EASE

EXAFS Analysis System for Emacs
Version 0.6.0

Bruce Ravel

The author of EASE, Bruce Ravel, can be reached at:

`<ravel@phys.washington.edu>`

The latest version of EASE can always be found at
`http://feff.phys.washington.edu/~ravel/ease/`

1 Introduction to EASE

This document describes the installation and use of EASE . EASE is the EXAFS Analysis System for Emacs. It is a user interface for FEFF, ATOMS, AUTOBK, FEFFIT and a few other programs.

This document is written with the assumption that the reader is familiar with EXAFS analysis, the use of FEFF and FEFFIT, and the use of Emacs. It is not expected that the reader is a guru in any of these topics, but a certain knowlege of each is necessary as their basic concepts will not be explained here.

There are two companion documents to this one. One of these is a tutorial (see section "Introduction" in *The Ease Tutorial*) and demonstrates the use of EASE by walking you through an analysis of copper metal using EASE along with FEFF and FEFFIT. The other is a quick reference card, '`quickref.ps`', which describes the key sequences and user configurable variables in EASE . These three documents together provide a complete reference to EASE .

The general purpose of a user interface to these (or any) programs is to to add value to the programs by making their use easier and more efficient. To this end, I have broken down the interface to FEFF and FEFFIT into five conceptual areas of functionality. These are

Generation of input files
: The input structures to these programs are rather complicated and offer many options for the user. This is particularly true for FEFFIT, which uses a highly abstracted metalanguage for constraining parameters in a fitting model. Much of the functionality of EASE is dedicated to the generation and manipulation of the text of the various input files. The creation of input files is the most difficult part of using these programs. Thus I was motivated to use Emacs as the application/program interface. Along with good integration of system level features such as file handling and process handling, Emacs possesses extremely powerful text manipulation capabilities and a flexible keyboard and mouse based user model.

Program execution
: EASE uses the Emacs system interface to execute the programs and to display their run-time messages. Although each of the programs requires that input files have filenames specific to the program (i.e. FEFF requires that its input file be called '`feff.inp`'), EASE allows you to use input files with arbitrary names. It uses a command wrapper to temporarily rename input files to the name expected by the executed program.

Organization of graphical output
: EASE uses GNUPLOT to display data. Scripts for running GNUPLOT are generated automatically from the contents of the input files and may be saved for future use. Many display options exist for FEFFIT for plotting results in different spaces and for plotting the contributions from individual scattering paths.

Organization of textual output
: Analysis of data is performed to obtain answers to physical questions. Pretty pictures are very important, but often a number is the bottom line. EASE has

several features for organizing the textual output (such as the contents of log files) of the programs into useful formats.

Error handling and recovery from mistakes
Any program that can be used can be misused. EASE has features for examining the contents of input files for errors or inconsistencies. It also allows the user to examine the run-time and saved-file output of the programs.

EASE is an ongoing project. I hope that, in its present form, it will help you analyze your data. For some hints of what EASE might do in the future, take a peek at Chapter 6 [Future], page 21. If you have any questions, comments, complaints, or suggestions, please contact me. There is a bug report function built into EASE which sends me email using the Emacs mailer. It is invoked by *`C-c C-b b`* of from the `Input - miscellaneous` menu. Use that function or any of my contact information shown on the second page of this document.

2 Installing EASE

2.1 Unpacking the distribution

EASE is distributed in two compressed archive formats. One uses the standard unix TAR and GZIP utilities, the other uses ZIP. Their contents are the same. Use whichever one is more convenient for you.

You can unpack the distribution any place on your disk. One of the steps of the installation discussed in See Section 2.2 [Making], page 4 will move all of the necessary files to their final home. To unpack the TARred and GZIPped file, `cd` to your lisp directory and execute these commands:

```
> gunzip ease.tar.gz
> tar xvf ease.tar
```

To unpack the ZIPped file, execute this commands:

```
> unzip ease.zip
```

After doing one of those steps, you will find that the EASE distribution has been unpacked into a subdirectory called '`ease-#.#.#`', where the '`#`' signs denote the current version number of EASE .

Now *cd* into the '`ease-#.#.#`' subdirectory. Here is what you will find in that directory.

- Several information files and scripts used during installation, including:

  '`Makefile.emacs`'
  '`Makefile.xemacs`'
  The instructions for the MAKE program, one for Emacs and one for XEmacs.

  '`configure`'
  A Bourne shell script used to configure the distribution for your machine. See Section 2.2 [Making], page 4.

  '`INSTALL`' A file with the installation instructions from Section 2.2 [Making], page 4.

  '`README`'
  '`COPYING`'
  '`HOOKS`' Some files with general information about EASE .
- A bunch of files that end in '`.el`'. These are the main programs in EASE .
- The '`docs/`' directory contains documentation for EASE as well as for ATOMS and FEFF in a variety of formats, including info, html, PostScript and plain text. You will also find quick reference cards in PostScript format for EASE and GNUPLOT here.
- The '`pixmaps/`' directory contains all of the icons used in the toolbars.
- The '`scripts/`' directory contains several shell and perl scripts used by EASE .
- The '`fortran/`' directory contains some programs useful for handling UWXAFS binary files as well as copies of the programs NORMAL for normalization and alignment of absorption data, and PHIT, an unsupported but possibly useful general purpose fitting program.

- The '`emulation/`' directory contains some files useful if you want Emacs to emulate VI, EDT, or CRISP while using EASE .
- The '`example/`' directory contains files that you will need for the tutorial, see section "Introduction" in *The Ease Tutorial*, and various example input files demonstrating features of EASE .

2.2 Configuring, making, and installing.

Before starting, you need to decide on a few things. During the first step in the installation process, you will be asked a few questions that EASE needs to know to install itself properly. These are

1. Will you be using Emacs or XEmacs?[1]
2. In which directory will you be installing EASE ? If you are installing EASE as a normal user, then this should be a directory where you keep personal Emacs lisp files. If you are installing EASE as root, then this should be the '`site-lisp`' directory. Telling the wrong place to the installation script will result in EASE not being accessible to Emacs.
3. What is the location of perl on your computer? The '`configure`' script will most likely be able to determine this, but if it fails you will need to supply its location. Perl is required to properly install various scripts the EASE uses to plot data and perform other chores. You can determine its location by ***`which perl`*** under tcsh or ***`type perl`*** under bash.
4. Do you want the installation script to edit your '`.emacs`' file so that EASE is automatically used with input files. To do so the script will write several lines to your '`.emacs`' file. If you have previously installed EASE then it is probably safe to answer no to this question. If you are installing this as root, you will have to edit the '`site-start.el`' file by hand with the lines printed to the screen at the end of the configuration step.[2]

Once you have decided on these points, type ***`configure`*** at the command line. This is an interactive script that asks you for the answers to the questions. Just follow the instructions printed on the screen.

Once the configuration is done, type ***`make`*** then ***`make install`***. Please note that if the location where you intend to have EASE installed is the same as the place where you unpacked it, you should skip the `make install` step. Nothing bad will happen if you don't skip it, but you will get a lot of non-critical but alarming looking error messages.[3]

The lines that are added to the '`.emacs`' file look something like this, with `~/lisp/ease/` replaced by the installation location on your computer:

```
(setq load-path
      (append (list "~/lisp/ease/" ) load-path))
(setq auto-mode-alist
      (append (list (cons "\\.inp$" 'input-mode))
```

[1] By Emacs, I mean the version from the Free Software Foundation, `http://www.gnu.org`. XEmacs refers to the version from `http://www.xemacs.org`.

[2] A future version of EASE will handle this for the root installation, but the current version does not.

[3] That is a bug that will be fixed in a future version.

```
                     auto-mode-alist))
      (autoload 'ease-mode "ease" t)
      (autoload 'input-mode "input" t)
      (add-hook 'dired-load-hook
                '(lambda () (load-library "ease-dired")))
```

The first two lines tell emacs where to find EASE . The next three lines tells Emacs to use EASE for files ending in '`.inp`'. The two '`autoload`' lines tell Emacs how to start using EASE . The last allows for batch processing of input files using dired mode in Emacs.

2.3 Customizing EASE

There are lots of variables which can be set to customize the appearance and behavior of EASE . The variables are listed in the quick reference card which comes with EASE or can be listed using the '`variable apropos`' function in Emacs or the '`hyper apropos`' function in XEmacs. There are at least four ways to do so, some more convenient than others.

1. At any time while using EASE , you can type *`M-x set-variable`* and respond with the name of the variable you want to change and the value you want to change it too. This solution, however, does not last between Emacs sessions. To make a permanent change you need to do one of the other things in this list.
2. Add lines to your '`.emacs`' file. Take a look at the file '`dot-emacs`' that comes with EASE for an example of this. The best way to set a user variable in '`.emacs`' is

   ```
   (add-hook 'ease-mode-hook '(lambda ()
                 (setq input-comment-char "% ")
                 (setq input-stanza-indent 2)   ))
   ```

   In this example I set two user variables to their default values.
3. Create a file called '`.ease`'. This file is read when a buffer first enters Input mode. It contains Emacs lisp statements for setting the values of the variables. An example of a '`.ease`' file is given with the EASE distribution in a file called '`dot-ease`'. In this example, every user configurable is set to its default value. The lines of the '`.ease`' file look like these two:

   ```
   (setq input-comment-char "% ")
   (setq input-stanza-indent 2)
   ```

   This is a little redundant with the '`.emacs`' file, but it is convenient to have another place to put your EASE customizations and keep clutter out of '`.emacs`'.
4. If you are using GNU Emacs 20 or XEmacs 20, or if you have the '`custom`' package specially installed for version 19 of either program, you can use it to set and save changes to EASE 's user variables. This is the easiest way of customizing EASE or any other Emacs package. Just type *`M-x customize-group`* and answer *`ease`*. You will then be presented with a hypertext buffer for setting the EASE user variables. EASE is in the *Local* customization group.

3 The syntax and appearance of input files

3.1 The syntax of the programs

Each of the programs requires an input file as part of its input structure. This input file is always an ASCII text file which contains all of the information required by the program at run-time. This information might include the names and locations of any other input files, the names to be assigned to output files, and any physical or run-time parameters required by the program.

Although there are some differences in how each program interprets its input file, there are several common rules governing the syntactic structure of the input files. The input files are always parsed. This means that there is considerable freedom in the order in which information is placed in the input file. Each program reads a line of input and searches for specially recognized words called *keywords*. When a recognized keyword is found, the next one or more words in the input file are read and interpreted in a context appropriate to the keyword. For example, in AUTOBK the word '`data`' is used to specify the name of the file containing the input chi(k) data. When AUTOBK encounters the word '`data`' it interprets the following word as the name of a computer file.

In all of the programs keywords and their values are separated by white space. The formal definition of white space in all of the programs is any number of space or tab characters followed by zero or one comma or equals sign followed by any number of spaces or tabs.[1] Here is an example of how this rule might be implemented:

```
    keyword1 = value1 keyword2 = value2
```

In this example, the equals sign ('`=`') is used to emphasize the relationship between the keyword and its value, and a tab is inserted before the second keyword.

In most cases there can be any number of keyword/value pairs on a line and the keywords can come in any order. There are a few notable exceptions to this rule:

1. In FEFF there can only be one keyword per line and it must come at the beginning of the line. Anything on a line after the keyword and its value is ignored by FEFF.
2. In ATOMS the keyword '`atoms`' (or possibly '`basis`') must be the last keyword in the file.
3. In ATOMS the 10 characters following the keyword '`space`' are read as the value of that keyword. This is because space group notation contains spaces, but no space group symbol is longer than 10 characters.
4. In FEFFIT all path parameters and the words '`set`', '`guess`', and '`local`' must be the only keywords on their lines and must come at the beginning of the line.
5. In FEFFIT there is one keyword that contains white space. It is '`next data set`' and is used to denote the boundary between data sets.

For complete details on the keywords recognized by these programs and the syntax of their values, see the documents for each program.

Each of the programs recognizes certain characters as comment characters. This means that any text on a line following one of these characters will be ignored by the program.

[1] In case you are *really* interested, the regular expression is: `[ \t]*[ \t,=][ \t]*`

In FEFF '*' is the comment character. In ATOMS and AUTOBK any of '*', '#', '%', or '!' are comment characters. In FEFFIT all of '*', '#', '%', or '!' are beginning of line comment characters, while '#', '%', or '!' are comment characters in the middle of the line. The reason for this distinction in FEFFIT is that '*' is used in math expressions.

3.2 Additional syntax rules imposed by EASE

The syntax rules described above are an integral part of EASE and are used to determine the behavior of most of its features. There is a major syntactic rule imposed by EASE for FEFFIT input that is not a requirement of the program. This rule is:

> All path parameters sharing a common path index (i.e. all those referring to the same scattering path as calculated by FEFF) must be contiguous in the input file. Such a grouping of path parameters is called a *path paragraph.* The first line in a path paragraph **must** be for the path parameter '`path`', which is used to identify the name of the FEFF output file containing the calculation for that scattering path. Path paragraphs are separated by lines containing only comment and whitespace characters (i.e. space, tab, comma, =, *, #, %, and !).

This definition of the path paragraph is **required** for several of the most powerful editing features in FEFFIT minor mode. I strongly encourage you to observe the syntactic structure of the path paragraph. If you prefer to group together all '`path`' lines separately from all '`sigma2`' lines, then you will not find FEFFIT minor mode to be particularly helpful. Sorry.

3.3 Syntax colorization

Each minor mode has rules for syntax colorization of the text. EASE supports both font-lock and hilit19 for syntax colorization. Keywords are cast in one of several colors, with conceptually similar keywords sharing colors. Comments are set in a different color, which is red by default. See the installation instructions for how to enable your Emacs session to use EASE 's syntax colorization.

If you already use either *font-lock* or *hilit19*, then EASE will display your input files with syntax colorization. If you would like to start using syntax colorization, the set up is easy. To enable syntax coloring of the text of your input files using font-lock, place the following line in your '`.emacs`' file:

```
(global-font-lock-mode t)
```

To enable syntax coloring of the text of your input files using the hilit19 package, place the following lines in your '`.emacs`' file:

```
(cond (window-system
       (setq hilit-mode-enable-list  '(not text-mode)
             hilit-background-mode   'light
             hilit-inhibit-hooks     nil
             hilit-inhibit-rebinding nil)
       (require 'hilit19) ))
```

I find that font-lock offers better performance than '`hilit19`'. Both packages are supported by EASE , although hilit19 is no longer supported by its author.

3.4 Associating programs with input files

Each of the programs requires that the input file have a particular name. For example FEFF requires that its input file be called '`feff.inp`'. You might, however, wish to use file names which are mnemonically associated with the contents of the file. For instance, the '`feff.inp`' file used to model data on metallic copper might be called '`cu-feff.inp`'. Because EASE uses minor modes specific to each program for editing the input files, it is necessary for EASE to determine which for which program each edited input file is intended. To determine this, it first looks at the file name, if the filename is obviously indicative of a particular program (e.g. '`autobk.inp`' is almost certainly intended for use with AUTOBK) then that program is used. If the filename is not obviously indicative, then EASE may prompt you for a program name when you edit the file for the first time. It is important to answer this question correctly. Every input file is edited in INPUT major mode and in a minor mode appropriate to the program. To enable the full functionality of EASE , the associated program must be identified so that the appropriate minor mode can be used.

3.5 Automatic configuration

There are several features in EASE that are routinely set in the course of editing an input files that are convenient to retain between editing sessions. To do this, EASE writes special comment lines to your input files that are read when the file is initially loaded by Emacs and used to set variables which appropriately alter the behavior of EASE . The comment lines are generally written to the end of the input file and begin with a special string of comment characters. They look something like this:

```
!!&& Local Variables:
!!&& input-program-name: "autobk"
!!&& End:
```

There are several variables which are set in this manner. The most important is the one that identifies the program for which the input file is intended. In the example above, the comment line identifies AUTOBK as the program for the input file.

There are currently five other pieces of information that are stored in these special comment lines. These are (1) the directory path to the location of files from a FEFF run, (2) the directory path to input data files, (3) the directory path to out files, (4) the default k-weight to use when chi(k) data is plotted, and (5) the name of the master file for an include file in a multi-file input file. In the future more automatic configuration possibilities may be added. EASE will ignore any other variable values in the list when it updates it's own automatically configured variables.

The automatic configuration lines are updated each time an input file is saved to disk. Each of the variable set by these lines can be altered by functions built into EASE during the course of editing. These are found in the '`Input`' menu and described in the quick reference card. One of the variables in EASE , '`input-prohibit-autoconfig-flag`', can be set to prohibit the writing of the auto-configuration lines. Use this if you object to having EASE insert text into your input file. I recommend, though, that you allow it to do so. It is very convenient to not have to re-enter this information every time the file is edited.

3.6 Keywords and keyword parsing

One of the difficulties of using FEFF and FEFFIT programs is remembering the names of the keywords recognized by the various programs and what values each of the keywords takes. EASE can help you. It has knowledge of all keywords used by each of the programs hardwired in and offers several functions to let you use that knowledge.

Templates Each minor mode offers template functions which insert necessary keywords into your input file with blank spaces for you to fill in appropriate values. Using the templates assures that your input file will at least run the program to completion without neglecting any crucial information. The templates are described in more details in the chapters on the minor modes.

All of the templates in EASE are made using the '`tempo`' package. This means that each place where a value needs to be inserted by the user is a *hotspot*. The hotspots are marked by salmon colored rectangles. Once the hotspots are filled in you can clear the salmon colored rectangles with *C-c C-t c*. The functions '`tempo-forward-mark`' and '`tempo-backward-mark`', bound in EASE to *M-n* and *M-p*, can be used to move among the hotspots.

Keyword completion

The function '`input-complete-keyword`' (normally bound to *M-*⟨tab⟩) will attempt to complete a partially typed keyword. If the string already typed matches only one possible keyword, that keyword will be completed, colored according to syntax, and a brief explanation of the keyword will be offered in the echo area. If the typed string does not match any keyword, you will be told in the echo area. If more than one keyword is matched, all possible matches will be offered in the echo area. As an example, when editing a FEFFIT input file, if you type `g` then *M-*⟨tab⟩, '`guess`' will be inserted into your input file and you will told in the echo area that '`guess`' is used to set the name and initial value of a fitting parameter.

Argument descriptions

The function '`input-arg-for-this-keyword`' (normally bound to *M-?*) will offer a brief description of the keyword underneath the cursor along with a description of the sort of argument it takes. The is written to the echo area.

Verification of keyword values

The function '`input-check-this-keyword-arg`' (normally bound to *M-*⟨ret⟩) will perform a simple check of the value of the keyword underneath the cursor. For example, if the keyword is supposed to take a numeric value, this function will check to see that the value is a number. If the keyword specifies an input data file, it will check to see that the value is the name of a readable file.

Verification of input files

This functionality is not yet a part of EASE , *but in a future version a function will exist to verify an entire input file by repeatedly using* '`input-check-this-keyword-arg`'*. If any mistakes are found, error messages will be written to a second window. I plan to provide a simple way of jumping from messages in the error window to the appropriate point in the input file.*

Display of all possible keywords

The function '`input-display-keywords`' (normally bound to *`C-c C-b k`*) opens a second window and displays all possible keywords for the program associated with the input file. The keywords are tabulated along with brief descriptions.

3.7 Indentation and separation

In the first section of this chapter I described the rules recognized by the programs for separating keywords and values. As long as at least one whitespace character separates words in the input files, the programs are quite content. Merely meeting the minimum requirement, however, will make for a messy looking and hard to read input file. Because you or some other human will eventually read your input file, it is convenient to adopt certain conventions about indenting text and about separating textual elements within the input files.

To make input files easier to read thus easier to understand, EASE uses several configurable rules for determining proper indentation and separation. The choices EASE makes about indentation and separation are context-dependent. That is, the indentation of, say, a line containing a '`guess`'ed variable in FEFFIT may be different from the indentation of a line in an AUTOBK input file. Variables in EASE with names ending in either '`indent`' or '`separate`' control the amount and type of whitespace used in various situation by many EASE functions.

Each of these variables takes an integer value. A positive integer specific how many *spaces* will be inserted as indentation or separation. A negative number specifies the number of *tabs* to insert. For indentation variables, a value of 0 means that no whitespace will be inserted, i.e. the text will be flush against the left side of the screen. For separation variables, a value of 0 will default to -1, i.e. a single tab character. While you are certainly free to choose absurdly large numbers, for example -73 for an indentation value, I strongly recommend against this. For one thing, an indentation of 73 tabs will be ugly and difficult to read. For another, each of the programs has a hard-wired limit on the length of a text line. For example, ATOMS only reads the first 78 characters of each line.

Each minor mode has a '`cleaning`' function which can be used to standardize the appearance of the file. Each cleaning function will alter each line in the file by deleting all existing indentation and separation on the line and inserting the appropriate whitespace as determined from the user configuration variables. All of the variables have defaults that will make any input file tidy and easy to read. The default values for variables used by the FEFF minor mode are chosen to make the file look like one generated by ATOMS.

Several other functions, such as those that insert templates, also use the indentation and separation variables.

3.8 Master files and include files

FEFFIT, AUTOBK, and PHIT allow the use of include files. EASE allows you to specify the relationship between include files and their master files by setting an automatic configuration variable. Currently this variable is used only by a few functions. The program running function will use the master file as the input file. Also the functions for jumping to log and prm files jump to the files appropriate to the master file. Plotting and paragraph manipulation functions do not currently use the master file.

4 Using EASE

4.1 Key Sequences, Menus, and Toolbars

As with most packages in Emacs, there are a number of different ways of accessing all of the functions in EASE . The most primitive manner of accessing functions is to type *M-x* followed by the name of the function. Additionally, most user functions are bound to key sequences. Most of these involve typing *C-c* followed by another control character and then followed by a single character. These are lengthy key sequences. Major modes typically use *C-c* sequences. To have enough options available, I felt it necessary to have all of the EASE functions use a second control prefix.

C- means to strike the control key while striking the following character. *M-* means to strike the ⟨esc⟩ key before striking the following key. Alternatively, you can hold down the ⟨alt⟩ or ⟨meta⟩ key while striking the following key.

The list of key sequences is rather lengthy and has been excluded from this document. The tutorial, see section "Introduction" in *The Ease Tutorial*, demonstrates the use of many functions. All of the key sequences recognized by EASE are listed in the quick reference card.

Virtually all of the functions in EASE are bound to either the `Input` menu or to one of the program menus. Whenever you edit and input file with EASE there will be a pull down menu labels `Input` and one labelled with the name of the program corresponding to that input file.

EASE makes use of the ability of XEmacs to display a toolbar. By default, a toolbar appears at the left side of the screen when a buffer in input-mode is displayed. The toolbar has a several of the most commonly used functions bound to it, including functions for making templates, running programs, plotting output, and examining log files.

4.2 Program Execution

As discussed in the introduction, one of the fundamental purposed of a user interface to a set of programs is providing an environment in which to run the programs. EASE allows you to run any of the programs with a key or mouse sequence. When a program is run, a second window is opened to display all of the run-time messages written to standard output and standard error by the program. This allows you to follow the progress of the program and to see when the program has finished.

The programs are always invoked using the shell's '`time`' command. Upon completion of the program, this causes a summary of the time consumed by the program to be written to the run-time display window. Generally this summary is rather cryptic but usually includes the real time elapsed and the amount of CPU time used. It would be nice to parse this string into a more readable form, but unfortunately its format is different under different shells and operating systems. Even so, it is sufficiently useful to know this information that I offer the unparsed string despite its crypticness.

The advantages of having the run-time messages written to a buffer in the manner described above is that it records a running log of work done during the EASE session. This log can be saved to a file using '`input-save-run-log`' (normally bound to *C-c C-b l* and found in the '`Input:miscellaneous`' menu). This runtime log is also written whenever Emacs is exited so that your last session can be reviewed.

If you are working in a windowing environment, the run-time messages buffer will be displayed in a separate frame from the frame containing the input files. This frame is only about 20 lines tall and lacks much of the normal decoration of an emacs frame. The idea is that it is a display-only frame which takes up only a small amount of screen real estate. This frame may be shared with the gnuplot script buffer, but typically is not. This is controlled with the variable '`input-use-frames-flag`'.

4.2.1 Basic run command

Several functions and features are used by EASE to control program execution.

- The function '`input-run-any-program-any-file`' (normally bound to *C-c C-r* a and found in the '`Input`' menu) is a general purpose program execution function. When you invoke it, you will be prompted for the name of a program to run and the name of a input file to use as input to the program.
- The function '`input-run-this-program-this-file`' (normally bound to *C-c C-r r* and found in the program menu) is used to run the program associated with the current file on the current file. This is the way that programs are most commonly executed in EASE .
- The function '`input-kill-program`' (normally bound to *C-c C-r k* and found in both the '`Input`' and program menus) is used to kill the currently running process.
- Only one program can run at a time. This is, admittedly, a silly restriction, since Unix certainly does not care how many programs are running. The bookkeeping chores involved in running more than one program at a time are rather complicated. Perhaps in a future version I will allow for execution of multiple concurrent programs. Of course, you could run the programs from the command line either in a virtual terminal or in an Emacs shell buffer (*M-x shell* or *C-c s*).
- EASE includes a nifty wrapper for program execution which allows you to name your input files with any name you want and to use them to run the programs without renaming them to the file name required by the program. EASE does this by renaming your input file to the appropriate name, then renaming it back when the program is done. The log file is also renamed to the same name as the input file, but with '`.log`' substituted for '`.inp`'. For FEFFIT the '`prm`' file is also renamed. For ATOMS, if the input files called, say, '`cu.inp`', the FEFF input file generated by the program will be called '`cu-feff.inp`'. Care is taken not to overwrite any existing files. Suppose a file called '`atoms.inp`' already exists. Before the '`cu.inp`' is renamed to '`atoms.inp`' the already existing '`atoms.inp`' will be renamed to a temporary, randomly generated file name. Output files are also renamed in this manner. Note that the output files of FEFF are not protected in this manner in the current version of EASE . Also the output data files of AUTOBK and FEFFIT are not protected (e.g. the output background file from AUTOBK will not be protected by this command wrapper.)
- In the future it will be nice if EASE is able to parse the screen output of each of the programs for error messages and to provide a way of jumping to the place in the input file that cause the problem.

- AUTOBK has a feature not shared by the other three programs. It is possible to run AUTOBK repeatedly with a single input file. Each of the blocks of text controlling a single run of autobk is called a *stanza*. AUTOBK minor mode offers a function for running AUTOBK just on the stanza currently occupied by the cursor. The function '`autobk-run-stanza`' (usually bound to `C-c C-r s` and found in the Autobk menu) does this by copying the current stanza to a file called '`ease-stanza.inp`' then running autobk on that file. The log file for the single stanza run is thus called '`ease-stanza.log`'

4.2.2 Batch processing

EASE is able to batch process input files by sequentially running the appropriate program on any number of input files. This is accomplished using a dired buffer. See the Emacs documentation for details about dired.

To enable EASE 's batch processing capabilities, you must have this lines somewhere in your '`.emacs`' file

```
(add-hook 'dired-load-hook '(lambda () (load "ease-dired")))
```

Once that is done, any time you are in a dired buffer you will be able to run any of the programs covered by EASE . First mark some number of input files using the '`m`' command (or any other marking command) in dired. Then invoke '`ease-dired-run-marked`' by typing `C-c r`. EASE will then load each marked file and run the program appropriate to that file. Just sit back and watch.

Note that the batch run may require some interactive response in certain situations. For example, if EASE cannot figure out the program associated with the input file (see Section 3.5 [Autoconfiguration], page 9), EASE will query you for it.

4.3 Plotting

4.3.1 Using EASE's Plotting Utilities

EASE plots the output of FEFF, FEFFIT, and AUTOBK by parsing for the input file for information about the data that needs to be plotted and constructing a GNUPLOT script based on what it finds. It then pipes the contents of this script to GNUPLOT. The scripts can be saved by moving the cursor to the script buffer and typing `C-x C-s`.

When one of the plotting options is chosen, a second window or a separate frame opens up showing the contents of the GNUPLOT script. Assuming there are no problems, the plot will be displayed and control of the cursor will return to the window containing the input file.

If you are working in a windowing environment, the run-time messages buffer will be displayed in a separate frame from the frame containing the input files. This frame is only about 20 lines tall and lacks much of the normal decoration of an emacs frame. I assume that you only rarely will edit the gnuplot script, so displaying it in a separate and rather small frame seems preferable. This frame may shared with the run-time messages buffer, but typically is not. This behavior is controlled by the variable '`input-use-frames-flag`'.

Plotting in ATOMS mode
: There is nothing to plot in ATOMS, so there are no plotting functions. An interface to a ball-and-stick plotter such as RASMOL or XMOL would sure be nice.

Plotting in FEFF mode
: In FEFF minor mode there are two plotting options. One is to plot chi(k) from '`chi.dat`' and the other is to plot mu and mu0 from '`xmu.dat`'. The scripts GNUFIX and KW are used. Both of these files are written by the fourth module of FEFF. '`xmu.dat`' is only written if the XANES calculation is enabled by having the '`xanes`' keyword in in the input file.

Plotting in AUTOBK mode
: In AUTOBK minor mode there are four plotting options, (1) plot chi(k) as specified in the stanza currently occupied by the cursor, (2) plot the data and AUTOBK's estimation of mu0 from the current stanza along, (3) plot chi(k) together with the fitting standard from the current stanza, (4) over-plot all chi(k) functions from all stanzas in the input file. The KW script is used to apply k-weight to the chi(k).

Plotting in FEFFIT mode
: In FEFFIT minor mode there are options to plot fit results in any of k, R, and back-transformed k spaces. The data and the full fit in the chosen space are always plotted. Plotting in any of the spaces requires that the appropriate control keyword ('`kspout`', '`rspout`', or '`qspout`') be set to true.

  By default only the data at full fit are shown. If FEFFIT's background fitting option is selected by setting '`bkgout`' to true, then the background function will be added to the plot. The individual paths can also be plotted by marking path paragraphs that you want to plot. This is done with the *`C-c C-p m`* command. When a path is marked, the '`path`' keyword will be highlighted and when any of the three plotting functions is called, the marked paths will be added to the GNUPLOT script. Hitting *`C-c C-p m`* again clears the mark. *`S-mouse-2`* marks and unmarks the paragraph under the mouse cursor. *`C-c C-p a`* marks all paths in the current data set and *`C-c C-p c`* clears all marks.

Using GNUPLOT mode
: The GNUPLOT script is written in two parts to a buffer which is, by default, called '`input-mode.gp`'. The first part initially has no GNUPLOT commands in it and is never deleted in subsequent plots. The second part contains all of the commands for the requested plot. The second part of the script is deleted for each subsequent plot. These two parts are separated by a line beginning with these characters: '`#-#-#-`'. Any GNUPLOT commands that you wish to issue with each plot can be typed into the first part of the script.

  The automatically generated scripts are always plotted when one of the plotting options is chosen by key or mouse sequence. You can interact with GNUPLOT directly by moving the cursor to the buffer containing the script. You can then edit the script as you wish and re-send it to GNUPLOT. There are three key sequences for communicating commands to GNUPLOT. '`send-buffer-to-gnuplot`', bound to *`C-c C-b`* and in the '`gnuplot`' menu, sends all commands in the buffer to GNUPLOT. '`send-region-to-gnuplot`', bound to *C-c C-r* and in the '`gnuplot`' menu, sends all commands in a selected region to GNUPLOT. Finally, '`send-line-to-gnuplot`', bound to *C-c C-l* and in the '`gnuplot`' menu, sends all commands on the line occupied by the cursor to GNUPLOT.

EASE adds a specialized history list to gnuplot-mode. Every time and entire script is sent to GNUPLOT, that script gets pushed onto a history list. You can navigate up and down the script history list, thus revisiting previous plots, by using *C-c C-p* and *C-c C-n* in the gnuplot-mode buffer or by hitting the up and down arrow icons in the XEmacs toolbar. The depth of the history list is determined by the value of `ease-gnuplot-history`.

Hitting *C-c C-f k* will terminate the GNUPLOT process and close the frame containing the gnuplot script.

For more details about using gnuplot-mode, see the reference card '`gpelcard.ps`' which comes with EASE .

Scripts used in plotting

Several scripts external to Emacs are used in the EASE plotting utilities, KW, GNUFIX, and MR. With Unix, GNUPLOT is able to pipe data through external scripts using the system '`popen`' function. This is the purpose of the '`<`' character that appears in several of EASE 's plotting scripts.

KW is a simple AWK script for k-weighting chi(k) data. It simply multiplies each y-value in the data file by some power of the x-value. GNUFIX is a Bourne shell script containing a single SED command. It prepends a comment character ('`#`') to each line in the header of '`chi.dat`' or '`xmu.dat`' so that these files may be properly plotted using GNUPLOT. MR is a Bourne shell script used as an interface to a Fortran program for reading a record from a UWXAFS binary file.

Another script, ESHIFT, comes with EASE although it is not used in any of the built-in script generating functions. The purpose of this script is to e0-shift mu(E) data. In GNUPLOT it works in a manner analogous to KW. The following GNUPLOT command plots some data with a 5 volt e0 shift:

```
pl '<eshift esh=5 data.xmu'
```

Note that any of these scripts can be used at the command line by redirecting their output to a file. For example:

```
# kw w=5 data.chi > data.chi.weighted
# gnufix esh=5 xmu.dat > xmu.data.fixed
# eshift esh=5 data.xmu > data.xmu.shifted
# mr data.xmu, 1 > data.xmu.ascii
```

These scripts are all in the '`scripts`' directory in the distribution. The directory containing these scripts is the value of `ease-bin-location`. This directory is added to the `exec-path` when EASE is started.

4.3.2 Limitations to EASE's Plotting Utilities

EASE is supposed to have a mechanism for detecting problems with the commands sent to GNUPLOT. In truth, I never got this to work properly and found that it was not entirely necessary. If something goes wrong with the plot, look at the '`gnuplot-program`' buffer by simply switching to that buffer or by using the '`gnuplot-jump-to-gnuplot-buffer`' function, normally bound to *`C-c C-e`* and in the '`Gnuplot`' menu in the gnuplot-buffer.

EASE 's plotting functions are able to interact with either ASCII data or data in the UWXAFS binary format. Because of certain features of the UWXAFS binary format, the GNUPLOT scripts generated by EASE may not be what you want. For example, a script for a FEFFIT fit is written assuming that the output data was written to a new file rather than appended to a pre-existing file. Thus the fit script uses numeric keys ('`nkeys`') beginning with 1.

Interaction with the UWXAFS binary files is one of the weaker areas in EASE . In truth, I rarely use the binary files and my memory of how the programs read and write them is a bit fuzzy. In particular, I don't recall how AUTOBK writes its many possible output files when binary output is used. I have a few ideas about how to improve plotting from the binary files. It could be possible to set an nkey offset variable to handle pre-existing files. Another possibility would be something like Dired mode in Emacs, where the contents of the binary file are displayed to a buffer and records could be marked and subsequently plotted.

5 Glossary of terms

Here are definitions of terms found in this document. Please note that these are not dictionary definitions, but rather what they mean in the context of using EASE .

- **Cleaning**: Fixing the indentation and separation on every line in a region or an input file.
- **Data set**: The portion of a FEFFIT input file devoted to fitting a particular data file. FEFFIT allows the user to co-refine parameters using multiple data sets. A data set is delimited by the top of the buffer, the bottom of the buffer or the keyword '`next data set`'.
- **Indentation**: Whitespace inserted at the beginning of a line. Different textual areas in input files use different, user-configurable rules to determine the type and amount of whitespace.
- **Keyword**: A word recognized by a program used to signal the program how to interpret the following word or words.
- **List**: A region of text in an input file whose format is rigidly interpreted by the program. Typically alist begins with a line containing a keyword telling the program to interpret the following lines as a list.
- **Path paragraph**: A region of text in a FEFFIT input file used to set the path parameters associated with a particular scattering path. The path paragraph must begin with the '`path`' keyword and is delimited by lines containing only whitespace or comment characters.
- **Separation**: Whitespace inserted between textual areas in an input file for the purpose of enhancing readability. Typically separation is inserted between columns in a list or between fields in a '`set`', '`guess`', or '`local`' line in a FEFFIT input file. Different textual areas in input files use different, user-configurable rules to determine the type and amount of whitespace.
- **Snag**: Insertion of text at the location of the cursor that is identical to that found in a similar textual area elsewhere in the input file.
- **Stanza**: The unit of text in an AUTOBK used to describe the background removal from a single data file. An AUTOBK input file can have any number of stanzas. For the sake of indentation and other functions, any part of an input file that is not otherwise defined is considered to be part of a stanza.
- **Swap**: Exchange of a portion of text with related, possibly opposite text. For example to exchange the keyword '`guess`' with '`set`' in a FEFFIT input file.
- **Template**: A unit of text containing one or more keywords without their values. Templates are used in EASE as a guideline for constructing a minimal input file.
- **Visit**: Open a new buffer containing a file related to the current input file. There are functions for visiting log files, data files, and other sorts of files. Also referred to as *jumping*.

6 Future work

I do things in EASE as I need them for my own work. That puts a severe limit on how much gets done. If I ever develop a user base, some of the ideas that I have that I can live without would perhaps get done. I find Emacs to be a really wonderful application interface and I have not really used all of its potential or implemented all of my ideas. Any of these things may happen some day. If I ever get some kind of real user base, that would be incentive for finishing off this list and even tackling some new problems.

File parsing
Do a simple syntax checking of the entire buffer – make sure that values match the expectation of a keyword, that named files exist, that keywords are not dangling, and the like. This already works for individual keywords, but I would like EASE to check entire files. A display of problems and a facility for jumping to the location of the problem, *a la* compilation-mode, would be great.

Execution parsing
The run-time output of all the programs is displayed in a run-time buffer. This output could be parsed in the event of an unsuccessful run much in the manner of the standard compilation utility which allows you jump to the problem line in the source code at a keystroke. The inconsistent screen messages of FEFF and the other programs make this difficult.

Better UWXAFS binary integration
A dired-like mode for interacting with uwxafs binary files (or even a packed ASCII representation of the many-records-one-file concept) would be quite slick. This is way down on my priority list because (1) I rarely use the UWXAFS format and (2) I would rather see it go away (or be replaced by something better, packed ASCII, zip, other?).

More plotting options
Gnuplot is really rather primitive, but it is clean and easy to interact with. Another freely available and fairly common plotting option is XMGR. It really would not be too hard to make an XMgr interface in a similar vein to the current gnuplot interface. XMgr has the advantages of, among other things, a cursor and focusing. Others possibilities are PGPLOT integration and . . . ?

NTEmacs integration
It'd be pretty cool if EASE worked on NT. I can't imagine it would take *that* much work.

and . . . I'm open to suggestions!

Index

Table of Contents