FEFFIT

Using FEFF to model XAFS data

Contents

1	Introduction	1
2	Input and Output Files	3
3	Keywords and Controls for *feffit.inp*	5
4	Variables and Parameters in Fitting	11
5	Goodness of Fit and Uncertainties in the Variables	16
6	The XAFS Equation and FEFFIT Algorithms	21
A	Examples	26
B	Suggestions for Building Physical Models with FEFFIT	31
C	Program Notes	37
D	Simultaneous Fits of Multiple Data Sets	38

FEFFIT was written with the guidance and encouragement of Edward A. Stern. This program would not be possible without FEFF, written by John Rehr and Steve Zabinsky, who let me use some of their code and many of their ideas. I also thank Charles Bouldin, Anatoly Frenkel, Pēteris Līviņš, Maoxu Qian, Bruce Ravel, Hans Stragier, Fuming Wang, Yizhak Yacoby, and Yanjun Zhang for many useful discussions and helpful suggestions.

Matthew Newville
University of Chicago
GSE-CARS, Bldg 434A
APS, Argonne National Lab
Argonne, IL 60439
newville@cars.uchicago.edu
(630) 252-0431

FEFFIT version 2.32
updated: July 3, 1998

Introduction

FEFFIT will fit the calculations from FEFF to XAFS $\chi(k)$ data, giving a method of determining the local structure around an atom. As with all XAFS analysis programs, FEFFIT can determine interatomic distances, coordination numbers and atomic species of neighboring atoms. Finer details of atomic configurations, including detailed descriptions of two-body distribution functions and certain aspects of three-body correlations, are also measurable with FEFFIT. Experience has shown that the FEFF calculations are good enough to be used effectively to fit real XAFS data from a wide range of experimental systems. It should be kept in mind, however, that FEFFIT compares experimental XAFS to theoretical calculations. This requires some considerations that can be neglected when comparing experimental data to experimental standards.

FEFFIT uses the *feffnnnn.dat* files output from FEFF as the basis calculations with which to fit the XAFS data. In this way, FEFFIT is highly dependent on FEFF, so that a good understanding of the FEFF calculation is important in being able to use FEFFIT well. Each of the *feffnnnn.dat* files represents the $\chi(k)$ for a scattering path, created by FEFF (version 5 or higher) when the "Control Card" `MFEFF` is set to 1. There is no distinction between single- or multiple-scattering paths in FEFF or FEFFIT. Those unfamiliar with FEFF or the path expansion should refer to the FEFF documentation and standard XAFS references.

The XAFS contribution of each scattering path (read from each *feffnnnn.dat* file) is adjusted by applying standard XAFS parameters such as coordination number, change in distance, Debye-Waller Factor, and shift in energy origin) until the best-fit to the data is found. Standard numerical techniques are used to find the set of variables that minimizes the sum of squares of the difference between model and data χ and to estimate the uncertainties in the variables. Fitting can be done in either R-space or back-transformed k-space, with Fourier Transforms done by FEFFIT on both the XAFS data and the calculated model during the fitting process. Real and imaginary parts of the Fourier Transformed χ are used in the fit with equal weight. The model $\chi(k)$ used to compare to data is evaluated as a sum over paths

$$\chi_{\rm model}(k) = \sum_{\rm Paths} \chi_{\rm path}(k, {\rm Amp}(k), {\rm Phase}(k), {\rm Path\ Parameters}).$$

$\chi_{\rm path}$ is the the XAFS contribution for each path, and depends on the scattering amplitude and phase-shift from FEFF (from *feffnnnn.dat*) and on standard XAFS Path Parameters. These Path Parameters are the physical quantities used to alter the $\chi_{\rm path}$, such as refinements of the XAFS due to changes in the atomic distribution. FEFFIT allows the XAFS for each path to be adjusted by the following Path Parameters:

`e0`	shift of energy origin : $k \rightarrow \sqrt{k^2 - \mathtt{e0}(2m_e/\hbar^2)}$
`ei`	imaginary energy shift (to give additional broadening)
`S02`	constant amplitude factor
`delR`	change in distance (1st cumulant)
`sigma2`	mean-square-displacement (2nd cumulant), or Debye-Waller Factor
`third`	3rd cumulant (from anharmonicity in the atomic distribution)
`fourth`	4th cumulant (from anharmonicity in the atomic distribution)

Up to 100 scattering paths may be combined to fit the data. Since each path gets its own set of Path Parameters, there could be up to 700 potential variables in the fit. But XAFS data

contains much less information than this about the local atomic structure around the central atom (typically 10 to 20 variables can be determined in a fit) so constraints will need to be placed on some of the Path Parameters. There is no general way to decide what should be varied or what the best constraints are for a given system. With FEFFIT, the user chooses the variables in the fit, and writes mathematical expressions for the Path Parameters in terms of these user-chosen variables. This conceptual distinction between variables and XAFS Path Parameters gives a powerful ability to put constraints on system studied and to use more physically meaningful variables in the fit.

1.1 How to use this document

FEFFIT is a fairly complex XAFS fitting program with lots of bells and whistles, and a few concepts that may be new even to experienced XAFS analysts. This document is intended both as a reference manual for the experienced user and as a tutorial for those who are knowledgeable about XAFS but are new to FEFFIT. A new user should probably skim all the chapters, and then go through the worked examples in appendix A. It's a fairly simple XAFS problem, but it will get you started. After the examples have been examined, a more careful reading of chapter 4 and appendix B should give you ideas about how to apply FEFFIT to your own XAFS problems, and chapter 5 should help you interpret the fit results. The remaining chapters are mostly written for reference. Chapter 2 discusses data file formats. Chapter 3 gives the complete list of FEFFIT options and how to implement them. Chapter 6 gives the FEFFIT version of the XAFS equation and an overview of the mathematical algorithms of the program.

1.2 Considerations When Using Theoretical Standards

FEFFIT compares experimental XAFS data to a theoretical standard, which has become the preferred analysis method in the XAFS community. While theoretical standards are convenient and often more reliable than experimental standards, they do require some special considerations. These are discussed in greater detail in the XAFS literature, but will be outlined here.

The experimental $\chi(k)$ should be as free as possible from any systematic errors (such as detector saturation or glitches). The appropriate corrections (especially for data taken in fluorescence) to the data should be made before trusting the results from FEFFIT. Such systematic errors and corrections are more important when using theoretical standards rather than experimental ones, because they will tend to cancel out (at least to first order) when using experimental standards. The UWXAFS3.0 program ATOMS (written by Bruce Ravel) will calculate most of these corrections, as well make a good first draft of a `feff.inp` file for crystalline materials. The corrections given by ATOMS will be in the form of additional amplitude corrections which can be used in FEFFIT to give the FEFF calculation the same amplitude reduction as is expected for the data. See appendix B and the ATOMS documentation for more details.

The second consideration is that the FEFF calculations are not perfect, and make assumptions in the calculations (most notably the muffin-tin approximation of the atomic potentials) that might be inadequate for some systems. While FEFF and FEFFIT has been demonstrated to give good results on many "standard" compounds, it is still a good idea to measure a suitable experimental "standard" compound and to fit it with a FEFF calculation before trusting the results for a completely unknown structure. Such fits to experimental "standards" generally prove invaluable in pointing out the the best ways to modify the FEFF calculations, and therefore how to get the best information out of the system being studied.

Input and Output Files

2.1 Input Files

FEFFIT uses the input file *feffit.inp* to control the running of the program. If this file cannot be found, FEFFIT will stop and complain. The form and contents of *feffit.inp* will be further discussed in chapter 3. In addition, FEFFIT needs a set of *feffnnnn.dat* files (such as *feff0001.dat*) to use as the model paths for building the model $\chi(k)$. Finally, a file containing the $\chi(k)$ data to be fit may be specified. (If no data file is given, FEFFIT will simply combine the *feffnnnn.dat* according to the inputs, without any fitting, which makes a convenient and flexible alternative to FF2CHI, the final module of FEFF). Note that the data contains $\chi(k)$, not $\mu(E)$, not $\chi(k)$ that is k-weighted, and not $\tilde{\chi}(R)$. The data is expected to be given on an evenly spaced k-grid of 0.05Å, and will be interpolated onto this grid if it is not. The names of the *feffnnnn.dat* and input data files can be any file name (including subdirectory paths) allowed by the operating system up to 70 characters long. In summary, there are three inputs:

1. *feffit.inp*, the input file for the program.
2. A set of *feffnnnn.dat* files to sum to make the total $\chi(k)$.
3. A data file containing $\chi(k)$ for the data to be fit.

2.2 Running FEFFIT, Output Messages, Warnings, and Errors

You should be able to run FEFFIT by using the command `feffit` from any directory with a file named *feffit.inp*. As FEFFIT runs, messages will be sent to the screen telling what the program is doing, and may include warning or error messages. If FEFFIT does something you didn't expect or doesn't run to completion, these output messages will provide the best diagnostic clues about what happened. FEFFIT should never break without giving messages that will tell the user how to fix the problem. But if it does, please send me the input file and output messages along with your bitter complaints.

2.3 Output Files

As soon as the fit is done, FEFFIT will write *feffit.log*, giving a summary of the inputs (such as Fourier Transform parameters) and the fit results for the variables, Path Parameters, and so on. This file is the only place where the fitting information such as uncertainties in the fit variables will be given. Output data files for the input data, full final fit, and the fit contribution from each path will be written in k-space, R-space, and backtransformed k-space. The names and contents of the output data files will depend on the file format used as discussed in the next section. The documentation in the output data files for the individual paths will include the Path Parameters used for that path. Again, the outputs are:

1. Run time messages written to standard output.
2. *feffit.log*, giving all important numerical results.
3. $\chi(k)$, $\tilde{\chi}(R)$, and $\tilde{\chi}(k)$ for the data.
4. $\chi(k)$, $\tilde{\chi}(R)$, and $\tilde{\chi}(k)$ for the full model fit.
5. $\chi(k)$, $\tilde{\chi}(R)$, and $\tilde{\chi}(k)$ for the fit contribution from each path.

2.4 Data File Formats

As for all UWXAFS3.0 data analysis programs, there are two options for the format of the data files. The data may be in either a specially formatted binary file known as a UWXAFS file (also called an RDF file), or in a specially formatted ASCII column file. More information on these file formats, including the format specifications and a discussion of the relative merits of the two file formats can be found in the UWXAFS3.0 document ***files.doc***. The two file handling formats can be mixed in FEFFIT, so that the input data can be in the UWXAFS format and the output data can be in the ASCII format, or vice versa.

If the input data is in UWXAFS format, it must be in a file with file type 'CHI' (such as is output from AUTOBK). Both the file name and record key (either nkey or skey) must be specified for the input. Outputs files in the UWXAFS format will be written to files the data, full fit, and contribution from each path in sequential records. If the user specifies the output file name by `out = Test`, the output files will be ***Test.chi*** (with file type 'CHI') for all $\chi(k)$, ***Test.rsp*** (with file type 'RSP') for all $\tilde{\chi}(R)$, and ***Test.env*** (with file type 'ENV') for all $\tilde{\chi}(k)$.

If the input data is in ASCII format, it must be in a file with one or more document lines, followed by a *required* line of minus signs ('-'), followed by an ignored line (for column labels) , and then columns of numerical data for k, and $\chi(k)$. Note that each k value must be given, that the second column is not k-weighted $\chi(k)$ or separated magnitude and phase of $\chi(k)$, and that only one data pair can be given per line. Data past the second column will be ignored. The data will be linearly interpolated onto an even k-grid of 0.05 Å^{-1} before being used.

Output files in the ASCII format will each contain only one set of data, and will be named according to the requested file name, the contents of data, and the space in which the data is written. If the user specifies the output file name by `out = Test`, the output files will be ***Testk.dat***, ***Testr.dat***, and ***Testq.dat*** for the data $\chi(k)$, $\tilde{\chi}(R)$, and $\tilde{\chi}(k)$, ***Testk.fit*** and ***Testr.fit*** for the full fit $\chi(k)$, $\tilde{\chi}(R)$, and $\tilde{\chi}(k)$, and ***Testk.nnn*** and ***Testr.nnn*** for the $\chi(k)$, $\tilde{\chi}(R)$, and $\tilde{\chi}(k)$from path ***nnn***. Files for the complex data of $\tilde{\chi}(R)$ and $\tilde{\chi}(k)$ will have five columns. The first column will contain the abscissa (either R or k), followed by columns of real, imaginary, amplitude, and phase of the complex data.

Keywords and Controls for FEFFIT.INP

3.1 General Format of FEFFIT.INP

Input commands to FEFFIT will be read from the file *feffit.inp*. These inputs will name the data files and the *feffnnnn.dat* files to use for each path, describe what to vary in the fit, and how the variables will alter these XAFS contribution from each path. FEFFIT uses keywords to describe and assign values to all program parameters. The use of keywords allows the input file to be easily read and the values of the program parameters to be easily modified. The keywords usually have fairly transparent meanings. Most program parameters are assigned values with keyword "sentences" which have syntax:

`keyword` ⟨delimiter⟩ `value`.

The `keyword` must be one of the valid keywords listed below. The ⟨delimiter⟩ is an equal sign or white space (a blank or TAB) surrounded by any number of white spaces. The `value` is provided by the user and will be interpreted as a number, a logical flag, or a character string, depending on the nature of the keyword — the list below will indicate what kind of value each keyword takes. Logical flags all have values `true` or `false` (`t` and `f` will work, too). If a keyword's `value` is a number or logical flag (but not a character string), the assigning keyword sentence can be put on the same line as other numerical and logical keyword sentences. Keywords that take character strings as their value must occur on their own line.

FEFFIT does not distinguish keywords, variable names or character strings containing Math Expressions by case. To accommodate many operating systems, it does distinguish the names of external files by case. Keyword sentences are allowed to occur in any order in the file. Internal comments can be written anywhere in *feffit.inp*, including end-of-line comments. Inputs to FEFFIT can be read from files other than *feffit.inp* by using a keyword sentence of the form "`include myfile.inp`" inside *feffit.inp*. This allows commonly used assignments (such as for Path Parameters) to be kept in *myfile.inp* and to be used for many different fits.

A *feffit.inp* file has quite a bit of structure to it, and the listing of Path Parameters makes it look a bit like a spreadsheet or programming language. Because of this built-in structure, a tool can be developed to help write the *feffit.inp* files. One such tool exists in the form of special macros (called FEFFIT.EL, written by Bruce Ravel, and part of the UWXAFS3.0 distribution) for the Emacs text editor. It's quite useful, and if you use Emacs, I highly recommend trying it. I hesitate to say that you should learn Emacs just to use these macros, but it's a very good editor anyway.

There is some syntax checking in FEFFIT, and if it gets confused by any inputs, it will report this as a run-time message, telling which line of the input file caused the confusion, and trying to describe which words it did not understand. There is also some syntax checking of the math expressions used and the variables defined. The syntax checking isn't foolproof, but it does catch the most simple mistakes.

3.2 Summary of Keywords

Here is a brief list of all the keywords for all the program controls and parameters in FEFFIT with a brief description of the meaning of their values. The form of the values taken by these keywords are indicated by c, n, or l for character strings, numbers, and logical flags, respectively. Where appropriate, valid options for the values are given in parentheses and default values are given in brackets. The following sections give more detailed explanations for each keyword.

General/Miscellaneous:

`%/!`	-	End-of-Line Comment: ignore everything on the line after `%` or `!`	
`#`	-	Comment Line: ignore line if `#` is in 1st column	
`End`	-	End-of-File: ignore rest of input file, start fitting	
`Include`	c	name of file to read more input commands from	[none]

Data Input and Output:

`Title`	c	Title line to write to output files	[none]
`Formin`	c	File Format of input data file (uw/ascii)	[found from input data]
`Formout`	c	File Format of output data file (uw/ascii)	[same as input format]
`Format`	c	File Format of both input and output data file (uw/ascii)	[none]
`Data`	c	Name of input data file (containing $\chi(k)$)	[none]
`Out`	c	Name of output data file	[same as input]
`Allout`	l	Flag for writing outputs for all individual paths	(T/F) [T]
`Kspout`	l	Flag for writing $\chi(k)$ output files	(T/F) [T]
`Rspout`	l	Flag for writing $\tilde{\chi}(R)$ output files	(T/F) [T]
`Qspout`	l	Flag for writing $\tilde{\chi}(k)$ output files	(T/F) [T]
`Rlast`	l	Maximum R-value for $\tilde{\chi}(R)$ output file	[10.0]

Fitting Control Flags:

`Rspfit`	l	Flag for fitting in R-space	(T/F)[T]
`Qspfit`	l	Flag for fitting in backtransformed k-space	(T/F)[F]
`Nodegen`	l	Flag for not using the Path Degeneracies from FEFF.	(T/F)[F]
`Noout`	l	Flag for not writing any output data files	(T/F)[F]
`Nofit`	l	Flag for not fitting, use initial guesses as final values	(T/F)[F]
`Norun`	l	Flag for not fitting and not writing output files	(T/F)[F]

Fourier Transform Parameters and Fitting Ranges:

`Rmin`	n	$R_{\rm min}$ for R-space fit and $R \rightarrow k$ FT	[0.0]
`Rmax`	n	$R_{\rm max}$ for R-space fit and $R \rightarrow k$ FT	[0.0]
`Kmin`	n	$k_{\rm min}$ for k-space fit and $k \rightarrow R$ FT	[first data point]
`Kmax`	n	$k_{\rm max}$ for k-space fit and $k \rightarrow R$ FT	[last data point]
`Kweight`	n	k-weight for $k \rightarrow R$ FT	[1.0]
`Dk1`	n	Low-k Window Parameter for $k \rightarrow R$ FT	[0.0]
`Dk2`	n	High-k Window Parameter for $k \rightarrow R$ FT	[0.0]
`Dk`	n	Both `Dk1` and `Dk2`	[0.0]
`Dr1`	n	Low-R Window Parameter for $R \rightarrow k$ FT	[0.0]
`Dr2`	n	High-R Window Parameter for $R \rightarrow k$ FT	[0.0]
`Dr`	n	Both `Dr1` and `Dr2`	[0.0]
`Ikwindo`	n	Integer to select Window Function for $k \rightarrow R$ FT	[0, Hanning]
`Irwindo`	n	Integer to select Window Function for $R \rightarrow k$ FT	[0, Hanning]
`Iwindo`	n	Both `Ikwindo` and `Irwindo`	[0, Hanning]
`Mftwrt`	n	Number of array points in FFT for writing out data	[2048]
`Mftfit`	n	Number of array points in FFT for fit	[512 or 1024]

Error Analysis:

`Epsdat`	n	Measurement uncertainty in $\chi(k)$	[found from high-R]
`Epsr`	n	Measurement uncertainty in $\tilde{\chi}(R)$	[found from high-R]
`Cormin`	n	Smallest correlation to report in *`feffit.log`*.	[0.50]

Variables and User Defined Functions:

Each variable and User Defined Function must be on its own line

`Guess`	n	Initial guess for a variable	[0.0]
`Set`	c	Math Expression For a User Defined Function	[none]

Path Parameters:

Each Path Parameter must be on its own line with syntax
`Path Parameter` ⟨delimiter⟩ `Path Index` ⟨delimiter⟩ `Character String`
where `Path Index` indicates which path each parameter is assigned to

`Path`	c	Name of *`feffnnnn.dat`* file for this path	[none]
`Id`	c	User Identification Label for this path	[none]
`S02`	c	Constant amplitude factor	[1.0]
`E0`	c	E_0 shift	[0.0]
`Ei`	c	Imaginary energy shift (broadening)	[0.0]
`Delr`	c	ΔR, or 1st cumulant	[0.0]
`Sigma2`	c	mean-square displacement of path distance, σ^2	[0.0]
`Third`	c	Third Cumulant	[0.0]
`Fourth`	c	Fourth Cumulant	[0.0]

3.3 General and Miscellaneous Keywords

`%` or `!` indicates a comment anywhere in *`feffit.inp`*, including end-of-line comments.

`*` or `#` indicates a comment line in *`feffit.inp`* if it is the first character on the line.

`End` stop reading inputs, and ignore everything in *`feffit.inp`* after this line.

`Include` read more inputs from another file. The syntax is `include myfile.inp`, and can be used in *`feffit.inp`* for standard definitions, or to break up long input files. If you're fitting lots of similar data sets, you may find this useful.

3.4 Data Input and Output Keywords

`Title` user-chosen title line which will be written to the output files. 10 title lines can be used, each of up to 70 characters. Everything on a line after the keyword `title` will be included in the title, even if it contains other keywords.

`Formin` file format to use for the input data files. The choices are UWXAFS and ASCII. See chapter 2 and the UWXAFS3.0 document on data files for more details. The default is for FEFFIT to find the input format itself, from the input data. This does not need to be on its own line.

`Formout` file format to use for the output data files. The choices are the same as for `Formin`, and the default is to use the format used as the input format. This does not need to be on its own line.

`Format` sets both `Formin` and `Formout`.

`Data` input data file name. For UWXAFS format files, either the nkey or skey must also be given, so that the syntax must be something like:

`Data = cu.chi, 1` or `Chi = cu.chi , TROUT`.

For ASCII input data format, only the input file name is needed. This should be on its own line.

`Out` prefix for the output file name. See chapter 2 for more details, and an explanation of the file name suffixes. This does not need to be on its own line.

`Allout` logical flag for writing output data for the individual paths to all output data files. The default is True.

`Kspout` logical flag for writing output data $\chi(k)$, the XAFS in unfiltered k-space. The default is True.

`Rspout` logical flag for writing output data $\tilde{\chi}(R)$, the XAFS in R-space. The default is True.

`Qspout` logical flag for writing output data $\tilde{\chi}(k)$, the XAFS in backtransformed or filtered k-space. The default is True.

`Rlast` last R value for which output data $\tilde{\chi}(R)$ will be written. The default is 10.0 Å. Output k-space data will be written over the input data k-range.

3.5 Fitting Control Flags

`Rspfit` logical flag for fitting in R-space. The default is True.

`Qspfit` logical flag for fitting in backtransformed k-space. The default is False.

`Nodegen` logical flag for ignoring the path degeneracies in all the *`feffnnnn.dat`* files, effectively setting all degeneracies to 1. The default is False, so that the degeneracies in the *`feffnnnn.dat`* files are used.

`Nofit` logical flag for skipping the fit, so that the initial guesses of variables are used as final values. The log file and data outputs are written. This is useful for your own error checking, and effectively changes all `guesses` to `set`. The default is False, so that fitting is done.

`Noout` logical flag for not writing any data file outputs. The default is False, so that outputs are written.

`Norun` logical flag for both skip fitting and not writing any data file outputs. This has the same effect as setting both `Nofit` and `Noout`.

3.6 Fourier Transform Parameters and Fitting Ranges

`Rmin` low-R value of the R-space range for either R-space fit or for $R \rightarrow k$ Fourier Transform. The default is 0.

`Rmax` high-R value of the R-space range for either R-space fit or for $R \rightarrow k$ Fourier Transform. The default is 0.

`Kmin` low-k value of the k-space range for both k-space fit (for fits to unfiltered or filtered data) and for $k \rightarrow R$ Fourier Transform. The default is the first k-value of the data.

`Kmax` high-k value of the k-space range for both k-space fit (for fits to unfiltered or filtered data) and for $k \rightarrow R$ Fourier Transform. The default is the last k-value of the data.

`Kweight` k-weighting for the $k \rightarrow R$ Fourier Transform. The default is 1.

`Dk1` low-k Fourier Transform Window Parameter (window "sill") for the $k \rightarrow R$ Fourier Transform. The default is 0.0.

`Dk2` high-k Fourier Transform Window Parameter (window "sill") for the $k \rightarrow R$ Fourier Transform. The default is 0.0.

`Dk` sets both `Dk1` and `Dk2` to the same value. The default is 0.0.

`Dr1` low-R Fourier Transform Window Parameter (window "sill") for the $R \rightarrow k$ Fourier Transform. The default is 0.0.

`Dr2` high-R Fourier Transform Window Parameter (window "sill") for the $R \rightarrow k$ Fourier Transform. The default is 0.0.

`Dr` sets both `Dr1` and `Dr2` to the same value. The default is 0.0.

`Ikwindo` integer index to specify which of the possible Window Types to use for the $k \rightarrow R$ Fourier Transform. The default is 0, indicating Hanning Windows. See chapter 6 for details, and a complete list of possible Window Types.

`Irwindo` integer index to specify which of the possible Window Types to use for the $R \rightarrow k$ Fourier Transform. The default is 0, indicating Hanning Windows. See chapter 6 for details, and a complete list of possible Window Types.

`Iwindo` sets both `Ikwindo` and `Irwindo`. The default is 0.

`Mftfit` number of points to use in the FFT when doing the FFT for the actual fit. This affects the R-spacing between data points $\delta R = 10\pi/$`MFTFIT`. The default is the smallest power of 2 such that there are not less than $N_{\rm idp}$ points in the fit R-range. `MFTFIT` will be reset to a power of 2. This will affect the speed of the calculation, but shouldn't substantially change the fit results.

`Mftwrt` the number of points to use in the Fourier Transform when doing the final FFT for writing the output files. This affects the R-spacing between data points, given by $\delta R = 10\pi/$`MFTWRT`. The default is 2048, and the number will be set to a power of 2.

3.7 Error Analysis Keywords

`Epsdat` ϵ_k, the uncertainty in the measurement of $\chi(k)$, used to scale χ^2 in the fit and estimate of the variables. By default, it is found from the rms value of $\tilde{\chi}(R)$ between 15 and 25 Å, as described in chapter 5.

`Epsr` ϵ_R, the uncertainty in the measurement of $\tilde{\chi}(R)$, used to scale χ^2 in the fit and estimate of the variables. By default, it is found from the rms value of $\tilde{\chi}(R)$ between 15 and 25 Å, as described in chapter 5.

`Cormin` smallest absolute value of correlation between any two variables to report in *`feffit.log`*. The default is 0.50.

3.8 Variable and User Defined Function Keywords

All variables and User Defined Functions must be on their own line in *feffit.inp*. More information on Math Expressions, variables and User Defined Functions is in chapter 4.

`Guess` Initial guess for a variable. This statement defines the variable, and so is required for every variable. The syntax is:

`Guess` ⟨delimiter⟩ `Variable Name` ⟨delimiter⟩ `Number`

where the `Number` will be used as the initial guess.

`Set` Math Expression to be used as a User Defined Function. The syntax is:

`Set` ⟨delimiter⟩ `Function` ⟨delimiter⟩ `Expression`

3.9 Path Parameter Keywords

All Path Parameters must be on their own line in *feffit.inp*, with the syntax

`Path Parameter` ⟨delimiter⟩ `Path Index` ⟨delimiter⟩ `Character String`

where `Path Index` is an integer. For a mathematical description of the effect of each of these parameters on $\chi(k)$, see chapter 6.

`Path` Path File Name, the name of the *feffnnnn.dat* file to use for the path. This is required for a path to be used. There is no default.

`Id` User Label for the path, used only for ease of identification. This is optional, but very convenient. The default Label will include the Path Name, Half Path Length, and Degeneracy.

`S02` Math Expression for a constant multiplicative factor for $\chi(k)$.

`E0` Math Expression for the E_0 shift of the path.

`Ei` Math Expression for the imaginary energy shift of the path, which will mostly broaden $\chi(k)$. The mean-free-path, $\lambda \sim 1/\sqrt{E_i}$.

`Delr` Math Expression for a correction to $R_{\rm eff}$, the half the path length. Note that this mostly affects the Phase, but that the Amplitude is also affected.

`Sigma2` Math Expression for the mean-square displacement, or second cumulant (or Debye-Waller Factor).

`Third` Math Expression for the third cumulant.

`Fourth` Math Expression for the fourth cumulant.

Variables and Parameters in Fitting

In order to allow general and flexible constraints of the physically interesting quantities in the fit, FEFFIT uses three separate kinds of numerical values in its calculations. This formalism exists to make the physical structure of the system under study easier to model. I hope that this rather arcane formalism is useful enough to make learning it worthwhile. Appendix A and appendix B have a few examples and suggestions for how to use this aspect of FEFFIT to put physically meaningful constraints on the fit of XAFS data.

First, we have the Path Parameters. These are numbers that have a pre-defined name and effect on the XAFS function for a path. The Path Parameters represent the physical quantities in the XAFS equation usually associated with XAFS measurements of local structure, such as ΔR and σ^2. Chapter 6 has FEFFIT's version of the XAFS equation, giving the numerical effect of each of the Path Parameters. Each path used in the sum over multiple-scattering paths gets a set of its own Path Parameters, so that each path can have a different value of ΔR, etc.

Second, there are the variables. These are the quantities that are actually varied to get the best fit. They are chosen and named by the user, and how they alter any of the Path Parameters is also chosen by the user. The Path Parameters are not varied directly in FEFFIT, but are instead written in terms of the variables. This allows two convenient things to happen. First, a single variable can be used in different Path Parameters, making constraints of different Path Parameters easy. Second, it makes it easy to change what is varied and what is held constant in the fit.

To make the separation of Path Parameters and variables even easier, there is a third kind of numerical values which I'll call User Defined Functions (less formally, you can think of the variables as "guessed" quantities and these user defined functions as "set" quantities). These are intermediate quantities that are neither true variables nor Path Parameters. Like variables, they are chosen and named by the user, and can represent something similar to or very different from the usual XAFS Path Parameters. In general, they are written in terms of the variables, and other User Defined Functions, and they can be as simple or complex as you choose to make them. User Defined Functions are very convenient for both constraining Path Parameters and changing what is varied in the fit.

4.1 Path Parameters and Path Indices

There are nine Path Parameters associated with each path: The Path File Name, a User Label, and seven Numerical Path Parameters. All statements for the Path Parameters in *feffit.inp* take two arguments, with syntax:

`Path Parameter` ⟨delimiter⟩ `Path Index` ⟨delimiter⟩ `Character String`.

where the Path Index is an integer between 1 and 999, and ties the different Path Parameters together. The Path Indices may be ordered according to any convention, and need not be related to the FEFF path index The FEFF Path Indices are a convenient choice for many simple applications, but cannot always be used (if, for example, there are more than two central atom sites in the system being studied). The Path Index is the index that FEFFIT uses in the sum over paths to make the total $\chi(k)$, and the index used to order and write the output data for the individual paths.

The Path Parameter is one of the following:

`Path`	name of *feffnnnn.dat* file to use as theoretical calculation for this path
`Id`	optional user-supplied path identification
`e0`	shift of energy origin : $k \rightarrow \sqrt{k^2 - \texttt{e0} \times 2m_e/\hbar^2}$

`ei`	imaginary energy shift (to give additional broadening)
`S02`	constant amplitude factor
`delR`	change in half path length (1st cumulant, added to $R_{\rm eff}$)
`sigma2`	mean-square-displacement (2nd cumulant), or Debye-Waller Factor
`third`	Third cumulant
`fourth`	Fourth cumulant

The character string for a path file name is the file name for the *`feffnnnn.dat`* for that Path Index. The file name can be up to 70 characters long. Subdirectory paths can be included in the file name, and for case-sensitive systems, case conventions must be followed when naming files. If one of these files cannot be found, FEFFIT will tell you which files are missing before it stops. The User path identification label is available for the user's convenience — it will be written to the outputs, but has no internal function. The character strings for the rest of the Path Parameters (all those except `Path` and `Id`) are interpreted as Math Expressions used to evaluate the Path Parameter, and are written in terms of the variables, user defined functions, intrinsic functions, and numerical constants. Math Expressions will be described in more detail in section 4.5. Here are some typical Path Parameter statements (see also appendix A):

```
  Path    1  =  feff0001.dat     % 1st output path from feff
  Id      1  =  Path #1: Single Scattering - first neighbor
  e0      1  =  e0shift
  delr    1  =  delr_1
  sigma2  1  =  sig_1
%
  Path    2  =  feff0002.dat    % 2nd output path from feff
  Id      2  =  Path #2: Single Scattering - second neighbor
  e0      2  =  e0shift
  delr    2  =  delr_1 * sqrt(2)
  sigma2  2  =  sig_2
```

4.2 Defaults for the Path Parameters, and the 0th Path

Every Path Parameter must be specified for every path in the fit. This is often inconvenient for Path Parameters that are same for all paths, as will often happen for the Path Parameters `S02`, `e0`, and `ei`. And since there is already plenty of opportunities for typing mistakes in *`feffit.inp`*, anything to avoid useless repetition is worthwhile. So, you can assign default values for each of the Numerical Path Parameters, using the 0th Path (*i.e.* the path with Path Index 0). The Numerical Path Parameters for the 0th Path are interpreted as Math Expressions, as for any other path. If any Numerical Path Parameter is not explicitly written in *`feffit.inp`*, the value for that Parameter will be taken from the value for 0th Path. For example, writing

`S02 0 amplitude`,

and then simply not mentioning `S02` for any other paths assigns `S02` the value of `amplitude` for all paths. It is important to remember that the 0th path gives the *default* value, not an *overall* value. If a Numerical Path Parameter is explicitly mentioned for any path, the default value will not be used for that path. The defaults for the 0th Path Parameters are 0.0 for all Path Parameters except `S02`, which has default 1.0. This means that if you don't mention a Numerical Path Parameter for any path, including the 0th Path, then that Parameter will be set to zero (or one in the case of `S02`) in the fit. The Path Parameter `Id` has a default which is a label made from the path's file

name and the half-path-length ($R_{\rm eff}$), number of equivalent paths, and number of scattering legs in the path. The Path Parameters `Path` does not have a default.

4.3 Variables

The variables in the fitting problem are chosen by the user and are conceptually separate from the Path Parameters. This formalism lets the variables be the physically interesting quantities for the system, and also allows constraints to be easily placed on the different Path Parameters in the problem. All variables must be defined in *`feffit.inp`*, or the program will complain that it doesn't know what you want it to do. To define a variable, the keyword `guess` is used with the following syntax:

`guess` ⟨delimiter⟩ `Variable Name` ⟨delimiter⟩ `Initial Guess`

where the Initial Guess is the real number that will be used as the starting value for this variable in the fit. The final results should not depend strongly on the value of the initial guess for most physically reasonable variables. The initial guess can affect the computation speed. Variable names are character strings up to 40 characters long that meet these two requirements

1. contain no white spaces (blanks and/or TABs), or +, -, *, /, ^, (,), !, or %.
2. The first character is not a numeral.

Variables are checked before the fit begins to make sure that they are defined and that they are used in the fit. If any named value has not been specifically defined as a variable or User Defined Variable (that is, if it hasn't been `guess`ed or `set`), FEFFIT will stop and complain that some variable was undefined. Variables that are defined but not used by any Path Parameters or User-Defined Functions will cause a warning, but FEFFIT will not stop. Variables that are defined and used but end up having no effect on the fit are not investigated before the fit is done. Any such "null variables" will prevent uncertainties from being estimated for all variables.

4.4 User Defined Functions

User Defined Functions are like variables, only they are "set" so that their value is not directly adjusted in the fit. You can use up to 200 of them. Like variables, they are chosen by the user, not by FEFFIT. Like the Path Parameters, they are written as Math Expressions of the variables, real numbers, and other User Defined Functions. This means that their values might change during the fit (if they depend on any of the changing variables), but they don't count as separate variables. The names of the User Defined Functions follow the same rules as the names of the variables. The syntax for defining a User Defined Function is:

`set` ⟨delimiter⟩ `User Defined Function` ⟨delimiter⟩ `Math Expression`.

Some examples of User Defined Functions are:

```
set  hbar_c       = 1973.                % set to a constant
set  golden_mean  = (1.0 + sqrt(5)) / 2  % calculate a constant number
set  halfpi       = pi/2
set  sinxpi       = sin(x*halfpi) ** 2   % these depend on other named
set  b2_&_c2      = b**2 + c**2          % values which could be variables
set  max_x_y      = max(x, y)            % or other user-defined functions
```

User Defined Functions can be used to name constants (like `halfpi`) or to break up formulas. They use other named values in their definitions, and that the status of the named value as a variables or User Defined Function does not matter to the definition. In the above definition of `sinxpi`, `b2_&_c2` and `max_x_y` do not care whether `x`, `b`, `c`, or `y` are variables or other User Defined Functions. Only the numerical values matters.

One of the best uses of the User Defined Functions is to make a flexible way of constraining variables. As an example, a User Defined Function can be assigned to each Path Parameter that is to be changed in the problem. Some of these can be true variables in the fit, and some can be dependent on the true variables. Here is part of a *feffit.inp* to help illustrate this:

```
  sigma2 1  = sig1                % these are all path parameter
  sigma2 2  = sig2                % statements for the sigma2
  sigma2 3  = sig3                % parameters of paths 1, 2, 3, and 4
  sigma2 4  = sig4                %
%
  guess   sig1 =  0.00000         % a variable
  set     sig2 =  sqrt(3) * sig1  % a user-defined function
%
  guess   sig3 =  0.00000
  set     sig4 =  sqrt( sig3^2 +  2*sig2^2 )
```

There are two variables for the four Debye-Waller factors. But it is easy to change `set` to `guess` to change the number of variables. The point is that the effect of the quantities (`sig1,...,sig4`) on the XAFS Path Parameter doesn't change, only their status as variables.

In general, both User Defined Functions and Path Parameters have values that will change as variables in the fit change, even though they are not directly varied in the fit. How they depend on the set of variables is left entirely up to you. You choose what gets varied, and how the physically important part of the system (presumably what you're trying to measure), will alter the XAFS signal in terms of the boring Path Parameters. Please try to come up with a set of variables better than Debye-Waller Factors and neighbor distances, so that FEFFIT can help you measure the physical parameters you're interested in, and that people who've never done XAFS can understand.

Finally, a warning should be given about recursive definitions of the User Defined Functions (that is some User Defined Function referring to itself, even if through intermediate steps). These are difficult to check for — so be careful. I've only see this problem once (and that was a typo of my own), but beware of doing something like this:

```
set a = b + 1
set b = a
```

because `a` and `b` will diverge as the User Defined Functions are repeatedly evaluated!

4.5 Math Expressions

The character strings for the Numerical Path Parameters and User Defined Functions are interpreted as Math Expressions. These are made up of simple algebraic expressions, using numbers, named values (no distinction is made whether a named value was `guess`ed or `set`), simple math operations, and intrinsic functions. FORTRAN syntax is followed, and the case of the strings is ignored. The supported math operations are `*, /, +, -, **`,and `^`. Exponentiation can be done with `**` or `^`. Supported intrinsic functions are `abs, exp, log, sqrt, sin, cos, tan, asin, acos, atan, sinh, cosh,` and `tanh`. All trigonometric functions use radians. The value of π can be accessed with the named constant `pi`. The two-component intrinsic functions `min` and `max` are supported, and return the minimum and maximum value of two arguments, separated by a comma, as in `min(x,y)`. All math is done in single precision.

Standard math precedence (quantities inside parentheses first, from inner to outer parentheses, then `**` and `^`, followed by `*` and `/`, and then finally `+` and `-`) is followed, but parentheses are encouraged to avoid confusion. If anything does confuse these routines (undefined variables,

arguments out-of-range, or nonsense math operations), they will return an error message and a value of zero. Further questions or suggestions about this part of FEFFIT are welcome.

Besides the standard math intrinsic functions, there are a few additional intrinsic functions that are useful for XAFS analysis. The first of these is the constant `reff`, which gives the value of the half-path-length, $R_{\rm eff}$, for the "current path". (The current path is the one that FEFFIT is considering as it sums over paths.) The utility of `reff` is easily demonstrated with the 0th path. For example, saying

```
delr 0 = reff * expansion_parameter
```

gives a convenient (and fool-proof!) method for modeling a lattice expansion without having to manually enter all the different path lengths. It might get confusing to use `reff` in User Defined Functions (`set` statements). It's always OK to use `reff` in any Path Parameter statement, including those for the 0th path.

The second XAFS intrinsic function will calculate a value for σ^2 for a path given a value for the temperature and Debye Temperature using the correlated Debye Model implemented by Rehr, *et al.* in FEFF. The function is called `debye`, and has the syntax

```
debye( temp, theta)
```

where `temp` and `theta` represent the temperature and Debye Temperature, respectively (both in Kelvin). The comma between the temperature and the Debye Temperature is required. The temperature and Debye Temperature are actually Math Expressions themselves, and can be numbers or named quantities that have a set or variable value. This points to real utility of this function — the Debye Temperature can be a fitting variable. It should be noted that the Debye-Waller Factor depends not only on the temperature and Debye Temperature, but also on the physical details of the path (where the atoms are, and what their masses are), so that somehow these path details need to be used. This information is in fact given in the *`feffnnnn.dat`* files, and FEFFIT simply uses the values from the "current" path. This means that, like `reff`, it is probably clearer to use the `debye` function in a Path Parameter line. Using it for the 0th path is always OK.

The third (and last — but we're taking suggestions) XAFS intrinsic function will also calculate σ^2 for a path, and is very similar in use to the Debye function above, but uses the somewhat simpler Einstein model. This model could actually be done by hand as,

$$\sigma^2 = \frac{(\hbar c)^2}{2k_B}\frac{\coth(\theta/2T)}{M_{\rm R}\theta} \tag{4.1}$$

where $M_{\rm R}$ is the reduced mass, θ is the Einstein temperature, and T is the temperature. The built in function is easier, because it gets the units right the first time and it figures out the reduced mass for the current path. The syntax is

```
eins( temp, theta).
```

For multiple scattering paths, the reduced mass of the whole path is used (that is, by adding the reciprocals of the masses). Like for the `debye` function, `temp` and `theta` are Math Expressions, and can be numbers, "set" values, or variables. The Einstein model seems to work better than the Debye model for single scattering paths with small disorders, but your mileage may vary. At this point we recommend trying both the Einstein and Debye model, and seeing which gives better results.

Goodness of Fit and Uncertainties in Variables

This chapter deals with statistics and error analysis, a field that is by its nature uncertain. The procedures used by FEFFIT are as close to the "standard techniques of data fitting and error analysis" as possible. See *Data Reduction and Error Analysis for the Physical Sciences* by Philip R. Bevington and *Numerical Recipes* by Press, *et al.* for good introductions to these topics. If you think that any issues of fitting or error analysis are being overlooked or could be improved, please let us know. The topics in this chapter are extremely important to XAFS data analysis and we welcome any discussion of them.

5.1 χ^2 as a measure of the Goodness of Fit

The best set of variables in FEFFIT will minimize the sum of the squares of the difference of model and data XAFS. The statistic called chi-square, written χ^2, is a scaled measure of the sum of squares of a function, is generally considered the best figure of merit to judge the quality of the fit. The standard definition of χ^2,

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{f_i}{\epsilon_i} \right)^2, \tag{5.1}$$

requires 3 quantities: (1) f_i, the function to minimize; (2) N, the number of function evaluations; and (3) ϵ_i, the uncertainties in the function to minimize. FEFFIT allows the fit to be done in R- or k-space, but there is no conceptual difference in the way the fit is done. In either case, the function to minimize consists of the real and imaginary parts of the difference between data and full model XAFS (either $\tilde{\chi}(R)$ or $\tilde{\chi}(k)$) over the fit range. To be specific, when fitting in R-space the function to minimize is

$$f(R_i) = \tilde{\chi}_{\rm data}(R_i) - \tilde{\chi}_{\rm model}(R_i), \qquad R_{\rm min} \le R_i \le R_{\rm max}, \tag{5.2}$$

and when fitting in k-space, the function to minimize is

$$f(k_i) = \tilde{\chi}_{\rm data}(k_i) - \tilde{\chi}_{\rm model}(k_i), \qquad k_{\rm min} \le k_i \le k_{\rm max}. \tag{5.3}$$

For the rest of this chapter, I'll use f_i as the elements of the function to minimize, without specifying which of the two options is used. Note that these elements are squared in Eq. (5.1), so that the sign of f_i is unimportant.

Since there is one real and one imaginary evaluation for each data point, the number of evaluations is $N = 2(R_{\rm max} - R_{\rm min})/\delta R$ when fitting in R-space and $N = 2(k_{\rm max} - k_{\rm min})/\delta k$ when fitting in k-space. Here δR and δk are the grid spacing in R- and k-space (δk is set to 0.05 Å^{-1}). $R_{\rm min}$ and $R_{\rm max}$ (or $k_{\rm min}$ and $k_{\rm max}$) are the bounds of the fitting range. Since δR and δk are chosen arbitrarily, N has no physical significance and is not the right number to use if the scale of χ^2 is to be meaningful. The best number to use is the number of relevant independent measurements, given by the amount of information in the data concerning the atomic distribution around the central atom. Note that although the points of $\mu(E)$ are all independent measurements of absorption, they are *not* independent measurements of the atomic distribution function, which is what we're interested in when analyzing data. From basic information theory, the number of independent measurements in a spectrum is given by

$$N_{\rm idp} = \frac{2(k_{\rm max} - k_{\rm min})(R_{\rm max} - R_{\rm min})}{\pi} \quad + \quad 2. \tag{5.4}$$

The qualitative arguments for this are (1) that the conjugate Fourier variables are k and $2R$; (2) that since we're measuring real and imaginary parts of $\tilde{\chi}(R)$, the information must be an even number of points; and (3) we must have at least one pair of points, even for an infinitesimally small R-range. χ^2 is then

$$\chi^2 = \sum_{i=1}^{N_{\rm idp}} \left(\frac{f_i}{\epsilon_i}\right)^2 = \frac{N_{\rm idp}}{N} \sum_{i=1}^{N} \left(\frac{f_i}{\epsilon_i}\right)^2. \tag{5.5}$$

We are still left with ϵ_i, the uncertainty in the measurement, which we'll return to in the section 5.2. To simplify matters (and because we don't know anything better to do) FEFFIT uses a single value ϵ for all values of ϵ_i. If the uncertainties are dominated by random fluctuations in the data, then a single value for ϵ is the best that can be done anyway. Assuming for the moment that we have a reasonable estimate for ϵ, χ^2 is then given by

$$\chi^2 = \frac{N_{\rm idp}}{N\epsilon^2} \sum_{i=1}^{N} \Big\{ \big[{\rm Re}(f_i)\big]^2 + \big[{\rm Im}(f_i)\big]^2 \Big\}. \tag{5.6}$$

This is the definition of χ^2 used by FEFFIT, and is the primary figure-of-merit to characterize the goodness of the fit. There is a related figure-of-merit, called reduced chi-square, denoted χ^2_ν. This is equal to χ^2/ν, where $\nu = N_{\rm idp} - N_{\rm varys}$ is the number of degrees of freedom in the fit, (where $N_{\rm varys}$ is the number of variables in the fit).

χ^2 and χ^2_ν are useful for comparing the quality of different fits. The basic rule is that the fit with the lowest χ^2_ν is the best. This comparison works even if two fits have different number of variables. The criterion for assessing if a particular variable is useful in the fit is that χ^2_ν will be lowered for useful variables. If adding a variable causes χ^2 to decrease but χ^2_ν to increase, the fit is not improved.

If the errors are dominated by random fluctuations in the data, a good fit should have $\chi^2_\nu \sim 1$. If you want to get picky, the expected deviation of χ^2_ν is roughly $\sqrt{2/\nu}$, so that any $\chi^2_\nu > 1+2\sqrt{2/\nu}$ would clearly indicate a poor fit. Our experience is that χ^2_ν is rarely this close to 1 for concentrated samples, even for fits that look excellent by eye. We usually find χ^2_ν to be more like 10 or 100! This means that the difference between the data and fit is much bigger than the estimated uncertainty in the data (again, the pesky ϵ). The most likely reasons for a χ^2_ν very different from 1, are: (1) the FEFF model is not a good representation of the data, (2) ϵ is a poor estimate of the measurement uncertainty of the data, or (3) the fit R-range does not reflect the paths specified in the fit. Our current thinking is that, sorry to say, FEFF is poor enough that it will not match the data of concentrated samples to within the measurement uncertainty. (In the example in appendix A, a fit to the first shell of Cu metal gives $\chi^2_\nu \approx 20$.)

A poorly scaled χ^2 is not a big deal if it is used only to compare the goodness of fit between different models. And we're mostly willing to say that, even though FEFF doesn't match our data to within the measurement uncertainties, we can still rely on the structural parameters that a fit to a FEFF model will give. But we do run into a serious problem when trying to interpret the meaning of a $\chi^2_\nu \gg 1$. Specifically, it is not clear from the value of χ^2_ν alone if hard-to-estimate systematic errors are drowning out the random measurement errors or if the fit is truly bad. To help distinguish these two very different conclusions, it is convenient to introduce an $\mathcal{R}$-factor, which is

scaled to the magnitude of the data itself,

$$\mathcal{R} = \frac{\sum_{i=1}^{N} \left\{ \left[\mathrm{Re}\left(f_i\right)\right]^2 + \left[\mathrm{Im}\left(f_i\right)\right]^2 \right\}}{\sum_{i=1}^{N} \left\{ \left[\mathrm{Re}\left(\tilde{\chi}_{\mathrm{data}_i}\right)\right]^2 + \left[\mathrm{Im}\left(\tilde{\chi}_{\mathrm{data}_i}\right)\right]^2 \right\}}, \tag{5.7}$$

This number is directly proportional to χ^2, and gives a sum-of-squares measure of the fractional misfit. (We should mention that most of the other XAFS analysis programs use a number more like this $\mathcal{R}$ for their definition of χ^2.) Since $\mathcal{R}$ does not depend on N, N_{idp}, or ϵ, it has a different interpretation than χ^2. As long as the measurement uncertainty isn't a significant fraction of the measurement itself (so that the signal-to-noise ratio is much less than 1) we can be confident that any fit with an $\mathcal{R}$-factor bigger than a few percent is not a very good fit. For good fits to carefully measured data on concentrated samples, $\mathcal{R} \lesssim 0.02$ and $\chi^2_\nu > 10$ are common. Such fits are clearly quite good, as the theory and data agree within a percent. But since the misfit is much larger than the random fluctuations in the measured data, we're left with the conclusion that systematic errors dominate such fits.

5.2 The measurement uncertainty problem

Estimating ϵ, the measurement uncertainty in the data over the fit range, is the main difficulty in the error analysis in FEFFIT. ϵ contains both random fluctuations and systematic errors in the data. The random fluctuations of the data in R-space can be estimated by evaluating the rms value of the $\tilde{\chi}(R)$ between 15 and 25Å. This assumes that the fluctuations are white noise, and that they are much bigger than the signal past 15Å.

Systematic errors in the data are much more difficult to estimate. (If you could accurately estimate their size you could probably eliminate them). Some things that may dominate the systematic errors of $\tilde{\chi}(R)$ are (1) leakage of an imperfect background into the first few shells, and (2) systematic errors in measurements of $\mu(E)$. You may be able to estimate the size of these systematic errors by trying different "reasonable" background removals, which, though tedious, will give an estimate of the first systematic error. Analyzing different data scans (taken under different experimental conditions) may help give an estimate of the second kind of systematic error. Though strictly not a systematic error in the data, a third source of systematic errors in the fit comes from the FEFF calculation itself. Such errors are important because they do contribute to the small amount of misfit expected in a good fit.

The scale of ϵ depends on the Fourier Transform parameters used (such as k-weight, ranges, and window functions), which makes ϵ difficult to interpret, and not a very intuitive quantity. Assuming that the noise is dominated by random fluctuations ϵ_R is linearly related to ϵ_k (the fluctuations in the unfiltered data $\chi(k)$), the measurement uncertainty in the k-space data, according to

$$\epsilon_k = \epsilon_R \sqrt{\frac{\pi\,(2w+1)}{\delta k\,(k_{\mathrm{max}}^{2w+1} - k_{\mathrm{min}}^{2w+1})}}, \tag{5.8}$$

where w is the k-weighting, and δk is the spacing between points in k-space. ϵ_q, the random fluctuations in filtered k-space, are found using the same kind of linear relation between ϵ_q and ϵ_R.

If, for any reason, you have an improved estimate of ϵ, (either ϵ_R or ϵ_k), you should definitely put it into *feffit.inp* with either the keyword `epsdat` or `epsr`. Note that all contributions to ϵ should be added in quadrature, and that the value used by default is only the random fluctuation component. If you specify ϵ_k, Eq. (5.8) will be used to convert this to ϵ_R.

5.3 Error estimation for the variables

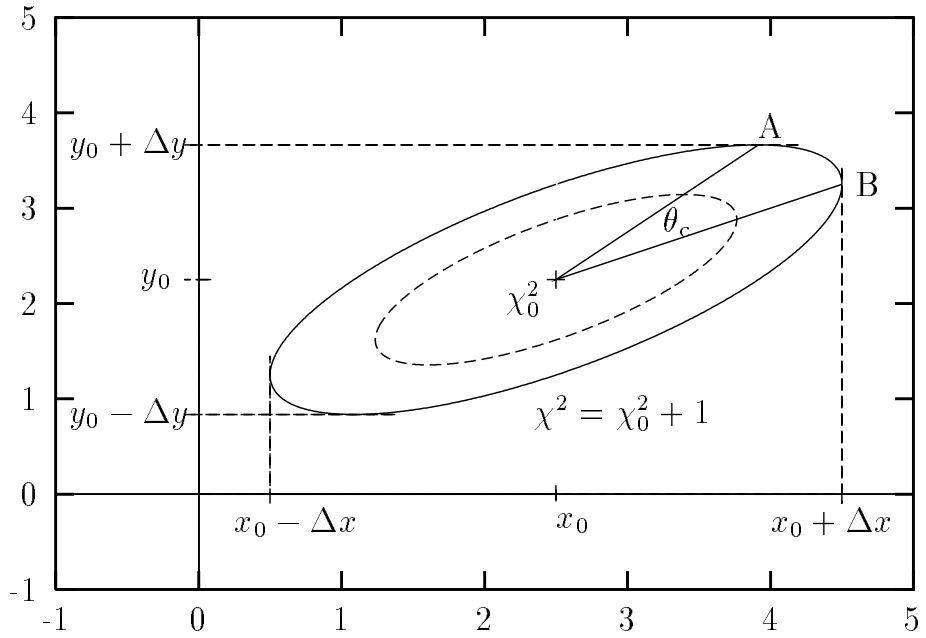


Figure 5.1 A contour map of χ^2 as a function of two variables, x and y. The uncertainties in the variables (Δx and Δy, respectively) are chosen so as to require that χ^2 is increased by 1 from its best value, χ^2_0. The correlation between the variables is given by $\cos(\theta_c)$.

FEFFIT will estimate the uncertainties in the variables immediately after the best-fit values of the variables are found. Occasionally the error estimation will fail, which means that at least one of the variables does not significantly change the model XAFS. Such "null variables" must be taken out of the fit for the uncertainties in the rest of the variables to be calculated. FEFFIT will try to tell you which variables are causing the problem if this happens.

The uncertainties in the variables are estimated using a standard technique of error analysis. This is well-explained in the standard references, but I'll summarize it here. The goal of the fit is to minimize χ^2 in each of its N_{varys} dimensions (where N_{varys} is the number of variables in the fit). Algorithms such as the Levenberg-Marquardt method are able to find a minimum for multi-dimensional χ^2 without too much difficulty. In order to do this, the first and second derivatives of χ^2 are found with respect to each of the variables (second derivatives are found for each pair of variables). These derivatives are used for finding the next estimate of the best variables, and turn out to be useful for estimating the uncertainties in the variables after the best fit has been found.

At the best-fit solution, χ^2 will be roughly parabolic in each of its N_{varys} dimensions. The $(N_{\mathrm{varys}} \times N_{\mathrm{varys}})$ matrix of second derivatives of χ^2 around the solution gives the curvature of the χ^2 surface. Figure 5.1 shows a crude rendition of a contour plot of the χ^2 surface for a two-variable problem. At the solution, the variables x and y have values x_0 and y_0, and $\chi^2 = \chi^2_0$. As x or y move away from their best-fit solution, χ^2 increases. For "normally distributed" uncertainties, the contours of constant χ^2 will be ellipses for two dimensions (and higher order ellipsoids for more than two dimensions). The uncertainty in the value of a variable is the amount by which it can be

increased and still have χ^2 below some limit. For randomly distributed errors, $\chi_0^2 + 1$ is a common criterion, and is the one used in FEFFIT. From Fig. 5.1, the uncertainties in x and y are Δx and Δy, and are those values which ensure that χ^2 is increased by 1 from its best value.

Note that when evaluating the uncertainty in a variable, all the other variables are allowed to vary, so that the correlations between variables can be taken into account. The correlation is a measure of how much the best-fit value of one of the variables changes in response to changing another variable away from its best-fit value. In Fig. 5.1, the correlation of the variables x and y is something like $\cos(\theta_c)$, the "projection" of Δx on Δy. If the variables were completely uncorrelated, the ellipse in Fig. 5.1 would have its major and minor axes parallel to the x and y axes. The point of this discussion is that if the correlations were ignored, and y were held constant, the uncertainty in x would be estimated to be $\Delta x'$. This is considerably smaller than Δx, and is a worse estimate of the uncertainty in x because a fit with x set to $x_0 + \Delta x$ will give a $\chi^2 = \chi_0^2 + 1$.

Algebraically, the uncertainties in the variables are given by the inverse of the curvature matrix (the matrix of the second derivatives), called the correlation matrix. The uncertainties in the variables are the square roots of the diagonal terms, and the correlations between pairs of variables are given by the off-diagonal terms of this matrix. This is very easy and useful to do, and gives a good estimate of the uncertainties, as long as the curvature matrix can be inverted. (Matrix inversion will fail if a variable does not affect the fit because the second derivative of χ^2 will be zero, and the curvature matrix will be singular.) Although it may not be obvious, the matrix inversion technique gives values for the uncertainties that will increase χ^2 by 1, as shown in Figure 5.1 (the key is that matrix inversion is division by 1).

Since χ^2 increases by 1 to give the uncertainties in the variables, the scale of χ^2 is very important. The scale of ϵ is therefore critical in getting good estimates of the uncertainties, and we're back where we were at the beginning of the chapter. Unless ϵ is correctly estimated, χ^2 will be wrong, and then the estimates for the uncertainties will be wrong. But there is away around this problem if we are convinced that a fit is good (based on a small $\mathcal{R}$) even if χ_ν^2 is much larger than 1.0, so that we assert that ϵ that is too small (because we did not include systematic errors). The trick is that the value of ϵ can be rescaled by a factor of $(\sqrt{\chi_\nu^2})$ so that χ_ν^2 will be forced to be 1. But we don't need to redo the fit or matrix inversion, we can just multiply the uncertainties themselves by $\sqrt{\chi_\nu^2}$. The numbers reported by FEFFIT for all the uncertainties in `feffit.log` are rescaled in this way by $\sqrt{\chi_\nu^2}$. It is important to remember that this trick gives reasonable estimates for the uncertainties at the expense of using χ_ν^2 for measuring the goodness of fit. It *assumes* that the fit is good (by forcing χ_ν^2 to 1), and that significant systematic contributions to ϵ were ignored.

Uncertainties are calculated only for the variables in the fit, not for the User-Defined Functions. Because the User-Defined Functions can depend on the variables in fairly complicated ways, the uncertainties in them are too hard to work out in general. You'll need to use the standard techniques of partial derivatives (see, Bevington's book, for example) to work out the propagation of errors in the errors to errors in functions of the errors.

All of the error analysis parameters discussed in this chapter will be written to `feffit.log`. The values for N_{idp}, N_{varys}, ν, ϵ, χ^2, and χ_ν^2 and $\mathcal{R}$ will all be written to this file. The uncertainties (already rescaled by $\sqrt{\chi_\nu^2}$) are listed with the best-fit values. Correlations between variables are sorted so that the most highly correlated are listed first. One warning about correlation of two variables should be mentioned. If two variables are completely correlated (*i.e.*, the correlation is greater than 0.999 or less than -0.999), then these two variables are not really different, and one of them can be eliminated.

The XAFS Equation and FEFFIT Algorithms

In this chapter I'll give the details the numerical procedures and algorithms used in FEFFITin as much detail as possible. This chapter is designed to tell you "what FEFFIT really does" in some part of the code. I'll try to be coherent and concise, so any further questions about the algorithms are welcome. Suggestions for improving any part of FEFFIT will be greatly appreciated. I'll start with the XAFS equation FEFFIT uses to determine the model XAFS function in terms of the variables. Then I'll go through the larger structure of the program. Finally, I'll discuss some the important algorithms FEFFIT uses including Fourier Transforms and interpolation.

6.1 The XAFS equation

The model calculation for $\chi(k)$ in FEFFIT is given by

$$\chi_{\rm model}(k) = \sum_{\rm Paths} \chi_{\rm path}(k, {\rm Amp}(k), {\rm Phase}(k), {\rm PathParameters}).$$

$\chi_{\rm model}(k)$ is the sum over paths of $\chi_{\rm path}(k)$, which is a function of the Phase and Amplitude from the ***feffnnnn.dat*** file for the path (which are functions of k), and the Numerical Path Parameters, which are the physical quantities that FEFFIT will use to vary $\chi_{\rm path}(k)$. As described in chapter 4, the Path Parameters are the physical quantities which alter the XAFS of a scattering and are written in terms of variables and User Defined Functions. The FEFFIT model uses the values of the Path Parameters `S02`, `e0`, `ei`, `delR`, `sigma2`, `third`, and `fourth` to write $\chi_{\rm path}(k)$ as:

$$\chi_{\rm path}(k) = {\rm Im}\left\{ \frac{{\rm Amp}(k) \times N_{\rm degen} \times \mathtt{S02}}{k(R_{\rm eff} + \mathtt{delR})^2} \exp\Big(-2p'' R_{\rm eff} - 2p^2\mathtt{sigma2} + \tfrac{2}{3}p^4\mathtt{fourth}\Big) \right. \\ \left. \times \exp\left\{ i\left[2kR_{\rm eff} + {\rm Phase}(k) + 2p\left(\mathtt{delR} - 2\frac{\mathtt{sigma2}}{R_{\rm eff}}\right) - \tfrac{4}{3}p^3\mathtt{third}\right]\right\}\right\}. \quad (6.1)$$

p' and p'' are the real and imaginary components of the the complex momentum with respect to E_0 (the bottom of the conduction band), and are evaluated as

$$p = p' - ip'' = \sqrt{\left\{ {\rm Re(p)}(k) - \frac{i}{\lambda(k)}\right\}^2 - i\,\mathtt{ei}\left(\frac{2m_e}{\hbar^2}\right)}. \quad (6.2)$$

k is the real momentum respect to $E_{\rm Fermi}$), evaluated as

$$k = \sqrt{k_{\rm FEFF}^2 - \mathtt{e0}\left(\frac{2m_e}{\hbar^2}\right)}. \quad (6.3)$$

The quantities Amp, Phase, Re(p), and λ in the above equations are all functions of $k_{\rm FEFF}$, and are taken from the ***feffnnnn.dat*** file for that path (as is $k_{\rm FEFF}$ itself. $N_{\rm degen}$, and $R_{\rm eff}$ are also taken from this file. The value of $N_{\rm degen}$ will be taken from FEFF for all paths unless the `Nodegen` flag is set, in which case all values of $N_{\rm degen}$ will be set to 1.

The fact that both the purely real k and the complex p are used for this calculation of $\chi_{\rm path}(k)$ needs some explanation. p is the preferred momentum since it is more like the Hermitian conjugate to r, the position operator of the photo-electron. The difference between the two is fairly small for

momentums above a few Å^{-1} and it could probably be argued that using a constant energy origin is a more serious problem than which origin to use. We use p consistent with FEFF. This is shown explicitly everywhere that anything from FEFF is changed. But we use k everywhere where the the FEFF calculation is being reconstructed. FEFF actually calculates $\tilde{\chi}(p)$ as a complex function of the complex momentum and then breaks this into amplitude and phase terms as a functions of k when writing its outputs. So all the occurrences of k in Eq. (6.1) are to carefully reconstruct the FEFF calculation before altering it with the Path Parameters.

The phase term $-4p\,\texttt{sigma2}/R_{\rm eff}$ in Eq. (6.1) comes about because the usual cumulant expansion ignores the $1/R^2$ dependence of the XAFS signal. This term is the first-order correction to ignoring this term. It is usually quite small, but can be important for systems with large disorders. For more on this, see the article by Rehr, Ingalls, and Crozier in *X-Ray Absorption, v.92 of Chemical Analysis*, edited by Koningsberger and Prins.

6.2 The FEFFIT Procedure

FEFFIT is a simple program in that it reads an input file, reads some other files, and then does a single calculation. It always proceeds exactly the same way through the calculation, and only skips steps if explicitly told to do so. It does not run interactively, and there is no way to stop in the middle or go half way back to change something. Here are the steps FEFFIT takes from beginning to end:

1. FEFFIT reads *feffit.inp*. All the input flags and commands described in chapter 3 are set. Math Expressions for User Defined Functions and Path Parameters are translated into quickly decodable form as they are read. The Math Expressions are checked for syntax errors and to ensure that all variables are defined and used.
2. The Experimental data file is read. If needed, the data are interpolated to uniform k-spacing with $\delta_k = 0.05$ Å^{-1}. Values for the fit and Fourier Transform ranges may be adjusted slightly to reflect these data ranges.
3. The *feffnnnn.dat* files are read.
4. The best-fit values for the variables are found.
5. The uncertainties in the variables and correlations between variables are estimated by inverting the curvature matrix, as discussed in chapter 5.
6. *feffit.log* is written. This will contain best-fit values for the variables, their uncertainties and correlations, and goodness-of-fit statistics. User Defined Functions and the Path Parameters for each path will also be written.
7. Output data files are written. These will contain data, full fit, and the contribution to the fit from each path. The outputs will be written in $k-$, $R-$, and backtransformed $k-$space.

Step 4 is the hard part (deciding how to improve the values of the variables, and when these values are good enough to quit) is done by standard non-linear least squares routines (from MINPACK), so we don't have to worry too much about it.

But we do need to provide the function to be minimized for a set of variables so that the least-squares black-box can evaluate this function for any set of variables. Let me represent the set of variables be the vector $\mathbf{x}$ (there may be more than one). The function to be minimized , $\mathbf{f}$ (which is a vector because it is a complex function of either R or k), is found for a given $\mathbf{x}$ in the following way:

1. $\chi_{\rm model}(k)$ is formed by summing the contribution from each path over the Path Index. For each path:
   a. The User Defined Functions are determined in terms of **x**. Note that this is done for each path, so that path dependent quantities like `reff` can be used in the User Defined Functions.
   b. The Path Parameters, (`S02`, `delR`, etc.) are determined in terms of **x** and the User-Defined Functions.
   c. $\chi_{\rm path}(k)$ for this path is found using Eq. (6.1).
   d. This $\chi_{\rm path}(k)$ is added to the total $\chi_{\rm model}(k)$.
2. The total $\chi_{\rm model}(k)$ is Fourier Transformed into R-space (and then into k-space if fitting in k-space), giving the real and imaginary parts of $\tilde{\chi}_{\rm model}$.
3 The fitting function, **f** is determined by subtracting $\tilde{\chi}_{\rm model}$ from the $\tilde{\chi}_{\rm data}$ (which was already formed from the data $\chi_{\rm data}(k)$ before the fit was started) over the fitting range. This is a complex function of R or k, found using either Eq. (5.2) or Eq. (5.3) .

This function **f** is used by the least-squares algorithm to select an improved set of variables, **x**, and these steps are repeated until the sum of squares of the elements of **f** is minimized. If you're up to reading FORTRAN, the steps above occur in the subroutines FITFUN and CHIPTH.

6.3 Fourier Transforms

Fourier Transforms are pretty common in XAFS analysis, but they need to be discussed here for completeness. Most XAFS analysis (FEFFIT included) uses

$$\tilde{\chi}(R) = \frac{1}{\sqrt{2\pi}} \int_0^\infty k^w \, \chi(k) \, {\rm W}(k) \, e^{i2kR} \, dk. \tag{6.4}$$

and

$$\tilde{\chi}(k) = \frac{1}{\sqrt{2\pi}} \int_0^\infty \chi(R) \, {\rm W}(R) \, e^{-i2kR} \, dR. \tag{6.5}$$

Of course, discrete forms of these are really used so that the Fast Fourier Transform can be exploited. The k-space grid is $\delta k = 0.05\,\text{Å}^{-1}$, and array sizes for $\chi(k)$ and $\tilde{\chi}(R)$ are $N_{\rm fft} = 512$, 1024, or 2048. The array for $\chi(k)$ is "padded" with zeros past the range of measured data. This "zero padding" has the effect of smoothly filling in data points in R-space. It also gives an R-space grid of $\delta R = \pi/N_{\rm fft}\,\delta k$, and we write $k_n = n\delta k$ and $R_m = m\,\delta R$. The discrete Fourier Transforms used are

$$\tilde{\chi}(R_m) = \frac{i\delta k}{\sqrt{\pi N_{\rm fft}}} \sum_{n=1}^{N_{\rm fft}} \chi(k_n) \, {\rm W}(k_n) \, k_n^w e^{2\pi i n m/N_{\rm fft}}, \tag{6.6}$$

and

$$\tilde{\chi}(k_n) = \frac{2i\delta R}{\sqrt{\pi N_{\rm fft}}} \sum_{m=1}^{N_{\rm fft}} \tilde{\chi}(R_m) \, {\rm W}(R_m) \, e^{-2\pi i n m/N_{\rm fft}}, \tag{6.7}$$

These normalizations preserve the symmetry properties of the Fourier Transforms with conjugate variables k and $2R$.

While the fitting is being done, the array size $N_{\rm fft}$ (which is the number of points between 0 and 10π Å in R-space) is set to `MFTFIT`, which is usually 512 or 1024. This keeps the spacing between data points not much smaller than the spacing between independent points so that N is not too much bigger than $N_{\rm idp}$ (though it is guaranteed to be bigger), and speeds up the code. Changing the value of `MFTFIT` should not significantly affect the fit results. If you have a small number of independent points in the fit range (fewer than 10), and you find a truly awful fit by eye gives a small χ^2, it might be that `MFTFIT` is too small, and that the fitting got stuck in a "false minimum". This should be a very rare occurrence, but if it happens, you should increase `MFTFIT`. When writing the output R-space data files, the value of $N_{\rm fft}$ is set to `MFTWRT`, which will normally be 2048. Both `MFTWRT` and `MFTFIT` can be set by the user to be 512, 1024, or 2048.

To transform from $\chi(k)$ to $\tilde{\chi}(R)$, $\chi(k)$ may be weighted by k^w. For both transforms, a Fourier Transform Window is used to select a finite data range. This Window is used to smooth out the data while maintaining some peak separation. The functional form of the Window depends on `Ikwindo`) (which gives the the functional form of the window), `Kmin`, `Kmax`, `Dk1`, and `Dk2`, for the forward transform ($k \rightarrow R$), and on `Irwindo`, `Rmin`, `Rmax`, `Dr1`, and `Dr2`, for the back transform ($R \rightarrow k$). There and currently 8 options for the functional form of the Window. Anything said in favor of one of the Window types is little more than folklore, with the exception of the Lorentzian Window (3), which is probably not worth using for XAFS analysis. If using different Windows gives different numbers for your fit, there is probably something wrong.

Here are the functional forms of the available Fourier Transform Windows in FEFFIT. For simplicity, all are written as functions of k (The R-space windows are exactly analogous to these with `Ikwindo`, `kmin`, `kmax`, `Dk1`, and `Dk2` replaced by `Irwindo`, `Rmin`, `Rmax`, `Dr1`, and `Dr2`):

`Ikwindo` Window Type and functional form

`0` Hanning Window Sills: The Default Window Type.

$$\mathrm{W}(k) = \begin{cases} \sin^2\left(\dfrac{\pi\,(k - \mathtt{Kmin} + \mathtt{Dk1}/2)}{2\ \mathtt{Dk1}}\right), & \mathtt{Kmin} - \mathtt{Dk1}/2 \le k < \mathtt{Kmin} + \mathtt{Dk1}/2 \\ 1.0, & \mathtt{Kmin} + \mathtt{Dk1}/2 \le k \le \mathtt{Kmax} - \mathtt{Dk2}/2 \\ \cos^2\left(\dfrac{\pi\,(k - \mathtt{Kmax} + \mathtt{Dk2}/2)}{2\ \mathtt{Dk2}}\right), & \mathtt{Kmax} - \mathtt{Dk2}/2 < k \le \mathtt{Kmax} + \mathtt{Dk2}/2 \end{cases}$$

`1` Hanning Window Fraction: `Dk1` is the fraction of the window range that is not held at 1.00. In the formula below, $\gamma = \mathtt{Dk1}(\mathtt{Kmax} - \mathtt{Kmin})/2$

$$\mathrm{W}(k) = \begin{cases} \sin^2\left(\dfrac{\pi\,(k - \mathtt{Kmin} + \mathtt{Dk1}/2)}{2\ \mathtt{Dk1}}\right), & \mathtt{Kmin} \le k < \mathtt{Kmin} + \gamma \\ 1.0 & \mathtt{Kmin} + \gamma \le k \le \mathtt{Kmax} - \gamma \\ \cos^2\left(\dfrac{\pi\,(k - \mathtt{Kmax} + \mathtt{Dk1}/2)}{2\ \mathtt{Dk1}}\right), & \mathtt{Kmax} - \gamma < k \le \mathtt{Kmax} \end{cases}$$

`2` Gaussian Window: Note that W(k) never goes to zero. `Iwindo` = 7 gives an alternate form for the Gaussian window.

$$\mathrm{W}(k) = \exp\left(-\mathtt{Dk1}\left\{\frac{2k - \mathtt{Kmax} - \mathtt{Kmin}}{\mathtt{Kmax} + \mathtt{Kmin}}\right\}^2\right)$$

3 Lorentzian Window: Note that W(k) never goes to zero.

$$\mathrm{W}(k) = \left(1.0 + \mathtt{Dk1} \left\{ \frac{2k - \mathtt{Kmax} - \mathtt{Kmin}}{\mathtt{Kmax} + \mathtt{Kmin}} \right\}^2 \right)^{-1}$$

4 Parzen Window: This window has linear "sills".

$$\mathrm{W}(k) = \begin{cases} \dfrac{k - \mathtt{Kmin} + \mathtt{Dk1}/2}{\mathtt{Dk1}}, & \mathtt{Kmin} - \mathtt{Dk1}/2 \le k < \mathtt{Kmin} + \mathtt{Dk1}/2 \\ 1.0 & \mathtt{Kmin} + \mathtt{Dk1}/2 \le k \le \mathtt{Kmax} - \mathtt{Dk2}/2 \\ 1.0 - \dfrac{k - \mathtt{Kmax} + \mathtt{Dk2}/2}{\mathtt{Dk2}}, & \mathtt{Kmax} - \mathtt{Dk2}/2 < k \le \mathtt{Kmax} + \mathtt{Dk2}/2 \end{cases}$$

5 Welch Window: This window has quadratic "sills".

$$\mathrm{W}(k) = \begin{cases} \left\{ \dfrac{k - \mathtt{Kmin} + \mathtt{Dk1}/2}{\mathtt{Dk1}} \right\}^2, & \mathtt{Kmin} - \mathtt{Dk1}/2 \le k < \mathtt{Kmin} + \mathtt{Dk1}/2 \\ 1.0 & \mathtt{Kmin} + \mathtt{Dk1}/2 \le k \le \mathtt{Kmax} - \mathtt{Dk2}/2 \\ 1.0 - \left\{ \dfrac{k - \mathtt{Kmax} + \mathtt{Dk2}/2}{\mathtt{Dk2}} \right\}^2, & \mathtt{Kmax} - \mathtt{Dk2}/2 < k \le \mathtt{Kmax} + \mathtt{Dk2}/2 \end{cases}$$

6 Sine Window: This gives a half-period over the window range.

$$\mathrm{W}(k) = \sin \left(\frac{\pi \left(\mathtt{Kmax} + \mathtt{Dk2} - k \right)}{\mathtt{Kmax} + \mathtt{Dk2} - \mathtt{Kmin} + \mathtt{Dk1}} \right)$$

7 Gaussian Window: An alternate version of the Gaussian window.

$$\mathrm{W}(k) = \exp \left(- \mathtt{Dk1} \left(k - \mathtt{Dk2} \right)^2 \right)$$

6.4 Public Domain Software and Further Reading

The Fast Fourier Transform and nonlinear last-squares routines used in FEFFIT are public domain software. The FFT routines used are part of FFTPACK, written by Paul N. Swarztrauber at the National Center for Atmospheric Research. I changed some of the dimension statements in these routines to more closely reflect the ANSI standard. The least-squares routines used are part of MINPACK, written by B. S. Garbow, K. E. Hillstrom, and J. J. More. at Argonne National Lab in 1980. I changed some of the machine-dependent parameters. Both of these packages were taken electronically from NETLIB, at AT&T Bell Labs in Murray Hill, NJ which has a large selection of public domain numerical software that can be taken for free by e-mail, ftp, or over the Web. Send the e-mail message `send index` to `netlib@research.att.com`, or give Mosaic the URL `ftp://netlib.att.com/home.html` to get more information.

Examples

This is the most important part of this document. I'll go through a simple example of pure Cu metal in detail, which should get you started on using FEFFIT for any real XAFS problem. The rest of the document should become much clearer after you start using the program. Appendix B gives some suggestions for using FEFFIT to solve typical XAFS problems as well as showing some of the more sophisticated things that FEFFIT can be used for.

All files mentioned in this chapter should have been included in your distribution of FEFFIT. If you don't have these files, contact us and we'll get them to you. All examples use the ASCII file type, and have been renamed to prevent FEFF or FEFFIT from easily overwriting them. Since FEFFIT requires its input file to be named *feffit.inp*, you'll have to copy each of the files *fit*.inp* to *feffit.inp* to do these example fits.

The distributed files include *atoms-cu.inp* which, if copied to *atoms.inp*, can be run through ATOMS. This will write a *feff.inp*, which can be run though FEFF. The twelve distributed *feffcu*.dat* files should be equivalent to the *feffnnnn.dat* that FEFF generates. Running ATOMS and FEFF and examining the outputs for this simple example is recommended.

A.1 Pure Cu example #1: FEFFIT without fitting

Below is *fit1.inp*, which is about as simple as FEFFIT gets. This is not really a fit at all, and just uses FEFFIT to add up three *feffnnnn.dat* files and write the results to k-, R-, and backtransformed k-space, applying some Debye-Waller Factors, and using an overall S_0^2. Still, it shows what a real *feffit.inp* file looks like, and how FEFFIT runs. Although FEFFIT is a fitting program, using it to add up FEFF files without doing any fitting is useful, and is a good way to start any analysis. It is more powerful than using FEFF itself for adding up *feffnnnn.dat* files to get a theoretical XAFS signal, and can do Fourier Transforms too.

```
  title = Example #1: Cu at 10K & the 1st 3 paths from feff !! NO FIT !!
  title = Setting  S02 = 0.90 , Using correlated Debye Model
%
  data   = cutest.dat                % input data file name
  out    = cu1.dat                   % output file name
%  fit R-range and FFT parameters:
  rmin   = 1.75   rmax    = 3.25
  kmin   = 2.0    kmax    = 19.0   dk     = 2      kweight = 1
%-----------------------------------------------------------------
   set    e0          =   0.0       % e0 offset
   set    s02         =   0.9       % constant amplitude factor, S02
   set    temp        =  10.        % temperature
   set    Debye_Temp  = 315.        % Debye temperature for Cu
   set    sigm_mcm    =   0.00052   % McMaster correction from feff.inp
%-----------------------------------------------------------------
```

```
% begin path parameter lists: Parameter,  Path Index , character string
   Path    1  feffcu01.dat
   Id      1  single scattering, R = 2.552, Degen= 12
   e0      1  e0
   S02     1  s02
   sigma2  1  debye(temp, Debye_temp) + sigm_mcm
%
   Path    2  feffcu02.dat
   Id      2  single scattering, R = 3.609, Degen=  6
   e0      2  e0
   S02     2  s02
   sigma2  2  debye(temp, Debye_temp) + sigm_mcm
%
   Path    3  feffcu03.dat
   Id      3  double scattering, R = 3.828, Degen= 48
   e0      3  e0
   S02     3  s02
   sigma2  3  debye(temp, Debye_temp) + sigm_mcm
%%%%%%%%%%%%%%%%%%%%%%%%%%  end  of FIT1.INP  %%%%%%%%%%%%%%%%%%%%%%%%%%
```

Three paths from FEFF are added together in this "null" fit, with each path being modified by the Path Parameters `S02` and `sigma2`. Values of `sigma2` are found for each path by using the Correlated Debye Model (see chapter 4) and the "McMaster correction" as calculated by ATOMS. At the bottom of the *feffit.log* file generated by FEFFIT (*fit1.log* in the distribution), you'll find that the numerical values for `sigma2` are different for the different paths even though the same Math Expression is used. The Debye function uses the positions and masses of the atoms in the path as well as the temperature and Debye temperature to give σ^2. The value used for the McMaster correction here is taken directly from the top of the *feff.inp* written by ATOMS. As further discussed in both the ATOMS and AUTOBK documents, this addition to `sigma2` accounts for the expected decay of the background absorption coefficient $\mu_0(E)$ that was not included in the AUTOBK background removal. Since AUTOBK normalizes $\chi(k)$ by a single number, not by a function that decreases slightly with energy (as $\mu_0(E)$ does), the resulting $\chi(k)$ for the data decreases slightly more rapidly than it should, and so the theoretical $\chi(k)$ should also be made to decrease.

There are five User-Defined Functions (`e0`, `S02`, `temp`, `Debye_temp`, and `sigm_mcm`) though in this simple example, they are all set to constants. It does not matter that there are User-Defined functions (`e0` and `S02`) which are also names of Path Parameters. The syntax of the Path Parameter statements means that they will never be confused. The third thing to notice is that all paths have the same Math Expression for Path Parameters `e0`, `S02` and `sigma2`, making them excellent candidates for the "0th" path. The portion of *fit1.inp* describing the Path Parameters could have been written as

```
   e0      0  e0
   S02     0  s02
   sigma2  0  debye(temp, Debye_temp) + sigm_mcm
%
   Path    1  feffcu01.dat
   Path    2  feffcu02.dat
   Path    3  feffcu03.dat
```

which is a more economical and fool-proof way of writing the same information.

A.2 Pure Cu example revisited: FEFFIT with fitting

Now let's add some variables to the Cu example and do a real fit. The file *fit2.inp* is pretty similar to *fit1.inp*. The important differences are the inclusion of several more paths (12 now, not just 3) and that some of the User-Defined Functions have been changed to variables (simply by changing `set` to `guess`!), so that *fit2.inp* will do a fit. There's also an additional variable to give a change in near-neighbor distance. Here is most of *fit2.inp*:

```
%%%%%%%%%%%%%%%%%%%%%%%%% Start of FIT2.INP  %%%%%%%%%%%%%%%%%%%%%%%%%
title = Example #2: Cu at 10K & the 1st 3 paths from FEFF5
title = Fitting energy0, S02, deltaR_1, and Debye Temperature
title = set temp =10 Kelvin,  using correlated Debye Model
%
data   = cutest.dat                % input data file name
out    = cu2.dat                   % output file name
%  fit R-range and FFT parameters:
  rmin   = 1.75   rmax    = 3.25
  kmin   = 2.0    kmax    = 19.0   dk     = 2      kweight = 1
%-----------------------------------------------------------------------
   guess  e0          =   0.0       % e0 offset
   guess  deltaR_1    =   0.0       % change in near-neighbor distance
   set    R_nn1       =   2.5478    % 1st neighbor distance
   guess  s02         =   0.9       % constant amplitude factor, S02
   set    Temp        =  10.        % temperature
   guess  Debye_Temp  = 315.        % Debye temperature for Cu
   set    sigm_mcm    =   0.00052   % McMaster correction from feff.inp
%-----------------------------------------------------------------------
% begin path parameter lists: Parameter,  Path Index , character string
   e0      0  e0
   delR    0  deltaR_1 *  ( reff / R_nn1 )
   S02     0  s02
   sigma2  0  debye(temp, Debye_temp) + sigm_mcm
%
   Path    1  feffcu01.dat
%
   Path    2  feffcu02.dat
%
   Path    3  feffcu03.dat
%
   Path    4  feffcu04.dat
%
%       It goes on like this up to path 12
%%%%%%%%%%%%%%%%%%%%%%%%%  end  of FIT2.INP  %%%%%%%%%%%%%%%%%%%%%%%%%
```

This is a reasonably good template to begin any XAFS analysis. The important things to notice about the modeling for XAFS data are the use of the "0th" path, `reff`, and the Correlated Debye Model. Further suggestions for modeling XAFS data are discussed in appendix B. I also suggest assigning every number with "`set`" rather than just writing the numbers in wherever they're needed,

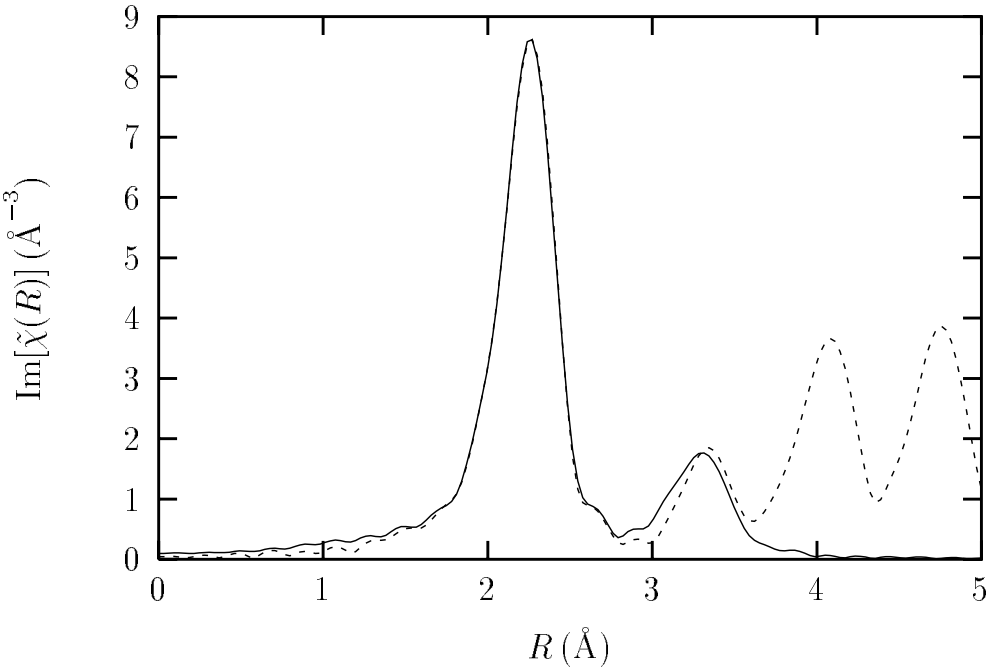


Figure A.1 $|\tilde{\chi}(R)|$ of data (dashed) and best-fit (solid) for pure Cu at 10 K. Fitting was done using *fit2.inp*, over an R-range of $[1.75, 3.25]$ Å. The results of this fit are further discussed in the text.

and not using any numbers in the Path Parameter section. For example, `R_nn1` and `temp` here are set as numbers here even though they're only used once. This makes it easier to change the numbers. And maybe later on you'll want to vary something that you originally thought had a constant value.

The results of the fit from *fit2.inp* is shown in Fig. A.1, which shows an excellent agreement between theory and data over the fit range, $R = [1.75, 3.25]$ Å, and even reasonable agreement for R past 3.25 Å, even though the high-R part of the spectrum is not being fit. If you're using ASCII files, this plot is of the fourth columns of *cu2r.dat* and *cu2r.fit*. For UWXAFS files, this data is held in records with nkey=1 and 2, respectively, of *cu2.rsp*. Please make sure that you can generate a picture similar to Fig. A.1 — you really can't do data analysis without looking at the fit results. You are also strongly encouraged to look at the contributions from the twelve individual paths that make up this fit, and especially where in R-space the different paths show up. The fit results in original and backtransformed k-space should also be look at.

Goodness-of-fit parameters for this fit and the best-fit values and estimated uncertainties in the variables can all be be found in *feffit.log*. From this file, we read that the $N_{\mathrm{idp}} = 16$, and $N_{\mathrm{varys}} = 4$, so that $\nu = 12$. We also see $\mathcal{R} \approx 0.0022$, which means the data and theory agree to 2 parts in a thousand over the fit range, indicating a very good fit. The values for χ^2 and χ^2_ν are ≈ 312 and ≈ 26, respectively. Following the discussion in chapter 5 about the need to include systematic errors when scaling χ^2(and the assertion that they were not included in the estimate of the measurement uncertainty ϵ), the uncertainties estimated are chosen to increase χ^2 by χ^2_ν.

The values found for the four variables and the uncertainties found (having already been rescaled by FEFFIT) are then `e0` $\approx -0.10 \pm 0.26$ eV, `deltar_1` $\approx 0.0097 \pm 0.0016$ Å, `s02` $\approx 0.943 \pm 0.026$, and `debye_temp` $\approx 314.1 pm 15.2$ K. The variables `e0` and `deltar_1` are found to be significantly correlated ($C \approx 0.83$), while `s02` and `debye_temp` are anti-correlated ($C \approx -0.87$).

After verifying that you can get these values and look at the fit results (there's always a hard part), you should be able to edit and play with *`fit2.inp`*, and become an expert at fitting Cu XAFS. Then you'll be ready to analyze your own data, doing sophisticated multiple-scattering fits with ingenious constraints. But first, ere are some suggestions for how to play with the Cu data to get a better feel for what FEFFIT can do. These aren't required, only suggested, and they're not listed in any particular order:

1 Change the k-range, dk, k-weighting, and even the Fourier Transform Window type (using `iwindo`). If the results change what does that mean? Is knowing how a variable depends on k-weighting important? (It is.) Notice that `Kmax` and `Kmin` determine $N_{\rm idp}$, but have slightly different meanings in some of the Fourier Transform Windows.

2 Increase the R-range, so that the fit is done over all twelve paths. First try this with the Correlated Debye Model. Notice that all the linear paths at twice the near-neighbor distance have the same Debye-Waller Factors, and that this is twice that of the near-neighbor. This is a general result of this model, and a very useful way to constrain Debye-Waller Factors.

3 Try fitting without the Debye Model, fitting the individual Debye-Waller Factors separately. Then try the Einstein Model.

4 Fit in k-space, trying some of the different ways to calculate the Debye-Waller Factors.

5 Go all the way back to the AUTOBK example, and remove the background and then analyze the 50K and 150K Cu data. This is almost like starting over, and should get you completely ready for your own analysis.

The basic ideas here are probably useful for general data analysis — start small, with the first neighbor distance, and work your way out in R-space until you can't get anything else from the data. Depending on how complicated your system is, you may need to start with more than just one or two paths, and you may not be able to fit the first shell without multiple scattering. But you can still start small, and work your way up.

Suggestions for Building Physical Models with FEFFIT

The examples in appendix A are intended as tutorial introductions to FEFFIT. You are no doubt trying to fit something more interesting and difficult than Cu. And although the tutorial examples show many aspects of using FEFFIT to model XAFS data, there are some subtleties and tricks that you may want to use to get the most out of your data. The hardest part of using FEFFIT is figuring out how to cut down the number of variables, and how to make a reasonable set of constraints of the Path Parameters so that some physics is put in the problem. In this appendix I will try to explain some of the modeling ideas and physical insight we've come up with for FEFFIT. We hope that this appendix will provide the kinds of suggestions that can inspire you to find a creative solution to your modeling needs. And if you come up with some creative solutions to XAFS problem using FEFFIT, we'd love to hear about it.

The examples in appendix A did, in fact, use a few modeling tricks. The use of the Debye function and `reff` in the Cu example allowed many Path Parameters to be reduced to two variables (the Debye Temperature and ΔR_1, with a linear expansion model). Though simple, these are the types of tricks that will be discussed in this Appendix. All the tricks will be illustrated with parts of input files, and are included in the file *suggest.inp*.

B.1 Simple Numerical Constraints

Often times you know ahead of time that some variable or Path Parameter has a range of values that is "reasonable" and a range that is "unreasonable". A typical example would be the assertion that no number for σ^2 should be negative. Non-linear least-squares fitting does not usually allow you put any constraints on the variables in the fit (*i.e.*, the things that are `guess`ed are unconstrained), so that it is not possible to tell the program to allow `sigma2` to vary, but to make sure that it is not negative. But the User-Defined Functions can be constrained, so that you can write a function of the variable `sigma2` that is constrained and use that constrained User-Defined Function for the Path Parameter. There are a few ways to write a function that is guaranteed to be non-negative. Here's one solution:

```
% ---------------------------
   guess    ss2      = 0.0        % ss2 will vary without constraint
   set      sigma2_1 = abs(ss2)   % while sigma2_1 will be non-negative
%
   sigma2  1  sigma2_1
% ---------------------------
```

This is a particularly simple case. What if you want `sigma2_1` to be bigger than some value, `x`? Just use `set sigma2_1 = abs(ss2 + x)`. But now what if you want to place both upper and lower constraints on some value? Say, for example, that you think `S02` should not be smaller than 0.5, but you also want to prevent it from being larger than 1.0? Again, there a few ways to do this. The solution I prefer uses both the `max` and `min` functions:

```
% ---------------------------
   guess  amp     = 0.70                 % amp will vary freely
   set    lower   = 0.50                 % lower bound for s02
   set    upper   = 1.00                 % upper bound for s02
%
   s02   0  max(lower, min(upper, amp))  % constrained path parameter
% ---------------------------
```

This allows the upper and lower limits to be changed easily. Be careful when doing this. Mixing up `min` and `max` or setting `lower` to be greater than `upper` will not allow `s02` to vary at all, and will probably make `amp` a null variable that kills the error analysis. Finally, after telling you how to do these things, I'll ask why you need them? If you really need the constraint, the best fit must want to give values for the variables that you consider to be physically unreasonable. This should disturb you and inspire you to think more carefully about the model you're using.

B.2 Using More Than One E0 Shifts

This is easy to put in *`feffit.inp`*. You simply put use different `e0`s for different paths. The question is: Why would you ever need more than one `e0`? FEFF makes various approximations which can be roughly corrected by shifts of `e0`, including incomplete core-hole shielding, a lack of angular variations of the valence charge distribution, and a lack of charge transfer between atoms in polar materials. Such approximations are worst for insulating materials with covalent or ionic bonds. In such cases it is probably important to use one `e0` for the near-neighbor, and another for the rest of the neighbors, which will compensate for the incomplete shielding of the core-hole. You might even might try more than two `e0`s for some materials. We've found that using two `e0` improves the fit quite a bit for a variety of ionic and insulating materials, with the typical result that the two are a few Volts apart and well outside their estimated uncertainties. In $BaZRO_3$ (see Haskel *et al.*, in press), as many as four `e0`s were found to each significantly improve the fit. These have been interpreted as accounting for angular variations of the valence charge density as well as incomplete shielding of the core-hole at the first neighbor.

B.3 Measuring the Number of Near Neighbors

This is a particularly important and common problem in XAFS analysis. But there are a few complications in getting the coordination number from XAFS data. The first is that both the number of near neighbors and the passive electron amplitude reduction factor ($N_{\rm degen}$ and S_0^2 in equation Eq. 6.1) contribute to the XAFS for a given path in exactly the same way, meaning that they are almost completely correlated quantities. This is why there are not separate `S02` and `N_degen` Path Parameters in FEFFIT. The upshot of this is that the number of near neighbors cannot be precisely and accurately determined without an equally precise and accurate measurement of S_0^2. The most likely possibility for overcoming this problem is to get S_0^2 by some other means. To a good approximation, S_0^2 is transferable between different systems with the same central atom. In principle, you ought to be able to measure S_0^2 for each element once (as on a sample for which the coordination number is not in doubt), and set that number in all other fits to get good measurements of the coordination number. However, such measurements must be done carefully to minimize experimental distortions in the XAFS amplitude.

But what if you don't have any idea what S_0^2 is, and you still want to measure the coordination number? One possibility is to assert that S_0^2 is the same for all paths in the solid. This removes at least some of the interdependence between S_0^2 and the coordination number, so that you can fit S_0^2 for a few different paths and the coordination number for just one (say the shortest path), but this still isn't perfect. A better way to get a good value for S_0^2 is to use the temperature-dependence of the XAFS (over some range for which $N_{\rm degen}$ is also constant). S_0^2 will not depend on temperature, and the temperature dependence of the σ^2 for the first neighbor (or the neighbors with the strongest backscattering) should be fairly simple. Most bonds are well-approximated by an Einstein oscillator so that σ^2 is given by Eq. (4.1). As a last resort, or a zeroth order approximation, the value of S_0^2 can be set to 0.9, which is expected to be correct to about 10% for all systems.

The second complication is a procedural artifact of the way FEFFIT sums over FEFF paths. FEFFIT uses the value of $N_{\rm degen}$ from the *`feffnnnn.dat`* file, which are going to be totally inap-

propriate if you're trying to fit this number. There are two choices: either to keep track of what was in the *feffnnnn.dat* file for $N_{\rm degen}$ and figure it out later or use the nodegen flag which will set $N_{\rm degen}$ to 1 in Eq. (6.1) for all paths, I prefer the second option.

Here is part of a *feffit.inp* file that will fit N assuming that S_0^2 has been given to us:

```
% ----------------------------
   nodegen = true             % set all values of n_degen = 1.0
   set    s02    =  0.9       % s02 from divine providence
   guess  n1     =  9.0       % fitting the coordination number
%
   path   1   feff0001.dat   %
   s02    1   s02 * n1
% ----------------------------
```

The key point here is that the constant amplitude for the near neighbor is the product of S_0^2 and the coordination number. Now, here's part of a *feffit.inp* file that will fit both S_0^2 and N for the first shell, assuming that N for the second shell is known.

```
% ----------------------------
   nodegen = true             % set all values of n_degen = 1.0
   guess  s02    =  0.9       % fitting s02
   guess  n1     =  6.0       % fitting 1st shell coordination number
   set    n2     = 12.0       % setting 2nd shell coordination number
%
   path   1   feff0001.dat    %
   id     1   a few (6?)  near neighbors
   s02    1   s02 * n1
%
   path   2   feff0002.dat    %
   id     2   twelve second neighbors, no doubt about it
   s02    2   s02 * n2
 ----------------------------
```

B.4 Combining Two Types of Near Neighbors

This is also a common problem in XAFS analysis. The solution is pretty similar to the above problem, but let's do this one too. As an example, let's say we have a Au-Ag alloy, with data on the Au edge, and a mixture of Au and Ag near-neighbors, at roughly the same distance (so that FEFF calculations are roughly transferable) in an FCC crystal. The problem is: How many Au-Ag near neighbors are there, and how many Au-Au neighbors are there? Since Au and Ag form a substitutional alloy at all concentrations, I'm going to assert that there are 12 total near-neighbors.

This problem definitely needs the nodegen flag. It could be solved without this flag, but it's much too painful. FEFF will give you two different *feffnnnn.dat* files at the first neighbor distance, which I'll call *feffAuAu.dat* and *feffAuAg.dat*. Depending on how you do the FEFF calculation, these two files could have nearly any path degeneracy, so it's best just to turn off this confusion, and control it all within FEFFIT. Here's the important part of *feffit.inp* for this problem, ignoring anything else like distances changes:

```
%
  nodegen = true                  % set all values of n_degen = 1.0
  set    s02     =  0.9           % from pure Au measurements (?)
  guess  n_au    =  2.0           % fit the number of Au near-neighbors
  set    n_total = 12.0           % set the number of total neighbors
  set    n_ag    = n_total - n_au % the number of Ag near-neighbors
%
   path    1  feffAuAu.dat
   id      1  Au-Au single scattering,
   s02     1  s02 * n_au
%
   path    2  feffAuAg.dat
   id      2  Au-Ag single scattering, degen = 1.0
   s02     2  s02 * n_ag
%
```

B.5 Linear Interpolation

The above technique actually has some important aspects that should be further discussed. Note that two FEFF calculations (and they might be from different FEFF runs, too) are used in FEFFIT as two paths that are combined to give a single physical shell. FEFFIT is an extension of FEFF, and can combine paths from different runs. Also note that this technique is an example of a linear combination of paths, and that the "linear coefficients" `n_au` and `na_ag` have the physical meaning of the relative weights for the two different neighbor atoms.

This is a particularly simple case of linear interpolation because the numbers `n_au` and `n_ag` directly affect the amplitude of the XAFS signal, so that using `n_au` and `n_ag` seems natural to associate with the Path Parameter `s02`. But linear interpolation only adjusts relative weights of two different FEFFIT paths, and they don't necessarily have to be interpreted as an amplitude factor. For instance, the linear interpolation technique *could* be used to measure a distance change, by doing two FEFF calculations with slightly different distances, and linearly combining them. The input file would look something like this:

```
%
  nodegen = true                  % set all values of n_degen = 1.0
  set    s02     =  0.9           % set s02
  guess  n_R1    =  2.0           % fit the number of neighbors with R1
  set    n_total = 12.0           % set the number of total neighbors
  set    n_R2    = n_total - n_R1 % the number of neighbors with R2
%
   path    1  feff00R1.dat
   id      1  atoms at R1
   s02     1  s02 * n_R1
%
   path    2  feff00R2.dat
   id      2  atoms at R2
   s02     2  s02 * n_R2
%
  set    R_fitted = ( n_R1 * R1  + n_R2 * R2 ) /n_total
%
```

Note that the values of `delr` don't change in this example, but that we're still, in some sense, measuring a distance change. The value of `R_fitted` will give the resulting value of near neighbor distance and will be written to *`feffit.log`* even though it's not actually used in the fit. Although I don't recommend this as a general way to measure distance (the Path Parameter `delr` is easier and more accurate), this does illustrate the general technique of linear interpolation between two *`feffnnnn.dat`* files, that has shown itself to be fairly useful in lots of disordered systems. Two different "known" FEFF calculations are done, and are linearly combined by adjusting the relative weights, which go in the `s02` parameter, and which can be given some physical significance.

B.6 Quadratic Interpolation

What if, for some reason, you don't trust the linear combination technique to give you a good enough answer? If this seems far-fetched, let me say that there is at least one important case where it is known to be unreliable on physical grounds. The example case involves focused multiple scattering paths where the photo-electron scatters at angle near 180°. The analysis challenge is to measure the "buckling" angle, which is how far from collinearity the three atoms are. The complication is that the scattering amplitude for such nearly-collinear scattering is known to vary quadratically with θ. The linear interpolation trick discussed above is liable to give poor results, unless we start with two FEFF calculations with θ's very close to the right value, which isn't very useful. The solution we came up with (the problem we used this on was mixtures of alaki-halide compounds and is discussed by Frenkel, *et al.* in Phys. Rev. B **49**, p. 11662, 1994 was to extend the linear combination of two FEFF paths to the quadratic interpolation of three FEFF paths. Quadratic interpolation is a pretty standard math method. But you may have to blow the dust off of some old math handbook to find it, so I'll just spell it out for you.

To do this, we first need to make a set of *`feffnnnn.dat`* basis functions for slightly different scattering angles. The easiest way to do this is to edit the file *`paths.dat`* output by FEFF, which has the complete path geometry for each path. Finding the right path in this file isn't too hard, and then you can edit the list of paths to make up any paths you want. So you simply pick some good "basis" angles for θ and figure out where the atoms are. We figured that the buckling angle θ would be around 5-10°, and certainly less than 20°, so I used basis angles of 0, 4, and 16°. Here's the important part of the doctored *`paths.dat`* file:

```
-----------------------------------------------------------------
 100    3  24.000  index, nleg, degeneracy, r=  5.1053
    x           y           z     ipot  label
  5.105311    .000000   .000000   1 'Br    '
  2.552655    .000000   .000000   2 'Rb    '
   .000000    .000000   .000000   0 'Br    '
 104    3  24.000  index, nleg, degeneracy, r=  5.1053
    x           y           z     ipot  label
  5.105311    .000000   .000000   1 'Br    '
  2.552655    .089141   .000000   2 'Rb    '
   .000000    .000000   .000000   0 'Br    '
 116    3  24.000  index, nleg, degeneracy, r=  5.1053
    x           y           z     ipot  label
  5.105311    .000000   .000000   1 'Br    '
  2.552655    .358752   .000000   2 'Rb    '
   .000000    .000000   .000000   0 'Br    '
-----------------------------------------------------------------
```

Running the third module of FEFF will then create the files *feff0100.dat*, *feff0104.dat*, and *feff0116.dat* for the angles 0, 4, and 16°. I changed the path indices here so that FEFF wouldn't overwrite any other files, and so the angle could be seen in the file name. This procedure needs to be done for all multiple scattering paths at this length, not just this 3-leg paths, but I'll skip over the rest of the paths for the sake of brevity. Now we're ready for the quadratic interpolation inside *feffit.inp* to measure the angle. Here it is:

```
%
   set    s02     =  0.9         % set the value of s02
   guess  theta   =  5
%
   set  theta1  =  0
   set  theta2  =  4
   set  theta3  = 16
%
   set  t21     = theta2 - theta1
   set  t31     = theta3 - theta1
   set  t23     = theta2 - theta3
%
   path 100  feff0100.dat
   id   100  theta =  0   feff calculation
   amp  100  s02 * (theta - theta2)*(theta - theta3) / ( t21*t31 )
%
   path 104  feff0104.dat
   id   104  theta =  4   feff calculation
   amp  104  s02 * (theta - theta1)*(theta - theta3) / ( t21*t23 )
%
   path 116  feff0116.dat
   id   116  theta = 16  feff calculation
   amp  116  s02 * (theta - theta1)*(theta - theta2) / (-t23*t31 )
%
```

This sort of interpolation would, of course need to be done for all focused multiple scattering paths at this distance that are affected by the change in buckling angle.

As a check of this procedure (and of course, anything this complicated should be checked), you could make paths at angles ranging from 0 to 20°, and generate mock $\chi(k)$ data files for each of these known distortions (by running the *feffnnnn.dat* file you generate for each angle through FEFFIT once and using the k-space output as $\chi(k)$). Each of these "data files" can then be fit using the *feffit.inp* above, where θ is a fitting variable. When I did this, FEFFIT got the right value for θ to within 1° for all angles below 16°.

Program Notes

This appendix is intended for those who want or need to deal with the source code of FEFFIT, probably to change some of it because it doesn't work on their machine, or because they've thought of some way to make the code better fit their needs. If you are setting out to change the source code, this appendix will probably not be nearly enough guidance, so feel free to contact me.

C.1 Code Portability and Code Compilation

The 1977 ANSI Standard for FORTRAN has been followed closely, so that FEFFIT should easily compile on any machine and run without any problems. The only significant departures from FORTRAN 77 are the assumption of the ASCII character set and the use of `INTEGER*2` variables for the UWXAFS binary file handling routines.

There are, unfortunately, aspects of FORTRAN which are machine- and compiler-dependent by design. One such aspect occurs in FEFFIT in the form of a compiler-dependent dimension for the "word-length" of the data in the UWXAFS binary files. The code cannot easily be made truly standard without significant changes to the UWXAFS binary file handling routines. The distributed code will, however, work on most machines, with the notable exception of a Vax. Changing the first executable statement of FEFFIT from `vaxflg = .false.` to `vaxflg = .true.` will make the code work on a Vax.

The UWXAFS binary file handling routines also use character strings which are 2048 characters long. Though standard, some compilers need to be told to accept character strings this long. The notable example of such a compiler is XLF (for AIX, IBM's Unix flavor), which needs the compiler switch "-qcharlen=2048". While compiling on any machine, we recommend including some form of array bounds checking. And if you have any problems with the compilation, it may be worthwhile to turn off compiler optimization flags. There may be some persistent, benign compiler warnings when you compile FEFFIT. There may be an "inconsistent variable type" warning in the routines from FFTPACK (routines with names like PASSF3 and CFFTI). There may also be "comparison is always false" warnings when using f2c. These can both be safely ignored.

As shipped, FEFFIT requires about 2 Mb of *available* RAM. Thus, it may be necessary to change some of the default dimensions when putting FEFFIT on machines with a small about memory such as PC's. Dimensions of all arrays are set in parameter statements, so changing them means changing several identical lines of code, (once for each of the principle subroutines of FEFFIT).

C.2 Adding More Data Types to FEFFIT

If the two data file formats (UWXAFS, ASCII) are not acceptable or convenient to your needs (that is, if you prefer using some other format), other choices could be added with a minimal amount of coding. The input and output of data files is fairly well-isolated, with subroutine INPDAT and OUTDAT controlling which data format to use. If you'd like another file format either contact us about it or follow the example of the routines INPCOL and OUTCOL, which read and write files in the ASCII column data format.

Simultaneous Fits of Multiple Data Sets

D.1 Why read this appendix?, or I'm Sick of This Document

First of all, do not read this appendix until you've written and used your own *feffit.inp* files for at least 3 systems. I don't mean this to be condescending, but this stuff is going to get uglier than the examples in Chapter 6. Furthermore, this is the first release of these FEFFIT features, so if you've read this far, you're a beta tester! This appendix explains how to use the new features, and the problems I've run into already.

The basic idea of this appendix is to fit more than one data set at the same time, presumably constraining physical parameters to be related for the different data sets. Although lots of possibilities come to mind (scans on the same sample at different polarizations, or of different edges in the same material, etc.) I'm going to restrict the discussion here to the temperature dependence of a single sample. This should be a fairly obvious and familiar use of multiple data sets. You should have the example Cu data at 10K, 50K, and 150K, and the example *feffit.inp* discussed later in this appendix. If you don't, contact me and I'll get them to you.

Once we start to think about fitting more than one data set at a time, the physical model we want to fit the data with will almost certainly change. For the different temperatures of the Cu data, we should use the same value of S_0^2. We may want to use the same value of R, or we may want to use a simple model for thermal expansion. The Debye-Waller Factors will, of course, be different for the different temperatures, but we may want to use just one Debye Temperature to parameterize them all instead of using three independent Debye-Waller Factors. Different E_0 values might be used, but the data may be aligned well enough that this is not necessary. The important point of this is that, with multiple data sets we can reduce the total number of variables used in the fit all the data sets, and therefore make better measurements of them. But in order to reduce the number of variables, it is important to rethink the model used to fit the XAFS.

D.2 The *feffit.inp* file, or Think Locally, Fit Globally

OK, once we want to fit multiple data sets, how do we do it? The simplest thing would be to just append the *feffit.inp* files for the different data sets together. I've tried to make the input file as close to this as possible but there are some important differences. While it might be useful to concatenate the different *feffit.inp* files and then edit the result, it might be just as easy to start over with a good editor (*i.e.*, one with cut and paste). Since it's always important to know what the input file is going to make the program do, I recommend against concatenation of files, for the simple reason that it's too easy to make mistakes that are hard to find. I'm telling you this from experience.

The first difference is that you need to tell FEFFIT when to stop thinking about one data set and to start thinking about the next set. The way you do this is to put the line

```
next data set
```

between the information for each data set. **This must occur on its own line** This line breaks the completely free format of the single data set *feffit.inp*, but I think that it's not too big of a hardship. It just means you have to group the file according to each data set, which you would almost certainly do anyway.

The second difference is that you have to now consider which keyword parameters affect the entire fit (a *global* parameter), and which affect only one data set (a *local* parameter). Frankly, some of this is just arbitrary, and you're just going to need to remember which are local and which

are global. There is a complete list at the end of this section, but first let me give a few general rules:

1 . Variables and User-Defined Functions (guesses and sets) are global.

2 . Path Indices and Path Parameters are local.

3 . Fourier Transform Parameters are local.

It is important to remember that local parameters are different for each data set. The default for each of them are the same as given in chapter 3, the default value is **NOT** the value from the previous data set.

You must make sure that the variables and user-defined are well-defined for all the data sets. Remember that the input file is read all at once, so that if a value is guessed or set more than one time in the file, the last definition is used for the entire fit. For example, if you want to set the temperature to 10K for one data set, 50K for another, and 150K for another, you will want to say something like

```
set t010 = 10
set t050 = 50
set t150 = 150
```

and you most certainly do **not** want to say `set temp = 10` for the 10K data, then `set temp = 50` for the 50K data, then `set temp = 150` for the 150K data. This will set the value of `temp` to 150 for the whole fitting. So beware.

The fact that the Path Indices and Parameters are local parameters requires some comment. Even though you may want to use the same set of *`feffnnnn.dat`* files for each data set in the fit, you *must* explicitly state all the paths that are used for each data set. You do not need to worry about reusing Path Indices, as these are local. So Path 1 for Data set 1 can be the same as Path 1 for data set 2, but it does not have to be.

The rest of the keywords are not so well defined, so here's a complete list (beta testers - any preferences which of these are local?) :

Global Keywords

guess	set	end	next data set
format	formin	formout	all
rlast	mftfit	mftwrt	degen/nodegen

Local Keywords

data	title	out	weight
kspout	rspout	kspfit	rspfit
kmin	kmax	kweight	dk/dk1/dk2
rmin	rmax	iwindo	sigdat/sigr
path	id	s02	e0
delr	sigma2	3rd	4th
ei	sinqr	cosqr	dafs

Besides the keyword phrase `next data set` to separate the successive data sets, there is only one new keyword for use with the multiple data sets. This is the keyword `weight` which gives the relative weight to give to each data set. The default value is 1.0 for each data set. If, for example, you wanted to give the 10K Cu data twice as much weight as the 50K data, then you would just

say `weight = 2` for the 10K data. The mathematical effect of `weight` on the fit will be discussed in the next section.

D.3 Matt rants about Information Theory, or It's Only Noise

Even though fitting multiple data sets at a single time is a very desirable thing to do, it turns out that there's sort of a serious problem that I haven't yet resolved, and I'm not sure a good solution even exists. The problem is: How do you count the information for more than one data set? For a single data scan the number of independent points is pretty clearly given as in Chapter 5.

This may seem like a technicality that's not critical to getting a good fit, and it actually should not effect the fit results. But because we don't have a good estimate of the measurement uncertainties, (and often end up rescaling the reported uncertainties by $\sqrt{\chi^2_\nu}$) it does end up affecting the uncertainties in the fit results, and so it ends up being quite important to know how many measurements have been made. Mostly because it's easy to do, FEFFIT simply adds the number of independent points from each data set.

Now I'll talk about why simply adding the independent data points from each data set is probably not correct, and why I don't know of a better way to do it. Your job as a beta tester includes reading this (read: suffering through my rant) and scratching your head at least for a while about the best way to estimate the number of measurements is, and just what exactly the different measurements are measuring. I'm open to any and all suggestions.

First of all, it's easy to trick FEFFIT by fitting the same data set twice. Almost everyone would say that the number of independent points is NOT doubled. But what if you used the same data set twice and used different k-weighting or different fitting regions? Those seem like slightly different fits, but certainly not completely different. From a practical point of view, there is a problem with FEFFIT figuring out whether two data sets are equivalent - you could simply remove one data point.

Now let's say you have two successive data scans. Are these really different measurements of the same information? Thinking of the number of independent points as the number of parameters you can fit, it seems ridiculous to say that you can fit twice as many parameters as you can with just one data scan. But thinking about how well the parameters are measured, it does seem that you've made better measurements of the parameters. A single scan with 2 second integration time should probably measure the parameters as well as two scans with 1 second integration times each. And you should also get the same uncertainties (ignoring systematic errors). But how can FEFFIT possibly distinguish two successive scans from two very different scans?

Next, let's say you have two scans at two different temperatures. Clearly the local atomic structure has changed, so some of the information about the local structure is probably changed, but they're still not completely different for the two different scans. If you were to characterize the temperature dependence by only a single Debye Temperature, then it could be argued that using 6 scans at different temperatures only improves the measurement of Θ_D compared to using 5 scans, but does not add any new information.

Finally, if you measure the edges of two different elements in one material then you've certainly added new information, but you'd need to constrain at least some of the structural parameters, so the information is probably not doubled. Anyway, you get the point that it's not easy to tell how much information there should be in two data sets. And even if it were it would be easy to fool the program.

With the important warning that the amount of information has a large uncertainty, here's how FEFFIT evaluates the information and how the fitting function is formulated when using multiple data sets. Refer to Chapter 5 for background information. With the subscript j referring to a

single data set, the number of independent points that FEFFIT uses is given by

$$N_{\rm idp} = \sum_{j=1}^{N_{data}} N_{{\rm idp}\,j} = \sum_{j=1}^{N_{data}} \left(\frac{2(k_{{\rm max}\,j} - k_{{\rm min}\,j})(R_{{\rm max}\,j} - R_{{\rm min}\,j})}{\pi} \quad + \quad 2 \right),$$

and the fitting function is an array with real and imaginary parts of the difference between data and model over the fitting range for each data set,

$$f_{ij} = f_j(R_i) = \tilde{\chi}_{{\rm data}\,j}(R_i) - \tilde{\chi}_{{\rm model}\,j}(R_i), \qquad R_{{\rm min}\,j} \le R_i \le R_{{\rm max}\,j}.$$

That is, $f(R)$ can be thought of as a matrix containing the difference of model and data for the different data sets. (The rows of this matrix would correspond to the different data sets, and the columns to the different R-values, and the elements of the matrix would be complex numbers representing the difference between model and data.) The χ^2 function that is actually minimized and used as a measure of the goodness of fit and for estimating uncertainties in the fit parameters is then given by

$$\chi^2 = \sum_{j=1}^{N_{data}} \left(\frac{{\rm weight}_j N_{{\rm idp}\,j}}{N_j \sigma_j^2} \sum_{i=1}^{N_j} (f_{ij}^2) \right).$$

The use of `weight` should now be slightly clearer. Again, the problem of how many independent measurements there are in the set of scans does not seem well-defined. If we rescale the reported errors by $\sqrt{\chi^2_\nu}$ this will certainly come back to haunt us. One option would be to get better estimates of the measurement uncertainties (or improve FEFF !) so that the value of χ^2 were more reliable. (Though, of course, to know how good the fit is we would compare χ^2 to the number of degrees of freedom in the fit, so we're back where we started.) But let's forget about all that for now and look at an example.

D.4 An example, or Copper Again?

OK, at this point an example will almost certainly help. Because I haven't gone through many examples myself, here's an example of the Cu data at 10K, 50K and 150K. The model I'm using is that:

1. S_0^2 is the same for all temperatures
2. The lattice expansion has no temperature dependence.
3. One value of E_0 for all the data sets is good enough.
4. The Debye-Waller Factors can all be fit with a single Debye Temperature.

I will discuss the reliability of this model, including how to test whether or not to include an `e0` for each data set, and whether ignoring the temperature dependence of the lattice spacing is appropriate. But first, here's the example ***feffit.inp*** file to fit all three Cu data sets.

```
!
```

```
!!!!!!!!!!!!!!!!!!!!!!!! Start of FEFFIT.INP  !!!!!!!!!!!!!!!!!!!!!!!!
!
! global parameters:
     format = ascii           ! format for all files in problem
     all    = no              ! don't write all outputs
!
! variables:
     guess    s02         =      0.8
     guess    thetad      =      200    !  Debye Temperature
     guess    e0_1        =      0.
     set      e0_2        =   e0_1
     set      e0_3        =   e0_1
     guess    alpha_0     =      0.001  !  constant lattice expansion
     guess    alpha_temp  =      0.000  !  thermal expansion
! set parameters
     set      t010        =     10
     set      t050        =     50
     set      t150        =    150
!-----------------------------------------------------------------
!
!   data set #1
!
   title  =  3 data-set fit     10K Cu data
!
   data   = cu10k.dat                    !  input data file name
   out    = 10k.dat                      !  output file name
   rmin   = 1.75    rmax    = 3.25       !  fit R-range
   kmin   = 2.0     kmax    = 19.0       !  FFT parameters
   dk     = 2       kweight = 2          !  FFT parameters
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
! Use the 0th path for the constant amplitude factor and e0.
   s02     0  s02
   e0      0  e0_1
   delr    0  reff * (alpha_0 + t010 * alpha_temp / 1000)
   sigma2  0  debye(t010, thetad)
!
   path    1  feffcu01.dat
   id      1  single scattering,  R = 2.552, Degen = 12
!
!
```

```
!
   path    2  feffcu02.dat
   id      2  single scattering,  R = 3.609, Degen =  6
!
   path    5  feffcu03.dat
   id      5  double scattering,  R = 3.828, Degen = 48
!
!-----------------------------------------------------------------
!
    next data set
!
!-----------------------------------------------------------------
!
!   data set #2
!
   title  =  3 data-set fit     50K Cu data
   data   = cu50k.dat                    !  input data file name
   out    = 50k.dat                      !  output file name
   rmin   = 1.75    rmax    = 3.25       !  fit R-range
   kmin   = 2.0     kmax    = 19.0       !  FFT parameters
   dk     = 2       kweight = 2          !  FFT parameters
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
! Use the 0th path for the constant amplitude factor and e0.
   s02     0  s02
   e0      0  e0_2
   delr    0  reff * (alpha_0 + t050 * alpha_temp / 1000)
   sigma2  0  debye(t050, thetad)
!
   path    1  feffcu01.dat
   id      1  single scattering,  R = 2.552, Degen = 12
!
   path    2  feffcu02.dat
   id      2  single scattering,  R = 3.609, Degen =  6
!
   path    5  feffcu03.dat
   id      5  double scattering,  R = 3.828, Degen = 48
!
!-----------------------------------------------------------------
     next data set
!-----------------------------------------------------------------
!
!   data set #3
!
   title  =  3 data-set fit    150K Cu data
   data   = cu150k.dat                   !  input data file name
   out    = 150k.dat                     !  output file name
   rmin   = 1.75    rmax    = 3.25       !  fit R-range
   kmin   = 2.0     kmax    = 19.0       !  FFT parameters
   dk     = 2       kweight = 2          !  FFT parameters
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!
```

```
! Use the 0th path for the constant amplitude factor and e0.
   s02     0  s02
   e0      0  e0_3
   delr    0  reff * (alpha_0 + t150 * alpha_temp / 1000.)
   sigma2  0  debye(t150, thetad)
!
   path    1  feffcu01.dat
   id      1  single scattering,  R = 2.552, Degen = 12
!
   path    2  feffcu02.dat
   id      2  single scattering,  R = 3.609, Degen =  6
!
   path    5  feffcu03.dat
   id      5  double scattering,  R = 3.828, Degen = 48
!!!!!!!!!!!!!!!!!!!!!!!!! End of FEFFIT.INP  !!!!!!!!!!!!!!!!!!!!!!!!!
!
```

All the `set` and `guess` statements are put together, at the top of the file. This is mostly to keep things easier to decipher, but it also makes it easier to change the details of the model later on. Also notice that I've used different variables for each data set's E_0 , but I've used `set` to make them all be the same number. This will make it easier to allow them each to float independently.

I've used `reff` to allow a simple lattice expansion by using the 0th path. I've also included the possibility of a temperature dependent ΔR, but that I've set the thermal expansion coefficient to zero, at least for now. I'm also fitting the Debye Temperature.

If you run this *`feffit.inp`* file as given, *`feffit.log`* should include these lines for the goodness of fit, best-fit values, and estimated uncertainties in the best-fit values:

```
-------------------------------------------------------------------------
 fit results, goodness of fit, and error analysis:
   number of data sets           =     3
   independent points in data    =    48
   number of variables in fit    =     5
   degrees of freedom in fit     =    43
   chi-square                    =   1899.47558594
   reduced chi-square            =     44.17385101
   feffit found the following values for the variables:
       variable            best fit value    uncertainty  initial guess
     s02             =       .91485572       .00145800       .80000001
     thetad          =    317.85824585       .47187614    200.00000000
     e0_1            =       .84959078       .01968322       .00000000
     alpha_0         =      -.00205574       .00003628       .00100000
     alpha_temp      =       .00108976       .00032104       .00000000
```

As mentioned above, the value of the number of independent points in the data is somewhat suspect, and therefore so is the number of degrees of freedom in the fit. Both are almost certainly too high, but it's not clear how much too high. This makes a numerical assessment of the fit a bit troubling. (It also means that $\chi^2_\nu = 44$ is a lower bound.) But we can compare two different models

with a fair amount of confidence by saying that any model that lowers χ^2_ν is an improvement over the other. This is true if we change what the variables are, or even if we change the number of variables. Because χ^2_ν is not quite correct, models which change χ^2_ν by a only small amount (say, a few percent) are not clearly improvements (we could at this point lapse into a wonderful discussion of confidence intervals and F-tests, but I'll spare you that nightmare).

So let's just pretend for the moment that χ^2_ν is correct, and then use the $\sqrt{\chi^2_\nu}$ trick from Chapter 5 to rescale the uncertainties in the variables. This will give

S_0^2	=	0.915	$\pm$	0.010 ,
Θ_D	=	318	$\pm$	3 ,
E_0	=	0.85	$\pm$	0.13 ,
α_0	=	-0.0021	$\pm$	0.0002 ,
α_{temp}	=	0.0011	$\pm$	0.0021

From this we can conclude that α_{temp} is consistent with zero, and that we could probably improve χ^2_ν by removing it from the fit. Another thing to notice is that the fitted Debye Temperature is fairly well estimated, and that it agrees with the value given in standard textbooks (Θ_D is typically reported as 315K). While I fully acknowledge the criticism that Cu is an easy case, I want to suggest that the concept for a Debye Temperature for a single bond may be useful in many materials, not just monatomic metals. In fact the `debye` function could be used to give different Debye Temperatures to different bonds in the same material. Using this technique may greatly improve the understanding of the temperature dependence of many complex systems.

Here are some suggestions for other things to try with the Cu data. First, try fitting with three different E_0's. This should **not** improve χ^2_ν, but see for yourself. You can also remove α_{temp} from the fit, setting it to either 0.0 or 0.0011.

Second, if you examine the *`feffit.log`* file in some detail, you'll notice that the measurement uncertainty is reported as being *smaller* for the 150K data than for the 10K and 50K data. This is a peculiar artifact of the fact that the measurement uncertainty is estimated from the high-R region, (15 Å to 25 Å), and that the low temperature Cu data is so good, and the Debye-Waller Factors so small, that the signal is considerably larger than the noise in the high-R region. Happily (or unhappily depending on your point of view), this will almost never happen for any real data. You could set all the values of `sigdat` to be equal. This will give truly equal weight to each of the temperatures. Using `sigdat` = 0.002 (as the measurement uncertainty in $\chi(k)$) seems like a reasonable value. This will actually increase χ^2_ν, but the fit results don't change very much.

As a final suggestion, include the rest of the 12 paths, including the multiple scattering ones, and fit out to the fourth "shell". This will be a little bit of work, but will probably make you quite familiar with all the details of fitting multiple data sets, and make you quite sick of Cu XAFS.

```
The FEFFIT document is finished.
                                Have a nice day.
```