination.
  INPUTS:
    dest            -   object reference | par to be coupled.
```

```
    source          -   object reference | pars to couple 'dest' to.
    expr            -   string (keyword) | expression defining how the 'dest'
                        par is to be coupled.
  OUTPUTS:
    returns         -   integer | returns a 1 or a 0 to signify success and
                        failure respectively.
  SIDE EFFECTS:
    Creates an 'ffs_couple' object.
.....................................................................
[replace]
  PURPOSE:
    Provides an appropriate replacement object for the input reference.
  INPUTS:
    ref             -   object reference | element to be replaced.
  OUTPUTS:
    returns         -   object reference | replacement element
  SIDE EFFECTS:
    Potentially calls some / all of: 'expand_add', 'expand_broaden', 'duplicate'
    and 'couple'. These have object creation side effects.
.....................................................................
```

Listing D.10: Public methods for class *ffs_fit*

```
PUBLIC METHODS:
In addtion to the methods listed below, this object inherits methods from
ffs_primitive - refer to this class' documentation for more details.
.....................................................................
[apply]
  PURPOSE:
    Main method to begin fitting model to data (uses a Levenberg-Marquardt
    algorithm).
  INPUTS:
    model           -   object reference for the ffs_model to be used in
                        fit.
    data            -   object reference for the ffs_data to be used in fit
                        (temporarily this will actually be a structure with
                        three fields 'x', 'y' and 'error'.
  OUTPUTS:
    Returns         -   integer | returns a 1 or a 0 to signify success and
                        failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getstatus]
  PURPOSE:
    Gets the fit status - indicates the manner in which the fit completed.
  INPUTS:
    None.
  OUTPUTS:
    Returns         -   integer |
                        0   "Fail."
                        1   "Reached relative tolerance (difference in
                            successive chisq) 'reltol'."
                        2   "Reached max number of iterations 'maxiter'."
  SIDE EFFECTS:
    None.
.....................................................................
```

```
[getstatusmsg]
  PURPOSE:
    Returns the message associated with a given fit status.
  INPUTS:
    status          -   integer | the status number to get the message for.
                        if none supplied, current status will be used.
  OUTPUTS:
    Returns         -   string | the requested status message.
  SIDE EFFECTS:
    None.
.....................................................................
[setmaxiter]
  PURPOSE:
    Sets maximum number of iterations the fitting routine will perform.
  INPUTS:
    maxiter         -   integer | maximum number of iterations.
  OUTPUTS:
    Returns         -   integer | returns a 1 or a 0 to signify success and
                        failure respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[getmaxiter]
  PURPOSE:
    Returns the maximum number of iterations that the fitting routine will
    perform.
  INPUTS:
    None.
  OUTPUTS:
    Returns         -   integer | maximum number of iterations.
  SIDE EFFECTS:
    None.
.....................................................................
[geterrors]
  PURPOSE:
    Returns the errors in the fitted parameter values.
  INPUTS:
    None.
  OUTPUTS:
    Returns         -   double | errors in the fitted parameter values.
                        Value of !values.d_nan signifies failure.
  SIDE EFFECTS:
    None.
.....................................................................
[getcor]
  PURPOSE:
    Returns the parameter correlation matrix.
  INPUTS:
    None.
  OUTPUTS:
    Returns         -   double | matrix of correlation between the parameters.
                        Value of !values.d_nan signifies failure.
  SIDE EFFECTS:
    None.
.....................................................................
[getscovar]
  PURPOSE:
```

```
    Returns the scaled parameter covariance matrix.
  INPUTS:
    None.
  OUTPUTS:
    Returns         -   double | matrix of covariances between the (scaled)
                        free parameters.
                        Value of !values.d_nan signifies failure.
  SIDE EFFECTS:
    None.
.....................................................................
[getcurvm]
  PURPOSE:
    Returns the curvature matrix.
  INPUTS:
    None.
  OUTPUTS:
    Returns         -   double | curvature matrix.
                        Value of !values.d_nan signifies failure.
  SIDE EFFECTS:
    None.
.....................................................................
[getchisqder]
  PURPOSE:
    Returns the matrix comprising of the values of the partial derivatives
    (wrt each of the parameters) of chi-square.
  INPUTS:
    None.
  OUTPUTS:
    Returns         -   double | chisqder matrix.
                        Value of !values.d_nan signifies failure.
  SIDE EFFECTS:
    None.
.....................................................................
[getdof]
  PURPOSE:
    Returns the number of degrees of freedom of the fit (number of data
    points - number of free parameters).
  INPUTS:
    y               -   data (really just any array of number of data
                        points in length though). OPTIONAL (required with p)
    p               -   free parameters (really just any array with number
                        of parameters in length though).OPTIONAL (required with y)
  OUTPUTS:
    Returns         -   double | degrees of freedom.
                        Value of -1 signifies failure.

  SIDE EFFECTS:
    None.
.....................................................................
[getnchisq]
  PURPOSE:
    Returns the normalised chi-square value (chi-square/degrees of freedom).
  INPUTS:
    None.
  OUTPUTS:
    Returns         -   double | normalised chi-square.
                        Value of !values.d_nan signifies failure.
```

```
  SIDE EFFECTS:
    None.
------------------------------------------------------------------------
[getchisq]
  PURPOSE:
    Returns the chi-square value.
  INPUTS:
    None.
  OUTPUTS:
    Returns         -   double | chi-square.
                        Value of !values.d_nan signifies failure.
  SIDE EFFECTS:
    None.
------------------------------------------------------------------------
[getp]
  PURPOSE:
    Returns the fitted parameter values.
  INPUTS:
    None.
  OUTPUTS:
    Returns         -   double | parameter values.
                        Value of !values.d_nan signifies failure.
  SIDE EFFECTS:
    None.

PRIVATE METHODS:
Although IDL does not support private routines, these are labelled as such
As it is not envisaged that users will typical call these methods
directly.
------------------------------------------------------------------------
[setsvals]
  PURPOSE:
    Sets the scaled parameter values.
  INPUTS:
    svals           -   double | scaled parameter values.
  OUTPUTS:
    Returns         -   integer | 1 (there is no error checking for this
                        method).
  SIDE EFFECTS:
    None.
------------------------------------------------------------------------
[getsvals]
  PURPOSE:
    Returns the scaled parameter values.
  INPUTS:
    None.
  OUTPUTS:
    Returns         -   double | scaled parameter values (there is no error
                        checking for this method).

  SIDE EFFECTS:
    None.
------------------------------------------------------------------------
[setslimits]
  PURPOSE:
    Sets the scaled parameter value limits.
  INPUTS:
```

```
    svals           -   double | scaled parameter value limits.
  OUTPUTS:
    Returns         -   integer | 1 (there is no error checking for this
                        method).
  SIDE EFFECTS:
    None.
.....................................................................
[getslimits]
  PURPOSE:
    Returns the scaled parameter value limits.
  INPUTS:
    None.
  OUTPUTS:
    Returns         -   double | scaled parameter value limits (there is no
                        error checking for this method).

  SIDE EFFECTS:
    None.
.....................................................................
[setivals]
  PURPOSE:
    Sets the initial parameter values.
  INPUTS:
    svals           -   double | initial parameter values.
  OUTPUTS:
    Returns         -   integer | 1 (there is no error checking for this
                        method).
  SIDE EFFECTS:
    None.
.....................................................................
[getivals]
  PURPOSE:
    Returns the initial parameter values.
  INPUTS:
    None.
  OUTPUTS:
    Returns         -   double | initial parameter values (there is no error
                        checking for this method).

  SIDE EFFECTS:
    None.
.....................................................................
[scalepd]
  PURPOSE:
    Determines the partial derivative of the model w.r.t. the scaled
    parameters, given the partial derivatives w.r.t. the real parameter
    values. Essentially applies the chain rule for derivatives.
  INPUTS:
    pd              -   partial derivatives of the model w.r.t. the real
                        parameter values.
    pars            -   object ref | array of ffs_par object references.
  OUTPUTS:
    spd             -   the scaled partial derivaties.
  SIDE EFFECTS:
    None.
.....................................................................
[unscale]
```

```
  PURPOSE:
    Recovers true parameter values from the scaled values used to improve
    numerical stability of the fitting algorithm.
  INPUTS:
    pars            -   object ref | array of ffs_par object references.
  OUTPUTS:
    Returns         -   integer | a 1 or a 0 to signify success and failure
                        respectively.
  SIDE EFFECTS:
    Calls 'setp' to store unscaled values.
.....................................................................
[scale]
  PURPOSE:
    Scales parameter values such all are of the same order to improve
    numerical stability of the fitting algorithm.
  INPUTS:
    pars            -   object ref | array of ffs_par object references.
  OUTPUTS:
    Returns         -   integer | a 1 or a 0 to signify success and failure
                        respectively.
  SIDE EFFECTS:
    Calls 'setsvals', 'setslimits' and 'setivals'. To store relevant
    quantities.
.....................................................................
[setstatus]
  PURPOSE:
    Sets the fit status - the manner in which the fit completed.
  INPUTS:
    status          -   integer | the fit status.
  OUTPUTS:
    Returns         -   integer | a 1 or a 0 to signify success and failure
                        respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[setcurvm]
  PURPOSE:
    Stores the curvature matrix.
  INPUTS:
    curvm           -   double | curvature matrix.
  OUTPUTS:
    Returns         -   integer | a 1 or a 0 to signify success and failure
                        respectively.
  SIDE EFFECTS:
    None.
.....................................................................
[calcmat]
  PURPOSE:
    Calculates the curvature matrix and the chi-square derivative matrix
    (w.r.t. each parameter).
  INPUTS:
    y               -   double | data intensity values.
    yerror          -   double | error in data intensity values.
    f               -   double | modelled intensity values.
    nfree           -   integer | number of free parameters.
    pd              -   double | partial derivatives of the model wrt free.
                        free parameters, for each data point.
```

```
  OUTPUTS:
    Returns         -   integer | a 1 or a 0 to signify success and failure
                        respectively.
    Keywords:
    curvm           -   double | curvature matrix.
    chisqder        -   double | chisqder matrix.
  SIDE EFFECTS:
    None.
.....................................................................
[setdof]
  PURPOSE:
    Sets / calculates
  INPUTS:
    y               -   double | data intensity values.
    yerror          -   double | error in data intensity values.
    f               -   double | modelled intensity values.
    nfree           -   integer | number of free parameters.
    pd              -   double | partial derivatives of the model wrt free.
                        free parameters, for each data point.

  OUTPUTS:
    Returns         -   integer | a 1 or a 0 to signify success and failure
                        respectively.
    Keywords:
    curvm           -   double | curvature matrix.
    chisqder        -   double | chisqder matrix.
  SIDE EFFECTS:
    None.
```

Listing D.11: Public methods for class *ffs_executor*

```
PUBLIC ROUTINES:
execute
  PURPOSE:
    Executes an FFS cmd on a model
  INPUTS:
    String containing command, may or
    may not be surrounded by brackets
  OUTPUTS:
    Returns 1 if execution ok
    Returns 0 if error occurred, use geterrmsg
    method for details of the error.
  SIDE EFFECTS:
    None

setmodel
  PURPOSE:
    Sets the model to use
  INPUTS:
    The model
  OUTPUTS:
    Returns 1 if model is succesfully used etc.
    Returns 0 if error occurred, use geterrmsg
    method for details of the error.
  SIDE EFFECTS:
    If the model changes then the executor will
```

```
    not notice, it needs to be explicitly
    re-initalised via the setmodel method.
```

Appendix E

ADAS Glossary

Taken from the ADAS manual [8]:
The Atomic Data and Analysis Structure (ADAS) is an interconnected set of computer codes and data collections for modelling the radiating properties of ions and atoms in plasmas and for assisting in the analysis and interpretation of spectral measurements. The three components of the package are an interactive system, a library of key subroutines, and a very large database of fundamental and derived atomic data. The interactive part provides immediate display of important fundamental and derived quantities used in analysis together with a substantial capability for preparation of derived data. It also allows exploration of parameter dependencies and diagnostic prediction of atomic population and plasma models. The second part is non-interactive but provides a set of subroutines which can be accessed from the user's own codes to draw in necessary data from the derived ADAS database. The database spans most types of data required for fusion and astrophysical application.

E.1 Acronyms

ADAS uses a number of acronyms and mnemonics for certain data formats and sub-classes. There are also some simple abbreviations. These are quite widely used in the fusion community but may not be generally familiar. They include:

- *ADF*: ADAS data format
- *CXS*: charge exchange spectroscopy, normally beam driven
- *MSE*: motional Stark effect, affecting beam atom emission

- *RR*: radiative recombination - ADF08
- *DR*: dielectronic recombination - ADF09
- *CR*: collisional–radiative coefficients - ADF11. Subclasses:
  - *ACD*: effective recombination
  - *SCD*: effective ionisation
  - *CCD*: effective charge exchange recombination
  - *PLT*: effective low-level line power
  - *PRB*: effective recom + Brems power
  - *PRC*: effective charge exchange power
- GCR - generalised collisional–radiative coefficients - ADF11. Subclasses as CR +:
  - *QCD*: effective metastable cross-coupling
  - *XCD*: effective parent metastable cross-coupling
- *CD*: collisional-dielectronic (synonym for collisional–radiative)
- *QEF*: effective emission coefficient (from CXS normally) - ADF12
- *SXB*: ionisation per photon coefficient - ADF13
- *PEC*: photon emissivity coefficient - ADF15
- *BMS*: beam stopping coefficient - ADF21
- *BME*: beam emission coefficient - ADF22

E.2 ADAS Main Program Series

A summary of the main ADAS program suite is given here. The list here is correct for the current version of ADAS (version 3.1) but note that some of the entries are placeholders for future development and are subject to change. Refer to the latest version of the ADAS manual [8] and / or webpage for up-to-date information.

ADAS Series 1 — Atomic Data Entry and Verification
ADAS101: Electron Impact Excitation Cross Section — Graphing and Rate Evaluation

ADAS102: Electron Impact Excitation Rate — Graphing and Interpolation
ADAS103: Dielectronic Recombination — Graphing and Interpolation
ADAS105: Electron Impact Ionisation Cross Section — Graphing and Rate Evaluation
ADAS106: Electron Impact Ionisation Rate — Graphing and Interpolation
ADAS107: Charge Exchange Cross Section — Graphing and Rate Evaluation
ADAS108: Electron Impact Excitation of Neutrals and Molecules — Graphing and Rate Evaluation

ADAS Series 2 — General Z Data and Population Processing
ADAS201: Specific Z Excitation File — Graph and Fit Coefficient
ADAS202: General Z Recom./Ionis. File — Extraction from General Z File
ADAS203: General Z Excitation File — Extraction from General Z File
ADAS204: Specific Z Recom./Ionis. File — Process ACD,SCD and Population
ADAS205: Specific Z Excitation File — Process Meta./Excit. Population
ADAS206: Specific Z Excitation File — Process Line/Total Power
ADAS207: Meta./Excit. Population File — Process Line Emissivities
ADAS208: Specific Z Excitation File — Advanced Population Processing
ADAS209: General Level Bundling File — Process Effective Collision Strengths
ADAS210: General Level Unbundling File — Process Effective Collision Strengths
ADAS211: Radiative Recombination — Process for Specific Ion File
ADAS212: Dielectronic Recombination — Process for Specific Ion File
ADAS213: Collisional Ionisation — Process for Specific Ion File
ADAS214: Escape Factors — Convert Specific ion File
ADAS215: Temperature Regrid — Convert Specific Ion File
ADAS216: Error Processing — Examine Errors in Specific ion Files

ADAS Series 3 — Charge Exchange Processing
ADAS301: QCX File — Graph and Fit Cross Section
ADAS302: IONATOM File — Graph and Fit Cross Section
ADAS303: QEF File — Graph and Fit Coefficient
ADAS304: BMS File — Graph and Fit Coefficient
ADAS306: QCX File — Process Effective Coefficient: J Resolved
ADAS307: QCX File — Process Effective Coefficient: J Resolved/Scan
ADAS308: QCX File — Process Effective Coefficient: L Resolved
ADAS309: QCX File — Process Effective Coefficient: L Resolved/Scan
ADAS310: BEAM STOPPING — Process Stopping Coefficient: H Beam
ADAS311: BEAM STOPPING — Process Stopping Coefficient: He Beam

ADAS312: BDN File — Tabulate and Graph BMS and BME data: H beam
ADAS313: BNL File — Tabulate and Graph BMS and BME data: He beam
ADAS314: QCX File — Convert QCX to effective cross sections
ADAS315: QCX File — Arbitrary species CX cross sections
ADAS316: QCX File — Bundle-n CX emissivity sections

ADAS Series 4 — Recombination and Ionisation Processing
ADAS401: Iso-Electronic Sequence — Graph and Fit Coefficient
ADAS402: Iso-Nuclear Sequence — Graph and Fit Coefficient
ADAS403: Iso-Electronic Master File — Merge Partial Iso-Elec. Files
ADAS404: Iso-Nuclear Master File — Extract from Iso-Elec. Master Files
ADAS405: Equilibrium Ionisation — Process Meta. Pops. and Emis. Funcs.
ADAS406: Transient Ionisation — Process Meta. Pops. amd Emis. Funcs.
ADAS407: Iso-Nuclear Param. Sets — Prepare Optimised Power Params.
ADAS408: Iso-Nuclear Master Data — Prepare from Iso-Nuc. Param. Sets
ADAS409: Equilibrium Ionisation — Prepare G(Te,Ne) Function Tables
ADAS410: Dielectronic Recombination — Graph and Fit Data
ADAS411: Radiative Recombination — Graph and Fit Data
ADAS412: Equilibrium Ionisation — Prepare G(Te) Function Tables
ADAS413: Collisional Ionisation — Graph and Fit Data
ADAS414: Prepare Soft X-ray Filter File
ADAS415: Display Spectral Filter File
ADAS416: Superstages — Repartition adf11 and emissivity datasets
ADAS417: Apply Filter to F-PEC Coefficients and F_G(Te,Ne) Functions

ADAS Series 5 — General ADAS Interrogation Routines
ADAS501: SXB File — Graph and Fit Ionizations per Photon
ADAS502: SZD File — Graph and Fit Ionization Rate-Coefficients
ADAS503: PEC File — Graph and Fit Photon Emissivities
ADAS504: PZD File — Graph and Fit Radiated Powers
ADAS505: QTX File — Graph and Fit Thermal Charge Exch. Coefft.
ADAS506: GFT File — Graph and Fit G(TE) Function
ADAS507: GCF File — Graph and Fit General. Contribution Function
ADAS508: GTN File — Graph and Fit G(TE,NE) Function
ADAS509: SCX File — Graph and Fit Charge Exchange Cross-sections
ADAS510: F-PEC File — Graph Envelope Feature Photon Emissivity Coefficients
ADAS511: F-GTN File — Graph Envelope Feature Photon Emissivity Function

ADAS Series 6 — Data Analysis Programs

ADAS601: Differential Emission Measure Analysis

ADAS602: Spectral Line Profile Fitting

ADAS603: Zeeman Feature and Spectral Line Profile Fitting

ADAS604: Dielectronic Satellite Line Profile Fitting

ADAS605: General ADAS feature inspection (AFG)

ADAS606: Helium Series Limit Feature Fitting

ADAS607: Non-Maxwellian Electron Distribution Function Fitting

ADAS608: Molecular Band Profile Fitting

ADAS Series 7 — Creating and Using Dielectronic Data

ADAS701: AUTOSTRUCTURE

ADAS702: Postprocessor — AUTOSTRUCTURE > adf09 file

ADAS703: Postprocessor — AUTOSTRUCTURE > adf04 file

ADAS704: Postprocessor — AUTOSTRUCTURE > adf18 file

ADAS705: Merge and Bundle Partial Files > adf04 file

ADAS706: Calculate Doubly Excited State Populations and Satellite Line Feature File

ADAS707: Doubly Excited Populations — Process Line Emissivities

ADAS Series 8 — Structure and Excitation Calculations

ADAS801: Cowan Structure Code — adf04 type 1 and 3

ADAS802: Calculate Distorted Wave Cross Sections

ADAS803: Post-Process Distorted Wave Cross Sections > adf04 file

ADAS804: Prepare Cross Sections and Rate Coefficients

ADAS805: Calculate Radiative Gaunt Factors

ADAS806: Merge, Clean and Check adf04 file

ADAS807: Prepare Cross-Referencing Files

ADAS808: Prepare driver files for ADAS801

ADAS809: Non-Maxwellian Modelling — Change adf04 file type

ADAS810: Generate Envelope Feature Photon Emissivity Coefficient

ADAS811: Graph and Compare adf04 Files

ADAS812: Compare adf04 Files (scatterplot)

E.3 ADAS Data Formats

ADF00: Ground configurations and ionisation potentials
ADF01: Bundle-n and bundle-nl charge exchange cross-sections
ADF02: Ion impact cross-sections with named participant
ADF03: Recombination, ionisation and power parameter sets
ADF04: Resolved specific ion data collections
ADF05: General z excitation data collections
ADF06: General z recombination/ionisation data collections
ADF07: Direct resolved electron impact ionisation data collections
ADF08: Direct resolved radiative recombination coefficients
ADF09: Direct resolved dielectronic recombination coefficients
ADF10: Iso-electronic master files
ADF11: Iso-nuclear master files
ADF12: Charge exchange effective emission coefficients
ADF13: Ionisation per photon coefficients
ADF14: Thermal charge exchange coefficients
ADF15: Photon emissivity coefficients
ADF16: Generalised contribution functions
ADF17: Condensed projection matrices
ADF18: Cross-referencing data
ADF19: Zero density radiative power
ADF20: G(Te) functions
ADF21: Effective beam stopping coefficients
ADF22: Effective beam emission coefficients
ADF23: State selective electron impact ionisation coefficients
ADF24: State selective charge transfer cross-sections
ADF25: Driver data-sets for ADAS204 calculations
ADF26: Bundle-n and bundle-nl populations of excited states in beams
ADF27: Driver data-sets for ADAS701 calculations
ADF28: Driver data-sets for ADAS702 calculations
ADF29: Driver data-sets for ADAS707 calculations
ADF30: Driver data-sets for ADAS708 postprocessing
ADF31: Feature files for spectral simulation
ADF32: Driver data-sets for ADAS802 calculations
ADF33: Driver data-sets for ADAS803 postprocessing
ADF34: Driver data-sets for ADAS801 calculations

ADF35: Spectral filter data

ADF36: Hydrogenic series limit feature file

ADF37: Non-Maxwellian distribution function files

ADF38: Opacity project extension: photoexcitation-autoionisation rate coefficients

ADF39: Opacity project extension: photoionisation cross-sections

ADF40: Envelope feature photon emissivity coefficients

ADF41: Driver data-sets for offline ADAS8#1 calculations

ADF42: Driver data-sets for ADAS810 calculations

ADF44: Envelope feature photon emissivity functions